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

Adel Nabli [1 2] Eugene Belilovsky [1] Edouard Oyallon [2]

[1]Mila, Concordia University

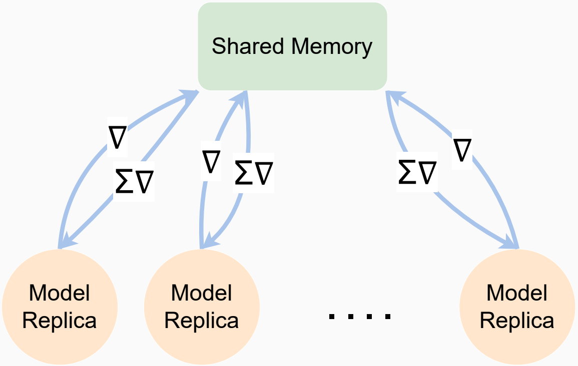
[2]Sorbonne University, ISIR, CNRS

Distributed Training of DNN

Data parallel: // opt. of model's parameters $x \in \mathbb{R}^d$ across n workers.
$\hookrightarrow t_{\text{train}} \propto \frac{1}{n}$: large minibatch training [Goyal et al., 2017].

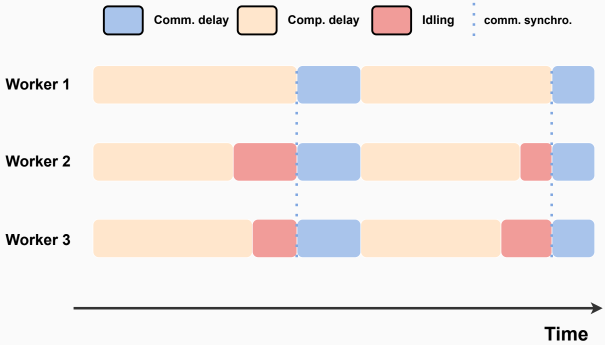
$$\inf_{x_i = x_1 \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(x_i)\,.$$

Standard methods (*e.g.*, Pytorch's DDP): centralized, synchronous.



Distributed Training of DNN

Typical example of timeline for centralized, synchronous methods:



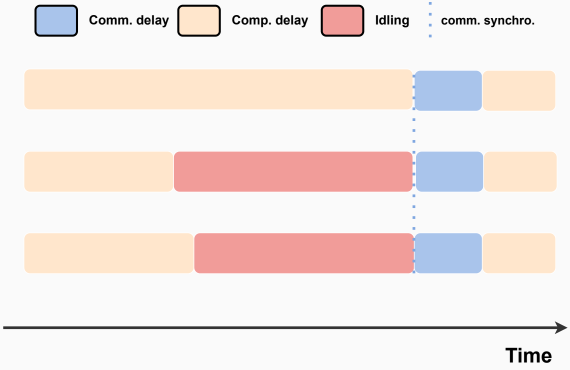
Problems :

- Averaging new ∇: $\Delta_t^{\text{TOT}} = \Delta_t^{\text{grad}} + \Delta_t^{\text{comm.}}$.
- Synchronous: Wait for straggler.
- Centralized: Communication bottleneck when scaling up n.

Straggler Problem

Problem :

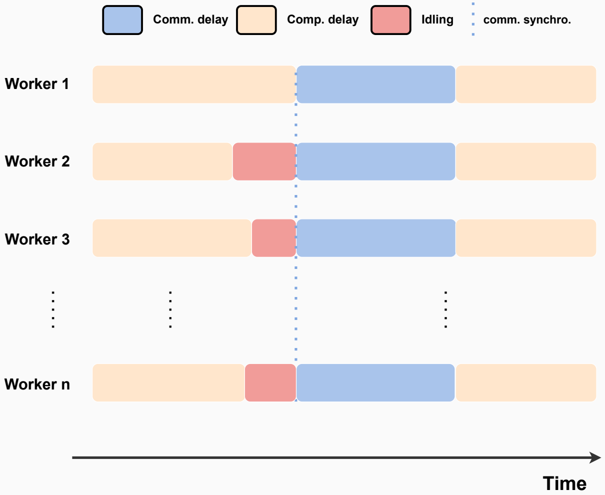
- Synchronous: Wait for straggler.



