A^2CiD^2: Accelerating Asynchronous Communication in Decentralized Deep Learning

Adel Nabli
Concordia University, Mila
Sorbonne University, ISIR, CNRS
adel.nabli@sorbonne-universite.fr

Eugene Belilovsky
Concordia University, Mila

Edouard Oyallon
Sorbonne University, ISIR, CNRS

Abstract

Distributed training of Deep Learning models has been critical to many recent successes in the field. Current standard methods primarily rely on synchronous centralized algorithms which induce major communication bottlenecks and synchronization locks at scale. Decentralized asynchronous algorithms are emerging as a potential alternative but their practical applicability still lags. In order to mitigate the increase in communication cost that naturally comes with scaling the number of workers, we introduce a principled asynchronous, randomized, gossip-based optimization algorithm which works thanks to a continuous local momentum named $\mathbf{A^2CiD^2}$. Our method allows each worker to continuously process mini-batches without stopping, and run a peer-to-peer averaging routine in parallel, reducing idle time. In addition to inducing a significant communication acceleration at no cost other than adding a local momentum variable, minimal adaptation is required to incorporate $\mathbf{A^2CiD^2}$ to standard asynchronous approaches. Our theoretical analysis proves accelerated rates compared to previous asynchronous decentralized baselines and we empirically show that using our $\mathbf{A^2CiD^2}$ momentum significantly decrease communication costs in poorly connected networks. In particular, we show consistent improvement on the ImageNet dataset using up to 64 asynchronous workers (A100 GPUs) and various communication network topologies.

1 Introduction

As Deep Neural Networks (DNNs) and their training datasets become larger and more complex, the computational demands and the need for efficient training schemes continues to escalate. Distributed training methods offer a solution by enabling the parallel optimization of model parameters across multiple workers. Yet, many of the current distributed methods in use are synchronous, and have significantly influenced the design of cluster computing environments. Thus, both the environments and algorithms rely heavily on high synchronicity in machine computations and near-instantaneous communication in high-bandwidth networks, favoring the adoption of centralized algorithms [7].

However, several studies [27, 44, 2, 28] are challenging this paradigm, proposing decentralized asynchronous algorithms that leverage minor time-delays fluctuations between workers to enhance the parallelization of computations and communications. Unlike centralized algorithms, decentralized approaches allow each node to contribute proportionally to its available resources, eliminating the necessity for a global central worker to aggregate results. Combined with asynchronous peer-to-peer (p2p) communications, these methods can streamline the overall training

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

process, mitigating common bottlenecks. This includes the Straggler Problem [42], the synchronization between computations and communications [9], or bandwidth limitations [47], potentially due to particular network topologies like a ring graph [43]. However, due to the large number of parameters which are optimized, training DNNs with these methods still critically requires a considerable amount of communication [22], presenting an additional challenge [32]. This work aims to address these challenges by introducing a principled acceleration method for pair-wise communications in peer-to-peer training of DNNs, in particular for cluster computing. While conventional synchronous settings accelerate communications by integrating a Chebychev acceleration followed by Gradient Descent steps [37], the potential of accelerated asynchronous pair-wise gossip for Deep Learning (DL) remains largely unexplored. Notably, the sophisticated theory of Stochastic Differential Equations (SDEs) offers an analytical framework for the design and study of the convergence of these algorithms [12]. We introduce a novel algorithm **A^{2}CiD^2** (standing for **A**ccelerating **A**synchronous **C**ommunication **i**n **D**ecentralized **D**eep Learning) that requires minimal overhead and effectively decouples communications and computations, accelerating pair-wise communications via a provable, accelerated, randomized gossip procedure based on continuous momentum (i.e., a mixing ODE) and time [12, 34]. We emphasize that beyond the aforementioned hardware superiority, stochastic algorithms also allows us to theoretically reach sublinear rates in convex settings [10], which opens the possibility to further principled accelerations. In practice, our method enables a virtual doubling of the communication rate in challenging network topologies without any additional cost, simply by maintaining a local momentum variable in each worker (see Fig. 1).

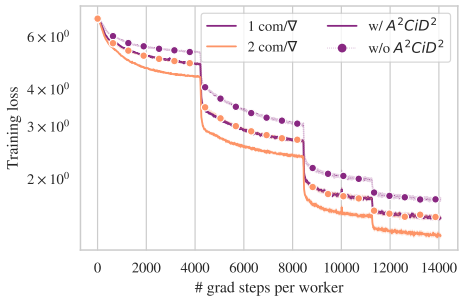


Figure 1: Adding **A^{2}CiD^2** has the same effect as doubling the communication rates on ImageNet on the ring graph with 64 workers. See Sec. 4.

Our key contributions are as follows: **(1)** We extend the continuized framework [12] to the non-convex setting, in order to obtain a neat framework to describe asynchronous decentralized DL training. **(2)** This framework allows us to refine the analysis of a baseline asynchronous decentralized optimization algorithm. **(3)** We propose a novel and simple continuized momentum which allows to significantly improve communication efficiency in challenging settings, which we name **A^{2}CiD^2**. **(4)** We demonstrate that our method effectively minimizes the gap between centralized settings in environments hosting up to 64 asynchronous GPUs. **(5)** Our code is implemented in Pytorch [35], remove locks put on previous asynchronous implementations by circumventing their deadlocks, and can be found in an open-source repository: https://github.com/AdelNabli/ACiD.

This paper is structured as follows: Sec. 3.1 outlines our model for asynchronous decentralized learning, while Sec. 3.2 discusses the training dynamic used to optimize our Deep models. Sec. 3.4 offers a comprehensive theoretical analysis of our method, which is validated empirically in Sec. 4.

2 Related Work

Large-scale distributed DL. Two paradigms allow to maintain high-parallelization. On one side, model-parallelism [9, 25], which splits a neural network on independent machines, allowing to use local learning methods [4, 3]. On the other hand data-parallelism, which accelerates learning by making use of larger mini-batch splitted across multiple nodes [38] to maximally use GPU capacities. This parallelization entailing the use of larger batch-sizes, it requires an important process of adapting hyper-parameters [16], and in particular the learning rate scheduler. Developed for this setting, methods such as [16, 46] allow to stabilize training while maintaining good generalization performances. However, they have been introduced in the context of centralized synchronous training using All-Reduce schemes for communication, which still is the default setting of many approaches to data parallelism.

Decentralized DL. The pioneer work [27] is one of the first study to suggest the potential superiority of synchronous decentralized training strategies in practice. In terms of implementation in the cluster

setting, decentralized frameworks have been shown to achieve higher throughput than optimized All-Reduce strategies [45, 38]. From the theoretical side, [21] propose a framework covering many settings of synchronous decentralized learning. However, as it consistently relies on using a global discrete iterations count, the notion of time is more difficult to exploit, which reveals crucial in our setting. Furthermore, no communication acceleration is incorporated in these algorithms. [22] provides a comprehensive methodology to relate the consensus distance, *i.e.* the average distance of the local parameters to the global average, to a necessary communication rate to avoid degrading performance and could be easily applied to our method. [29] is a method focusing on improving performance via a discrete momentum modification, which indicates momentum variables are key to decentralized DL.

Asynchronous Decentralized DL. There exist many attempts to incorporate asynchrony in decentralized training [48, 5, 8, 28, 2], which typically aim at removing lock barriers of synchronous decentralized algorithms. To the best of our knowledge, none of them introduce communication acceleration, yet they could be simply combined with our approach. Although recent approaches such as [2, 28] perform peer-to-peer averaging of parameters instead of gradients, thus allowing to communicate *in parallel* of computing (as there is no need to wait for the gradients before communicating), they are still coupled: parameter updates resulting from computations and communications are scheduled in a specific order, limiting their speed. Furthermore, in practice, both those works only implement a periodic averaging on the exponential graph (more favorable, see [43]) instead of investigating the influence of the graph's topology on the convergence of a randomized gossip method, as we do. In fact, AD-PSGD [28], the baseline algorithm in asynchronous decentralized DL, comes with a major caveat to avoid deadlocks in practice: they *require* a bipartite graph and schedule p2p communications in a pseudo-random manner instead of basing the decision on worker's current availability, hindering the advantage given by asynchronous methods in the mitigation of stragglers. Contrary to them, our implementation allows to pair workers in real time based on their availability, minimizing idle time for communications.

Communication reduction. Reducing communication overhead is an important topic for scalability [36]. For instance, [19, 20] allow to use of compression factor in limited bandwidth setting, and the local SGD communication schedule of [30] is shown to be beneficial. Those methods could be independently and simply combined with ours to potentially benefit from an additional communication acceleration. By leveraging key properties of the resistance of the communication network [14], [12] showed that standard asynchronous gossip [6] can be accelerated, even to give efficient primal algorithms in the convex setting [34]. However, this acceleration has never been deployed in the DL context, until now. RelaySum [41] is an approach which allow to average exactly parameters produced by different time steps and thus potentially delayed. However, it requires either to use a tree graph topology, either to build ad-hoc spanning trees and has inherent synchronous locks as it averages neighbor messages in a specific order.

Notations: Let $n \in \mathbb{N}$ and $d \in \mathbb{N}$ an ambient dimension, for $x = (x^1; \dots; x^n) \in \bigotimes_{i=1}^n \mathbb{R}^d$, we write $\bar{x} = \frac{1}{n}\sum_{i=1}^n x^i$ and $\mathbf{1}$ the tensor of ones such that $x = \frac{1}{n}\mathbf{1}^\top x$. $(\Omega, \mathcal{A}, P)$ is a probability space with measure P. $f(t) = O(1)$ means there is a $C > 0$ such that for t large enough, $|f(t)| \leq C$, whereas $\tilde{O}$-notation hides constants and polylogarithmic factors..

3 Method

3.1 Model for a decentralized environment

We consider a network of n workers whose connectivity is given by edges E. Local computations are modeled as (stochastic) point-wise processes N_t^i, and communications between nodes $(i, j) \in E$ as M_t^{ij}. We assume that the communications are symmetric, meaning that if a message is sent from node i to j, then the reverse is true. In practice, such processes are potentially highly correlated and could follow any specific law, and could involve delays. For the sake of simplicity, we do not model lags, though it is possible to obtain guarantees via dedicated Lyapunov functions [13]. In our setting, we assume that all nodes have similar buffer variables which correspond to a copy of a common model (e.g., a DNN). For a parameter x, we write x_t^i the model's parameters at node i and time t and $x_t = (x_t^1; \dots; x_t^n)$ their concatenation. In the following, we assume that each worker computes

Table 1: Comparison of convergence rates for strongly convex and non-convex objectives against concurrent works in the fixed topology setting. We neglect logarithmic terms. Observe that thanks to the maximal resistance $_2$ $_1$, our method obtains substantial acceleration for the bias term. Moreover, while our baseline is strongly related to AD-PSGD [28], our analysis refines its complexity when workers sample data from the same distribution.

Method	**Strongly Convex**	**Non-Convex**
Koloskova et al. [21]	$\frac{2}{n\ 2} + \sqrt{L}\frac{1+\sqrt{1}}{3=2\sqrt{}} + \frac{L}{}\ 1$	$\frac{L\ 2}{n\ 2} + L\frac{1+\sqrt{1}}{3=2} + \frac{L\ 1}{}$
AD-PSGD [28]	-	$L\frac{2+\ 2}{2} + \frac{n^2 L\ 1}{}$
Baseline **(Ours)**	$\frac{2+\ 1\ 2}{2} + \frac{L}{}\ 1$	$L\frac{2+\ 1\ 2}{2} + \frac{L\ 1}{}$
A^{2}CiD^2 (Ours)	$\frac{2+\sqrt{\ 1\ 2}\ 2}{2} + \frac{L}{}\sqrt{\ 1\ 2}$	$L\frac{2+\sqrt{\ 1\ 2}\ 2}{2} + \frac{L\sqrt{\ 1\ 2}}{}$

about 1 mini-batch of gradient per unit of time (not necessarily simultaneously), which is a standard homogeneity assumption [18], and we denote by ^{ij} the instantaneous expected frequency of edge $(i; j)$, which we assume time homogeneous.

