Breaking (Global) Barriers in Parallel Stochastic Optimization with Wait-Avoiding Group Averaging

Shigang Li, *Member IEEE,* Tal Ben-Nun, Giorgi Nadiradze, Salvatore Di Girolamo, Nikoli Dryden, Dan Alistarh, and Torsten Hoefler, *Senior Member IEEE*

Abstract—Deep learning at scale is dominated by communication time. Distributing samples across nodes usually yields the best performance, but poses scaling challenges due to global information dissemination and load imbalance across uneven sample lengths. State-of-the-art decentralized optimizers mitigate the problem, but require more iterations to achieve the same accuracy as their globally-communicating counterparts. We present Wait-Avoiding Group Model Averaging (WAGMA) SGD, a wait-avoiding stochastic optimizer that reduces global communication via subgroup weight exchange. The key insight is a combination of algorithmic changes to the averaging scheme and the use of a group allreduce operation. We prove the convergence of WAGMA-SGD, and empirically show that it retains convergence rates similar to Allreduce-SGD. For evaluation, we train ResNet-50 on ImageNet; Transformer for machine translation; and deep reinforcement learning for navigation at scale. Compared with state-of-the-art decentralized SGD variants, WAGMA-SGD significantly improves training throughput (e.g., 2.1$\times$ on 1,024 GPUs for reinforcement learning), and achieves the fastest time-to-solution (e.g., the highest score using the shortest training time for Transformer).

Index Terms—stochastic gradient descent, distributed deep learning, decentralized optimization.

1 Introduction

The introduction of deep learning is one of the most important advancements in science over the past two decades, powering industries from autonomous driving [1] to drug discovery [2]. With the rise of deep neural networks, their training evolved into a computationally-intensive task that consumes as many resources as modern complex high-performance computing problems [3]. As a result, an abundance of research has been conducted into its scaling and distribution [4].

The leading contenders for largest workloads in deep learning are Neural Language Models [5], [6], Deep Reinforcement Learning (RL) [7], [8] and Neural Architecture Search [9]. In these regimes, computation time is measured in thousands of "GPU days", with some utilizing hundreds of accelerators (GPUs, TPUs) for several weeks [7], [10], [11].

Distributed training is largely supported by data parallelism, where sample evaluation is partitioned across processors. In this parallelism mode, all participants must exchange their gradients or model, resulting in an `Allreduce` operation across a cluster [12]. The exchange communication dominates the overall runtime [10], especially in large-minibatch stochastic gradient descent (SGD). To exacerbate the problem, certain datasets and environments are inherently imbalanced, e.g., with different sentence/video lengths [13] or heterogeneous environments in RL [14].

In order to mitigate the wait time for gradient/weight exchange, existing approaches attempt to relax model consistency between processors [4], [15]. Examples include synchronous gossip-based SGD [16], [17], asynchronous SGD [18], [19], [20], [21], and asynchronous SGD with bounded staleness [22], [23], [24], [25]. Gossip-based SGD replaces the global allreduce by communicating with randomly selected neighbors. Asynchronous SGD breaks the global synchronization to mitigate the effect of stragglers (slow processes). However, most of these approaches adversely impact convergence, necessitating an increase in the number of iterations [17], [26], sometimes to the point where synchronous waits are preferable.

In this paper, we solve this problem by introducing Wait-Avoiding Group Model Averaging (WAGMA) SGD, a novel optimizer that combines group collective communication with bounded staleness, in order to ensure competitive performance with decentralized and asynchronous methods, while retaining the convergence rate of synchronous model-averaging SGD. WAGMA-SGD locally communicates model updates across subgroups of processors, mitigating the need for global communication at every training iteration. Specifically, we propose to use a group allreduce operation for model averaging, in which the fastest process will trigger exchanges within all subgroups. Grouping is performed dynamically to facilitate model update propagation, and

- *S. Li is with Department of Computer Science, ETH Zurich. E-mail: shigangli.cs@gmail.com*
- *T. Ben-Nun is with Department of Computer Science, ETH Zurich. E-mail: talbn@inf.ethz.ch*
- *G. Nadiradze is with IST Austria. E-mail: giorgi.nadiradze@ist.ac.at*
- *S. D. Girolamo is with Department of Computer Science, ETH Zurich. E-mail: salvatore.digirolamo@inf.ethz.ch*
- *N. Dryden is with Department of Computer Science, ETH Zurich. E-mail: ndryden@ethz.ch*
- *D. Alistarh is with IST Austria. E-mail: dan.alistarh@ist.ac.at*
- *T. Hoefler is with Department of Computer Science, ETH Zurich. E-mail: htor@inf.ethz.ch*

(Corresponding Author: Shigang Li.)

Published in IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol. 32, No. 7, July 2021, https://doi.org/10.1109/TPDS.2020.3040606

as a result not only speeds up communication, but also mitigates the effect of unbalanced workloads, all without harming convergence in practice.

We theoretically prove the convergence of WAGMA-SGD, showing that, for certain parameter values, its convergence rate is comparable to synchronous SGD with model averaging. Subsequently, we test the algorithm on a supercomputer equipped with GPUs for three different categories of deep learning: supervised image classification on the ImageNet dataset; semi-supervised language modeling on the WMT17 translation dataset; and deep reinforcement learning on the Habitat indoor navigation dataset. We show that both theoretically and empirically, WAGMA-SGD is favorable over other asynchronous algorithms and the baselines, which makes it an excellent approach for scaling up distributed deep learning.

Our main contributions are:

- We propose a novel asynchronous decentralized optimizer — WAGMA-SGD, and realize it based on a wait-avoiding group allreduce operation.
- We theoretically analyze the convergence of WAGMA-SGD, showing that, under reasonable parameter values, it converges at the same rate as SGD, with linear speedup due to parallel updates.
- Compared with state-of-the-art decentralized SGD, WAGMA-SGD significantly improves the training throughput (e.g., 2.1× on 1,024 GPUs on RL), and achieves the fastest time-to-solution for all three evaluated tasks (e.g., the highest score using the shortest training time for Transformer).

2 Background and Related Work

Deep neural networks are primarily trained with mini-batch SGD [27]. Let b be the batch size, W_t the neural network weights at step t, (x_i, y_i) a set of samples of size b, and ℓ a loss function. We compute the loss for each sample as $z_i = \ell(W_t, x_i, y_i)$ and then a stochastic gradient as

$$G_t = \frac{1}{b}\sum_{i=0}^{b} \nabla \ell(W_t, z_i).$$

SGD then iterates over steps such that $W_{t+1} = W_t - \eta_t G_t$. In more general terms, first-order stochastic gradient update rules can take different forms (e.g., by adding a momentum term), which is represented as $W_{t+1} = W_t + U\left(G_t, W_{(0,\ldots,t)}, t\right)$. In distributed environments with P processors, b denotes the *local* batch size per processor. We refer to Ben-Nun & Hoefler [4] for a general overview of distributed deep learning.

Thanks to the robustness of stochastic optimization, in distributed environments one can relax weight updates by varying several axes, trading off communication overhead for convergence. Data-parallel distributed SGD algorithms can be broadly identified by the following five questions:

Q1. What is averaged?

There are two typical approaches for aggregating distributed updates: gradient and model averaging. When performing gradient averaging, we compute G_t as the average over the global batch size. With standard model averaging, the SGD update is applied locally at the node, and then the resulting model W_{t+1} is averaged over all processors.

Complementary to these approaches is the degree of quantization or sparsity in the exchanged updates. As these concepts are out of the scope of this paper, we refer to Tang et al. [15] for a comprehensive survey.

Q2. Who is coordinating the averaging?

Earlier implementations of distributed SGD for deep learning [28] use a *centralized* coordination architecture, where a parameter server or other coordinator maintains a master copy of the model that workers use. As this approach does not scale to large numbers of processors, a *decentralized* global clock can be synchronized across workers, where each worker maintains a local replica of the model and communicates updates to other workers directly.

To mitigate the overheads of global communication and synchronization, several decentralized instances of SGD have been proposed, e.g., [16], [17], [20], [26], where each worker maintains a local model but communicates updates in separate schedules, rather than synchronizing globally.

Q3. How old (stale) can averaged components be?

In a *synchronous* system, model or gradient averaging occurs when all processes are on the same training iteration t. This does not guarantee that every worker uses the same parameters (i.e., consistent model), however, standard parameter server or globally-coordinated methods ensure all workers have a consistent model. In an *asynchronous* system, averaging can occur between workers at any point. We thus define the staleness of models/gradients by τ, indicating how many iterations have passed since the produced value's model was updated. A *bounded staleness* system mitigates convergence issues with asynchronous systems by ensuring that the difference in the number of training iterations between the slowest and fastest processor is bounded, using τ as a proxy.

Q4. How often is global averaging performed?

While bounded and unbounded staleness SGD variants do not adhere to rigid communication schedules, some algorithms may periodically synchronize all processors' model replicas. This ensures that not only the staleness is bounded by τ, but also the consistency of the model is retained throughout training, mitigating its divergence across processors. In other algorithms, this global consensus is achieved post-training, by choosing the model average or the model with best generalization scores. Note that under this nomenclature, the global average frequency of synchronous variants is one step.

Q5. How many learners are averaging at every step?

In the steps between the aforementioned global model averaging period, decentralized SGD variants perform local averages with a certain group (or *quorum*) size S, leveraging the fact that several averaging steps can be performed in parallel. Removing the global communication bottleneck in decentralized SGD has been shown to enable scaling to tens and even hundreds of nodes [16], [17], [20]. However, performing averaging in pairs does come at the cost of *worse convergence*: in particular, early proposals on decentralized algorithms [16], [20] lose accuracy with respect to the synchronous baseline at scale, while more recent work [17], [26] observe that the algorithms can achieve full accuracy if executed for *more iterations* than the synchronous baseline: in particular, they execute between twice and four times more SGD iterations in total, relative to the synchronous baseline,

TABLE 1: Classification of data-parallel SGD variants.

Coordination	Staleness	Gradient Averaging	Model Averaging
Centralized	None	Parameter server [29], P3 [30]	—
	Unbounded	Hogwild! [18], Downpour SGD [28], AASGD [31]	SAPS-PSGD [32]
	Bounded	SSP [22], Rudra [33], Softsync SGD [34], Gaia [35], k-async SGD [36], Qsparse-local-SGD [37], Hybrid sync/async [38]	EASGD [23], Federated learning [39], [40]
Decentralized, $S = P$	None	**Allreduce-SGD** [41], [42], [43]	BMUF [24]
	Unbounded	—	One-shot SGD [44], WP-SGD [45], SimuParallelSGD [46]
	Bounded	**Eager-SGD** [13], Gradient lag [47]	—
Decentralized, $S = \sqrt{P}$	None	—	—
	Unbounded	—	—
	Bounded	—	**★WAGMA-SGD★**
Decentralized, $S = \mathcal{O}(1)$	None	—	**D-PSGD** [16], **SGP** [17]
	Unbounded	GossipGraD [21], Choco-SGD [48]	**AD-PSGD** [20], Gossiping SGD [49], SwarmSGD [26]
	Bounded	CDSGD [50]	**Local SGD** [25], [51], [52]

erasing much of the speedup due to increased scalability. This decreased convergence behavior is connected to the analytical bounds provided by these algorithms: while the theoretical convergence rates suggest linear speedup with respect to the number of SGD steps and executing nodes, these rates only apply after a very large number of SGD steps have been taken, in order to allow the pairwise averaging process to "mix" well, thereby simulating all-to-all averaging. See Section 4.1 for a detailed discussion.

