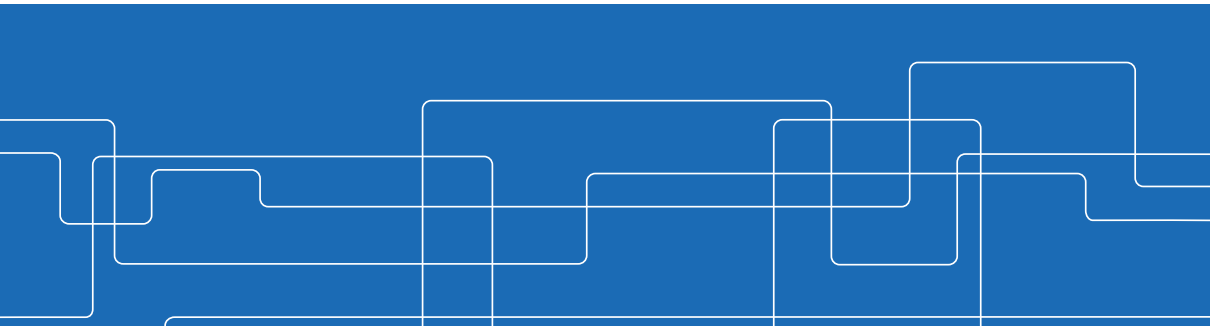


From core to clusters: tailoring optimization algorithms to modern compute architectures

Z. Wang, J. Lindbäck, V. Mai, J. Zhang, X. Wu, and Mikael Johansson



A hardware–algorithm disconnect

Modern ML requires solving larger optimization problems, faster:

- training models with billions of parameters using trillions of data tokens
- solving multi-million variable LPs in every training iteration for generative models

A hardware–algorithm disconnect

Modern ML requires solving larger optimization problems, faster:

- training models with billions of parameters using trillions of data tokens
- solving multi-million variable LPs in every training iteration for generative models

Compute scales and diversifies, but algorithms are often designed with idealized assumptions

- sequential execution, uniform memory, perfect communication.

A hardware–algorithm disconnect

Modern ML requires solving larger optimization problems, faster:

- training models with billions of parameters using trillions of data tokens
- solving multi-million variable LPs in every training iteration for generative models

Compute scales and diversifies, but algorithms are often designed with idealized assumptions

- sequential execution, uniform memory, perfect communication.

Hardware–algorithm disconnect creates practical paradoxes

- GPU-friendly $O(n^3)$ algorithms may outperform $O(n^2)$ methods
- “optimal” distributed algorithms spend 80% of time waiting while cores idle

Hardware-aware optimization is more than implementation

Hardware-aware optimization is more than just implementation tricks:

- bottlenecks often block the asymptotics from kicking in
- algorithms aligned with the compute fabric are more efficient in practice

Hardware-aware optimization is more than implementation

Hardware-aware optimization is more than just implementation tricks:

- bottlenecks often block the asymptotics from kicking in
- algorithms aligned with the compute fabric are more efficient in practice

This requires restructuring the algorithm design itself:

- reorganizing computation patterns for parallel execution (SIMD/SIMT)
- redesigning memory access for coalescence and locality
- overlapping communication with computation

Large gains from rethinking algorithms to respect hardware constraints!

This talk: two hardware-aware optimization designs

GPU acceleration: optimal transport solver

- Douglas-Rachford method restructured for GPU warp execution patterns
- memory access reorganized for coalescence and bandwidth efficiency
- **result:** 10–100× speedup over standard implementations

This talk: two hardware-aware optimization designs

GPU acceleration: optimal transport solver

- Douglas-Rachford method restructured for GPU warp execution patterns
- memory access reorganized for coalescence and bandwidth efficiency
- **result:** 10–100× speedup over standard implementations

Distributed training: a system-level design

- overlap communication and computation, use bottleneck-aware topology designs
- a decentralized Adam optimizer redesigned for small local batch effects and asynchrony
- **result:** 30–60% faster wall-clock time + better generalization

Results from concrete algorithm restructuring – not just efficient implementation!

Contents

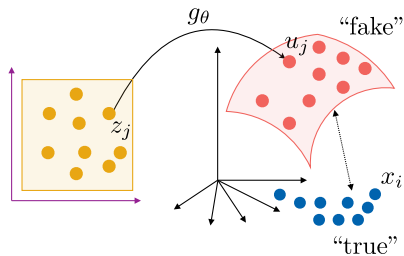
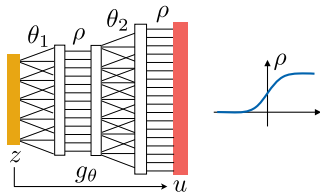
- Motivation
- A GPU-based optimal transport solver
- An efficient decentralized DNN training system
- Conclusions and outlook

Optimal transport in machine learning

Optimal transport (OT): cheapest way to transform one probability distribution into another

- defines a geometrically meaningful distance
- has tons of ML applications: domain adaptation, generative models, learning to rank, ...

$$z_j = \text{rand} \xrightarrow[\text{deep-net}]{g_\theta} u_j = g_\theta(z_j)$$



OT cost measures distance between distribution of true and generated images.