Definition 3.1 (Instantaneous expected Laplacian)**.** We define the Laplacian as:

$$, \sum_{(i;j)2E} {}^{ij}(e_i \quad e_j)(e_i \quad e_j)^{\mathsf{T}} : \tag{1}$$

In this context, a natural quantity is the algebraic connectivity [6] given by:

$$_1 , \sup_{kxk=1;x?1} \frac{1}{x^{\mathsf{T}} \quad x} : \tag{2}$$

For a connected graph ($i.e.$, $_1 < +1$), we will also use the maximal resistance of the network:

$$_2 , \frac{1}{2} \sup_{(i;j)2E} (e_i \quad e_j)^{\mathsf{T}} \quad ^+(e_i \quad e_j) \qquad _1 : \tag{3}$$

The next sections will show that it is possible to accelerate the asynchronous gossip algorithms from $_1$ to $\sqrt{\ _1\ _2}$ $_1$, while [12] or [34] emphasize the superiority of accelerated asynchronous gossips over accelerated synchronous ones.

3.2 Training dynamic

The goal of a typical decentralized algorithm is to minimize the following quantity:

$$\inf_{x2\mathbb{R}^d} f(x) , \inf_{x2\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} f_i(x) = \inf_{x_i=x_1} \frac{1}{n}\sum_{i=1}^{n} f_i(x_i) :$$

For this, we follow a first order optimization strategy consisting in using estimates of the gradient $r f_i(x_i)$ via i.i.d unbiased Stochastic Gradient (SG) oracles given by $r F_i(x_i; \ _i)$ s.t. $\mathsf{E}_{\ _i}[r F_i(x_i; \ _i)] = r f_i(x_i)$. The dynamic of updates of our model evolves as the following SDE, for $; \ ; \ ; \tilde{}$ some time-independent scalar hyper-parameters, whose values are found in our theoretical analysis and used in our implementation, and $dN_t^i(\ _i)$ some point processes on R_+ with intensity $dt \quad dP$:

$$dx_t^i = (\tilde{x}_t^i \quad x_t^i)dt \qquad \int_{\Xi} r F_i(x_t^i; \ _i)\, dN_t^i(\ _i) \qquad \sum_{j;(i;j)2E} (x_t^i \quad x_t^j) dM_t^{ij} ; \tag{4}$$

$$d\tilde{x}_t^i = (x_t^i \quad \tilde{x}_t^i)dt \qquad \int_{\Xi} r F_i(x_t^i; \ _i)\, dN_t^i(\ _i) \quad \tilde{} \sum_{j;(i;j)2E} (x_t^i \quad x_t^j) dM_t^{ij} :$$

We emphasize that while the dynamic Eq. 4 is formulated using SDEs [1], which brings the power of the continuous-time analysis toolbox, it is still *event-based* and thus discrete in nature. Hence, it can efficiently model practically implementable algorithms, as shown by Algo. 1. The coupling $\{x_t; \bar{x}_t\}$ corresponds to a momentum term which will be useful to obtain communication acceleration as explained in the next section. Again, $\int_\Xi r F_i(x_t^i; \xi_i) dN_t^i(\xi_i)$ will be estimated via i.i.d SGs sampled as N_t^i spikes. Furthermore, if $x_0 = \bar{x}_0$, then, $x_t = \bar{x}_t$ and we obtain a tracker of the average across workers which is similar to what is achieved through Gradient Tracking methods [19]. This is a key advantage of our method to obtain convergence guarantees, which writes as:

$$dx_t = \frac{1}{n}\sum_{i=1}^{n}\int_\Xi r F_i(x_t^i; \xi_i) dN_t^i(\xi_i): \tag{5}$$

3.3 Informal explanation of the dynamic through the Baseline case

To give some practical intuition on our method, we consider a baseline asynchronous decentralized dynamic, close to AD-PSGD [28]. By considering $\alpha = 0; \alpha = \tilde{\alpha} = \frac{1}{2}$, the dynamic (4) simplifies to:

$$dx_t^i = \int_\Xi r F_i(x_t^i; \xi_i) dN_t^i(\xi_i) \quad \frac{1}{2}\sum_{j;(i;j)\in E}(x_t^i \quad x_t^j) dM_t^{ij}: \tag{6}$$

In a DL setting, x^i contains the parameters of the DNN hosted on worker i. Thus, (6) simply says that the parameters of the DNN are updated either by taking local SGD steps, or by pairwise averaging with peers $j; (i; j) \in E$. These updates happen independently, at random times: although we assume that all workers compute gradients at the same speed *on average* (and re-normalized time accordingly), the use of Poisson Processes model the inherent variability in the time between these updates. However, the p2p averaging depends on the capabilities of the network, and we allow each link $(i; j)$ to have a different bandwidth, albeit constant through time, modeled through the frequency λ_{ij}. The gradient and communication processes are decoupled: there is no need for one to wait for the other, allowing to compute stochastic gradients uninterruptedly and run the p2p averaging in parallel, as illustrated by Fig.2. Finally, (4) adds a momentum step mixing the local parameters x^i and momentum buffer $\bar{x}^i$ before each type of update, allowing for significant savings in communication costs, as we show next.

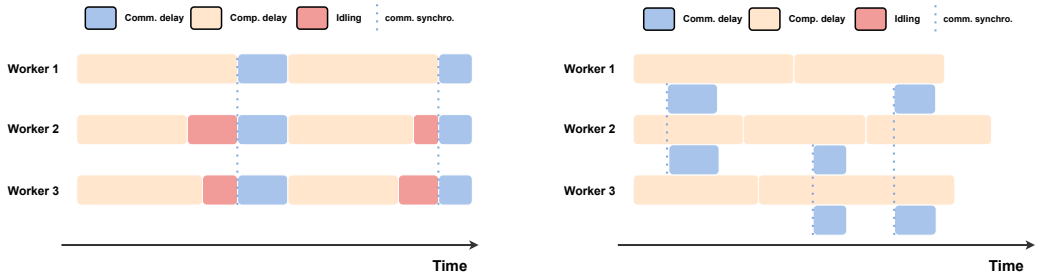


Figure 2: Example of worker updates in synchronous (**left**) and asynchronous (**right**) optimization methods. We remark that our asynchronous algorithm reduces idle time, and allow to communicate *in parallel* of computing gradient, only synchronizing two workers at a time for averaging parameters. Here, one p2p communication is performed per computation *in expectation*.

3.4 Theoretical analysis of A^{2}CiD^2

We now provide an analysis of our decentralized, asynchronous algorithm. For the sake of simplicity, we will consider that communications and gradients spike as Poisson processes:

Assumption 3.2 (Poisson Processes)**.** N_t^i, M_t^{ij} are independent, Point-wise Poisson Processes. The $\{N_t^i\}_{i=1}^n$ have a rate of 1, and for $(i; j) \in E$, M_t^{ij} have a rate λ_{ij}.

We also assume that the communication network is connected during the training:

Assumption 3.3 (Strong connectivity)**.** We assume that $_1 < 1$.

We will now consider two generic assumptions obtained from [21], which allow us to specify our lemma to convex and non-convex settings. Note that the non-convex Assumption 3.5 generalizes the assumptions of [28], by taking $M = P = 0$.

Assumption 3.4 (Strongly convex setting)**.** Each f_i is -strongly convex and L-smooth, and:

$$\frac{1}{n}\sum_{i=1}^{n} \mathsf{E}_{\ i}[k\ r\ F_i(x;\ _i) \quad r\ f_i(x)k^2] \quad ^2 \text{ and } \frac{1}{n}\sum_{i=1}^{n} k\ r\ f_i(x\ ) \quad r\ f(x\ )k^2 \quad ^2:$$

Assumption 3.5 (Non-convex setting)**.** Each f_i is L-smooth, and there exists $P; M > 0$ such that:

$$8x \in \mathbb{R}^d; \frac{1}{n}\sum_{i=1}^{n} k\ r\ f_i(x) \quad r\ f(x)k^2 \quad ^2 + Pk\ r\ f(x)k^2;$$

and,

$$8x_1; :::; x_n \in \mathbb{R}^d; \frac{1}{n}\sum_{i=1}^{n} \mathsf{E}_{\ i}k\ r\ F_i(x_i;\ _i) \quad r\ f_i(x_i)k^2 \quad ^2 + \frac{M}{n}\sum_{i=1}^{n} k\ r\ f_i(x_i)k^2:$$

We can now state our convergence guarantees. An informal way to understand our proposition, is that while gradient updates are non-convex, the communication updates are linear and thus benefit from local convexity; its proof is delayed to Appendix C.

Proposition 3.6 (Convergence guarantees.)**.** *Assume that $fx_t; x_tg$ follow the dynamic Eq. 4 and that Assumption 3.2-3.3 are satisfied. Assume that $\mathbf{1}x_0 = x_0 = x_0$ and let T the total running time. Then:*

- ***Non-accelerated setting**, we pick $= 0;\ = \sim = \frac{1}{2}$ and set $= {}_1$,*
- ***Acceleration (A^{2}CiD^2)**, we set $= \frac{p\ 1}{2^{p\ \frac{1}{1\ 2}}};\ = \frac{1}{2}; \sim = \frac{1}{2}\sqrt{\frac{-1}{2}}$, and $= p\frac{}{1\ 2}\ {}_1$.*

Then, there exists a constant step size > 0 such that if:

- *(**strong-convexity**) the Assumption 3.4 is satisfied, then $\frac{1}{16L(1+\ )}$ and:*

$$\mathsf{E}\left[kx_T \quad x\ k^2\right] = \mathcal{O}\left(kx_0 \quad x\ k^2 e^{\ \frac{T}{16L(1+\ )}} + \frac{^2 + \ ^2(1+\ )}{^2 T}\right);$$

- *(**non-convexity**) the Assumption 3.5 is satisfied, then there is $c > 0$ which depends only on $P; M$ from the assumptions such that $\frac{c}{L(\ +1)}$ and:*

$$\frac{1}{T}\int_0^T \mathsf{E}\left[k\ r\ f(x_t)k^2\right]\ dt = O\left(\frac{L(1+\ )}{T}(f(x_0) \quad f(x\ )) + \sqrt{\frac{L(f(x_0) \quad f(x\ ))}{T}(\ ^2 + (1+\ )\ ^2)}\right):$$

Also, the expected number of gradient steps is nT and the number of communications is $\frac{\mathrm{Tr}(\Lambda)}{2}T$.

Tab. 1 compares our convergence rates with concurrent works. Compared to every concurrent work, the bias term of $\mathbf{A}^2\mathbf{CiD}^2$ is smaller by a factor $\sqrt{\frac{-1}{2}}$ 1 at least. Yet, as expected, in the non-accelerated setting, we would recover similar rates to those. Compared to [20], the variance terms held no variance reduction with the number of workers; however, this should not be an issue in a DL setting, where it is well-known that variance reduction techniques degrade generalization during training [15]. Comparing directly the results of [2] is difficult as they only consider the asymptotic rate, even if the proof framework is similar to [28] and should thus lead to similar rates of convergence.

3.5 Informal interpretation and comparison with decentralized synchronous methods

Here, we informally discuss results from Prop. 3.6 and compare our communication rate with state-of-the-art decentralized synchronous methods such as DeTAG [31], MSDA [37] and OPAPC [23].