2.1 Training at Scale

An orthogonal challenge to distributed stochastic optimization is that of unbalanced workloads. Imbalance may be caused by the training system [13], [20], [53] or by the task itself [13], [14]. Training on multi-tenant cloud systems can suffer from performance variability due to resource sharing. Several deep learning tasks, such as video classification and machine translation, have inherent load imbalance, because input/output sequences have different lengths [13]. In deep reinforcement learning, an agent must interact with the environment to generate training data. For RL tasks using heterogeneous environments [14], the runtime of training data generation varies significantly. Quantitative profiling for the load imbalance of language modeling and RL is presented in Section 5.3 and Section 5.4, respectively.

2.2 Comparison Targets

In Table 1 we summarize and classify the distributed SGD algorithms most relevant to our work. Algorithms in **bold** are used for comparison in this work. Since decentralized algorithms typically scale and perform better on large-scale systems than centralized algorithms, we limit our comparison to decentralized algorithms. The algorithms with which we compare our approach are chosen specifically to be spread across the different answers to the above five questions, prioritizing popular algorithms with proven convergence, both in theory and in practice:

- Allreduce-SGD is the standard data-parallel training.
- Local SGD [25], [51], [52] performs a fixed number of local iterations of SGD (a hyperparameter determined by the user) and then averages the models over all processes with a standard allreduce. Several variants with different methods for determining the frequency of global averaging exist.
- Decentralized parallel SGD (D-PSGD) [16] uses a ring topology, where each process averages its local model with its two neighbors. Processes advance synchronously with a single global clock.
- Stochastic gradient push (SGP) [17] generalizes the topology used in D-PSGD to support more flexible, asymmetric communication patterns.
- Eager-SGD [13] uses partial collective allreduces over the gradients, allowing at most half processors to contribute stale gradients if not ready.
- For Asynchronous decentralized parallel SGD (AD-PSGD) [20], each worker runs a communication thread and a computation thread in parallel. The computation thread calculates the model updates while the communication thread exchanges models with a selected communication neighbor. AD-PSGD extends the idea of D-PSGD by allowing workers to exchange models at any time.

These cover nearly all varieties of consistency and averaging, and practical differences in communication patterns. We do not include a comparison to a decentralized, asynchronous, gradient averaging algorithm, as GossipGraD does not show good convergence for many applications and Choco-SGD has not yet been applied to deep learning.

2.3 Discussion

Following the discussion on the impact of quorum size on convergence (**Q5**), it is natural to ask whether performing decentralized averaging *in larger groups* would be able to provide the best of both worlds, enabling the full convergence of the synchronous algorithm, and the scalability of fully decentralized ones. There are two main barriers to this solution: the first one is at the implementation level, since, to our knowledge, no efficient non-blocking implementation of group model averaging exists. The second is

at the application level, since it is not clear whether group averaging would be able to achieve the same convergence as the synchronous solution (both in theory and in practice). In the following sections, we address both of these issues.

3 Wait-Avoiding Group Communication

The allreduce operation [12] is defined as a reduction whose results are shared among all participants. Although several optimizations [41], [54], [55], [56] have been designed to improve the performance of this collective, allreduce poses an implicit global synchronization point, which makes the operation vulnerable to stragglers during deep learning training. On larger systems, the performance of the compute nodes can be impacted by different internal (e.g., load imbalance) and external factors (e.g., OS or network [57] noise), potentially increasing the synchronization overhead. We define this collective as *synchronous allreduce*. While non-blocking collectives [58] can alleviate the synchronization overhead, they do not fully remove it and completion still waits. Even if the participating processes are perfectly synchronized, the optimal scaling of an allreduce of size N is at best $\mathcal{O}(\log P + N)$ for P processes [10], [59]. Therefore, growing process counts will reduce the parallel efficiency and eventually make the reduction a scaling bottleneck.

3.1 Wait-Avoiding Group Allreduce

To overcome the synchronization overhead and overall collective cost, we introduce a new class of *wait-avoiding group collectives*, focusing on *group allreduce* for the purpose of this work. We relax synchronization by making the collectives externally-triggerable [13], [60], namely, a collective can be initiated without requiring that all processes enter it, by externally activating the communication schedule of *late* processes with activation messages sent by the *early* ones. Once activated, a group allreduce does not perform a global reduction. Instead, it partially reduces the data within non-overlapping groups of processes, limiting the number of communications needed to implement the group collective.

3.1.1 Collective activation

In a wait-avoiding group allreduce, any process can make progress regardless of what the other processes are working on. This wait-avoidance is achieved by the activation component. We call the process reaching the collective call first the *activator*. The activator is in charge of informing the other processes that an allreduce operation has started and that they have to participate, regardless of whether they reached the collective call-site.

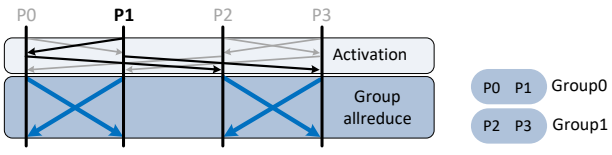

Fig. 1: Wait-avoiding group allreduce on four processes with a group size of two. P1 arrived first and activates the operation.

In a wait-avoiding group allreduce, any process can initiate the collective. We use a modified version of the recursive doubling algorithm that builds a butterfly topology, which can be seen as a set of overlapping binomial trees, one rooted at each process. Any node can activate the collective by sending activation messages along the binomial tree rooted at itself. Fig. 1 shows an example where P1 is the activator. In this case, P1 uses its broadcast tree and sends the activation messages to P0 and P3. Once activated, P0 first forwards the activation message to P2, after which P0 starts executing its group allreduce schedule.

It is possible that several processes arrive at the wait-avoiding group allreduce operation at close proximity, which means we may have more than one activator during the activation phase. To guarantee that a process does not execute the same collective twice, we assign each operation a version number that is increased every time the collective is executed. The collective version number is encoded in the tag of the activation messages: once an activation is received, the collective schedule is activated only if its version number is lower than or equal to the one carried by the activation message. The version number check is executed also when a process reaches the collective: if it fails, then the version of the collective that the process wants to activate has already been executed (and the process has passively participated in it). In this case, no activation messages are sent.

3.1.2 Asynchronous execution

To enable asynchronous execution of the custom collectives, we extend the `fflib` communication library [60], adding support for wait-avoiding group allreduce. `fflib` allows programmers to customize collective operations via a flexible, DAG-based representation of point-to-point and local compute operations, defined as *schedules*. The library provides a C-based interface for schedule creation and non-blocking invocation, using MPI as its primary backend, with additional support for network offloading engines such as sPIN [61]. Our defined schedule for group operations models both the activation and group allreduce phases.

3.2 Dynamic Grouping Strategy

As discussed in Section 2, in data-parallel SGD variants such as allreduce SGD [42], [43] and gossip SGD [16], [17], [20], each process keeps propagating local model updates to the other processes at every iteration to make global progress. We propose a dynamic grouping strategy to reduce the latency (in steps) of local update propagation. Together with the group allreduce operation, the grouping strategy guarantees that the local updates can be globally propagated within $\log P$ iterations. The larger the group size, the faster the updates are propagated. By carefully selecting the group size, we can achieve both lower latency than gossip SGD and efficient communication by reducing contention.

We define the dynamic grouping strategy in Algorithm 1. We assume the number of processes P is a power-of-two, which is a common case in current distributed training systems. The group size S ($\leq P$) is also set to a power-of-two. In line 2, we initialize the *mask*, and calculate the number of phases in a butterfly topology for P and S processes, respectively. Line 3 initializes the *shift*. In each training iteration t, the algorithm first initializes P groups, each of which contains one process (line 4). In line 8, an

equivalence relation between each pair of processes is found using the bitwise XOR operation. For a pair of processes with an equivalence relation (i.e., $p \equiv q$), we find the groups p and q belong to, respectively (line 9); if p and q are not in the same group, we merge the two groups into one using the union operation (lines 10–12). In line 15, the processes will have been partitioned into P/S groups in iteration t. Note that the initial value of *shift* is periodically changing in every iteration (line 3), which, in turn, changes the group composition in every iteration.

Algorithm 1 Dynamic grouping strategy

1: **Input:** Total P processes. S is the group size. t is the training iteration.
2: $mask = 1$, $global\_phases = log_2 P$, $group\_phases = log_2 S$
3: $shift = (t * group\_phases) \mod global\_phases$
4: ***make each process an individual group*** ▷ initialize P groups
5: **for** $r = 1$ **to** $group\_phases$ **do**
6: $mask <<= shift$ ▷ bitwise left shift on $mask$
7: **for** $p = 0$ **to** $P - 1$ **do**
8: $q = p$ XOR $mask$ ▷ equivalence relation $p \equiv q$
9: **Find groups**: $p \in group\_p$, $q \in group\_q$
10: **if** $group\_p \neq group\_q$ **then**
11: **Merge groups**: $group\_merge = group\_p \cup group\_q$
12: **end if**
13: **end for**
14: $shift = (shift+1) \mod global\_phases$
15: **end for** ▷ processes are partitioned into P/S groups in iteration t

To demonstrate dynamic grouping, we use $P = 8$ and $S = 4$ as an example. In iteration 0, all processes are initially partitioned into 8 groups. The set of equivalence relations includes $0 \equiv 1$, $2 \equiv 3$, $4 \equiv 5$, $6 \equiv 7$, $0 \equiv 2$, $1 \equiv 3$, $4 \equiv 6$, and $5 \equiv 7$. By recursively merging the two groups in which a pair of processes with a equivalence relation belongs to, we obtain two non-overlapping groups, which contain the processor sets {0, 1, 2, 3} and {4, 5, 6, 7}. In iteration 1, the set of equivalence relations changes; thus, the grouping changes accordingly (i.e., {0, 1, 4, 5} and {2, 3, 6, 7}).

Note that we only use Algorithm 1 to formally describe the grouping strategy. The grouping strategy together with allreduce within each group is implemented concisely following the phases of the butterfly topology, namely each pair of processes with a equivalence relation in a phase would exchange messages. We use the variable t to change the phases that should be executed in the current iteration. Fig. 2 presents the iterative execution of group allreduce with dynamic grouping in WAGMA-SGD, and grouping is shown on the right side. We can see that although the group size is fixed, the groups are dynamically changing during the iterations. Within each group, the allreduce is conducted following $\log_2 S$ phases of the butterfly topology. To maintain convergence with this communication scheme in data-parallel deep learning training, a standard synchronous allreduce across all processes is conducted every τ iterations, bounding the staleness of the weights. In the following section, we will present the algorithm in detail and discuss this periodic synchronization further.