Discrete optimal transport as a linear program

Given

- two discrete probability distributions $p \in \mathbf{R}^m$ and $q \in \mathbf{R}^n$,
- a matrix $C \in \mathbf{R}^{m \times n}$ where C_{ij} is the cost for moving mass from bin $p_i \mapsto$ bin q_j ,

find transportation plan X that solves

$$\begin{aligned} & \text{minimize} && \langle C, X \rangle \\ & \text{subject to} && X\mathbf{1} = p, \quad X^T\mathbf{1} = q \\ & && X \geq 0 \end{aligned}$$

For appropriate C , optimal value is Wasserstein distance: $W(p, q) = \langle C, X^* \rangle$

Discrete optimal transport as a linear program

Given

- two discrete probability distributions $p \in \mathbf{R}^m$ and $q \in \mathbf{R}^n$,
- a matrix $C \in \mathbf{R}^{m \times n}$ where C_{ij} is the cost for moving mass from bin $p_i \mapsto$ bin q_j ,

find transportation plan X that solves

$$\begin{aligned} & \text{minimize} && \langle C, X \rangle \\ & \text{subject to} && X\mathbf{1} = p, \quad X^T\mathbf{1} = q \\ & && X \geq 0 \end{aligned}$$

For appropriate C , optimal value is Wasserstein distance: $W(p, q) = \langle C, X^* \rangle$

A **huge** linear program: quickly many millions of variables.

- unless special structure in C , complexity is $O((m+n)^3)$.

Entropic regularization and the Sinkhorn algorithm

Replace non-negativity with entropic penalty term

$$\begin{aligned} W_\varepsilon(p, q) &= \min_X \quad \langle C, X \rangle - \varepsilon H(X) \\ &\text{subject to} \quad X\mathbf{1} = p, \quad X^T\mathbf{1} = q \end{aligned}$$

where $H = -\sum_i \sum_j X_{ij} \log X_{ij}$ is the entropy of X .

Entropic regularization and the Sinkhorn algorithm

Replace non-negativity with entropic penalty term

$$\begin{aligned} W_\varepsilon(p, q) = \min_X \quad & \langle C, X \rangle - \varepsilon H(X) \\ \text{subject to} \quad & X\mathbf{1} = p, \quad X^T\mathbf{1} = q \end{aligned}$$

where $H = -\sum_i \sum_j X_{ij} \log X_{ij}$ is the entropy of X .

A **convex** optimization problem with simple constraints.

An approximation to optimal transport. Can (often) be solved very quickly.

The Sinkhorn algorithm

The Sinkhorn algorithm, starts with $v_0 = \mathbf{1}$, and alternates

$$u_{k+1} = p/Kv_k, \quad v_{k+1} = q/K^T u_{k+1}$$

where division is elementwise and $K_{ij} = \exp(-C_{ij}/\epsilon)$.

The Sinkhorn algorithm

The Sinkhorn algorithm, starts with $v_0 = \mathbf{1}$, and alternates

$$u_{k+1} = p/Kv_k, \quad v_{k+1} = q/K^T u_{k+1}$$

where division is elementwise and $K_{ij} = \exp(-C_{ij}/\epsilon)$.

Optimal solution to entropy-regularized OT is $X_\epsilon^\star = \text{diag}(u^\star)K\text{diag}(v^\star)$.

The Sinkhorn algorithm

The Sinkhorn algorithm, starts with $v_0 = \mathbf{1}$, and alternates

$$u_{k+1} = p / K v_k, \quad v_{k+1} = q / K^T u_{k+1}$$

where division is elementwise and $K_{ij} = \exp(-C_{ij}/\epsilon)$.

Optimal solution to entropy-regularized OT is $X_\epsilon^\star = \text{diag}(u^\star) K \text{diag}(v^\star)$.

Widely used, but has many limitations:

- sensitive to ϵ : too small risks numerical instability; too large causes blurry X_ϵ
- solution (if found) does not define a proper distance
- cannot encourage sparsity or structure. . .

Can we design a scalable algorithm for the real OT problem?

Perhaps, if we make clever use of modern hardware, such as GPUs.

GPUs offer massive parallelism, but are memory bound

- Registers: fastest (1 cycle), few KB
- Shared memory: fast (100 cycles), 48–96KB/block
- Global memory: large but slow (500+ cycles)

Key to success: algorithm should fit in fast memory, minimize memory traffic, avoid branching

Our approach: Douglas-Rachford splitting

Douglas-Rachford splitting

Solves problems on the form

$$\underset{x}{\text{minimize}} \quad f(x) + g(x)$$

where f and g are closed convex functions, with cheap prox-operators.