As we normalize time so that each node takes one gradient step per time unit in expectation, one time unit for us is analogous to one round of computation (one "step") for synchronous methods. Synchronous methods such as [31, 37, 23] perform multiple rounds of communications (their *Accelerated Gossip* procedure) between rounds of gradient computations by using an inner loop inside their main loop (the one counting "steps"), so that the graph connectivity do not impact the total number of "steps" necessary to reach -precision. As Prop. 3.6 shows, the quantity $1 + \sqrt{\ _1[\ ]\ _2[\ ]}$ is a factor in our convergence rate. containing the information of both the topology E and the edge communication rates $_{ij}$, this is analogous to saying $\sqrt{\ _1[\ ]\ _2[\ ]} = O(1)$ for our method (*i.e.*, the graph connectivity does not impact the time to converge), which, given the graph's topology, dictates the communication rate, see Appendix D for more details. Tab. 2 compares the subsequent communication rates with synchronous methods.

Table 2: # of communications per "step"/time unit on several graphs.

Method	Star	Ring	Complete
Accelerated Synchronous (*e.g.*, [31, 37, 23])	$n^{3=2}$	n^2	n^2
$\mathbf{A^2CiD^2}$	n	n^2	n

4 Numerical Experiments

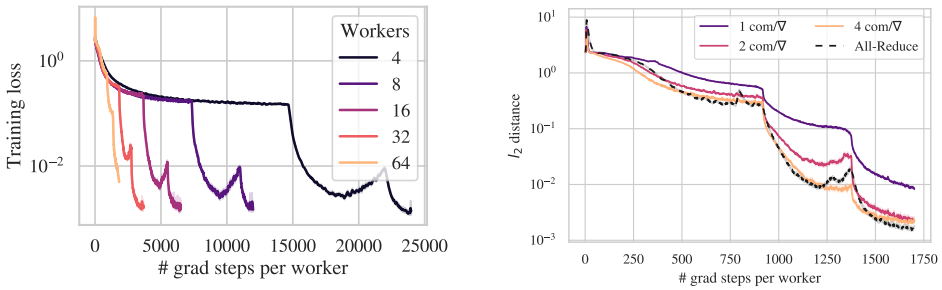
Now, we experimentally compare $\mathbf{A^2CiD^2}$ to a synchronous baseline All-Reduce SGD (AR-SGD, see [26]) and an *asynchronous baseline* using randomized pairwise communications (a variant of AD-PSGD [28], traditionally used in state-of-the-art decentralized asynchronous training of DNNs). In our case, the *asynchronous baseline* corresponds to the dynamic Eq. (6). Our approach is standard: we empirically study the decentralized training behavior of our asynchronous algorithm by training ResNets [17] for image recognition. Following [2], we pick a ResNet18 for CIFAR-10 [24] and ResNet50 for ImageNet [11]. To investigate how our method scales with the number of workers, we run multiple experiments using up to 64 NVIDIA A100 GPUs in a cluster with 8 A100 GPUs per node using an Omni-PAth interconnection network at 100 Gb/s, and set one worker per GPU.

4.1 Experimental Setup

Hyper-parameters. Training a DNN using multiple workers on a cluster requires several adaptations compared to the standard setting. As the effective batch-size grows linearly with the number of workers n, we use the learning-rate schedule for large batch training of [16] in all our experiments. Following [30], we fixed the local batch size to 128 on both CIFAR-10 and ImageNet.

Table 3: Training times on CIFAR10 (6s).

	n	4	8	16	32	64
Ours	t (min)	**20.9**	**10.5**	**5.2**	**2.7**	**1.5**
AR	t (min)	21.9	11.1	6.6	3.2	1.8



(a) (b)

Figure 3: (a) Training loss for CIFAR10 with minibatch size 128 on the complete graph, w/o $\mathbf{A^2CiD^2}$. As the number of worker increases, the loss degrades, especially for $n = 64$. (b) Focus on the training loss for the complete graph of size $n = 64$, w/o $\mathbf{A^2CiD^2}$. As the rate of communication increases, the gap with All-Reduce decreases. With 2 com/ r , a test accuracy of 94:6 0:04 is reached.

Our goal being to divide the compute load between the n workers, all methods access the same total amount of data samples, regardless of the number of local steps. On CIFAR-10 and ImageNet, this

Algorithm 1: This algorithm block describes our implementation of our Asynchronous algorithm with $\mathbf{A}^2\mathbf{CiD}^2$ on each local machine. p2p comm. and r comp. are run independently in parallel.

Input: On each machine $i \in \{1,\dots,n\}$, gradient oracle $r f_i$, parameters $\ ;\ ;\eta;\ ;T$.

Initialize on each machine $i \in \{1,\dots,n\}$:
  Initialize x^i, $\tilde{x}^i \leftarrow x^i; t^i \leftarrow 0$ and **put** $x^i; \tilde{x}^i; t^i$ in shared memory;
  Synchronize the clocks of all machines ;
In parallel *on workers* $i \in \{1,\dots,n\}$, **while** $t < T$, **continuously do:**
  In one thread *on worker* i **continuously do:**
    $t \leftarrow$ *clock*() ;
    Sample a batch of data via $_i$;
    $g_i \leftarrow r F_i(x_i;\ _i)$; `// Compute gradients`
    $\begin{pmatrix} x^i \\ \tilde{x}^i \end{pmatrix} \leftarrow \exp\left((t - t^i)\begin{pmatrix} & \\ & \end{pmatrix}\right)\begin{pmatrix} x^i \\ \tilde{x}^i \end{pmatrix}$;
    $x^i \leftarrow x^i - g_i$; `// Apply` $\mathrm{A}^2\mathrm{CiD}^2$
    $\tilde{x}^i \leftarrow \tilde{x}^i - g_i$; `// Take the grad step`
    $t^i \leftarrow t$;
  In one thread *on worker* i **continuously do:**
    $t \leftarrow$ *clock*() ;
    Find available worker j ; `// Synchronize workers` i `and` j
    $m_{ij} \leftarrow (x^i - x^j)$; `// Send` x^i `to` j `and receive` x^j `from` j
    $\begin{pmatrix} x^i \\ \tilde{x}^i \end{pmatrix} \leftarrow \exp\left((t - t^i)\begin{pmatrix} & \\ & \end{pmatrix}\right)\begin{pmatrix} x^i \\ \tilde{x}^i \end{pmatrix}$; `// Apply` $\mathrm{A}^2\mathrm{CiD}^2$
    $x^i \leftarrow x^i - m_{ij}$; `// p2p averaging`
    $\tilde{x}^i \leftarrow \tilde{x}^i - \tilde{\eta} m_{ij}$;
    $t^i \leftarrow t$;
return $(x_T^i)_{1 \le i \le n}$.

number is set to 300 and 90 epochs respectively, following standard practice [22]. To circumvent the fuzziness of the notion of epoch in the asynchronous decentralized setting, we do not "split the dataset and re-shuffle it among workers at each epoch" as done with our standard All-Reduce baseline. Rather, we give access to the whole dataset to all workers, each one shuffling it with a different random seed. We use SGD with a base learning rate of 0.1, a momentum value set at 0.9 and $5 \cdot 10^{-4}$ for weight decay. As advocated in [16], we do not apply weight decay on the learnable batch-norm coefficients. For ImageNet training with the SGD baseline, we decay the learning-rate by a factor of 10 at epochs 30, 60, 80 (epochs 50, 75 for CIFAR-10), and apply an analogous decay schedule with our asynchronous decentralized methods. All of our neural network parameters are initialized with the default Pytorch settings, and one All-Reduce averaging is performed before and after the training to ensure consensus at initialization and before testing. For our continuous momentum, we also need to set the parameters $\ ;\eta$. For all our experiments, we use the values given by Prop. 3.6. As advocated, the *asynchronous baseline* correspond to the setting without acceleration, i.e. with $= 0$ and $= \eta = \frac{1}{2}$, whereas using $\mathbf{A}^2\mathbf{CiD}^2$ leads to consider $= \frac{1}{2\sqrt{\frac{1}{1}\frac{1}{2}}};\ = \frac{1}{2};\eta = \frac{1}{2}\sqrt{\frac{1}{2}}$, where $_1;\ _2$ are set to their theoretical value given by (2), (3) depending on the communication rate and graph's topology, assuming that each worker chose their peers uniformly among their neighbors (we verify empirically that it is the case in practice, see Appendix E.2).

Practical implementation of the dynamic. The dynamic studied in Eq. (4) is a model displaying many of the properties sought after in practice. In our implementation, described in Algo. 1, each worker i has indeed two independent processes and the DNN parameters and momentum variable $\{x^i;\tilde{x}^i\}$ are locally stored such that both processes can update them at any time. One process continuously performs gradient steps, while the other updates $\{x^i;\tilde{x}^i\}$ via peer-to-peer averaging. The gradient process maximizes its throughput by computing forward and backward passes back to back. Contrary to All-Reduce based methods that require an increasing number of communications with the growing number of workers, inevitably leading to an increasing time between two rounds of computations,

we study the case where each worker has a fixed communication rate, given as hyperparameter in our implementation. We implement 3 different graph topologies: complete, ring, and exponential [28, 2], see Appendix E.1 for details. To emulate the P.P.Ps for the communications, each worker samples a random number of p2p averaging to perform between each gradient computation, following a Poisson law using the communication rate as mean. To minimize idle time of the communication process, workers are paired with one of their neighbors in a "First In First Out" manner in an availability queue (a worker is available when it finished its previous averaging and still has some to do before the next gradient step). To implement this, we use a central coordinator to store the availability queues and the graph topology (this is lightweight in a cluster: the coordinator only exchanges integers with the workers), but it could be done in different ways, *e.g.* by pinging each other at high frequency. As we assumed a unit time for the gradient process in our analysis, and that real time is used in our algorithm to apply our $\mathbf{A^2CiD^2}$momentum (see Algo. 1), we maintain a running average measure of the duration of the previous gradient steps to normalize time.

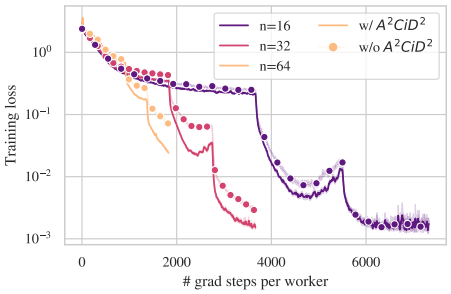


Figure 4: Training loss for CIFAR10 using a minibatch size of 128. We display the training loss with up to 64 workers, w/ and w/o $\mathbf{A^2CiD^2}$, on the challenging ring graph.

4.2 Evaluation on large scale datasets

CIFAR10. This simple benchmark allows to understand the benefits of our method in a well-controlled environment. Tab. 4 reports our numerical accuracy on the test set of CIFAR10, with a standard deviation calculated over 3 runs. Three scenarios are considered: a complete, an exponential and a ring graph. In Fig. 3 (a), we observe that with the asynchronous baseline on the complete graph, the more workers, the more the training loss degrades. Fig. 3 (b) hints that it is in part due to an insufficient communication rate, as increasing it allows to lower the loss and close the gap with the All-Reduce baseline. However, this is not the only causative factor as Tab. 4 indicates that accuracy generally degrades as the number of workers increases even for AR-SGD, which is expected for large batch sizes. Surprisingly, even with a worse training loss for $n = 64$, the asynchronous baseline still leads to better generalization than

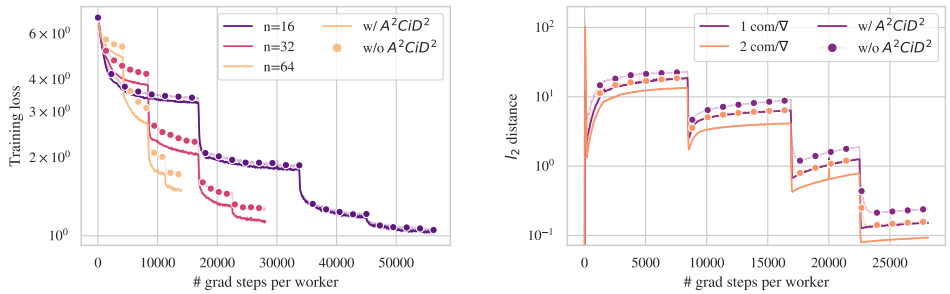
Table 5: Accuracy on ImageNet for a batch-size of 128. We compared a vanilla asynchronous pairwise gossip approach with and without $\mathbf{A^2CiD^2}$, demonstrating the improvement of our method. We also varied the communication rate.