4 Wait-Avoiding Group Model Averaging

Based on the insight that larger groups converge faster, and on the novel implementation of wait-avoiding group collectives, we design the Wait-Avoiding Group Model Averaging (WAGMA) SGD algorithm. WAGMA-SGD can be

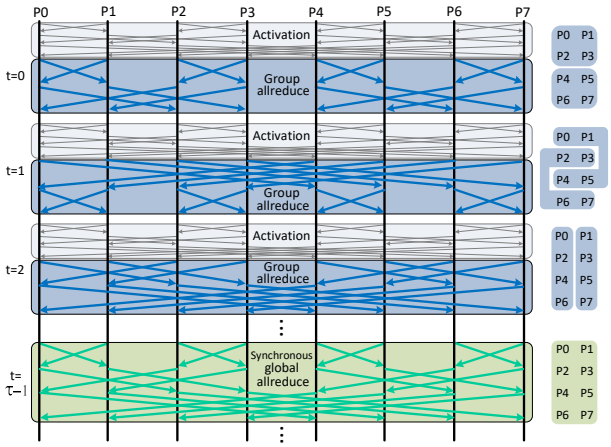

Fig. 2: Communication scheme of WAGMA-SGD. Total 8 processes and the group size is 4. Every τ iterations, the algorithm synchronizes globally.

Algorithm 2 WAGMA-SGD

1: **Input:** b is local batchsize for P processes. S is the group size of the processes. τ is the synchronization period.
2: **for** $t = 0$ **to** $T - 1$ **do**
3: $\vec{x}, \vec{y} \leftarrow$ Each process samples b elements from dataset
4: $\vec{z} \leftarrow \ell(W_t, \vec{x}, \vec{y})$
5: $G_t^{local} \leftarrow \frac{1}{b} \Sigma_{i=0}^{b} \nabla \ell(W_t, \vec{z}_i)$
6: $\Delta W_t \leftarrow U(G_t^{local}, W_{(0,\ldots,t)}, t)$
7: $W_t' \leftarrow W_t + \Delta W_t$
8: **if** $(t+1) \mod \tau \neq 0$ **then**
9: $W_t^{sum} \leftarrow wait\text{-}avoiding\_group\_allreduce(W_t', t)$
10: **if** W_t' *is not stale* **then**
11: $W_{t+1} \leftarrow \frac{1}{S} W_t^{sum}$
12: **else**
13: $W_{t+1} \leftarrow \frac{1}{S+1} (W_t^{sum} + W_t')$
14: **end if**
15: **else**
16: $W_{t+1} \leftarrow \frac{1}{P} sync\_allreduce(W_t')$
17: **end if**
18: **end for**

classified as a *model-averaging*, *bounded-staleness*, *decentralized* SGD with a *group size* of $S \propto \sqrt{P}$ and a *global communication period* of τ steps. As listed in Algorithm 2, WAGMA-SGD is similar to minibatch SGD, but makes a few important distinctions. In lines 3–7, each process calculates the local gradients G_t^{local} and then applies the local gradients to derive and apply the model update ΔW_t. Subsequently, the wait-avoiding group model averaging is conducted (lines 8–17) using the aforementioned wait-avoiding communication scheme. From an algorithmic perspective, WAGMA-SGD does not rely on certain choice of group members for the local collectives. However, instead of randomly choosing groups of processes, we use the butterfly strategy (Algorithm 1) for efficient and deterministic communication.

In each iteration, faster processes will trigger the model averaging immediately without waiting (line 9, t is used to control grouping), which may incur averaging the local models with some stale models from the slower processes. To both bound the staleness and mitigate divergence across local model replicas, we define a synchronization period τ, in which the models are averaged across all processes using a global allreduce (line 16). Empirically, we set the synchronization period τ to 10 training iterations, which

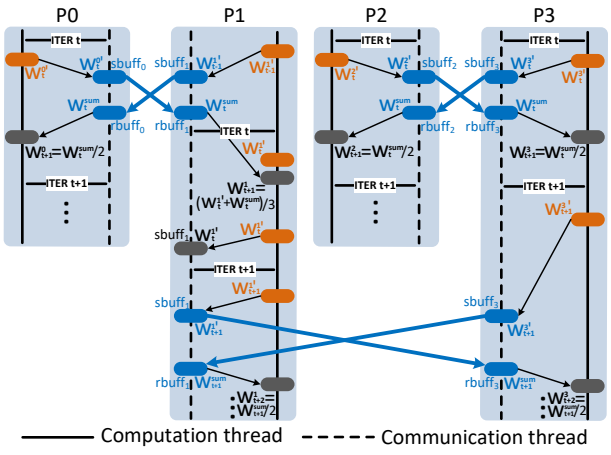

Fig. 3: WAGMA-SGD execution snapshot for P=4 and S=2.

balances model accuracy with training throughput, as we will show in Section 5.

From the algorithm perspective, there are four parameters in WAGMA-SGD, which can be used to approximately represent other parallel SGD variants by adjusting the parameters. The parameters include τ, a boolean variable α to indicate whether to use the wait-avoiding group allreduce, a boolean variable β to indicate whether to use a normal group allreduce (without activation component), and the group size S. Note that α and β can not be *true* at the same time. If α == β == *false*, WAGMA-SGD degrades to local SGD with global communication period τ. If α == *true*, β == *false*, S == O(1) and τ == ∞, WAGMA-SGD is similar to AD-PSGD but WAGMA-SGD requires to wait for the fastest process (i.e., the *activator*). If α == *false*, β == *true*, S == O(1) and τ == ∞, WAGMA-SGD is similar to D-PSGD and SGP.

An execution snapshot of WAGMA-SGD ($P = 4$ and $S = 2$) is presented in Fig. 3. Suppose P1 is a straggler. When the group allreduce in iteration t is triggered by any of the other three processes, P1 contributes the stale model parameters $W_{t-1}^{1'}$ while the other three processes contribute up-to-date models. In iteration t, P1 and P0 are in the same group; therefore, $W_{t-1}^{1'}$ and $W_t^{0'}$ will be added together to derive W_t^{sum}. P0 will use the averaged model W_{t+1}^0 (line 11 in Algorithm 1) for the next iteration of training. P1 subsequently finishes the calculation for the local updated model in iteration t (i.e., $W_t^{1'}$), but finds out that the group allreduce in iteration t is already finished by checking the value of a preset tag variable in the receive buffer. In this case, P1 will average the stale model $W_t^{1'}$ with W_t^{sum} (line 13 in Algorithm 1), and the averaged model W_{t+1}^1 will be used for the next iteration of training. Meanwhile, the data in the send buffer of P1 is updated by $W_t^{1'}$. If the group allreduce in iteration $t+1$ is triggered by some faster process at this time, P1 will continue to passively contribute the stale model $W_t^{1'}$. When a standard allreduce is called at the synchronization point, all processes are forced to contribute the model parameters after training for the same number of iterations. In Fig. 3, P1 catches up with the other processes in iteration $t+1$; thus, P1 will contribute the timely model $W_{t+1}^{1'}$ to P3, as they are in the same group.

4.1 Proof of Convergence

4.1.1 Algorithm Modelling

For analysis purposes, we will model the algorithm's execution as follows. We will proceed in steps, indexed by time $t \geq 0$. Each node i maintains its own local model W_t^i, and has a local partition of the data. In each step, a group of nodes of size S is chosen to interact. Each node takes a local gradient step, and then nodes average their models. This averaging step might be inconsistent, as per the above semantics. In the analysis, we will assume that the group of S interacting nodes is chosen uniformly at random—in the long run, the resulting interaction graph will have the same properties as the butterfly interaction strategy used in the implementation. While our analysis considers each interaction sequentially, in practice $\Theta(P/S)$ interaction steps can occur in parallel.

4.1.2 Setup and Analytic Assumptions

We will assume a standard setting in which we are given a dataset of D samples $\mathcal{D} = \{e_1, e_2, \ldots, e_D\}$, and to each associate a differentiable loss function $f_e : \mathbb{R}^d \to \mathbb{R}$. Each node i is given a random partition $\mathcal{D}_i$ of the dataset $\mathcal{E}$, and we wish to solve an empirical risk minimization problem by finding

$$x^\star = \text{argmin}_{x \in \mathbb{R}^d} \left[F(x) := \frac{1}{D} \sum_{e \in \mathcal{D}} f_e(x) \right].$$

Let $F_i = \frac{P}{D} \sum_{e \in \mathcal{D}_i} f_e$ be the loss function corresponding to the dataset of the ith node, and $F^\star = F(x^\star)$. To make the analysis tractable, we will make the following standard assumptions on the loss function.

Assumption 1. *We assume the following hold:*
1) (Lipschitz gradients) *All functions f_e have L-Lipschitz gradient, for some constant L.*
2) (Bounded Second Moment) *There exists a constant M such that for any node i and $w \in \mathbb{R}^d$, $\mathbb{E}_{e \in D_i} \|\nabla f_e(w)\|^2 \leq M^2$.*
3) (Bounded Staleness) *The staleness during the averaging step is upper bounded by a parameter τ. That is, for any node i participating in the averaging step at time t, averaging is performed with respect to model $W_{t'}^i$, where $t' \geq t - \tau + 1$, and every gradient update is applied globally at most τ steps after it was generated.*

4.1.3 Convergence result

We can now state the main convergence result. For readability, we state a simplified variant that highlights the relationship between parameter values, in particular the relation between the convergence time T, P processors, and the size of the interacting group S. The detailed proof will be presented in Section 4.1.4.

Theorem 4.1. *Consider the setting described above, in which we wish to optimize a non-convex function $F : \mathbb{R}^d \to \mathbb{R}$. Let S be the size of a communication group, and assume that the maximum staleness τ is constant. Fix a success parameter $\varepsilon > 0$. For each time t, we define $\mu_t = \sum_{i=1}^P W_t^i$ to be the average of local models at time t. Then there exists a setting of the learning rate in the order of $P/\sqrt{T}$ such that, if the algorithm has taken*

$$T = \Omega\left(\max\left\{\frac{(F(W_0)-F^\star)^2}{\epsilon^2}, \frac{S^4M^4L^4}{\epsilon^2}, P^4\tau^4\right\}\right) \text{ steps,}$$

then there exists an iterate $0 \leq T^\star \leq T$ *such that*

$$\mathbb{E}\|\nabla F(\mu_{T^\star})\|^2 \leq \varepsilon,$$

where the expectation is taken w.r.t. the randomness in the sampling and interactions.

At a high level, this claim shows that the algorithm will eventually reach a point where the model average has negligible gradient, i.e., is at a local minimum. While this does not guarantee convergence to a global minimum, it matches the best achievable guarantees for SGD in the non-convex setting [62]. The convergence proof, provided as additional material, generalizes the decentralized asynchronous framework of Nadiradze et al. [26] since we allow for group communication, whereas they only consider pairwise interactions.