For penalty parameter $\rho > 0$, repeat

$$\begin{aligned} x^{k+1} &= \text{prox}_{\rho f}(y^k) \\ y^{k+1} &= y^k + \text{prox}_{\rho g}(2x^{k+1} - y^k) - x^{k+1} \end{aligned}$$

Global convergence of $\{x_k\}$ to x such that $0 \in \partial f(x) + \partial g(x)$, for all values of $\rho > 0$.

How to split the optimal transport?

A result by Bauschke et al. (2021) gives a hint.

Lemma. Let $p, q \geq 0$ satisfy $\mathbf{1}^T p = \mathbf{1}^T q$. Then,

$$\mathcal{X} = \{X \in \mathbf{R}^{m \times n} \mid X\mathbf{1} = p, \quad X^T\mathbf{1} = q\}$$

is non-empty, and for every given $X \in \mathbf{R}^{m \times n}$, we have

$$P_{\mathcal{X}}(X) = X - \frac{1}{n} ((X\mathbf{1} - p)\mathbf{1}^T - \gamma\mathbf{1}\mathbf{1}^T) - \frac{1}{m} (\mathbf{1}(X^T\mathbf{1} - q)^T - \gamma\mathbf{1}\mathbf{1}^T) := X + \phi\mathbf{1}^T + \mathbf{1}\varphi^T$$

where $\gamma = \mathbf{1}^T(X\mathbf{1} - p)/(m + n) = \mathbf{1}^T(X^T\mathbf{1} - q)/(m + n)$.

Projection computed using only matrix-vector multiplies/rank-one updates.

DROT: DR splitting for OT

Re-writing the OT problem as

$$\underset{X}{\text{minimize}} \quad \underbrace{\langle C, X \rangle + \iota_{\mathbb{R}_+^{m \times n}}(X)}_{f(X)} + \underbrace{\iota_{\mathcal{X}}(X)}_{g(X)}$$

yields the DR iterations

$$\begin{aligned} X^{k+1} &= [Y^k - \rho C]_+ \\ Y^{k+1} &= X^{k+1} + \phi_{k+1} \mathbf{1}^\top + \mathbf{1} \varphi_{k+1}^\top \end{aligned}$$

Eliminating Y , we only need to manipulate a single large matrix

$$X^{k+1} = [X^k + \phi_k \mathbf{1}^\top + \mathbf{1} \varphi_k^\top - \rho C]_+$$

No exponentials, no divisions, only rank-one updates.

DROT GPU kernel

Organize GPU threads into a 2D grid of blocks

- each thread block (SP) handles a submatrix of X
- each thread handles a single column (for memory efficiency)

DROT GPU kernel

Organize GPU threads into a 2D grid of blocks

- each thread block (SP) handles a submatrix of X
- each thread handles a single column (for memory efficiency)

Respect the memory hierarchy

- registers for private data, shared memory for common data across threads (e.g. ϕ , φ)
- perform partial reductions (e.g. row sums, cost evaluation) during matrix updates
- fuse operations into a single kernel pass to minimize memory transactions

DROT GPU kernel

Organize GPU threads into a 2D grid of blocks

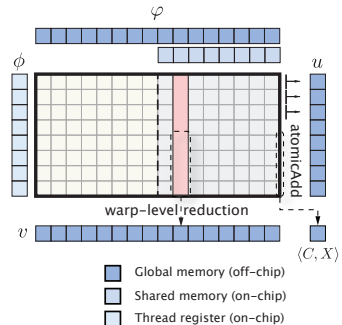
- each thread block (SP) handles a submatrix of X
- each thread handles a single column (for memory efficiency)

Respect the memory hierarchy

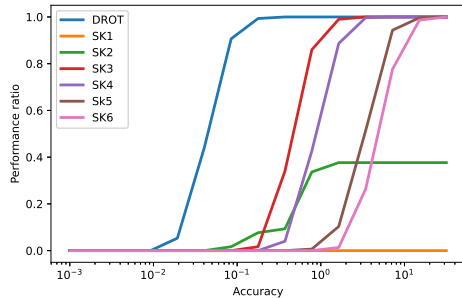
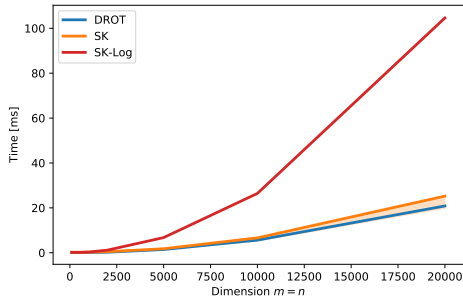
- registers for private data, shared memory for common data across threads (e.g. ϕ , φ)
- perform partial reductions (e.g. row sums, cost evaluation) during matrix updates
- fuse operations into a single kernel pass to minimize memory transactions

Reduce global memory access

- each element of X is read or written once per iteration
- can skip loading C every other iteration (by writing $X^+ - \rho C$)



Numerical results



Faster and more robust than Sinkhorn. Solves the true OT problem, returns sparse plans.