#Workers	#com/#grad	16	32	64
AR-SGD baseline	-	75.5	75.2	74.5
Complete graph				
Async. baseline	1	74.6	73.8	71.3
Ring graph				
Async. baseline	1	**74.8**	71.6	64.1
$\mathbf{A^2CiD^2}$	1	74.7	**73.4**	**68.0**
Async. baseline	2	74.8	73.7	68.2
$\mathbf{A^2CiD^2}$	2	**75.3**	**74.4**	**71.4**

Table 4: Accuracy of our method on CIFAR10 for a 128 batchsize with an equal number of pairwise communications and gradient computations per worker. We compared a vanilla asynchronous pairwise gossip approach with and without $\mathbf{A^2CiD^2}$, demonstrating the improvement of our method.

#Workers	4		8		16		32		64	
AR-SGD baseline	94.5	0.1	94.4	0.1	94.5	0.2	93.7	0.3	92.8	0.2
Complete graph										
Async. baseline	94.93	0.11	94.91	0.07	94.86	0.01	94.55	0.01	93.38	0.21
Exponential graph										
Async. baseline	95.07	0.01	94.89	0.01	94.82	0.06	94.44	0.02	93.41	0.02
$\mathbf{A^2CiD^2}$	**95.17**	0.04	**95.04**	0.01	**94.87**	0.02	**94.56**	0.01	**93.47**	0.01
Ring graph										
Async. baseline	**95.02**	0.06	**95.01**	0.01	95.00	0.01	93.95	0.11	91.90	0.10
$\mathbf{A^2CiD^2}$	94.95	0.02	**95.01**	0.10	**95.03**	0.01	**94.61**	0.02	**93.08**	0.20

AR-SGD, and consistently improves the test accuracy across all tested values of n. The communication rate being identified as a critical factor at large scale, we tested our continuous momentum on the ring graph, each worker performing one p2p averaging for each gradient step. Fig. 4 shows that incorporating $\mathbf{A^2CiD^2}$ leads to a significantly better training dynamic for a large number of workers, which translates into better performances at test time as shown in Tab. 4.



(a) (b)

Figure 5: (a) Training loss for ImageNet using 128 batch size, with an equal number of communications and computations per worker. We display the training loss for various number of workers (up to 64), using $\mathbf{A^2CiD^2}$, for the ring graph. (b) Comparison of consensus distances when $\mathbf{A^2CiD^2}$ is applied versus doubling the rate of communications on the rign graph with 64 workers: applying $\mathbf{A^2CiD^2}$ has the same effect as doubling communications.

ImageNet. For validating our method in a real-life environment, we consider the large-scale ImageNet dataset. Tab. 6 confirms the advantage of asynchronous methods by allocating less compute to the slowest workers, leading to faster training times. Tab. 5 reports our accuracy for the complete and ring graphs. As $_1 = _2$ for the complete graph, we simply run our baseline asynchronous method for reference. The case of the ring graph is much more challenging: for $n = 64$ workers, the accuracy drops by 10% compared to the synchronous baseline given by AR-SGD. Systematically, with $\mathbf{A^2CiD^2}$, the final accuracy increases: up to 4% absolute percent in the difficult $n = 64$ setting. This is corroborated by Fig. 5, which indicates that incorporating $\mathbf{A^2CiD^2}$ significantly improves the training dynamic on ImageNet. However, for reducing the gap with the AR-SGD baseline, it will be necessary to increase the communication rate as discussed next.

Table 6: Statistics of runs on Imagenet with 64 workers (for ours, on the exponential graph).

Method	t (min)	# r slowest worker	# r fastest worker
AR-SGD	1:7 10^2	14k	14k
Baseline (ours)	**1:5 10^2**	**13k**	14k
$\mathbf{A^2CiD^2}$(ours)	**1:5 10^2**	**13k**	14k

Consensus improvement. The bottom of Tab. 5, as well as Fig. 5 (b) study the virtual acceleration thanks to $\mathbf{A^2CiD^2}$. Not only increasing communications combined with $\mathbf{A^2CiD^2}$ allows to obtain competitive performance, but Fig. 1 shows that doubling the rate of communication has an identical effect on the training loss than adding $\mathbf{A^2CiD^2}$. This is verified in Fig. 5 (b) by tracking the consensus distance between workers: $\mathbf{A^2CiD^2}$ significantly reduces it, which validates the results of Sec. 3.4.

5 Conclusion

In this work, we confirmed that the communication rate is a key performance factor to successfully train DNNs at large scale with decentralized asynchronous methods. We introduced $\mathbf{A^2CiD^2}$, a continuous momentum which only adds a minor local memory overhead while allowing to mitigate this need. We demonstrated, both theoretically and empirically, that $\mathbf{A^2CiD^2}$ substantially improves performances, especially on challenging network topologies. As we only focused on data parallel methods for training Deep Neural Networks in a cluster environment, in a future work, we would like to extend our empirical study to more heterogeneous compute and data sources, as our theory could encompass local SGD methods [39] and data heterogeneity inherent in Federated Learning [33].

Acknowledgements

EO, AN, and EB's work was supported by Project ANR-21-CE23-0030 ADONIS and EMERG-ADONIS from Alliance SU. This work was granted access to the HPC/AI resources of IDRIS under the allocation AD011013743 made by GENCI. EB and AN acknowledge funding and support from NSERC Discovery Grant RGPIN- 2021-04104, FRQNT New Scholar, and resources from Compute Canada and Calcul Quebec. In addition, the authors would like to thank Olexa Bilaniuk and Louis Fournier for their helpful insights regarding our code implementation.

References

[1] L. Arnold. Stochastic differential equations. *New York*, 1974.

[2] M. Assran, N. Loizou, N. Ballas, and M. Rabbat. Stochastic gradient push for distributed deep learning. In *International Conference on Machine Learning*, pages 344–353. PMLR, 2019.

[3] E. Belilovsky, M. Eickenberg, and E. Oyallon. Decoupled greedy learning of cnns. In *International Conference on Machine Learning*, pages 736–745. PMLR, 2020.

[4] E. Belilovsky, L. Leconte, L. Caccia, M. Eickenberg, and E. Oyallon. Decoupled greedy learning of cnns for synchronous and asynchronous distributed learning. *arXiv preprint arXiv:2106.06401*, 2021.

[5] M. Blot, D. Picard, M. Cord, and N. Thome. Gossip training for deep learning. In *Advances in Neural Information Processing Systems*, volume 30, 2016.

[6] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, 2006.

[7] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[8] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya. Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent, 2018.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[10] A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in neural information processing systems*, 27, 2014.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] M. Even, R. Berthier, F. Bach, N. Flammarion, H. Hendrikx, P. Gaillard, L. Massoulié, and A. Taylor. A continuized view on nesterov acceleration for stochastic gradient descent and randomized gossip. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[13] M. Even, H. Hendrikx, and L. Massoulie. Decentralized optimization with heterogeneous delays: a continuous-time approach. *arXiv preprint arXiv:2106.03585*, 2021.

[14] A. Ghosh, S. Boyd, and A. Saberi. Minimizing effective resistance of a graph. *SIAM Review*, 50(1):37–66, 2008.

[15] R. M. Gower, M. Schmidt, F. Bach, and P. Richtárik. Variance-reduced methods for machine learning. *Proceedings of the IEEE*, 108(11):1968–1983, 2020.

[16] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[18] H. Hendrikx, F. Bach, and L. Massoulié. An accelerated decentralized stochastic proximal algorithm for finite sums. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[19] A. Koloskova, T. Lin, and S. U. Stich. An improved analysis of gradient tracking for decentralized machine learning. *Advances in Neural Information Processing Systems*, 34:11422–11435, 2021.

[20] A. Koloskova, T. Lin, S. U. Stich, and M. Jaggi. Decentralized deep learning with arbitrary communication compression. *arXiv preprint arXiv:1907.09356*, 2019.

[21] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. Stich. A unified theory of decentralized sgd with changing topology and local updates. In *International Conference on Machine Learning*, pages 5381–5393. PMLR, 2020.

[22] L. Kong, T. Lin, A. Koloskova, M. Jaggi, and S. Stich. Consensus control for decentralized deep learning. In *International Conference on Machine Learning*, pages 5686–5696. PMLR, 2021.

[23] D. Kovalev, A. Salim, and P. Richtarik. Optimal and practical algorithms for smooth and strongly convex decentralized optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18342–18352. Curran Associates, Inc., 2020.

[24] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017.

[26] S. Li and T. Hoefler. Near-optimal sparse allreduce for distributed deep learning. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 135–149, 2022.

[27] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in neural information processing systems*, 30, 2017.

[28] X. Lian, W. Zhang, C. Zhang, and J. Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

[29] T. Lin, S. P. Karimireddy, S. U. Stich, and M. Jaggi. Quasi-global momentum: Accelerating decentralized deep learning on heterogeneous data. *arXiv preprint arXiv:2102.04761*, 2021.

[30] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi. Don't use large mini-batches, use local sgd. In *International Conference on Learning Representations*, 2020.

[31] Y. Lu and C. De Sa. Optimal complexity in decentralized training. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7111–7123. PMLR, 18–24 Jul 2021.

[32] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[33] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, 2016.

[34] A. Nabli and E. Oyallon. DADAO: Decoupled accelerated decentralized asynchronous optimization. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 25604–25626. PMLR, 23–29 Jul 2023.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[36] M. Ryabinin, E. Gorbunov, V. Plokhotnyuk, and G. Pekhimenko. Moshpit sgd: Communication-efficient decentralized training on heterogeneous unreliable devices. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 18195–18211. Curran Associates, Inc., 2021.

[37] K. Scaman, F. Bach, S. Bubeck, Y. T. Lee, and L. Massoulié. Optimal algorithms for smooth and strongly convex distributed optimization in networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3027–3036. PMLR, 06–11 Aug 2017.

[38] A. Sergeev and M. D. Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[39] S. U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019.

[40] S. U. Stich. Unified optimal analysis of the (stochastic) gradient method, 2019.

[41] T. Vogels, L. He, A. Koloskova, S. P. Karimireddy, T. Lin, S. U. Stich, and M. Jaggi. Relaysum for decentralized deep learning on heterogeneous data. *Advances in Neural Information Processing Systems*, 34:28004–28015, 2021.

[42] Y. Yakimenka, C.-W. Weng, H.-Y. Lin, E. Rosnes, and J. Kliewer. Straggler-resilient differentially-private decentralized learning. In *2022 IEEE Information Theory Workshop (ITW)*, pages 708–713. IEEE, 2022.

[43] B. Ying, K. Yuan, Y. Chen, H. Hu, P. Pan, and W. Yin. Exponential graph is provably efficient for decentralized deep training. *Advances in Neural Information Processing Systems*, 34:13975–13987, 2021.

[44] B. Ying, K. Yuan, H. Hu, Y. Chen, and W. Yin. Bluefog: Make decentralized algorithms practical for optimization and deep learning. *arXiv preprint arXiv:2111.04287*, 2021.

[45] B. Ying, K. Yuan, H. Hu, Y. Chen, and W. Yin. Bluefog: Make decentralized algorithms practical for optimization and deep learning. 2021.

[46] Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

[47] B. Yuan, Y. He, J. Davis, T. Zhang, T. Dao, B. Chen, P. S. Liang, C. Re, and C. Zhang. Decentralized training of foundation models in heterogeneous environments. *Advances in Neural Information Processing Systems*, 35:25464–25477, 2022.