It is interesting to examine the rate at which convergence occurs — for standard parameter settings, i.e. constant τ, S, M and L, the convergence (i.e., the rate at which we get to a point of negligible gradient) matches that of standard SGD, and the speedup with respect to the number of nodes can be *linear* in P. More precisely, we can examine the three terms in the lower bound on the number of steps T to convergence given by the Theorem. The first is the standard convergence rate for SGD. (Recall however that we express this bound in terms of *total* steps, whereas P/S such steps can occur in parallel.) The second bounds the impact of the variance on the total number of iterations, whereas the third suggests that, to negate the impact of asynchrony, the algorithm has to execute for at least $P^4\tau^4$ steps. The convergence ensured by the first term is the best possible, and matches the rates for other decentralized algorithms, e.g. [16], [20].

4.1.4 Proof

In this section, we focus on providing a formal proof for the convergence statement provided in Theorem 4.1.

Preliminaries: Let $\mathcal{K}$ be a set all subsets of $\{1, 2, ..., P\}$, which have size S. if set $K \in \mathcal{K}$ is chosen for interaction. we have that for each $i \in K$

$$W^i_{t+1/2} = W^i_t/S + \sum_{j\in K/\{i\}} W^{j'}_t/S.$$

and

$$W^i_{t+1} = W^i_{t+1/2} - \eta\tilde{G}(W^i_t)$$

Where for each process i, if $t \geq q^i_t > t-\tau$ is the last time it interacted before the step t, we have

$$W^{i'}_t = W^i_{q^i_t+1/2} = W_{q^i_t+1} + \eta\tilde{G}(W^i_{q^i_t}) = W^i_t + \eta\tilde{G}(W^i_{q^i_t}). \quad (1)$$

Let $\mathbb{E}_t$ be expectation which is conditioned on the entire history up to and including step t. This means that

$$\mathbb{E}_t[\mu_{t+1}] = \mu_t + \frac{1}{\binom{P}{S}}\sum_{K\in\mathcal{K}}\sum_{i\in K}\mathbb{E}_t\Big[-\frac{\eta\tilde{G}(W^i_t)}{P} + \frac{\eta(S-1)\tilde{G}(W^i_{q^i_t})}{PS}\Big]$$
$$= \mu_t + \frac{S}{P}\sum_{i=1}^{P}\Big(-\frac{\eta\nabla f(W^i_t)}{P} + \frac{\eta(S-1)\tilde{G}(W^i_{q^i_t})}{PS}\Big). \quad (2)$$

Potential Bound: Now we derive bound on the potential $\Gamma_t = \sum_{i=1}^P \|W^i_t - \mu_t\|^2$ in expectation. Observe that for each t, $\Gamma_{r_t} = 0$, where $t \geq r_t > t-\tau$ is the last time global averaging happened. Hence we need to bound the change over τ steps. Using induction we show that:

Lemma 4.2. *for each* $t \geq 0$ *and process* i*:*

$$\mathbb{E}\,\|W^i_t - \mu_{r_t}\|^2 \leq 4\eta^2M^2(t-r_t)^2 \leq 4\eta^2\tau^2M^2. \quad (3)$$

Proof. Base case $t = r_t$ holds trivially. We assume that the claim holds for t (and every process i), and we show that it holds for $t+1$. Let K be a set chosen for averaging. If $i \in K$ then we have that

$$W^i_{t+1} = \sum_{j\in K} W^j_t/S - \eta\tilde{G}(W^i_t) + \sum_{j\in K/\{i\}}\eta\tilde{G}(W^j_{q^j_t})/S.$$

Hence, if $t > r_t$

$$\|W^i_{t+1} - \mu_{r_t}\|^2 =$$
$$\Big\|\sum_{j\in K}(W^j_t - \mu_{r_t})/S - \eta\tilde{G}(W^i_t) + \sum_{j\in K/\{i\}}\eta\tilde{G}(W^j_{q^j_t})/S\Big\|^2$$
$$\overset{Young}{\leq} \frac{1}{S^2}(1+\frac{1}{t-r_t})\Big\|\sum_{j\in K}(W^j_t - \mu_{r_t})\Big\|^2 +$$
$$(1+t-r_t)\Big\| - \eta\tilde{G}(W^i_t) + \sum_{j\in K/\{i\}}\eta\tilde{G}(W^j_{q^j_t})/S\Big\|^2$$
$$\overset{Cauchy-Schwarz}{\leq} \frac{1}{S}(1+\frac{1}{t-r_t})\sum_{j\in K}\Big\|W^j_t - \mu_{r_t}\Big\|^2$$
$$+ 2(1+t-r_t)\eta^2\Big\|\tilde{G}(W^i_t)\Big\|^2$$
$$+ \frac{2\eta^2(1+t-r_t)}{S^2}\Big\|\sum_{j\in K/\{i\}}\tilde{G}(W^j_{q^j_t})\Big\|^2$$
$$\overset{Cauchy-Schwarz}{\leq} \frac{1}{S}(1+\frac{1}{t-r_t})\sum_{j\in K}\Big\|W^j_t - \mu_{r_t}\Big\|^2$$
$$+ 2(1+t-r_t)\eta^2\Big\|\tilde{G}(W^i_t)\Big\|^2$$
$$+ \frac{2\eta^2(1+t-r_t)(S-1)}{S^2}\sum_{j\in K/\{i\}}\Big\|\tilde{G}(W^j_{q^j_t})\Big\|^2$$

Next we take expectations:

$$\mathbb{E}\,\|W^i_{t+1} - \mu_{r_t}\|^2 \leq \frac{1}{S}(1+\frac{1}{t-r_t})\sum_{j\in K}\mathbb{E}\Big\|W^j_t - \mu_{r_t}\Big\|^2$$
$$+ 2(1+t-r_t)\eta^2\,\mathbb{E}\Big\|\tilde{G}(W^i_t)\Big\|^2$$
$$+ \frac{2\eta^2(1+t-r_t)(S-1)}{S^2}\sum_{j\in K/\{i\}}\mathbb{E}\Big\|\tilde{G}(W^j_{q^j_t})\Big\|^2$$

Using the second moment bound we get

$$\begin{aligned}\mathbb{E}\left\|W_{t+1}^i-\mu_{r_t}\right\|^2 &\le \frac{1}{S}(1+\frac{1}{t-r_t})\sum_{j\in K}\mathbb{E}\left\|W_t^j-\mu_{r_t}\right\|^2\\ &+2(1+t-r_t)\eta^2M^2\\ &+\frac{2\eta^2(1+t-r_t)(S-1)^2M^2}{S^2}\\ &\le \frac{1}{S}(1+\frac{1}{t-r_t})\sum_{j\in K}\mathbb{E}\left\|W_t^j-\mu_{r_t}\right\|^2\\ &+4(1+t-r_t)\eta^2M^2\end{aligned}$$

Finally using the assumption that claim holds for t (and every i), we get that

$$\begin{aligned}\mathbb{E}\left\|W_{t+1}^i-\mu_{r_t}\right\|^2 &\le 4\eta^2M^2(t-r_t)^2(1+\frac{1}{t-r_t})\\ &+4(1+t-r_t)\eta^2M^2\\ &=4(t-r_t+1)^2\eta^2M^2.\end{aligned}$$

If i does not belong to K, claim for $t+1$ holds trivially (since i does not perform SGD step).

□

This allows us to show the following lemma:

Lemma 4.3.

$$\mathbb{E}\left\|\mu_t-\mu_{r_t}\right\|^2\le 4\eta^2M^2\tau^2.$$

Proof. Observe that since $\mu_t=\sum_{i=1}^P W_t^i/P$, by Jensen's inequality we have that

$$\mathbb{E}\left\|\mu_t-\mu_{r_t}\right\|^2\le\sum_{i=1}^P\mathbb{E}\left\|W_t^i-\mu_{r_t}\right\|^2/P\overset{\text{Lemma 4.2}}{\le}4\eta^2M^2\tau^2.$$

□

Next we need another lemma which upper bounds discrepancy between μ_t and $W_{q_t^i}^i$ for each process i:

Lemma 4.4. *For each $t\ge 0$ and process i we have that:*

$$\mathbb{E}\left\|W_{q_t^i}^i-\mu_t\right\|^2\le 16\eta^2M^2\tau^2.$$

Proof. Recall that for each t and i, $t\ge q_t^i>t-\tau$. Hence we have that $r_t=r_{q_t^i}$. From Lemmas 4.3 and 4.2 it follows that $\mathbb{E}\left\|\mu_{r_t}-W_{q_t^i}^i\right\|^2\le 4\eta^2M^2\tau^2$ and $\mathbb{E}\left\|\mu_t-\mu_{r_t}\right\|^2\le 4\eta^2M^2\tau^2$. Thus by using Cauchy-Schwarz inequality we get that

$$\begin{aligned}\mathbb{E}\left\|W_{q_t^i}^i-\mu_t\right\|^2&\le 2\,\mathbb{E}\left\|W_{q_t^i}^i-\mu_{r_t}\right\|^2+2\,\mathbb{E}\left\|\mu_t-\mu_{r_t}\right\|^2\\ &\le 16\eta^2M^2\tau^2.\end{aligned}$$

□

Finally we upper bound the Gamma potential in expectation:

Lemma 4.5. *For any $t\ge 0$, we have that*

$$\mathbb{E}[\Gamma_t]=\sum_{i=1}^P\mathbb{E}\left\|W_t^i-\mu_t\right\|^2\le 16P\eta^2M^2\tau^2. \tag{4}$$

Proof. For each process i, we have that:

$$\mathbb{E}\left\|W_t^i-\mu_t\right\|^2\le 2\,\mathbb{E}\left\|W_t^i-\mu_r^t\right\|^2+2\,\mathbb{E}\left\|\mu_t-\mu_r^t\right\|^2\le 16\eta^2M^2\tau^2. \tag{5}$$

Where first inequality is Cauchy-Schwarz and second inequality is combination of Lemmas 4.2 and 4.3. The upper bound we need to prove the lemma follows trivially. □

Convergence Proof: We are now ready to state and prove our main theorem. We provide a slightly more general version, which implies Theorem 4.1.