Adding regularization to DROT

Regularization: standard approach for encouraging structure in solution

$$\begin{array}{ll}\text{minimize} & \langle C, X \rangle + h(X) \\ \text{subject to} & X\mathbf{1} = p, \quad X^T\mathbf{1} = q \\ & X \geq 0\end{array}$$

Useful in practice, but non-trivial for Sinkhorn.

Adding regularization to DROT

Regularization: standard approach for encouraging structure in solution

$$\begin{array}{ll} \text{minimize} & \langle C, X \rangle + h(X) \\ \text{subject to} & X\mathbf{1} = p, \quad X^T\mathbf{1} = q \\ & X \geq 0 \end{array}$$

Useful in practice, but non-trivial for Sinkhorn.

We can extend the DROT approach to a family of useful regularizers.

Definition. h is sparsity promoting if $h(\tilde{X}) \leq h(X)$ for every \tilde{X} with $\tilde{X}_{ij} \in \{0, X_{ij}\}$

Includes weighted ℓ_1 , group-lasso, quadratic, constrained OT, and more ...

Sparsity-promoting regularizers

Proposition. Let $f(X) = \langle C, X \rangle + \mathbf{I}_{\mathbf{R}_+^{m \times n}}(X) + h(X)$ with h sparsity promoting. Then

$$\text{prox}_{\rho f}(X) = \text{prox}_{\rho h}([X - \rho C]_+)$$

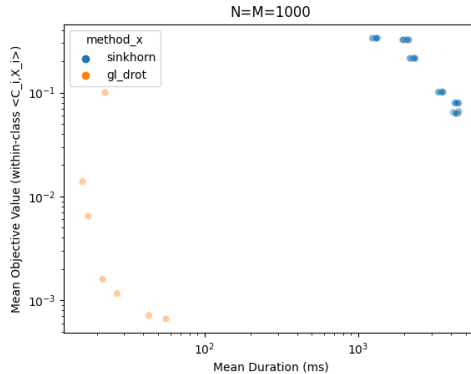
Consequence. splitting as

$$\underset{X}{\text{minimize}} \quad \underbrace{\langle C, X \rangle + h(X) + \iota_{\mathbf{R}_+^{m \times n}}(X)}_{f(X)} + \underbrace{\iota_{\mathcal{X}}(X)}_{g(X)}$$

results in a minimal update of the DROT approach. Efficient when prox is GPU-compatible.

Numerical results: domain adaptation using group-lasso regularized OT

Comparison with method of Courty et al. (sequential linearization, solved by SK)



Speedups of over 100x, with better accuracy!

Theoretical guarantees

Theorem. (informal) Let $\{X_k\}$ be generated by DROT. Then

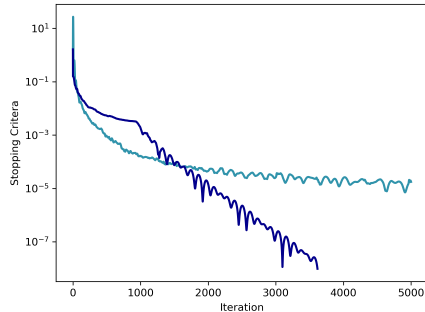
$$\langle C, \bar{X}_k \rangle + h(\bar{X}_k) - (\langle C, X^* \rangle + h(X^*)) \leq \frac{c}{k}$$

where $\bar{X}_k = \sum_{i=1}^k X_i/k$ and c is a constant depending on X_0, Y_0, X^* and Y^* .

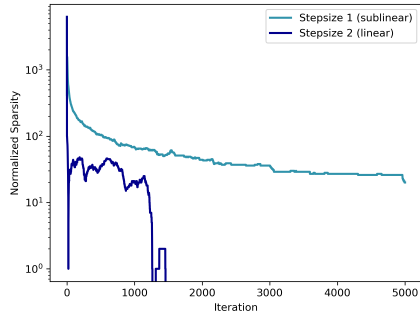
Significant improvement over SK's $\mathcal{O}(1/\sqrt{k})$, but not how method actually behaves.

Convergence behaviour in practice

Initial $1/k$ rate followed by fast linear convergence when sparsity stabilizes.