[48] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Appendix

Table of Contents

A Notations

For $n \in \mathbb{N}$ the number of workers and $d \in \mathbb{N}$ an ambient dimension, for all $t > 0$, the variable $x_t \in \mathbb{R}^{n \times d}$ is a matrix such that $x_t = [x_t^1; \dots; x_t^n]^\top$, with $x_t^i \in \mathbb{R}^d$ for all $i \in \{1; \dots; n\}$. We remind that $\mathbf{1}$ is the vector of n ones such that $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i = \frac{1}{n} x^\top \mathbf{1} \in \mathbb{R}^d$. With $\mathbf{I}$ the identity and $\|\cdot\|_F$ the matrix Frobenius norm, we write $\pi = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ the projection so that $\|x\pi\|_F^2 = \sum_{i=1}^n \|x_i - \bar{x}\|^2$.

For a continuously differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ and $a, b \in \mathbb{R}^d$, the Bregman divergence is defined with $d_f(a, b) = f(a) - f(b) - \langle \nabla f(b), a - b \rangle$. For $\Gamma \in \mathbb{R}^{n \times n}$, we denote by Γ^+ its pseudo inverse. For a positive semi-definite $\Gamma \in \mathbb{R}^{n \times n}$, and $x \in \mathbb{R}^{n \times d}$, we introduce $\|x\|_\Gamma^2 = \mathrm{Tr}(x^\top \Gamma x)$ and $\Gamma^{1/2}$ its square-root. We recall that the connectivity between workers is given by a set of edges E, and denote by e_i the i^{th} basis vector of $\mathbb{R}^n$.

For $x \in \mathbb{R}^{n \times d}$, we introduce:

$$\nabla F(x) = [\nabla f_1(x_1); \dots; \nabla f_n(x_n)]^\top \in \mathbb{R}^{n \times d} \text{ and } \nabla F(x; \xi) = [\nabla F_1(x_1; \xi_1); \dots; \nabla F_n(x_n; \xi_n)]^\top \in \mathbb{R}^{n \times d}.$$

Finally, to study the gradient steps taken on each individual worker independently, we introduce:

$$\nabla F_i(x; \xi) = [0; \dots 0; \nabla F_i(x_i; \xi_i); 0; \dots; 0]^\top \in \mathbb{R}^{n \times d}.$$

B Technical Preliminaries

We recall some basic properties that we will use throughout our proofs.

Lemma B.1 (Implications of Assumption 3.4)**.** *If each f_i is μ-strongly convex and L-smooth, we have, for any $a, b \in \mathbb{R}^d$:*

$$\frac{1}{2L}\|\nabla f_i(a) - \nabla f_i(b)\|^2 \leq d_{f_i}(a, b) \leq \frac{L}{2}\|a - b\|^2;$$

and

$$\frac{\mu}{2}\|a - b\|^2 \leq d_{f_i}(a, b) \leq \frac{1}{2\mu}\|\nabla f_i(a) - \nabla f_i(b)\|^2.$$

Lemma B.2 (Generalized triangle inequality)**.** *For any $a; b; c \in \mathbb{R}^d$ and continuously differentiable function $f : \mathbb{R}^d \to \mathbb{R}$, by definition of the Bregman divergence, we have:*

$$d_f(a; b) + d_f(b; c) = d_f(a; c) + \langle a - b;\ \nabla f(c) - \nabla f(b) \rangle :$$

Lemma B.3 (Variance decomposition)**.** *For a random vector $a \in \mathbb{R}^d$ and any $b \in \mathbb{R}^d$, the variance of a can be decomposed as:*

$$\mathbb{E}\left[\|a - \mathbb{E}[a]\|^2\right] = \mathbb{E}\left[\|a - b\|^2\right] - \left\|\mathbb{E}[a] - b\right\|^2 :$$

Lemma B.4 (Jensen's inequality)**.** *For any vectors $a_1; \ldots; a_n \in \mathbb{R}^d$, we have:*

$$\left\|\sum_{i=1}^{n} a_i\right\|^2 \leq n \sum_{i=1}^{n} \|a_i\|^2 :$$

Lemma B.5. *For any vectors $a; b \in \mathbb{R}^d$ and $\gamma > 0$:*

$$2\langle a; b\rangle \leq \gamma\|a\|^2 + \gamma^{-1}\|b\|^2 :$$

Lemma B.6. *For any vectors $a; b \in \mathbb{R}^d$ and $\gamma > 0$:*

$$\|a + b\|^2 \leq (1 + \gamma)\|a\|^2 + (1 + \gamma^{-1})\|b\|^2 :$$

Lemma B.7. *For any $A \in \mathbb{R}^{n \times d}$ and $B \in \mathbb{R}^{n \times n}$, we have:*

$$\|BA\|_F \leq \|A\|_F \|B\|_2 :$$

Lemma B.8 (Effective resistance contraction)**.** *For $(i; j) \in E$ and any $x \in \mathbb{R}^{n \times d}$, we have:*

$$\|(e_i - e_j)(e_i - e_j)^\top x\|_{\Lambda^+}^2 \leq \|(e_i - e_j)(e_i - e_j)^\top x\|_F^2 :$$

Proof. Indeed, we note that, by definition of $\|\cdot\|_2$ (3):

$$\begin{aligned}
\|(e_i - e_j)(e_i - e_j)^\top x\|_{\Lambda^+}^2 &= \mathrm{Tr}\left(x^\top (e_i - e_j)(e_i - e_j)^\top \Lambda^+ (e_i - e_j)(e_i - e_j)^\top x\right) && (7)\\
&\leq \|\Lambda^+\|_2 \mathrm{Tr}\left(x^\top (e_i - e_j)(e_i - e_j)^\top x\right) && (8)\\
&= \|\Lambda^+\|_2 \|(e_i - e_j)(e_i - e_j)^\top x\|_F^2 && (9)
\end{aligned}$$

□

Lemma B.9. *For any $x \in \mathbb{R}^{n \times d}$, and Λ the Laplacian of a connected graph, we have:*

$$\sum_{(i;j) \in E} w_{ij} \|(e_i - e_j)(e_i - e_j)^\top x\|_F^2 = 2\|x\|_\Lambda^2 :$$

Proof. Indeed, by definition of the Laplacian Λ (3.1), we have:

$$\begin{aligned}
\sum_{(i;j) \in E} w_{ij} \|(e_i - e_j)(e_i - e_j)^\top x\|_F^2 &= \sum_{(i;j) \in E} w_{ij} \mathrm{Tr}\left(x^\top (e_i - e_j)(e_i - e_j)^\top (e_i - e_j)(e_i - e_j)^\top x\right) && (10)\\
&= 2 \sum_{(i;j) \in E} w_{ij} \mathrm{Tr}\left(x^\top (e_i - e_j)(e_i - e_j)^\top x\right) && (11)\\
&= 2\mathrm{Tr}\left(x^\top \sum_{(i;j) \in E} w_{ij} (e_i - e_j)(e_i - e_j)^\top x\right) && (12)
\end{aligned}$$

□

Lemma B.10. *For any $x; y \in \mathbb{R}^{n \quad d}$, and the Laplacian of a connected graph, we have:*

$$2h \ x; y i \quad \frac{1}{4} k x k_{\Lambda}^2 + 4 \ _1 k y k_F^2$$

Proof. By property of the Laplacian of a connected graph, we have that $= (\ ^+)^{1=2} \ ^{1=2}$. Thus:

$$2h \ x; y i = 2h \ ^{1=2} x; (\ ^+)^{1=2} y i \tag{13}$$

$$\overset{(B.5)}{=} \frac{1}{4} k \ ^{1=2} x k_F^2 + 4 k (\ ^+)^{1=2} y k_F^2 \tag{14}$$

$$\overset{(B.7)}{=} \frac{1}{4} k x k_{\Lambda}^2 + 4 \ _1 k y k_F^2 \tag{15}$$

□

C Proof of the main result of this paper

We remind that we study the following dynamic:

$$dx_t^i = (\cancel{x}_t^i \quad x_t^i) dt \quad \int_{\Xi} r F_i(x_t^i; \ _i) \, dN_t^i(\ _i) \quad \sum_{j;(i;j) 2 E} (x_t^i \quad x_t^j) dM_t^{ij} ;$$

$$d\cancel{x}_t^i = (x_t^i \quad \cancel{x}_t^i) dt \quad \int_{\Xi} r F_i(x_t^i; \ _i) \, dN_t^i(\ _i) \quad \sim \sum_{j;(i;j) 2 E} (x_t^i \quad x_t^j) dM_t^{ij} ;$$

which simplifies to:

$$dx_t^i = \quad \int_{\Xi} r F_i(x_t^i; \ _i) \, dN_t^i(\ _i) \quad \sum_{j;(i;j) 2 E} (x_t^i \quad x_t^j) dM_t^{ij}$$

if $\mathbf{A}^2\mathbf{C}\mathbf{i}\mathbf{D}^2$is not applied. We also recall the main proposition, which we prove next:

Proposition C.1 (Convergence guarantees.)**.** *Assume that $f x_t; \cancel{x}_t g$ follow the dynamic Eq. 4 and that Assumption 3.2-3.3 are satisfied. Assume that $\mathbf{1} x_0 = x_0 = \cancel{x}_0$ and let T the total running time. Then:*

- ***Non-accelerated setting**, we pick $= 0; \quad = \sim = \frac{1}{2}$ and set $= \ _1$,*
- ***Acceleration ($\mathbf{A}^2\mathbf{C}\mathbf{i}\mathbf{D}^2$)**, we set $= \frac{p \frac{1}{1 \ 2}}{2}; \quad = \frac{1}{2}; \sim = \frac{1}{2} \sqrt{\frac{-1}{2}}$, and $= \frac{p}{1 \ 2} \quad _1$.*

Then, there exists a constant step size > 0 such that if:

- ***(strong-convexity)** the Assumption 3.4 is satisfied, then $\frac{1}{16 L (1+ \ )}$ and[1]:*

$$\mathbb{E} \left[k x_T \quad x^* k^2 \right] = \mathcal{O} \left(k x_0 \quad x^* k^2 e^{\frac{T}{16 L (1+ \ )}} + \frac{^2 + \ ^2 (1 + \ )}{^2 T} \right) ;$$

- ***(non-convexity)** the Assumption 3.5 is satisfied, then there is $c > 0$ which depends only on $P; M$ from the assumptions such that $\frac{c}{L (\ +1)}$ and:*

$$\frac{1}{T} \int_0^T \mathbb{E} \left[k \nabla f(x_t) k^2 \right] dt = \mathcal{O} \left(\frac{L (1 + \ )}{T} (f(x_0) \quad f(x^*)) + \sqrt{\frac{L (f(x_0) \quad f(x^*))}{T} (\ ^2 + (1 + \ ) \ ^2)} \right) :$$

Also, the expected number of gradient steps is nT and the number of communications is $\frac{\mathrm{Tr}(\Lambda)}{2} T$.

[1] $\tilde{\mathcal{O}}$-notation hides constants and polylogarithmic factors.