Theorem 4.6. *For a smooth non-convex objective function f, whose minimum x^* we are trying to find using WAGMA algorithm and the constant learning rate $\alpha=\frac{P}{\sqrt{T}}$, where $T\ge P^4\tau^4$ is the number of iterations:*

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(\mu_t)\right\|^2\le\frac{2(f(\mu_0)-f(x^*))}{\sqrt{T}}+\\ \frac{8S^2M^2}{\sqrt{T}}+\frac{64S^2L^2M^2}{\sqrt{T}}.\end{aligned}$$

Proof. Recall that $\mathbb{E}_t$ is expectation which is conditioned on the entire history up to and including step t. By descent Lemma we know that

$$\begin{aligned}\mathbb{E}_t[f(\mu_{t+1})]\le f(\mu_t)+\mathbb{E}_t\langle\nabla f(\mu_t),\mu_{t+1}-\mu_t\rangle\\ +\frac{L}{2}\,\mathbb{E}_t\left\|\mu_{t+1}-\mu_t\right\|^2.\end{aligned} \tag{6}$$

First, we look at $\mathbb{E}_t\left\|\mu_{t+1}-\mu_t\right\|^2$:

$$\begin{aligned}&\mathbb{E}_t\left\|\mu_{t+1}-\mu_t\right\|^2=\\ &\frac{1}{\binom{P}{S}}\sum_{K\in\mathcal{K}}\mathbb{E}_t\left\|\sum_{i\in K}\Big(-\frac{\eta\tilde{G}(W_t^i)}{P}+\frac{\eta(S-1)\tilde{G}(W_{q_t^i}^i)}{PS}\Big)\right\|^2.\end{aligned}$$

Next we apply Cauchy-Schwarz inequality and get:

$$\begin{aligned}&\mathbb{E}_t\left\|\mu_{t+1}-\mu_t\right\|^2\le\\ &\frac{S}{\binom{P}{S}}\sum_{K\in\mathcal{K}}\sum_{i\in K}\mathbb{E}_t\left\|-\frac{\eta\tilde{G}(W_t^i)}{P}+\frac{\eta(S-1)\tilde{G}(W_{q_t^i}^i)}{PS}\right\|^2\\ &=\frac{S^2\eta^2}{P^3}\sum_{i=1}^P\mathbb{E}_t\left\|-\tilde{G}(W_t^i)+\frac{(S-1)\tilde{G}(W_{q_t^i}^i)}{S}\right\|^2\\ &\overset{Cauchy-Schwarz}{\le}\frac{2S^2\eta^2}{P^3}\sum_{i=1}^P\mathbb{E}_t\left\|-\tilde{G}(W_t^i)\right\|^2+\\ &\frac{2(S-1)^2\eta^2}{P^3}\sum_{i=1}^P\mathbb{E}_t\left\|\tilde{G}(W_{q_t^i}^i)\right\|^2\\ &\le\frac{2S^2M^2\eta^2}{P^2}+\frac{2(S-1)^2\eta^2}{P^3}\sum_{i=1}^P\left\|\tilde{G}(W_{q_t^i}^i)\right\|^2.\end{aligned} \tag{7}$$

Now, we upper bound $\mathbb{E}_t\langle\nabla f(\mu_t),\mu_{t+1}-\mu_t\rangle$. Using (2), we

have that

$$\begin{aligned}
&\mathbb{E}_t\langle\nabla f(\mu_t), \mu_{t+1}-\mu_t\rangle = \\
&\frac{\eta S}{P^2}\sum_{i=1}^{P}\langle\nabla f(\mu_t), -\nabla f(W_t^i) + \frac{S-1}{S}\tilde{G}(W_{q_t^i}^i)\rangle \\
&= \frac{\eta S}{P^2}\sum_{i=1}^{P}\langle\nabla f(\mu_t), \frac{S-1}{S}\tilde{G}(W_{q_t^i}^i)\rangle + \\
&\frac{\eta S}{P^2}\sum_{i=1}^{P}\langle\nabla f(\mu_t), \nabla f(\mu_t) - \nabla f(W_t^i)\rangle - \frac{\eta S}{P}\|\nabla f(\mu_t)\|^2 \\
&\overset{Young}{\leq} \frac{\eta S}{P^2}\sum_{i=1}^{P}\langle\nabla f(\mu_t), \frac{S-1}{S}\tilde{G}(W_{q_t^i}^i)\rangle + \frac{\eta}{4P}\|\nabla f(\mu_t)\|^2 \\
&\quad + \frac{\eta S^2}{P^2}\sum_{i=1}^{P}\|\nabla f(\mu_t) - \nabla f(W_t^i)\|^2 - \frac{\eta P}{S}\|\nabla f(\mu_t)\|^2 \\
&\leq \frac{\eta(S-1)}{P^2}\sum_{i=1}^{P}\langle\nabla f(\mu_t), \tilde{G}(W_{q_t^i}^i)\rangle \\
&\quad + \frac{\eta S^2 L^2}{P^2}\sum_{i=1}^{P}\|\mu_t - W_t^i\|^2 - \frac{\eta(S-\frac{1}{4})}{P}\|\nabla f(\mu_t)\|^2.
\end{aligned}$$

Using the above inequality and (7) in inequality (6) we get that

$$\begin{aligned}
\mathbb{E}_t[f(\mu_{t+1})] \leq f(\mu_t) &+ \frac{2S^3M^2\eta^2}{P^3} + \frac{2(S-1)^2\eta^2}{P^3}\|\tilde{G}(W_{q_t^i}^i)\|^2 \\
&+ \frac{\eta(S-1)}{P^2}\sum_{i=1}^{P}\langle\nabla f(\mu_t), \tilde{G}(W_{q_t^i}^i)\rangle + \\
&\frac{\eta S^2L^2}{P^2}\sum_{i=1}^{P}\|\mu_t - W_t^i\|^2 - \frac{\eta(S-\frac{1}{4})}{P}\|\nabla f(\mu_t)\|^2.
\end{aligned}$$

Next we use $\mathbb{E}[f(\mu_{t+1})] = \mathbb{E}[\mathbb{E}_t[f(\mu_{t+1})]]$:

$$\begin{aligned}
\mathbb{E}[f(\mu_{t+1})] \leq \mathbb{E}[f(\mu_t)] &+ \frac{2S^2M^2\eta^2}{P^2} \\
&+ \frac{2(S-1)^2\eta^2}{P^3}\sum_{i=1}^{P}\mathbb{E}\|\tilde{G}(W_{q_t^i}^i)\|^2 \\
&+ \frac{\eta(S-1)}{P^2}\sum_{i=1}^{P}\mathbb{E}\langle\nabla f(\mu_t), \tilde{G}(W_{q_t^i}^i)\rangle \\
&+ \frac{\eta S^2L^2}{P^2}\sum_{i=1}^{P}\mathbb{E}\|\mu_t - W_t^i\|^2 \\
&- \frac{\eta(S-\frac{1}{4})}{P}\mathbb{E}\|\nabla f(\mu_t)\|^2. \qquad (8)
\end{aligned}$$

Recall that by the second moment bound $\mathbb{E}\|\tilde{G}(W_{q_t^i}^i)\|^2$ is at most M^2, and by Lemma 4.5 $\sum_{i=1}^{P}\mathbb{E}\|\mu_t - W_t^i\|^2 \leq 16P\eta^2M^2\tau^2$. Hence we can rewrite (8) as

$$\begin{aligned}
\mathbb{E}[f(\mu_{t+1})] \leq \mathbb{E}[f(\mu_t)] &+ \frac{2S^2M^2\eta^2}{P^2} + \frac{2(S-1)^2\eta^2M^2}{P^2} \\
&+ \frac{\eta(S-1)}{P^2}\sum_{i=1}^{P}\mathbb{E}\langle\nabla f(\mu_t), \tilde{G}(W_{q_t^i}^i)\rangle \\
&+ \frac{16\eta^3S^2L^2M^2\tau^2}{P} - \frac{\eta(S-\frac{1}{4})}{P}\mathbb{E}\|\nabla f(\mu_t)\|^2. \qquad (9)
\end{aligned}$$

We proceed by upper bounding $\sum_{i=1}^{P}\mathbb{E}\langle\nabla f(\mu_t), \tilde{G}(W_{q_t^i}^i)\rangle$:

$$\begin{aligned}
&\sum_{i=1}^{P}\mathbb{E}\langle\nabla f(\mu_t), \tilde{G}(W_{q_t^i}^i)\rangle = \sum_{i=1}^{P}\mathbb{E}\langle\nabla f(\mu_t), \nabla f(W_{q_t^i}^i)\rangle \\
&= \sum_{i=1}^{P}\mathbb{E}\langle\nabla f(\mu_t), -\nabla f(\mu_t) + \nabla f(W_{q_t^i}^i)\rangle + P\,\mathbb{E}\|\nabla f(\mu_t)\|^2 \\
&\overset{Young}{\leq} \frac{P}{4(S-1)}\,\mathbb{E}\|\nabla f(\mu_t)\|^2 + P\,\mathbb{E}\|\nabla f(\mu_t)\|^2 + \\
&\qquad (S-1)\sum_{i=1}^{P}\mathbb{E}\|\nabla f(\mu_t) - \nabla f(W_{q_t^i}^i)\|^2 \\
&\leq (S-1)L^2\sum_{i=1}^{P}\|W_{q_t^i}^i - \mu_t\|^2 + \frac{P(4S-3)}{4(S-1)}\,\mathbb{E}\|\nabla f(\mu_t)\|^2 \\
&\overset{\text{Lemma 4.4}}{\leq} 16P(S-1)L^2\eta^2M^2\tau^2 + \frac{P(4S-3)}{4(S-1)}\,\mathbb{E}\|\nabla f(\mu_t)\|^2.
\end{aligned}$$

Plugging the above inequality in (9) we get that

$$\begin{aligned}
\mathbb{E}[f(\mu_{t+1})] &\leq \mathbb{E}[f(\mu_t)] + \frac{2S^2M^2\eta^2}{P^2} + \frac{2(S-1)^2\eta^2M^2}{P^2} \\
&+ \frac{16\eta^3(S-1)^2L^2M^2\tau^2}{P} + \frac{\eta(S-1+\frac{1}{4})}{P}\,\mathbb{E}\|\nabla f(\mu_t)\|^2 \\
&+ \frac{16\eta^3S^2L^2M^2\tau^2}{P} - \frac{\eta(S-\frac{1}{4})}{P}\,\mathbb{E}\|\nabla f(\mu_t)\|^2 \\
&\leq \mathbb{E}[f(\mu_t)] + \frac{4S^2M^2\eta^2}{P^2} + \frac{32\eta^3S^2L^2M^2\tau^2}{P} \\
&\qquad - \frac{\eta}{2P}\,\mathbb{E}\|\nabla f(\mu_t)\|^2.
\end{aligned}$$

Next we sum the above inequality for $t = 0$ to $T-1$ and rearrange terms:

$$\begin{aligned}
\frac{\eta}{2P}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(\mu_t)\|^2 &\leq f(\mu_0) - \mathbb{E}[f(\mu_T)] + \frac{4S^2M^2\eta^2T}{P^2} \\
&\quad + \frac{32\eta^3S^2L^2M^2\tau^2T}{P}
\end{aligned}$$

Now we divide the above inequality by $\frac{\eta}{2PT}$ and use the fact that $\mathbb{E}[f(\mu_T)] \geq f(x^*)$:

$$\begin{aligned}
\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(\mu_t)\|^2 &\leq \frac{2P(f(\mu_0) - f(x^*))}{\eta T} + \frac{8S^2M^2\eta}{P} \\
&\quad + 64\eta^2S^2L^2M^2\tau^2.
\end{aligned}$$

By plugging $\eta = \frac{P}{\sqrt{T}}$ and noticing that for $T \geq P^4\tau^4$, $\eta \leq \frac{1}{P\tau^2}$ we get :

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(\mu_t)\|^2 \leq \frac{2(f(\mu_0) - f(x^*))}{\sqrt{T}} + \frac{8S^2M^2}{\sqrt{T}}\big(1 + 8L^2\big).$$

□

5 Experimental Evaluation

We conduct our experiments on the CSCS Piz Daint supercomputer. Each Cray XC50 compute node contains a 12-core Intel Xeon E5-2690 CPU with 64 GB RAM, and one NVIDIA Tesla P100 with 16 GB memory. The compute nodes are connected by Cray Aries interconnect in a Dragonfly

topology. The communication library is Cray MPICH 7.7.2. We use one MPI process per node and utilize the GPU for acceleration in all following experiments. We evaluate three different deep learning problems, including image classification (ResNet-50 [63] on ImageNet [64]), machine translation (Transformer [65] on WMT17), and deep reinforcement learning (PPO [14] for navigation in Habitat [66]). For throughput tests, we run the number of nodes until reaching a point where batch size is too large to converge [67].