Communication bottleneck

Problem :

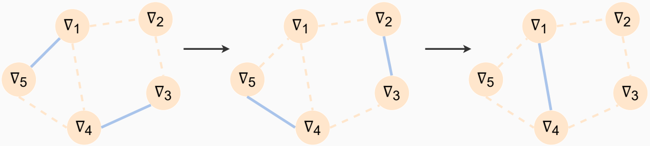
- Centralized: Communication bottleneck when scaling up n.

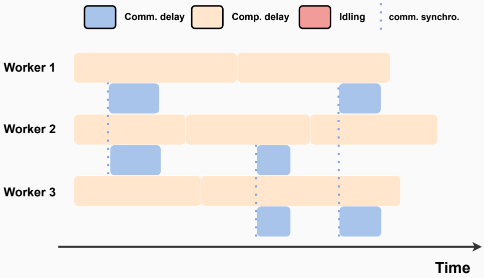


Decentralized Asynchronous p2p

- Decentralized: alleviate communication bottleneck.
- Asynchronous: reduce impact of slower workers.
- Params. averaging: Computations and communications in //.

↪ SOTA decentralized async. DNN training: AD-PSGD [Lian et al., 2018].



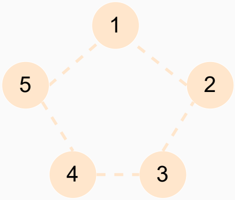


Decentralized Asynchronous p2p

Table 1: # of communications per "step"/time unit on several graphs.

Method	Star	Ring	Complete
Synchronous	n^2	n^3	n^2
Accelerated Synchronous	$n^{3/2}$	n^2	n^2
A^2CiD^2	n	n^2	n

Problem: Graph connectivity impacts the communication complexity.



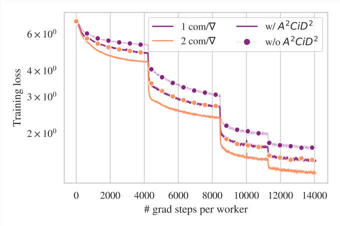


Figure 1: ImageNet cycle $n = 64$

Question: Can we reduce the impact of the graph's connectivity ?

A^2CiD^2 momentum

Using continuized framework [Even et al., 2021], model discrete updates at random times with Poisson Point Processes:

$$
\begin{aligned}
dx_t^i &= \eta(\tilde{x}_t^i - x_t^i)dt - \gamma \int_{\Xi} \nabla F_i(x_t^i, \xi_i)\, dN_t^i(\xi_i) - \alpha \sum_{j,(i,j)\in\mathcal{E}} (x_t^i - x_t^j) dM_t^{ij}\,, \\
d\tilde{x}_t^i &= \eta(x_t^i - \tilde{x}_t^i)dt - \gamma \int_{\Xi} \nabla F_i(x_t^i, \xi_i)\, dN_t^i(\xi_i) - \tilde{\alpha} \sum_{j,(i,j)\in\mathcal{E}} (x_t^i - x_t^j) dM_t^{ij}\,.
\end{aligned}
$$

Algorithm 1: This algorithm block describes our implementation of our Asynchronous algorithm with **A^2CiD^2** on each local machine. p2p comm. and ∇ comp. are run independently in parallel.

Input: On each machine $i \in \{1, ..., n\}$, gradient oracle ∇f_i, parameters $\eta, \alpha, \tilde{\alpha}, \gamma, T$.