Proof. The core of the proof is to introduce the appropriate Lyapunov potentials $_k(t; X)$, where X could be $(x; \star)$ or x depending on whether we apply $\mathbf{A}^2\mathbf{CiD}^2$or not. If we apply $\mathbf{A}^2\mathbf{CiD}^2$, we introduce the momentum matrix $A = \begin{pmatrix} & \\ & \end{pmatrix}$. Then, by Ito's lemma, given that all the functions are smooth, and remembering that all Point-wise Poisson Processes N_t^i have unit intensity, that the M_t^{ij} have intensity ^{ij} and that they are all independent, we obtain in a similar fashion to [12, 34]:

$$
\begin{aligned}
_k(T; X_T) \quad _k(0; X_0) = & \int_0^T @_t \ _k(t; X_t) + \underbrace{\langle A X_t; @_X \ _k(t; X_t)\rangle}_{\text{momentum term}} dt \\
& + \sum_{i=1}^n \int_0^T \underbrace{\int_\Xi \ _k\left(t; X_t \quad \begin{pmatrix} r\bar{F}_i(x_t; \ ) \\ r\bar{F}_i(x_t; \ )\end{pmatrix}\right) \quad _k(t; X_t)}_{\text{variation due to each independent gradient update}} d\mathrm{t}dP(\ ) \\
& + \sum_{(i;j)2E} \int_0^T \underbrace{\left[\ _k\left(t; X_t \quad \begin{pmatrix} (e_i \quad e_j)(e_i \quad e_j)^\intercal x_t \\ -(e_i \quad e_j)(e_i \quad e_j)^\intercal x_t\end{pmatrix}\right) \quad _k(t; X_t)\right]}_{\text{variation due to each independent p2p communication}} {}^{ij} dt \\
& + M_T \ ;
\end{aligned}
$$

where M_T is a martingale. In the case where $\mathbf{A}^2\mathbf{CiD}^2$is not applied, we set $A = 0$ to remove the momentum term, and all updates are done only along x as there is no $\star$. We remind that:

$$\int_0^t e^{\ u} du = \frac{1}{}(e^{\ t} \quad 1) \tag{16}$$

We now present our choice of potential for each cases:

- For the convex case in the non-accelerated setting, we introduce:

$$_1(t; x) \ , \ A_t kx \quad x\ k^2 + B_t k\ \ xk_F^2 :$$

- For the convex case with $\mathbf{A}^2\mathbf{CiD}^2$, we introduce:

$$_2(t; x; \star) \ , \ A_t kx \quad x\ k^2 + B_t k\ \ xk_F^2 + B_t k\star k_{\Lambda^+} :$$

- For the non-convex case in the non-accelerated setting, we introduce:

$$_3(t; x) \ , \ A_t d_f(x; x\ ) + B_t k\ \ xk_F^2 :$$

- For the non-convex case with $\mathbf{A}^2\mathbf{CiD}^2$, we introduce:

$$_4(t; x) \ , \ A_t d_f(x; x\ ) + B_t k\ \ xk_F^2 + B_t k\star k_{\Lambda^+}^2 :$$

C.1 Some useful upper-bounds

As the same terms appear in several potentials, we now prepare some intermediary results which will be helpful for the proofs.

Study of the $kx \quad x\ k^2$ terms:

First, we study the variations in the $kx \quad x\ k^2$ term appearing in $_1$ and $_2$. As the updates due to the communication are in the orthogonal of $\mathbf{1}$, it is only necessary to study the variations induced by the gradient steps. Thus, we define:

$$x \ , \ \sum_{i=1}^n k\overline{x \quad r\bar{F}_i(x; \ )} \quad x\ k^2 \quad kx \quad x\ k^2$$

We note that $\overline{r F_i(x;\ )} = \frac{1}{n} r F_i(x_i;\ _i)$, which, using $\sum_i r f_i(x\ ) = 0$, leads to:

$$\mathsf{E}_{\ _1;\dots;\ _n}[\ \ x] = \sum_{i=1}^{n} \ \ \frac{2}{n}\langle x\ \ x\ ;\ r f_i(x_i)\rangle + \frac{^2}{n^2}\mathsf{E}_{\ _i}[\|k r F_i(x_i;\ _i)\|^2] \tag{17}$$

$$= \sum_{i=1}^{n} \ \ \frac{2}{n}\langle x\ \ x\ ;\ r f_i(x_i)\ \ \ r f_i(x\ )\rangle + \frac{^2}{n^2}\mathsf{E}_{\ _i}[\|k r F_i(x_i;\ _i)\|^2] \tag{18}$$

$$\overset{(3.4);(B.3)}{=} \ \ \frac{^2}{n}\ ^2 + \sum_{i=1}^{n} \ \ \frac{2}{n}\langle x\ \ x\ ;\ r f_i(x_i)\ \ \ r f_i(x\ )\rangle + \frac{^2}{n^2}\|k r f_i(x_i)\|^2 \tag{19}$$

$$\overset{(B.2)}{=} \ \ \frac{^2}{n}\ ^2 + \sum_{i=1}^{n} \ \ \frac{2}{n}(d_{f_i}(x;x\ ) + d_{f_i}(x\ ;x_i)\ \ \ d_{f_i}(x;x_i)) + \frac{^2}{n^2}\|k r f_i(x_i)\|^2 \tag{20}$$

$$\overset{(B.4)}{\leq} \ \ \frac{^2}{n}\ ^2 + \sum_{i=1}^{n} \ \ \frac{2}{n}(d_{f_i}(x;x\ ) + d_{f_i}(x\ ;x_i)\ \ \ d_{f_i}(x;x_i)) + \frac{2}{n^2}\ ^2\|k r f_i(x\ )\ \ \ r f_i(x_i)\|^2 + \frac{2}{n^2}\ ^2\|k r f_i(x\ )\|^2 \tag{21}$$

$$\overset{(3.4);(B.1)}{\leq} \ \ \frac{^2}{n}\ ^2 + \frac{2}{n}\ ^2\ ^2 + \sum_{i=1}^{n} \ \ \frac{2}{n}(d_{f_i}(x;x\ ) + d_{f_i}(x\ ;x_i)\ \ \ d_{f_i}(x;x_i)) + \frac{4L}{n^2}\ ^2 d_{f_i}(x\ ;x_i) \tag{22}$$

$$\overset{(B.1)}{\leq} \ \ \frac{2}{n}\ ^2\ ^2 + \frac{2}{n}\ ^2\ ^2 \ \ \ \ \|kx\ \ x\ \|^2 + \frac{L}{n}\|k\ \ xk_F^2 + \sum_{i=1}^{n}\left(\ \ \frac{2}{n} + \frac{4L}{n^2}\ ^2\right) d_{f_i}(x\ ;x_i) \tag{23}$$

Study of the $d_f(x;x\ )$ terms:

Next, using the same reasoning as for $\ \ x$, we also need to only study the gradient updates in the non-convex setting for the first part of $\ _3$, $\ _4$. Thus, we set:

$$f\ ,\ \sum_{i=1}^{n} d_f(x\ \ \ \frac{1}{n} r F_i(x_i;\ _i);x\ )\ \ \ d_f(x;x\ ): \tag{24}$$

First, it is useful to note that under Assumption 3.5, using (B.3), we have:

$$\mathsf{E}_{\ _1;\dots;\ _n}\|k r F(x;\ )\|_F^2 \ \ \ n\ ^2 + (1+M)\sum_{i=1}^{n}\|k r f_i(x_i)\|^2 : \tag{25}$$

Then, using $\sum_i r f_i(x\ ) = 0$ and the L-smoothness of $f = \frac{1}{n}\sum_i f_i$, we get:

$$\mathsf{E}_{\ _1;\dots;\ _n}[\ \ f] \overset{(B.2)}{=} \sum_{i=1}^{n}\mathsf{E}[d_f(x\ \ \ \frac{1}{n} r F_i(x_i;\ _i);x)]\ \ \ \frac{}{n}\langle h r f_i(x_i);\ r f(x)\rangle \tag{26}$$

$$\overset{(B.1)}{\leq} \sum_{i=1}^{n}\frac{1}{2n^2}L\ \ ^2\mathsf{E}[\|k r F_i(x_i;\ _i)\|^2]\ \ \ \frac{}{n}\langle h r f_i(x_i);\ r f(x)\rangle \tag{27}$$

$$\overset{(25)}{\leq} \frac{L}{2n}\ ^2\ ^2 \ \ \ \ k r f(x)\|^2 + \sum_{i=1}^{n}\frac{M+1}{2n^2}L\ \ ^2\|k r f_i(x_i)\|^2 \ \ \ \sum_{i=1}^{n}\frac{}{n}\langle h r f_i(x_i)\ \ \ r f_i(x);\ r f(x)\rangle \tag{28}$$

$$\overset{(B.5)}{\leq} \frac{L}{2n}\ ^2\ ^2 + \frac{}{2n}L^2\|k\ \ xk_F^2 \ \ \ \frac{}{2}\|k r f(x)\|^2 + \sum_{i=1}^{n}\frac{M+1}{2n^2}L\ \ ^2\|k r f_i(x_i)\|^2 \tag{29}$$

As we also have:

$$\sum_{i=1}^{n} k\, r\, f_i(x_i)k^2 \overset{(B.4)}{} \sum_{i=1}^{n} 3(k\, r\, f_i(x_i) \quad r\, f_i(x)k^2 + k\, r\, f_i(x) \quad r\, f(x)k^2 + k\, r\, f(x)k^2) \tag{30}$$

$$\overset{(3.5)}{} 3L^2 k \;\; xk_F^2 \; + 3n \;^2 + 3n(1+P)k\, r\, f(x)k^2 \tag{31}$$

We get in the end:

$$\mathsf{E}_{1;\dots;n}[\;\; f] \quad \frac{L\;^2}{2n} \;^2 + \frac{3(M\;+1)}{2n} \;^2 L\;^2 + \left(\frac{3(M\;+1)}{2n^2} L^{3\;\;2} + \frac{}{2n} L^2\right) k \;\; xk_F^2 + \left(\frac{3(M\;+1)}{2n}(1+P)L\;^2 \quad \frac{}{2}\right) k\, r\, f(x)k^2 \tag{32}$$

Remark C.2. As observed in (5), note that for both the terms $kx \quad x\;k^2$ and $d_f(x;x\;)$, as we are considering x and $\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}} \;= 0$, Poisson updates from the communication process amount to zero. Moreover, as $\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}(x \quad \star) = 0$, the update from the momentum is also null for these terms.

Study of the $k \;\; xk_F^2$ terms:

We get from the Poisson updates for the gradient processes:

$$\mathsf{E}_{1;\dots;n}\left[\sum_{i=1}^{n} k\;\;(x \qquad r\bar{F}_i(x;\;\;))k_F^2 \quad k\;\; xk_F^2\right] = \quad 2\;\; h\;\; x;\, r\,F(x)i + \;\;^2\sum_{i=1}^{n} \mathsf{E}_{1;\dots;n}\, k \quad r\bar{F}_i(x;\;\;)k_F^2$$
$$2\;\; h\;\; x;\, r\,F(x)i + \;\;^2\mathsf{E}_{1;\dots;n}\, k\, r\,\bar{F}(x;\;\;)k_F^2\;;$$

and, using the definition of the Laplacian (3.1), we get from the communication processes:

$$\sum_{(i;j)2E}{}^{ij}\left(k\;\;(x \qquad (e_i \quad e_j)(e_i \quad e_j)^{\mathsf{T}}x)k_F^2 \quad k\;\; xk_F^2\right) = \quad 2\;\; hx;\;\; xi + \sum_{(i;j)2E}{}^{ij}\;\;^2 k(e_i \quad e_j)(e_i \quad e_j)^{\mathsf{T}}xk_F^2$$
$$\overset{(B.9)}{=} \quad 2\;\; hx;\;\; xi + 2\;\;^2 kxk_\Lambda^2$$
$$= 2\;\;(\qquad 1)kxk_\Lambda^2\,:$$

Putting together both types of Poisson updates, we define:

$$\mathsf{E}[\quad]\;, \quad 2\;\; h\;\; x;\, r\,F(x)i + \;\;^2\mathsf{E}_{1;\dots;n}\, k \quad r\bar{F}(x;\;\;)k_F^2 + 2\;\;(\qquad 1)kxk_\Lambda^2 \tag{33}$$

$$\overset{(B.10)}{} 4\;\;_1\;\;^2 k\, r\,F(x)k_F^2 + \left(\frac{1}{4} \quad 2\;\;(1 \qquad)\right) kxk_\Lambda^2 + \;\;^2\mathsf{E}_{1;\dots;n}\, k\, r\,\bar{F}(x;\;\;)k_F^2 \tag{34}$$

For $\;= \frac{1}{2}$, we get:

$$\mathsf{E}[\quad] \quad 4\;\;_1\;\;^2 k\, r\,F(x)k_F^2 \quad \frac{1}{4\;\;_1} k\;\; xk_F^2 + \;\;^2\mathsf{E}_{1;\dots;n}\, k\, r\,\bar{F}(x;\;\;)k_F^2 \tag{35}$$