5.1 Baselines

We compare our WAGMA-SGD with the state-of-the-art data-parallel SGD variants, including Allreduce-SGD [42], [43], local SGD [25], [51], gossip-based SGD variants (D-PSGD [16], AD-PSGD [20], and SGP [17]), and eager-SGD [13]. Unless mentioned specifically, the synchronization period of local SGD is set to one, namely calling a standard allreduce to average the models in each training iteration, which essentially is a synchronous SGD. For SGP, we evaluate its performance with different number of communication neighbors [17]. For more detailed discussion about the baselines, please refer to Section 2.

5.2 Image Classification with Simulated Workload Imbalance

Residual Networks (ResNet) [63] are pervasively used in computer vision tasks. To evaluate their performance, we train ResNet-50 v1 (total 25,559,081 trainable parameters) on ImageNet using TensorFlow [68] as the basic platform. Although the training workload is balanced due to the input size being fixed, performance variability is observed when training on multi-tenant cloud systems [13], [20], [69] due to resource sharing. To simulate the same degree of imbalance, we randomly select two processes at every training step to inject a certain amount of delay (320 ms). This simulated load imbalance also helps us to compare the robustness of the parallel SGD variants with the most popular deep learning benchmark. For WAGMA-SGD, we set the synchronization period $\tau = 10$, and the group size $S = \sqrt{P}$. This group setting is chosen to balance the trade-off between the faster mixing time, and improved convergence, provided by larger group size as discussed in the previous section, and the inherent increased communication overhead due to increasing S. In particular, this setting allows roughly $\sqrt{P}$ groups of $\sqrt{P}$ nodes each to perform reductions in parallel, on average. Both P and S are power-of-two in our experimental configuration.

Fig. 4 shows the training throughput as the number of GPU nodes increases from 4 to 256, and the top of the rectangle wrapping each cluster indicates the ideal throughput without communication overhead. Compared with local SGD, Allreduce-SGD (implemented in Deep500 [43]), D-PSGD, SGP (two communication neighbors), and eager-SGD when training on 64 GPU nodes, WAGMA-SGD achieves 1.25x, 1.26x, 1.23x, 1.25x, and 1.13x speedup, respectively. The speedup becomes larger as the number of GPU nodes increases to 256: WAGMA-SGD achieves up to 1.37x speedup. The only algorithm with higher throughput than WAGMA-SGD is AD-PSGD, in which the asynchronous

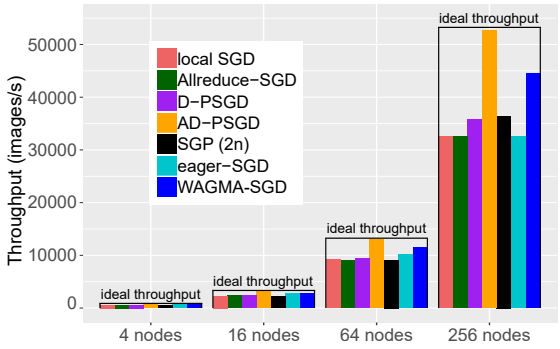

Fig. 4: Throughput comparison between different parallel SGD algorithms for ResNet-50 on ImageNet with simulated load imbalance. Local batch size is 128. (AD-PSGD achieves the highest throughput but with much less accuracy.)

communication is completely overlapped with the computation. These results show that WAGMA-SGD can better handle the unbalanced workload than the synchronous SGD algorithms (i.e., local SGD, Allreduce-SGD, D-PSGD, and SGP), as well as the bounded-staleness eager-SGD variant. In the latter case, while staleness is bounded, the algorithm still conducts a global collective communication for gradient averaging in each training iteration. In contrast, WAGMA-SGD keeps the collectives within each group, and thus has a better parallel scalability.

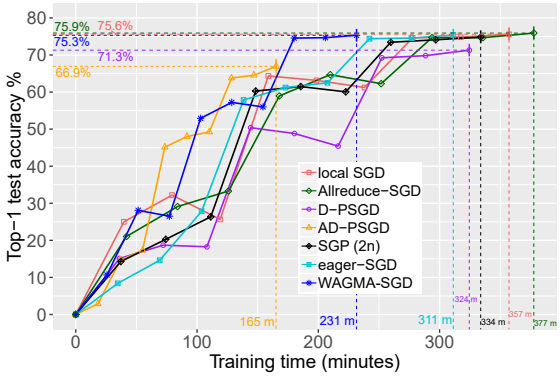

Fig. 5: Top-1 validation accuracy of ResNet-50 on ImageNet training for 90 epochs using 64 GPU nodes. Each point is at the boundary of every 10 epochs.

Fig. 5 presents the Top-1 validation accuracy when training for 90 epochs on 64 nodes, with a total batch size of 4,096 and simulated load imbalance. We can see that the accuracy of WAGMA-SGD (75.3%) is very close to the standard Allreduce-SGD (75.9%, in accordance with that in MLPerf [70]) and local SGD (75.6%), and the slightly decreased accuracy is caused by the staled model parameters in the unbalanced workload environment; WAGMA-SGD significantly reduces the training time (e.g., 1.45x, 1.54x, and 1.63x speedup over SGP, local SGD, and Allreduce-SGD, respectively). Gossip-based SGD algorithms, such as D-PSGD and the higher-throughput AD-PSGD, attain much lower accuracy than the other variants. This can be explained by the fact that the algorithms have not fully converged, requiring more steps to be taken to achieve comparable accuracy [26]. We further train 130 epochs for AD-PSGD

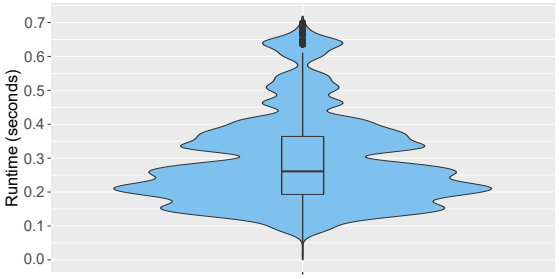

Fig. 6: Runtime distribution of different sentences on a P100 GPU for a Transformer network on WMT17.

and linearly scale the learning rate, and the accuracy is improved to 70.4%; however, it costs 238 minutes (longer than WAGAM-SGD which achieves a higher accuracy). Similarly, we improve the accuracy of D-PSGD to 74.4% by doubling the communication steps in each iteration, which costs 410 minutes for 90 epochs. For SGP, we set and tune the number of communication neighbors to achieve the highest generalization using a directed exponential graph [17], which causes it to achieve higher accuracy (74.8%) than D-PSGD and AD-PSGD, yet still lower than WAGMA-SGD. Note that the default setting for the number of communication neighbors in SGP is one, whereas we set it to two for better generalization performance. Overall, WAGMA-SGD achieves the highest accuracy-vs-time among all parallel SGD variants and quite robust to the unbalanced workloads.

We also train the model without simulated load imbalance for 90 epochs on 64 nodes, with a total batch size of 8,192. Local SGD, WAGMA-SGD, AD-PSGD, D-PSGD, and SGP achieve 1.52, 1.55, 1.61, 1.57, and 1.51 global_step/s for the training throughput, respectively; and 75.3%, 75.0%, 67.1%, 71.2%, and 74.5% for the Top-1 test accuracy, respectively. These results show that, when the workload is balanced, there is no big difference for the training throughput between the synchronous and asynchronous parallel SGD algorithms; WAGMA-SGD performs as well as the synchronous parallel SGD in terms of the model accuracy.

By setting the group size $S = \sqrt{P} = 8$, WAGMA-SGD has a faster model update propagation speed (globally propagate only using $\log_S P = 2$ iterations) than the gossip-based algorithms (globally propagate using at least $\log_2 P = 6$ iterations), which makes WAGMA-SGD achieve higher accuracy. This is consistent with our analysis in Section 4.1.

To analyze the convergence properties of WAGMA-SGD further, we conduct additional experiments. ❶ In the first experiment, we remove the wait-avoiding group collectives in WAGMA-SGD and only keep standard allreduce operations on the synchronization points, which is essentially equivalent to local SGD with a synchronization period $\tau = 10$. This causes the top-1 validation accuracy to sharply drop to 68.5%. ❷ In a second experiment, we execute group model averaging without using the dynamic grouping strategy (i.e., the groups are fixed). In this case, the top-1 validation accuracy drops to 72.2%. ❸ We also experiment with increasing the group size to 64 (i.e., a global collective). While accuracy does not increase, the throughput drops by factor of 1.24x. ❹ Lastly, we decrease the group size to 4 and observe that the top-1 validation accuracy drops to 72.8%.

The results from experiments ❶ and ❷ indicate that the combination of group allreduce operations and the dynamic grouping strategy is essential to achieve good generalization performance. The results from experiments ❸ and ❹ demonstrate that $S = \sqrt{P}$ empirically exhibits the best performance among different group size settings.

5.3 Machine Translation

Transformers are sequence-to-sequence transducers that can be used to translate a sequence of words from one language to another. We use the standard-sized Transformer network [65], which has 61,362,176 trainable parameters, to train English to German translation WMT17 dataset using TensorFlow as the basic platform. While training the model, the computation overhead changes with the length of the input and output sentences. The samples in the training dataset typically consist of sentences in various lengths and thus the training workload is unbalanced. As shown in Fig. 6, even when using a bucketing strategy to group sentences with similar lengths, there is a high variance in workload size between samples. Specifically, in our experiment each local batch contains equal number of sentences sampled from a randomly selected bucket, where the maximum local batch size is set to 8,192 tokens. For WAGMA-SGD, we set the synchronization period $\tau = 8$ and the group size $S = \sqrt{P}$.

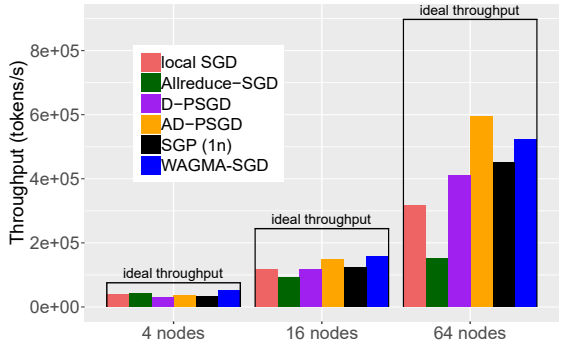

Fig. 7: Throughput comparison between different parallel SGD algorithms for Transformer on WMT17.