Convergence



Sparsity

Takeaways

DROT: GPU-native optimal transport solver

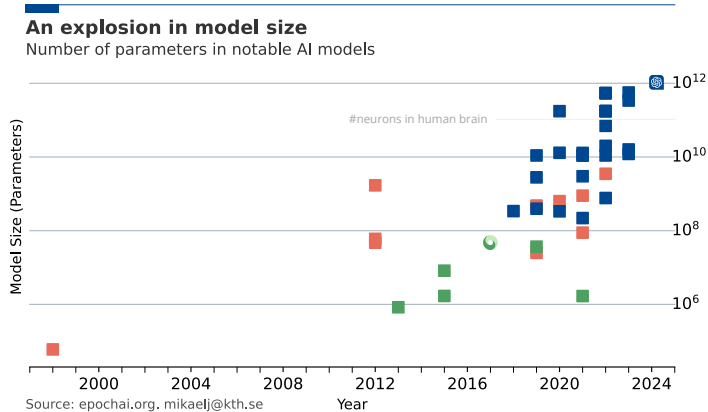
- accurate: solves true OT problem, not an approximation
- flexible: easily incorporates sparsity-promoting regularizers
- fast: up to 100x speedup vs Sinkhorn; avoids exponentials, divisions

Hardware-awareness is key: from a method that "does not work" to one that excels!

Contents

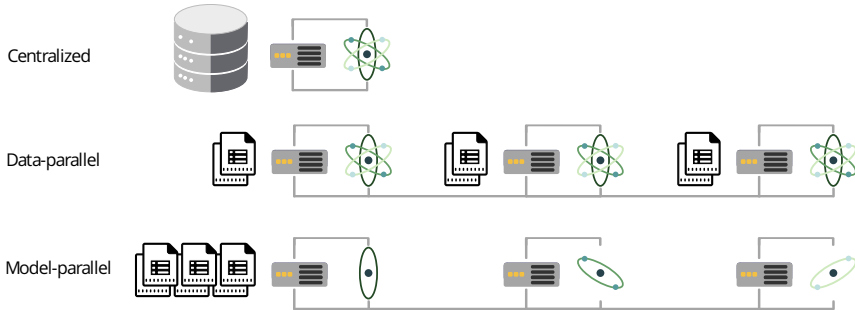
- Motivation
- A GPU-based optimal transport solver
- An efficient decentralized DNN training system
- Conclusions and outlook

The challenge of large-scale model training



Implies similar explosion in training data (Chinchilla scaling: 20 tokens/parameter)

Parallelizing large-scale training

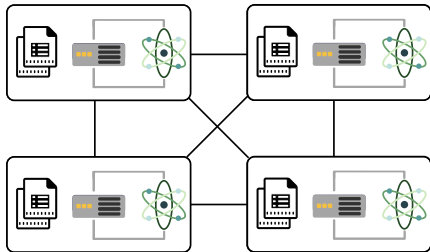


Two main approaches

- Distribute data: data parallelism (our focus)
- Distribute compute: model, pipeline, and tensor parallelism

Decentralized optimization for large-scale training

Q: Can we make decentralized optimization competitive for large-scale DNN training?



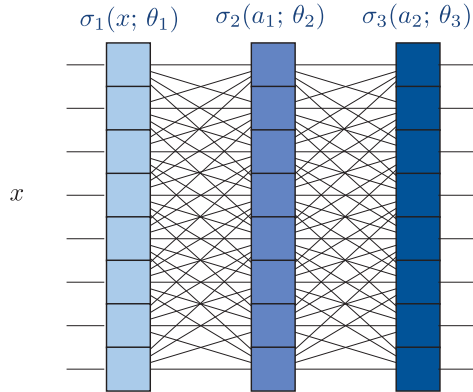
Earlier attempts fail to beat synchronous training on wall-clock time/validation loss.

Our recipe: eliminate all idle times, resist stragglers, respect heterogeneity.

Contents

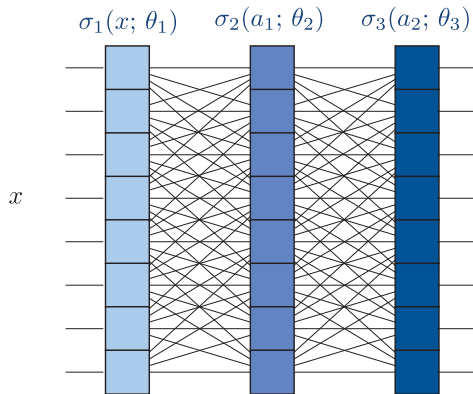
- Motivation
- Basics of DNN training
- Our proposal
- Conclusion

Basics of DNN training



$$y = f(x; \theta) = \sigma_3(\sigma_2(\sigma_1(x; \theta_1); \theta_2); \theta_3)$$

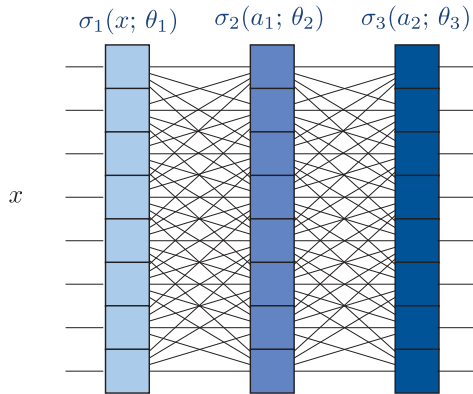
Basics of DNN training



Forward pass: evaluate f (and activations)

$$\begin{aligned}
 a_0 &= x \\
 a_1 &= \sigma_1(a_0; \theta_1) \\
 a_2 &= \sigma_2(a_1; \theta_2) \\
 a_3 &= \sigma_3(a_2; \theta_3) \\
 y &= a_3
 \end{aligned}$$