For $\mathbf{A}^2\mathbf{CiD}^2$, we add the momentum term $h\;\;(\star \quad x);2\;\; xi$ to define:

$$\frac{1}{\Lambda}\;, \; 2\;\; h\star;\;\; xi \quad 2\;\; k\;\; xk_F^2 \quad 2\;\; h\;\; x;\, r\,F(x)i + \;\;^2\mathsf{E}_{1;\dots;n}\, k\, r\,\bar{F}(x;\;\;)k_F^2 + 2\;\;(\qquad 1)kxk_\Lambda^2 \tag{36}$$

$$\overset{(B.5)}{} 2\;\; h\star;\;\; xi \quad \frac{3}{2}\;\; k\;\; xk_F^2 + \frac{2}{\;} \;^2 k\, r\,F(x)k_F^2 \quad 2\;\;(1 \qquad)kxk_\Lambda + \;\;^2\mathsf{E}_{1;\dots;n}\, k\, r\,\bar{F}(x;\;\;)k_F^2\,: \tag{37}$$

Study of the $k\star k_{\Lambda^+}^2$ terms:

These terms only appear in the Lyapunov potentials used when applying $\mathbf{A}^2\mathbf{CiD}^2$. From the Poisson updates and momentum, we get:

$$\begin{aligned} {}^{2}_{\Lambda} \;,\; & 2\ h x \quad x; x i_{\Lambda^+} \quad 2\ h x; \ {}^{+} r F(x; \ ) i + \ {}^{2} k\, r F(x; \ ) k^2_{\Lambda^+} \quad 2\sim h\ x; x i \\ & + \sim^2 \sum_{(i;j)2E} {}^{ij} k(e_i \quad e_j)(e_i \quad e_j)^{\mathsf{T}} x k^2_{\Lambda^+} : \end{aligned} \tag{38}$$

Taking the expectation and using (B.9), (B.8), (B.7), (B.5) leads to:

$$\begin{aligned} \mathsf{E}[\ {}^{2}_{\Lambda}] \quad & {}_1\ k\ x k^2_F + \left(\frac{}{2} \quad\right) k x k^2_{\Lambda^+} + \frac{2}{-}\ {}_1\ {}^2 k \quad r F(x) k^2_F \quad 2\sim h\ x; x i \\ & + 2\ {}_2 \sim^2 k x k_\Lambda + \ {}_1\ {}^2 \mathsf{E}_{\ {}_1;:::; \ {}_n} k\, r F(x; \ ) k^2_F \end{aligned} \tag{39}$$

C.2 Resolution: putting everything together

In this part, we combine the terms for each potential. We remind that with Assumption 3.4:

$$\sum_{i=1}^{n} \mathsf{E}\ {}_i [k\, r F_i(x_i; \ {}_i) k^2] = n\ {}^2 + \sum_{i=1}^{n} k\, r\ f_i(x_i) k^2 \tag{40}$$

$$n\ {}^2 + \sum_{i=1}^{n} 2k\, r\ f_i(x\ ) \quad r\ f_i(x_i) k^2 + 2k\, r\ f_i(x\ ) k^2 \tag{41}$$

$$n\ {}^2 + 2n\ {}^2 + 4L \sum_{i=1}^{n} d_{f_i}(x\ ; x_i) \tag{42}$$

Convex case, non-accelerated. We remind that

$${}_1(t; x; x) = A_t k x \quad x\ k^2 + B_t k\ x k^2 :$$

Then, using (23), (35) and defining ${}_1 \ , \ @_t\ {}_1(t; X_t) + \mathsf{E}[A_t \quad x + B_t \quad]$, we have:

$${}_1 \quad k x \quad x\ k^2 (A^0_t \qquad A_t) \tag{43}$$

$$+ k\ x k^2 \left(B^0_t + \frac{L}{n} A_t \quad \frac{1}{4\ {}_1} B_t \right) \tag{44}$$

$$+ \sum_{i=1}^{n} d_{f_i}(x\ ; x_i) \left(\quad \frac{2}{n} A_t + \frac{4L\ {}^2}{n^2} A_t + 4L \left(4\ {}_1\ {}^2 + \ {}^2\right) B_t \right) \tag{45}$$

$$+ \left(\frac{{}^2\ {}^2}{n} + \frac{2\ {}^2}{n}\ {}^2 \right) A_t + \left(n\ {}^2\ {}^2 + 2n\ {}^2 (4\ {}_1\ {}^2 + \ {}^2)\right) B_t \tag{46}$$

We pick $\quad = \frac{1}{2};\ B_t = \frac{1}{n} A_t$, with $A_t = e^{\ rt}$ (we denote by r the rate of the exponentials $A_t; B_t$). Then (43), (44) imply:

$$r \quad \min(\quad ; \frac{1}{4\ {}_1} \quad L\ ) \tag{47}$$

As we want (45) to be negative, we have:

$$1 + \quad \left(\frac{2L}{n} + 2L(4\ {}_1 + 1) \right) \quad 0 \tag{48}$$

which leads to:

$$\frac{1}{2L\left(\frac{1}{n} + 4\ {}_1 + 1\right)} \tag{49}$$

and taking $\quad \frac{1}{2} \frac{1}{2L(3+4\ {}_1+1)} = \frac{1}{16L(1+\ {}_1)}$ works. Now, as $\frac{1}{4\ {}_1} \quad L \quad \frac{3}{16\ {}_1}$ and $\quad \frac{1}{16\ {}_1}$, we pick $r = \quad$. As we have:

$$\mathsf{E}[\ {}_1(T; x_T) \quad {}_1(0; x_0)] = \int_0^T (A^0_t k x_t \quad x\ k^2 + B^0_t k\ x_t k^2 + A_t \mathsf{E}_{\ {}_1;:::; \ {}_n}[\ \ x] + B_t \mathsf{E}[\ \ ])\, dt \tag{50}$$

using (46) and (16) leads to:

$$\mathsf{E} k x_t \quad x\ k^2 \quad e \quad {}^t \left(k x_0 \quad x\ k^2 + \frac{1}{n} k\ x_0 k^2 \right) + - \left(\quad {}^2 (\frac{1}{n} + 1) + 2\ {}^2 (\frac{1}{n} + 4\ {}_1 + 1) \right) \tag{51}$$

Convex case with $\mathbf{A}^2\mathbf{CiD}^2$. We remind that

$$_2(t;x;\star) = A_t k x \quad x\ k^2 + B_t k\ x k^2 + \bar{B}_t k\star k_{\Lambda^+}:$$

Then, using (23), (37), (39) and defining $\ _2\ ,\ @_t\ _2(t;X_t) + \mathsf{E}[A_t\ \ x + B_t\ \ ^1_\Lambda + \bar{B}_t\ \ ^2_\Lambda]$, we have:

$$_2\quad kx\quad x\ k^2\,(A^0_t\qquad A_t) \tag{52}$$

$$+ k\ \ xk^2\left(B^0_t + \frac{L}{n}A_t\quad \frac{3}{2}\ B_t + \quad {}_1\bar{B}_t\right) \tag{53}$$

$$+ k\star k^2_{\Lambda^+}\left(B^0_t\quad \frac{}{2}B_t\right) \tag{54}$$

$$+ kxk^2_\Lambda\left(2\sim^2\ {}_2B_t\quad 2\ \ (1\qquad)B_t\right) \tag{55}$$

$$+ h\star;\ \ xi\left(2\ B_t\quad 2\sim B_t\right) \tag{56}$$

$$+ \sum_{i=1}^n d_{f_i}(x\ ;x_i)\left(\ \ \frac{2}{n}A_t + \frac{4L\ ^2}{n^2}A_t + 4L\left(\frac{2\ ^2}{}+\ \ ^2\right)(B_t + \ {}_1\bar{B}_t)\right) \tag{57}$$

$$+ \left(\frac{^2\ ^2}{n} + \frac{2\ ^2}{n}\ ^2\right)A_t + \left(n\ ^2\ ^2 + 2n\ ^2(\ ^2 + \frac{2\ ^2}{})\right)(B_t + \ {}_1\bar{B}_t) \tag{58}$$

Then, we assume $\ = \frac{1}{2}$, $\sim = \frac{1}{2}\sqrt{\frac{1}{2}}$, $\ = \frac{1}{2}\sqrt{\frac{1}{1\ \ 2}}$, $B_t = \frac{1}{n}A_t$, $\bar{B}_t = \frac{1}{1}B_t$, $A_t = e^{\ rt}$ (we denote by r the rate of the exponentials $A_t; B_t; \bar{B}_t$), which satisfies (55) and (56). Then (52), (53), (54) imply:

$$r\quad \min(\quad ;\frac{}{2};\frac{}{2}\quad L\ ) = \min(\quad ;\frac{}{2}\quad L\ ) \tag{59}$$

As we want (57) to be negative, we have:

$$1 + \quad\left(\frac{2L}{n} + 4L(\frac{2}{-} + 1)\right)\quad 0 \tag{60}$$

which leads to:

$$\frac{1}{2L\left(\frac{1}{n} + \frac{4}{-} + 2\right)} \tag{61}$$

and taking $\ \frac{1}{2L\left(6+\frac{4}{-}+2\right)} = \frac{1}{16L\left(1+\sqrt{\frac{1}{1\ \ 2}}\right)}$ works. Now, we have:

$$\frac{}{2}\quad L\quad \frac{1}{4\sqrt{\frac{1}{1\ \ 2}}}\left(1\quad \frac{4\sqrt{\frac{1}{1\ \ 2}}}{16\left(1+\sqrt{\frac{1}{1\ \ 2}}\right)}\right)\quad \frac{3}{16\sqrt{\frac{}{1\ \ 2}}} \tag{62}$$

As $\ \frac{1}{16L\left(1+\sqrt{\frac{1}{1\ \ 2}}\right)}\quad \frac{1}{16\sqrt{\frac{1}{1\ \ 2}}}$, taking $r =\ \ $ works. Finally, using (58) and (16) leads to:

$$\mathsf{E}kx_t\quad x\ k^2\quad e\quad ^t\left(kx_0\quad x\ k^2 + \frac{2}{n}k\ \ x_0k^2\right) + -\left(\ \ ^2(\frac{1}{n} + 2) + 2\ \ ^2(\frac{1}{n} + 8\sqrt{\frac{}{1\ \ 2}} + 2)\right) \tag{63}$$

Non-convex case, non-accelerated. We remind that:

$$_3(t;x) = A_t d_f(x;x\ ) + B_t k\ \ xk^2$$

Here, we pick $\ = \frac{1}{2}; A_t = 1; B_t = \frac{L}{n}A_t$. Thus, $A^0_t = B^0_t = 0$. Then, using (32), (35), (31), (25) we obtain:

$$A_t\mathsf{E}[\quad f] + B_t\mathsf{E}[\quad])\quad k\,r\,f(x)k^2\left(\ \ \frac{}{2}A_t + \frac{3}{2nL}\ ^2(M\ +1)(P\ +1)A_t + 3n\ ^2(4\ \ _1 + M\ +1)(P\ +1)B_t\right) \tag{64}$$

$$+ k\ \ xk^2\left(\frac{L^2}{2n}(1 + \frac{3}{n}(M\ +1)L\ \ )A_t + 3L^2\ \ ^2(4\ \ _1 + M\ +1)B_t\quad \frac{1}{4\ \ _1}B_t\right) \tag{65}$$

$$+\quad ^2\left(\ \ ^2 + 3(M\ +1)\ \ ^2\right)\left(\frac{L}{2n}A_t + nB_t\right) + 12n\ \ _1\ \ ^2\ \ ^2B_t \tag{66}$$