Initialize on each machine $i \in \{1, ..., n\}$:

- **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* $i \in \{1, ..., n\}$, **while** $t < T$, **continuously do:**

- **In one thread** *on worker* i **continuously do:**
  - $t \leftarrow clock()$;
  - Sample a batch of data via $\xi_i \sim \Xi$;
  - $g_i \leftarrow \nabla F_i(x_i, \xi_i)$; // Compute gradients
  - $\boxed{\begin{pmatrix} x^i \\ \tilde{x}^i \end{pmatrix} \leftarrow \exp\left((t - t^i)\begin{pmatrix} -\eta & \eta \\ \eta & -\eta \end{pmatrix}\right)\begin{pmatrix} x^i \\ \tilde{x}^i \end{pmatrix};}$
  - $x^i \leftarrow x^i - \gamma g_i$; // Apply A^2CiD^2
  - $\tilde{x}^i \leftarrow \tilde{x}^i - \gamma g_i$; // 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
  - $\boxed{\begin{pmatrix} x^i \\ \tilde{x}^i \end{pmatrix} \leftarrow \exp\left((t - t^i)\begin{pmatrix} -\eta & \eta \\ \eta & -\eta \end{pmatrix}\right)\begin{pmatrix} x^i \\ \tilde{x}^i \end{pmatrix};}$ // Apply A^2CiD^2
  - $x^i \leftarrow x^i - \alpha m_{ij}$; // p2p averaging
  - $\tilde{x}^i \leftarrow \tilde{x}^i - \tilde{\alpha} m_{ij}$;
  - $t^i \leftarrow t$;

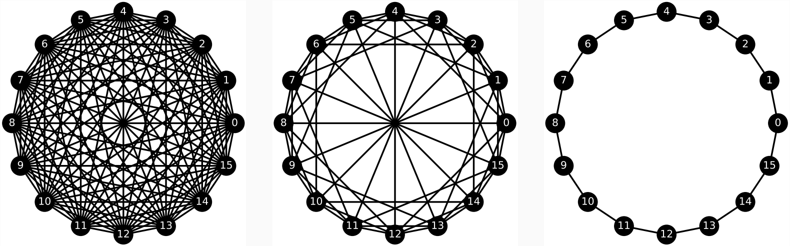
return $(x_T^i)_{1 \le i \le n}$.

- // ∇ and p2p comm.
- **A^2CiD^2** momentum mixes local var. $x_i, \tilde{x}_i$ at each update.

$\hookrightarrow$ Provably improves communication complexity compared to previous decentralized methods.

Experimental setting

- 3 graph's topology:



- ResNet18 on CIFAR-10 and ResNet50 on ImageNet.
- 1 worker / NVIDIA A100 GPU, cluster with 8 GPUs per node using an Omni-PAth interconnection network at 100 Gb/s.
- up to $n = 64$ workers, "effective" batch-size of $n \times 128$.

Experimental results

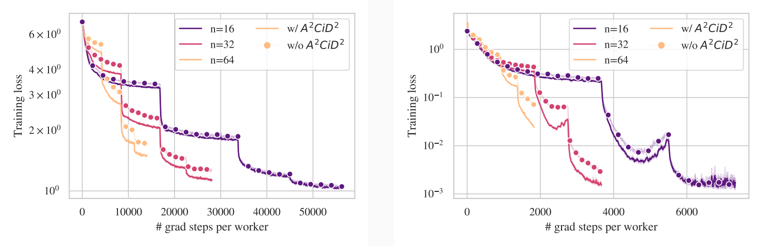


Figure 2: Training loss on cycle $n \leq 64$, ImageNet (left) and CIFAR-10 (right).

Table 2: ImageNet, $n = 64$.

Method	t (min)	# ∇ slowest worker	# ∇ fastest worker
AR-SGD	$1.7\,10^2$	14k	14k
Ours	$\mathbf{1.5\,10^2}$	**13k**	14k

Table 3: t_{train} on CIFAR10 ($\pm$ 6s).

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

Conclusion

We presented a local momentum $\mathbf{A^2CiD^2}$:

- Provably improves communication complexity of Decentralized Asynchronous DNN training algorithms (*e.g.*, AD-PSGD).
- Experimentally demonstrates that $\mathbf{A^2CiD^2}$ works efficiently for training DNN in decentralized settings.
- Release our code (github.com/AdelNabli/ACiD), circumvent locks put on previous implementations (*e.g.*, AD-PSGD).

Thank you,

Come see our poster !

References i

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

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

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