Basics of DNN training



Backward pass: compute gradients

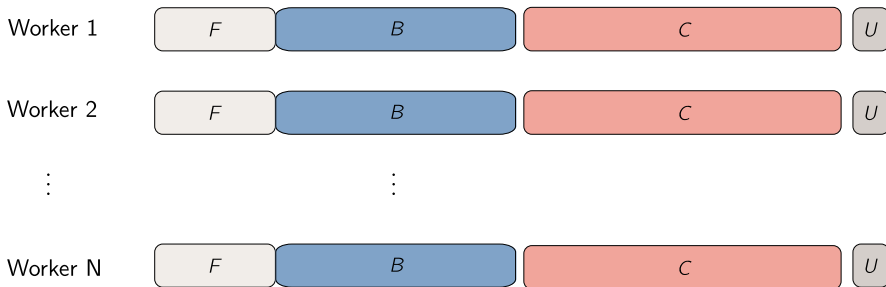
$$\frac{\partial \ell(y)}{\partial \theta_3} = \frac{\partial \ell}{\partial y} \cdot \frac{\partial y}{\partial \theta_3} = \frac{\partial \ell}{\partial y} \cdot \frac{\partial \sigma_3}{\partial \theta_3}(a_2)$$

$$\frac{\partial \ell(y)}{\partial \theta_2} = \frac{\partial \ell}{\partial y} \cdot \frac{\partial \sigma_3}{\partial a_2} \cdot \frac{\partial \sigma_2}{\partial \theta_2}(a_1)$$

$$\frac{\partial \ell(y)}{\partial \theta_1} = \frac{\partial \ell}{\partial y} \cdot \frac{\partial \sigma_3}{\partial a_2} \cdot \frac{\partial \sigma_2}{\partial a_1} \cdot \frac{\partial \sigma_1}{\partial \theta_1}(x)$$

DNN training using parallel gradient descent

$$\theta_{t+1} = \theta_t - \alpha \sum_{i=1}^N g_i(\theta_t)$$



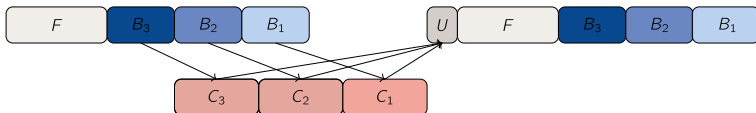
Reducing idle time

Bucketing: communicate gradient blocks (layers) when they are done

Naïve



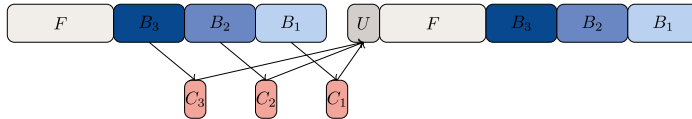
Bucketing



Simple idea, but can yield substantial improvements!

Why AllReduce can be hard to beat...

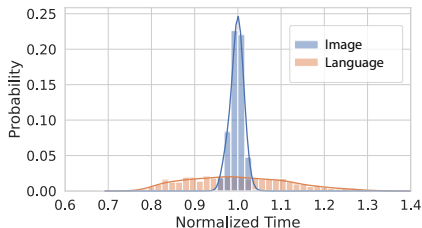
Parallel GD hard to beat with fast communication, homogeneous hardware, workloads.



Can happen with few workers, or in the very high-end of dedicated HPC clusters

Still need to wait until all gradients are computed and communicated.

...and why decentralized optimization could be superior



Natural language tasks introduce work-load variations.

Many HPC clusters have heterogeneous hardware, limited interconnect

Global synchronization (waiting for slowest node) becomes increasingly wasteful!

Our proposal

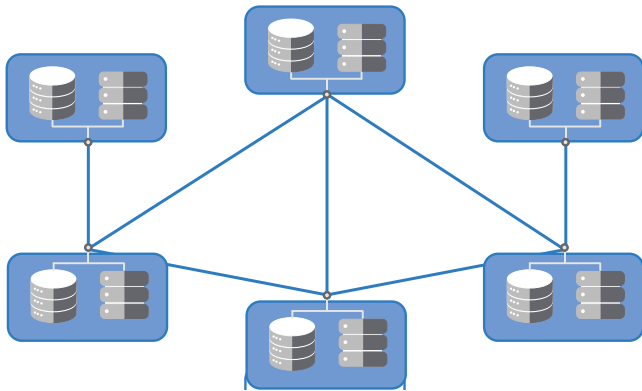
A framework for data-parallel training of DNNs based on:

- a decentralized system with GPUs as workers
- a decentralized Adam algorithm, with a twist
- use of time-varying (switched) communication topology to deal with heterogeneity
- an execution model for efficient system tuning

Resilient to workload variations, system noise and hardware heterogeneity.