Our goal is to use half of the negative term of (64) to cancel the positive ones, so that there remains at least $-\frac{1}{4}A_t k\nabla f(x)k^2$ in the end. Thus, we want:

$$3L^2\left(\frac{(M+1)(P+1)}{2n}+(4\ _1+M+1)(P+1)\right)\quad\frac{1}{4} \tag{67}$$

and taking $\frac{1}{48(M+1)(P+1)(1+\ _1)}$ works. We verify that with defined as such, (65) is also negative. Finally, we upper bound (66) with $3L^2\left(\ ^2+3(M+1+4\ _1)\ ^2\right)$. As we have:

$$\mathbb{E}[\ _3(T;x_T)\quad{}_3(0;x_0)]=\int_0^T(A_t^0 d_f(x_t;x^\ )+B_t^0 k\ x_t k^2+A_t\mathbb{E}[\ \ f]+B_t\mathbb{E}[\ \ ])\,dt \tag{68}$$

we note that if $\frac{c}{L(\ _1+1)}$ for some constant $c>0$ which depends on $M;P$, we will get:

$$\frac{1}{4}\int_0^T\mathbb{E}[k\nabla f(x_t)k^2]\,dt\quad\frac{L}{n}k\ x_0k^2+d_f(x_0;x^\ )+O\left(LT^2(\ ^2+(1+\ _1)\ ^2)\right) \tag{69}$$

which also writes:

$$\frac{1}{T}\int_0^T\mathbb{E}[k\nabla f(x_t)k^2]\,dt\quad\frac{4}{T}(f(x_0)\quad f(x^\ ))+O\left(L\ (\ ^2+(1+\ _1)\ ^2)\right) \tag{70}$$

Non-convex case, with $\mathbf{A}^2\mathbf{CiD}^2$. We have:

$${}_4(t;x)=A_t d_f(x;x^\ )+B_t k\ xk^2+B_t k\nabla k^2_{\Lambda^+}$$

Here, we pick $=\frac{1}{2};\ =\frac{1}{2p\frac{1}{1\ 2}};A_t=1;B_t=\frac{L}{n}A_t;B_t=\ _1B_t$ and an identical reasoning to the convex setting allows to say we can find a constant $c>0$ such that if $\frac{}{L(1+p\frac{}{1\ 2})}$, then:

$$\frac{1}{T}\int_0^T\mathbb{E}[k\nabla f(x_t)k^2]\,dt=O\left(\frac{1}{T}(f(x_0)\quad f(x^\ ))+L\ (\ ^2+(1+\frac{p}{1\ 2})\ ^2)\right) \tag{71}$$

C.3 Optimizing the step-size

In this part, we follow [40, 21] and optimize the step-size a posteriori. We set $=\ _1$ for the non-accelerated setting and $=p\frac{}{1\ 2}$ with $\mathbf{A}^2\mathbf{CiD}^2$.

Strongly-convex cases: From (51) and (63), we can write that, for $\frac{1}{16L(1+\ )}$ and initializing x_0 such that $x_0=0$, we have:

$$\mathbb{E}kx_t\quad x\ k^2=O\left(kx_0\quad x\ k^2e^{\quad t}+-(\ ^2+\ ^2(1+\ ))\right) \tag{72}$$

Then, taking the proof of [40] and adapting the threshold, we consider two cases (with $r_0\ ,\ kx_0\quad x\ k^2$):

- if $\frac{1}{16L(1+\ )}\quad\frac{\log(\max\{2;\ ^2r_0T=\ ^2\})}{T}$, then we set $=\frac{\log(\max\{2;\ ^2r_0T=\ ^2\})}{T}$.
  In this case, (72) gives:
  $$\mathbb{E}kx_T\quad x\ k^2=O\left(\frac{1}{^2T}(\ ^2+\ ^2(1+\ ))\right) \tag{73}$$
- if $\frac{1}{16L(1+\ )}<\frac{\log(\max\{2;\ ^2r_0T=\ ^2\})}{T}$, then we set $=\frac{1}{16L(1+\ )}$.
  Then, (72) gives:
  $$\mathbb{E}kx_T\quad x\ k^2=O\left(r_0e^{\ \frac{T}{16L(1+\ )}}+\frac{1}{16L}\frac{1}{(1+\ )}(\ ^2+\ ^2(1+\ ))\right) \tag{74}$$
  $$=O\left(r_0e^{\ \frac{T}{16L(1+\ )}}+\frac{1}{^2T}(\ ^2+\ ^2(1+\ ))\right) \tag{75}$$

Non-convex cases: From (70) and (71), we can write that, for some constant $c > 0$ depending on $M; P$ such that $\frac{c}{L(1+\ )}$, we have:

$$\frac{1}{T}\int_0^T \mathsf{E}[k\, r\, f(x_t)k^2]\,dt = O\left(\frac{1}{T}(f(x_0) \quad f(x\ )) + L\ (\ ^2 + (1+\ )\ ^2)\right) \tag{76}$$

Then, taking the proof of Lemma 17 in [21] and adapting the threshold, we consider two cases (with $f_0\ ,\ f(x_0) \quad f(x\ )$):

- if $\frac{c}{L(1+\ )} < \left(\frac{f_0}{TL(\ ^2+(1+\ )\ ^2)}\right)^{1=2}$, then we take $\ = \frac{c}{L(1+\ )}$, giving:

$$\frac{1}{T}\int_0^T \mathsf{E}[k\, r\, f(x_t)k^2]\,dt = O\left(\frac{L(1+\ )}{T}f_0 + L\left(\frac{f_0}{TL(\ ^2+(1+\ )\ ^2)}\right)^{1=2}(\ ^2+(1+\ )\ ^2)\right) \tag{77}$$

$$= O\left(\frac{L(1+\ )}{T}f_0 + \sqrt{\frac{Lf_0}{T}(\ ^2+(1+\ )\ ^2)}\right) \tag{78}$$

- if $\frac{c}{L(1+\ )} \quad \left(\frac{f_0}{TL(\ ^2+(1+\ )\ ^2)}\right)^{1=2}$, then we take $\ = \left(\frac{f_0}{TL(\ ^2+(1+\ )\ ^2)}\right)^{1=2}$, giving:

$$\frac{1}{T}\int_0^T \mathsf{E}[k\, r\, f(x_t)k^2]\,dt = O\left(\frac{f_0}{T}\left(\frac{TL(\ ^2+(1+\ )\ ^2)}{f_0}\right)^{1=2} + \sqrt{\frac{Lf_0}{T}(\ ^2+(1+\ )\ ^2)}\right) \tag{79}$$

$$= O\left(\sqrt{\frac{Lf_0}{T}(\ ^2+(1+\ )\ ^2)}\right) \tag{80}$$

□

D Comparison with accelerated synchronous methods

By definition of (3.1), our communication complexity (the expected number of communications) is simply given by $\frac{\mathrm{Tr}(\Lambda)}{2}$ per time unit. As discussed in Sec. 3.5, our goal is to replicate the behaviour of accelerated synchronous methods such as DeTAG [31], MSDA [37] and OPAPC [23] by communicating sufficiently so that the graph connectivity does not impact the time to converge, leading to the condition $\sqrt{\ _1[\ ]\ _2[\ ]} = O(1)$.

Now, let us consider a gossip matrix W as in [31, 37, 23] (*i.e.*, W is symmetric doubly stochastic) and its Laplacian $L = I_n \quad W$. Then, using $\ = \sqrt{\ _1[L]\ _2[L]}L$ is sufficient for having $\sqrt{\ _1[\ ]\ _2[\ ]} = O(1)$.

- **Synchronous methods:** between two rounds of computations ("steps"), the number of communication edges used is $\frac{jEj}{1}$ with $\ = \max\{j\ _2j; j\ _nj\}$ the eigenvalues of W.
- **Ours:** the number of communication edges used per time unit for our method is $\frac{\mathrm{Tr}(\Lambda)}{2} = \frac{1}{2}\sqrt{\ _1[L]\ _2[L]}\mathrm{Tr}(L)$.

As, in [31, 37, 23], each communication edge is used at the same rate, we can apply **Lemma 3.3** of [34] stating: $\sqrt{\ _1[L]\ _2[L]}\mathrm{Tr}(L) \quad \sqrt{kLk\ _1(n \quad 1)jEj}$. We have:

- W is stochastic: $kLk \quad 2$.
- the graph is connected: $n \quad 1 \quad jEj$.
- definition of $\ _1$ and $: 1 \quad \frac{1}{\ _1[L]}$

Thus, $\sqrt{\ _1[L]\ _2[L]}\mathrm{Tr}(L) \quad \frac{p\ \overline{2jEj}}{1}$, which proves that our communication complexity per time unit is at least as good as any accelerated synchronous method.

E Experimental details

E.1 Graph topologies

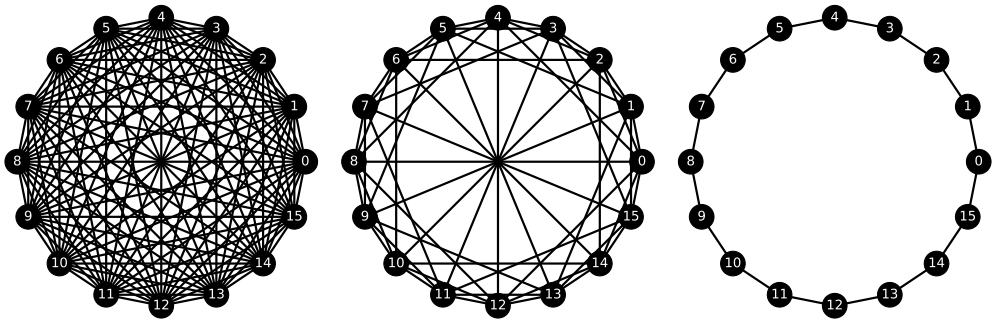


Figure 6: The three types of graph topology implemented. From left to right: complete, exponential, cycle, all with 16 nodes. From left to right, the approximate values of (χ_1, χ_2) with a communication rate of "1 p2p comm./ r comp." for each worker are: $(1, 1)$; $(2, 1)$; $(13, 1)$.

Fig. 6 displays an example of each of the three graph topologies implemented. The exponential graph follows the architecture described in [28, 2]. Note the discrepancy between the values of χ_1 and χ_2 for the cycle graph, highlighting the advantage of using $\mathbf{A}^2\mathbf{CiD}^2$ in the asynchronous setting (to lower the complexity from χ_1 to $\sqrt{\chi_1 \chi_2}$).

E.2 Uniform neighbor selection check

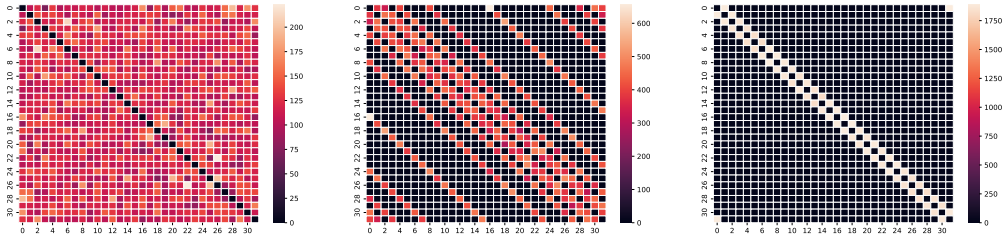


Figure 7: Heat-map of the communication history (showed through a weighted adjacency matrix) during the asynchronous training on CIFAR10 with 32 workers. We display the results for the complete graph (left), exponential (centre) and ring (right) graph.

Our asynchronous algorithm acts as follows: to reduce latency, the first two workers (*i.e.*, GPUs) in the whole pool that declare they are ready to communicate (*i.e.*, finished their previous communication and have to perform more before the next gradient step) are paired together for a p2p communication if they are neighbors in the connectivity network. During training, we registered the history of the pairwise communications that happened. Fig. 7 displays the heat-map of the adjacency matrix, confirming that our assumption of "uniform pairing among neighbors" (used to compute the values of χ_1, χ_2) seems to be sound.