Fig. 7 presents the training throughput as the number of GPU nodes increases from 4 to 64, where the top of the rectangle indicates the ideal throughput without communication overhead. On 16 GPU nodes, WAGMA-SGD achieves the highest throughput, compared with local SGD, Allreduce-SGD (implemented in Horovod [42]), D-PSGD, AD-PSGD, and SGP (one communication neighbor). When the number of GPU nodes increases to 64, as with image classification WAGMA-SGD exhibits a lower throughput than AD-PSGD but higher than the other variants. Observe that on 64 nodes, all algorithms perform far worse than the ideal throughput. We believe that this effect stems from the balance of the number of parameters (occupying 245 MB alone) vs. the operational intensity to compute back-propagation. Since transformer networks mostly consist of tensor contractions implemented as batched matrix products, which utilize GPUs well, communication overhead

dominates and not even AD-PSGD manages to overlap communication with computation.

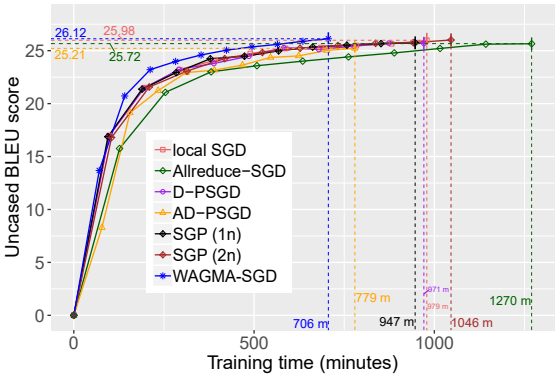

Fig. 8: Uncased BLEU score for Transformer on WMT17 training for 10 epochs using 16 GPU nodes. Each point is at the boundary of one epoch.

As for accuracy, Fig. 8 presents the BiLingual Evaluation Understudy (BLEU) score (higher is better) on the test dataset after training for 10 epochs on 16 nodes. D-PSGD and AD-PSGD, have lower score (25.69 and 25.21, respectively) than the other SGD variants, likely because of the slower model update propagation. SGP (1n, i.e., one communication neighbor) achieves higher score (i.e., 25.75) than D-PSGD and AD-PSGD. After increasing the number of communication neighbors to two in SGP (2n), the score increases to 26.01 (equivalent to local SGD, 25.98). However, this accuracy increase comes at the cost of reduced training speed compared with SGP (1n). Among all SGD variants, WAGMA-SGD not only achieves the highest score (i.e., 26.12, higher than 25.00 which is claimed in MLPerf [70]) but also uses the shortest training time (e.g., 1.48x, 1.10x and 1.39x speedup over SGP (2n), AD-PSGD, and local SGD, respectively).

We conduct additional experiments for WAGMA-SGD, similarly to Section 5.2: (1) Without using the dynamic grouping strategy (i.e., fixed groups), the score drops to 24.79; (2) By increasing the group size to 16 (i.e., global collective), accuracy does not improve and training throughput drops by a factor of 1.28x; and (3) By decreasing the group size to 2, the score drops to 24.53. These results reaffirm the conclusions from image classification.

5.4 Deep Reinforcement Learning

Due to the inherent characteristics of the problem, reinforcement learning poses a more challenging training process over supervised and semi-supervised learning. This also applies to the heterogeneity in workloads during training — since the problems in question involve interacting with an environment in episodes (where failure terminates an episode early), a variety of episode lengths may occur within a single minibatch, in a way that cannot be anticipated or categorized in buckets.

We use the popular Proximal Policy Optimization (PPO) policy gradient optimizer [14] to train a model for robot navigation on a meta-dataset called Habitat [66], which is composed of multiple heterogeneous environments. We first confirm previous claims [14] and our own in Fig. 9,

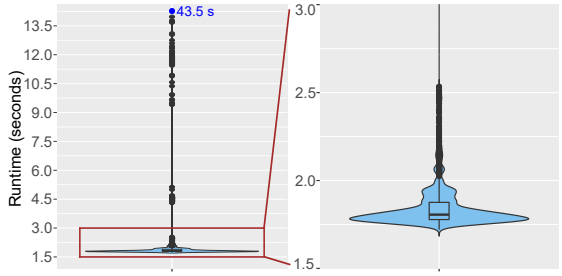

Fig. 9: Runtime distribution of experience collecting on a P100 GPU in heterogeneous environments.

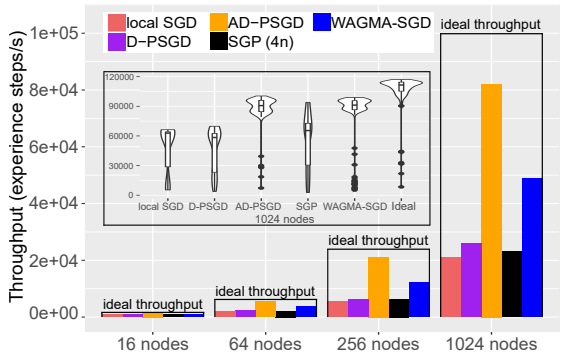

Fig. 10: Throughput comparison between different parallel SGD algorithms for DDPPO on Habitat. (AD-PSGD achieves the highest throughput but with a much lower score.)

where we collate the runtime distribution of 5,000 training iterations. The runtime is very widely distributed: from 1.7 seconds to 43.5, with a median below 2 seconds, which makes it an excellent use case for the load-rebalancing properties of WAGMA-SGD.

To evaluate the performance, we train a standard ResNet-LSTM model for navigation using PyTorch [71] as the basic platform. In particular, the network structure is composed of a ResNet-18 visual encoder, connected to a stack of two Long Short-Term Memory (LSTM) [72] recurrent units functioning as the policy, containing 8,476,421 trainable parameters. The measured heterogeneous environments in Habitat, Gibson [73] and Matterport3D [74], consist of interactive RGB-D datasets. We set the experience steps to 128 and use the two vectorized (namely, optimized) environments, which means each GPU node needs to collect 256 experience steps for each training iteration. We set the WAGMA-SGD synchronization period to $\tau = 8$.

Fig. 10 presents the training throughput as the number of GPU nodes increases from 16 to 1,024, where the top of the rectangle indicates the ideal throughput without communication overhead. Compared with local SGD, D-PSGD, and SGP (four communication neighbors) on 1,024 GPU nodes, WAGMA-SGD achieves 2.33x, 1.88x, and 2.10x speedup, respectively. The violin plot shows the throughput distribution. WAGMA-SGD only has lower throughput than AD-PSGD, since AD-PSGD is fully asynchronous. These results show that WAGMA-SGD excels in handling highly unbalanced workloads, achieving good scaling efficiency.

Complementary to training throughput, we study the Success weighted by Path Length (SPL) score (higher is

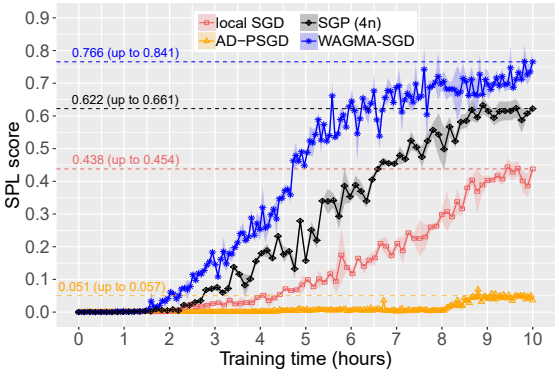

Fig. 11: SPL score comparison for DDPPO on Habitat. Training on 64 GPU nodes for 10 hours. Each point is at the boundary of every 50 updates.

better) after training the model for 10 hours on 64 GPUs. All models are tested four separate times to account for variability, and the average scores together with the standard deviation (shaded regions) over training time are plotted in Fig. 11. As the figure shows, after training for 10 hours, AD-PSGD converges to a lower SPL score (on average 0.051), which does not rise over the course of 13,000 iterations. This is probably caused by the unbounded staleness and the overuse of the stale models in AD-PSGD. On the other hand, WAGMA-SGD achieves the highest score over time. SGP scores higher than local SGD, but not as well as WAGMA-SGD, whose quorum size is larger.

Beyond our experiments, the current state-of-the-art (SOTA) SPL score is 0.922 [14], which is achieved after training on 2.5 billion experience steps. WAGMA-SGD consumes total 2.1 million experience steps after training for 10 hours using 64 GPUs, and achieves on average 83.1% (up to 91.2%) of the SOTA score. This indicates that WAGMA-SGD achieves generalization performance close to the SOTA using three orders of magnitude fewer iterations.

6 Collectives in Context

Collective operations have a core role in running applications efficiently at scale. As such, their optimization has led to several implementation and algorithmic variants [55], [58], [60], [75]. *Blocking* collectives [12] constitute the basic class of operations. In this case, the collective call is allowed to return only when the calling process has completed the actions needed to its participation in the operation. A first optimization to blocking collectives is to make them *non-blocking* [58], enabling processes to return immediately and overlap other activities with the ongoing collective.

Some collectives require all processes to invoke the operation in order to complete, e.g., a reduction cannot be computed before knowing all the values to reduce. Hence, their completion time can be influenced by any skewing (imbalance) among the processes. *Solo* collectives [60] remove this synchronization overhead by making the collectives externally-triggerable: once a process joins the collective, it sends an activation message to the other processes, making them to start the collective independently from their state. An issue of *solo* collectives is that they make triggering the collective possible, even if there is only one process joining it. *Majority* collectives [13] extend the *solo* idea by requiring that at least $P/2$ processes join the collective before triggering it. While these collectives are not guaranteed to be equivalent to their blocking or non-blocking counterparts, they are suited for machine learning tasks, due to the robustness of stochastic optimization to staleness.

Both *solo* and *majority* collectives aim to minimize the synchronization overhead. However, once activated, the collective is fully performed, making the application pay the full operation cost plus the activation overhead. *Wait-avoiding group* collectives (this work) utilize the approach of solo collectives to achieve asynchrony, and reduce the overall operation cost further by dynamically selecting subgroups of processes, each of which executing the collective independently from the others. By applying the idea of wait-avoiding group allreduce to the decentralized parallel SGD based on model averaging, we propose WAGMA-SGD. Our work is orthogonal to communication compression or quantization methods, which reduce the communication volume by using less number of bits to represent the data (module update) to be transferred.

7 Conclusion

We show, both theoretically and in practice, that stochastic optimization via group model averaging — asynchronously averaging the learned weights across subgroups of nodes — functions well in large clusters. We prove that the algorithm converges under the standard conditions of SGD, and through a careful implementation of wait-avoiding collectives, we use the topology of the network to attain the best scaling results without losing accuracy. With the same number of steps, WAGMA-SGD achieves equivalent (or even higher) generalization scores as the standard synchronous SGD, while significantly reducing the training time (e.g., up to 1.48× speedup on Transformer) over the previous state-of-the-art, gossip-based SGD. Similar results are observed on the models from various sub-fields, where WAGMA-SGD consistently achieves the fastest time-to-solution. These results empirically prove that this approach successfully tackles the unbalanced training workloads in large scales, and brings asynchronous decentralized SGD to the regime of supercomputers.