A few words about decentralized optimization

$$\theta_i^{(t+1)} = -\alpha \nabla f_i(\theta_i^{(t)}) + \sum_{j \in \mathcal{N}_i} w_{ij} \theta_j^{(t)}$$



A decentralized Adam algorithm


A decentralized Adam algorithm based on decentralized gradient descent

$$\theta_i^{(t+1)} = \alpha d_i^{(t)}(\theta_i^{(t)}) + \sum_{j \in \mathcal{N}_i} w_{ij} \theta_j^{(t)}$$

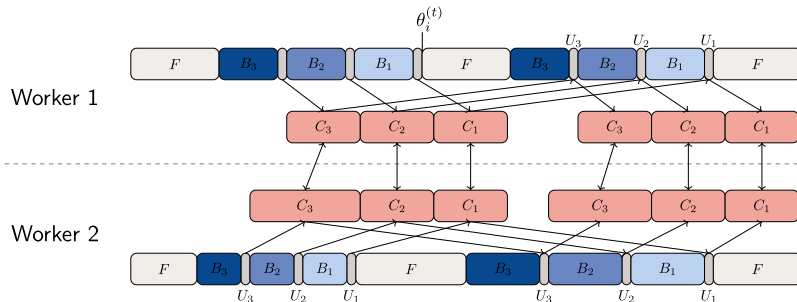
Rationale:

- Sharing models instead of gradients allows to increase overlap dramatically
- Nodes only share models with neighbors (in a communication topology)
- Adam essential for training transformer-based architectures

Resilient to stragglers, can approach 100% utilization



$$\theta_i^{(t+1)} = \alpha d_i^{(t)}(\theta_i^{(t)}) - \sum_{j \in \mathcal{N}_i} w_{ij} \theta_j^{(t)}$$



A decentralized Adam algorithm

Algorithm 1 Decentralized Adam on worker i

$$m_i^{(0)}, v_i^{(0)} \leftarrow 0; \theta_i^{(0)} \leftarrow \theta^{(0)}$$

for $t = 1, 2, \dots, T$ **do**

$$g_i^{(t)} \leftarrow \nabla \ell(\theta_i^{(t)}; \xi_i^{(t)})$$

$$m_i^{(t)} \leftarrow \beta_1 m_i^{(t-1)} + (1 - \beta_1) g_i^{(t)}$$

$$v_i^{(t)} \leftarrow \beta_2 v_i^{(t-1)} + (1 - \beta_2) [g_i^{(t)}]^2$$

$$\theta_i^{(t+1)} \leftarrow -\alpha \frac{m_i^{(t)}/(1-\beta_1^t)}{\sqrt{v_i^{(t)}/(1-\beta_2^t)+\epsilon}} + \sum_{j \in \mathcal{N}_i} w_{ij} \theta_j^{(t)}$$

end for

A decentralized Adam algorithm

Theorem. If Algorithm 1 uses $0 < \beta_1 < \beta_2 < 1$, then

$$\mathbf{E} \left[\|\nabla \ell(\bar{\theta}^{(\tau)})\|_2^2 \right] \leq \frac{4R}{\alpha \tilde{T}} \left(\ell(\bar{\theta}^{(0)}) - \ell^* \right) + E \left[\frac{1}{\tilde{T}} \ln \left(1 + \frac{R}{\epsilon(1 - \beta_2)} \right) - \frac{T}{\tilde{T}} \ln(\beta_2) \right], \quad (1)$$

where $\bar{\theta}^{(t)} = \frac{1}{N} \sum_{i=1}^N \theta_i^{(t)}$, τ is a random stopping time, and $E \sim \alpha^2$.

Note. Last term new compared to single-machine setting, but vanishes quickly with α .

The vanishing mini-batch problem

For good generalization performance, the batch size should not be too large.

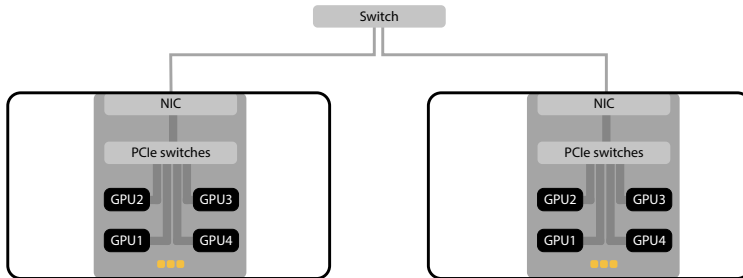
- with fixed batch-size, local mini-batches vanish in size as worker count increases
- small batches gives high-variance of momentum parameter updates

Communication in a typical HPC cluster

Simple graph abstraction of communication topology is not enough.

- intra-node communication (bus) is much faster (say, 20x) than inter-node
- congestion can occur at switches and interfaces

Need to respect this when designing communication pattern (and mixing weights w_{ij})!

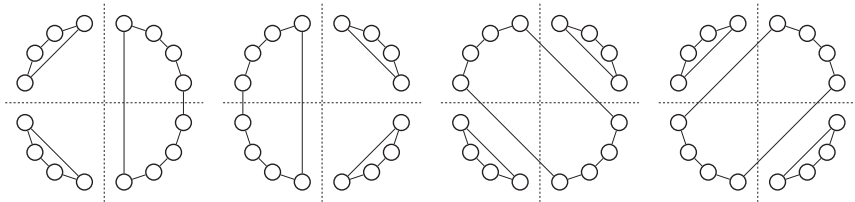


What is the optimal communication scheme?

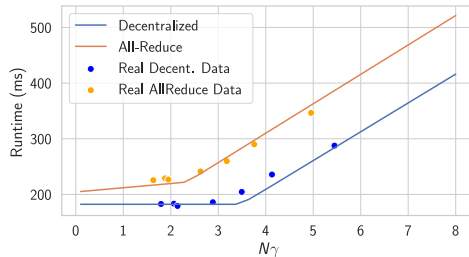
We do not know, but clearly

- limiting inter-node communication reduces costly communications and congestion
- increasing intra-node communication enhances convergence

We propose to use an alternating exponential ring:

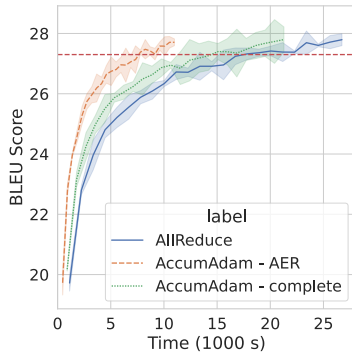
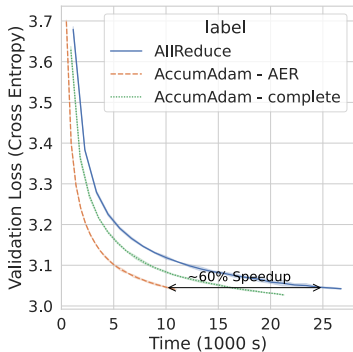


- should try to optimize per-iteration runtime
- captured by run-time model (communication/compute times, workload variations)



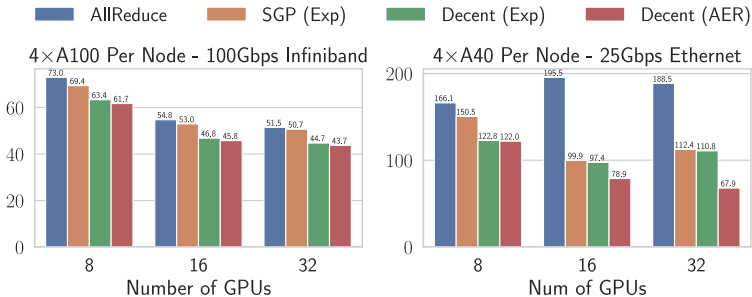
Evaluation

Training of transformer for English-to-German translation task (65M parameters)



Evaluation

Scaling in high-performance and communication-bound scenarios (GPT-2 training)



Takaways

Decentralized optimization for large-DNN training

- clearly useful, and perhaps more so in the years to come

Our solution:

- a decentralized system with GPUs as workers
- a decentralized Adam algorithm, with a twist
- use of time-varying (switched) communication topology to deal with heterogeneity
- an execution model for efficient system tuning

Strong practical performance, but many open theoretical problems

Summary and outlook

Bridging the hardware–algorithm disconnect:

- modern architectures require algorithms designed with hardware in mind.
- hardware-aware design as a core algorithmic principle?

Summary and outlook

Bridging the hardware–algorithm disconnect:

- modern architectures require algorithms designed with hardware in mind.
- hardware-aware design as a core algorithmic principle?

Two case studies, one message:

- a GPU-native OT solver designed for memory-aware computation, streamlined updates.
- decentralized training resilient to stragglers, hiding communication, eliminating idling

Summary and outlook

Bridging the hardware–algorithm disconnect:

- modern architectures require algorithms designed with hardware in mind.
- hardware-aware design as a core algorithmic principle?

Two case studies, one message:

- a GPU-native OT solver designed for memory-aware computation, streamlined updates.
- decentralized training resilient to stragglers, hiding communication, eliminating idling

Looking ahead:

- generalize the co-design principles to other classes of optimization problems
- explore tighter theoretical bounds for decentralized and DR-based algorithms
- investigate adaptability: can algorithms dynamically tune to hardware in real-time?