Acknowledgment

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 programme (grant agreement DAPP, No. 678880; EPiGRAM-HS, No. 801039; and ERC Starting Grant ScaleML, No. 805223). T.B.N is supported by the Swiss National Science Foundation (Ambizione Project No. 185778). N.D. is supported by the ETH Postdoctoral Fellowship. We also thank the Swiss National Supercomputing Center for providing the computing resources and technical support.

References

[1] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[2] A. W. Senior, R. Evans, J. Jumper, and et al., "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[3] D. Amodei and D. Hernandez, "AI and compute," May 2018, https://openai.com/blog/ai-and-compute/.

[4] T. Ben-Nun and T. Hoefler, "Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis," *ACM Comput. Surv.*, vol. 52, no. 4, Aug. 2019.

[5] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[7] O. Vinyals, I. Babuschkin, W. M. Czarnecki, and et al., "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[8] S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team, "An empirical model of large-batch training," *arXiv preprint arXiv:1812.06162*, 2018.

[9] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[10] Renggli, Cedric and Ashkboos, Saleh and Aghagolzadeh, Mehdi and Alistarh, Dan and Hoefler, Torsten, "SparCML: High-Performance Sparse Communication for Machine Learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, 2019.

[11] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *The Thirty-Third AAAI Conference on Artificial Intelligence, (AAAI 2019)*. AAAI Press, 2019, pp. 4780–4789.

[12] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard Version 3.1," 2015.

[13] S. Li, T. Ben-Nun, S. D. Girolamo, D. Alistarh, and T. Hoefler, "Taming unbalanced training workloads in deep learning with partial collective operations," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2020.

[14] E. Wijmans, A. Kadian, A. S. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames," in *ICLR*, 2020.

[15] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, "Communication-efficient distributed deep learning: A comprehensive survey," *arXiv preprint arXiv:2003.06307*, 2020.

[16] 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," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, 2017, pp. 5336–5346.

[17] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *Proceedings of the Thirty-sixth International Conference on Machine Learning (ICML)*, 2019, pp. 344–353.

[18] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems 24*, 2011, pp. 693–701.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[20] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 3043–3052.

[21] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, "GossipGraD: Scalable deep learning using gossip communication based asynchronous gradient descent," *arXiv preprint arXiv:1803.05880*, 2018.

[22] Q. Ho, J. Cipar, H. Cui, J. K. Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'13, 2013, pp. 1223–1231.

[23] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging SGD," in *Advances in neural information processing systems (NeurIPS)*, 2015, pp. 685–693.

[24] K. Chen and Q. Huo, "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering," in *2016 ieee international conference on acoustics, speech and signal processing (icassp)*. IEEE, 2016, pp. 5880–5884.

[25] S. U. Stich, "Local SGD converges fast and communicates little," in *Proceedings of the Seventh International Conference on Learning Representations (ICLR)*, 2019.

[26] G. Nadiradze, A. Sabour, D. Alistarh, A. Sharma, I. Markov, and V. Aksenov, "SwarmSGD: Scalable decentralized SGD with local updates," *arXiv preprint arXiv:1910.12308*, 2019.

[27] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, 2018.

[28] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in neural information processing systems (NeurIPS)*, 2012, pp. 1223–1231.

[29] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'14, 2014, pp. 583–598.

[30] A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko, "Priority-based parameter propagation for distributed DNN training," in *Proceedings of the 2nd SysML Conference*, 2019.

[31] D. Grishchenko, F. Iutzeler, J. Malick, and M.-R. Amini, "Asynchronous distributed learning with sparse communications and identification," *arXiv preprint arXiv:1812.03871*, 2018.

[32] Z. Tang, S. Shi, and X. Chu, "Communication-efficient decentralized learning with sparsification and adaptive peer selection," *arXiv preprint arXiv:2002.09692*, 2020.

[33] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016.

[34] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-SGD for distributed deep learning," in *25th International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

[35] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching lan speeds," in *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'17, 2017, pp. 629–647.

[36] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd," in *Proceedings of the International Conference on Artifical Intelligence and Statistics (AISTATS)*, ser. Proceedings of Machine Learning Research, vol. 84, 2018, pp. 803–812.

[37] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[38] T. Kurth, J. Zhang, N. Satish, E. Racah, I. Mitliagkas, M. M. A. Patwary, T. Malas, N. Sundaram, W. Bhimji, M. Smorkalov *et al.*, "Deep learning at 15PF: supervised and semi-supervised classification for scientific data," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2017.

[39] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, ser. Proceedings of Machine Learning Research, vol. 54, 2017, pp. 1273–1282.

[40] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NeurIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[41] A. Gibiansky, "Bringing hpc techniques to deep learning," *URL http://research. baidu. com/bringing-hpc-techniques-deep-learning*, 2017.

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

[43] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler, "A modular benchmarking infrastructure for high-

performance and reproducible deep learning," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2019, pp. 66–77.

[44] R. Mcdonald, M. Mohri, N. Silberman, D. Walker, and G. S. Mann, "Efficient large-scale distributed training of conditional maximum entropy models," in *Advances in neural information processing systems (NeurIPS)*, vol. 22, 2009, pp. 1231–1239.

[45] D. Cheng, S. Li, and Y. Zhang, "WP-SGD: Weighted parallel SGD for distributed unbalanced-workload training system," *Journal of Parallel and Distributed Computing*, vol. 145, pp. 202–216, 2020.

[46] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems (NeurIPS)*, 2010.

[47] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica *et al.*, "Exascale deep learning for climate analytics," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 2018, pp. 649–660.

[48] A. Koloskova, S. U. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," *arXiv preprint arXiv:1902.00340*, 2019.

[49] P. H. Jin, Q. Yuan, F. N. Iandola, and K. Keutzer, "How to scale distributed deep learning?" in *Workshop on Machine Learning Systems at NeurIPS 2016*, 2016.

[50] Z. Jiang, A. Balu, C. Hegde, and S. Sarkar, "Collaborative deep learning in fixed topology networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5904–5914.

[51] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi, "Don't use large mini-batches, use local sgd," *arXiv preprint arXiv:1808.07217*, 2018.

[52] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD," in *Proceedings of the Second SysML Conference*, 2019.

[53] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2011, pp. 104–113.

[54] S. Li, T. Hoefler, and M. Snir, "NUMA-aware shared-memory collective communication for MPI," in *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, 2013, pp. 85–96.

[55] R. Rabenseifner, "Optimization of collective reduction operations," in *International Conference on Computational Science*. Springer, 2004, pp. 1–9.

[56] S. Li, T. Hoefler, C. Hu, and M. Snir, "Improved MPI collectives for MPI processes in shared address spaces," *Cluster computing*, vol. 17, no. 4, pp. 1139–1155, 2014.

[57] D. De Sensi, S. Di Girolamo, and T. Hoefler, "Mitigating network noise on dragonfly networks through application-aware routing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–32.

[58] T. Hoefler, A. Lumsdaine, and W. Rehm, "Implementation and performance analysis of non-blocking collective operations for mpi," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. ACM, 2007, pp. 1–10.

[59] P. Patarasuk and X. Yuan, "Bandwidth Optimal All-reduce Algorithms for Clusters of Workstations," *J. Parallel Distrib. Comput.*, vol. 69, no. 2, pp. 117–124, Feb. 2009.

[60] S. Di Girolamo, P. Jolivet, K. D. Underwood, and T. Hoefler, "Exploiting offload enabled network interfaces," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 2015, pp. 26–33.

[61] T. Hoefler, S. D. Girolamo, K. Taranov, R. E. Grant, and R. Brightwell, "sPIN: High-performance streaming Processing in the Network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC17)*, Nov. 2017.

[62] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[63] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[64] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2009, pp. 248–255.

[65] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[66] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik *et al.*, "Habitat: A platform for embodied ai research," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9339–9347.

[67] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *Journal of Machine Learning Research*, vol. 20, pp. 1–49, 2019.

[68] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[69] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 460–471, 2010.

[70] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang *et al.*, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.

[71] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[72] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[73] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, "Gibson Env: real-world perception for embodied agents," in *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018.

[74] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3D: Learning from RGB-D data in indoor environments," *International Conference on 3D Vision (3DV)*, pp. 667–676, 2017.

[75] S. Li, Y. Zhang, and T. Hoefler, "Cache-oblivious MPI all-to-all communications based on Morton order," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 542–555, 2017.

Shigang Li (Member, IEEE) is a postdoctoral researcher in Department of Computer Science, ETH Zürich from August 2018 to now. He received the bachelor's degree in computer science and technology and the PhD degree in computer architecture from the University of Science and Technology Beijing, China, in 2009 and 2014, respectively. He was a joint PhD student at University of Illinois at Urbana-Champaign from September 2011 to September 2013 funded by CSC. He was an Assistant Professor (from June 2014 to August 2018) with the State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences. He is a member of ACM. His research interests focus on parallel and distributed computing, and parallel and distributed deep learning.

Tal Ben-Nun is a senior scientist in Department of Computer Science, ETH Zürich. He received his PhD degree from the Hebrew University of Jerusalem in 2016, where he worked on programming models for distributed memory environments. His research interests include large-scale machine learning for scientific computing, machine comprehension of code via deep learning, and designing data-centric programming models for heterogeneous architectures.

Giorgi Nadiradze is a PhD student at IST Austria. His research centers on fundamental problems in theoretical computer science, with an emphasis on dynamic load balancing processes and their applications. In particular, his work considers practical applications to concurrent data structures and distributed machine learning.

Salvatore Di Girolamo is a PhD student at ETH Zürich. He works on high-performance networking with special focus on communication and computation network offloading.

Nikoli Dryden is an ETH Postdoctoral Fellow in the Scalable Parallel Computing Lab at ETH Zürich. He received his PhD from the University of Illinois at Urbana-Champaign in 2019, where he worked on large-scale training of deep neural networks. His research interests include distributed machine learning, machine learning for computational science, parallel algorithms and runtimes, and communication and performance optimization.

Dan Alistarh is an Assistant Professor at IST Austria, where he leads the Distributed Algorithms and Systems Group. His research focuses on concurrent data structures and distributed algorithms, and spans from algorithms and lower bounds, to practical implementations. Before IST, he was a researcher at ETH Zurich and Microsoft Research, Cambridge, UK. Prior to that, he was a Postdoctoral Associate at MIT CSAIL.

Torsten Hoefler is a Full Professor of Computer Science at ETH Zürich, Switzerland. He directs the Scalable Parallel Computing Lab. He is a key member of the MPI Forum where he chairs the Collective Operations and Topologies working group. His research interests revolve around the central topic of Performance-Centric Software Development and include scalable networks, parallel programming techniques, and performance modeling.