



Exceptional service in the national interest

Distributed Generalized Canonical Polyadic Decomposition

With Asynchrony and Less Communication

Cannada Lewis, Eric Phipps

Sep 24, 2021

HPEC 2021

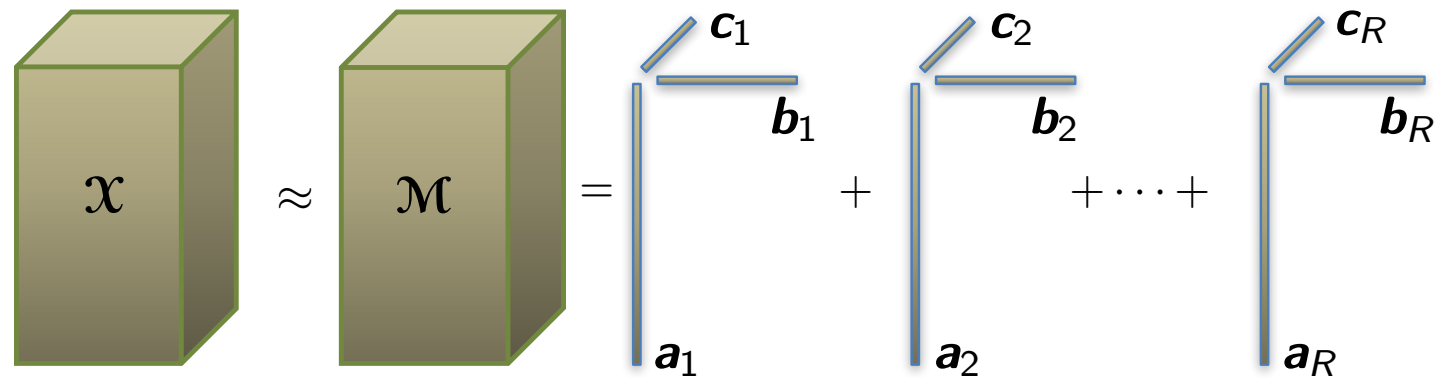




The Canonical Polyadic (CP) Decomposition

The CP decomposition seeks to represent a tensor as a sum of outer products, similar to low rank matrix decompositions.

Traditionally it has used the Gaussian metric to determine the best fit.



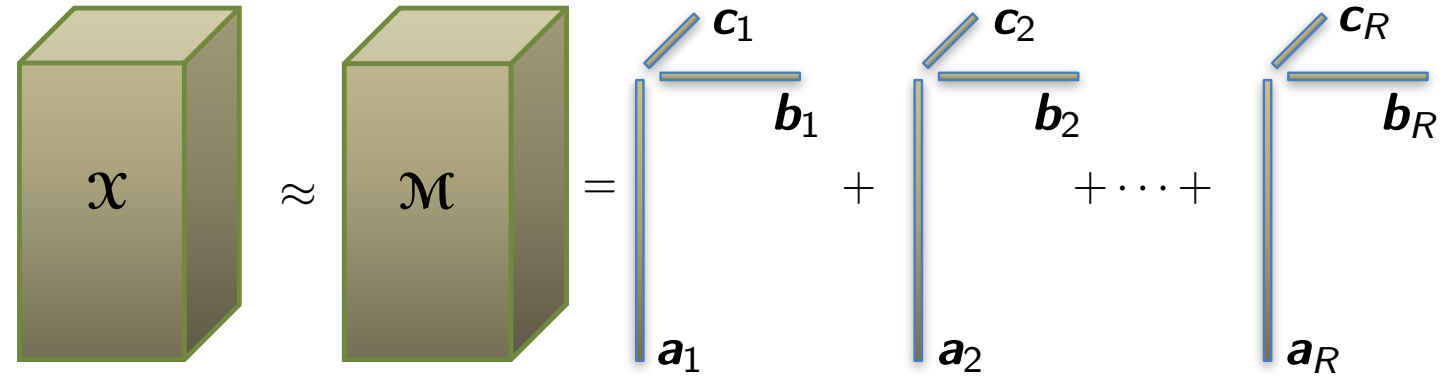
$$\min_{\mathcal{M}} F(\mathcal{X}, \mathcal{M}) = \|\mathcal{X} - \mathcal{M}\|_F^2 = \sum_i (x_i - m_i)^2$$

$$\text{s.t. } \mathcal{M} = \llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket = \mathbf{a}_1 \circ \mathbf{b}_1 \circ \mathbf{c}_1 + \dots + \mathbf{a}_R \circ \mathbf{b}_R \circ \mathbf{c}_R$$



The Generalized Canonical Polyadic* (GCP) Decomposition

Generalization allows for the use of other loss functions, but you lose access to alternating least squares (ALS).



$$x_i \sim p(x_i | \theta_i) \Rightarrow \min_{\mathcal{M}} F(\mathcal{X}, \mathcal{M}) = \sum_i f(x_i, m_i)$$

$$f(x_i, m_i) \equiv \log p(x_i | \ell^{-1}(m_i)) \quad \text{s.t. } \mathcal{M} = [\mathbf{A}, \mathbf{B}, \mathbf{C}]$$

| Distribution | Link function | Loss function | Constraints |
|----------------------------|-------------------------------|---|--------------------------------------|
| $\mathcal{N}(\mu, \sigma)$ | $m = \mu$ | $(x - m)^2$ | $x, m \in \mathbb{R}$ |
| Gamma(k, σ) | $m = k\sigma$ | $x / (m + \epsilon) + \log(m + \epsilon)$ | $x > 0, m \geq 0$ |
| Poisson(λ) | $m = \lambda$ | $m - x \log(m + \epsilon)$ | $x \in \mathbb{N}, m \geq 0$ |
| | $m = \log \lambda$ | $e^m - xm$ | $x \in \mathbb{N}, m \in \mathbb{R}$ |
| Bernoulli(ρ) | $m = \rho / (1 - \rho)$ | $\log(m + 1) - x \log(m + \epsilon)$ | $x \in \{0, 1\}, m \geq 0$ |
| | $m = \log(\rho / (1 - \rho))$ | $\log(1 + e^m) - xm$ | $x \in \{0, 1\}, m \in \mathbb{R}$ |

*Hong, Kolda, Duersch. Generalized Canonical Polyadic Tensor Decomposition. SIAM Review, 2019.



Stochastic Gradient Descent (SGD) for GCP

- Without access to ALS one simple method to solve GCP is Stochastic Gradient Descent.
 - Allows for sparse gradient tensors which wouldn't be possible with gradient descent
 - Other methods are possible and may offer improved convergence.

SGD randomly samples a subset of indices, The Update can use many different strategies

Gradient Descent Terms

Form the gradient tensor $\mathcal{Y}y(i_1, \dots, i_d) = y_i = \frac{\partial f}{\partial m}(x_i, m_i)$

Then factor gradients are:

MTTKRP!

$$\mathbf{G}_k = \frac{\partial F}{\partial \mathbf{A}_k} = \mathcal{Y}_{(k)}(\mathbf{A}_d \odot \dots \odot \mathbf{A}_{k+1} \odot \mathbf{A}_{k-1} \odot \dots \odot \mathbf{A}_1)$$

Algorithm GD/SGD (in pseudocode):

```
// Given Tensor T, Factors F, and Gradients G  
while(!converged):
```

```
    Y = Tensor(T.shape())
```

```
    for idx in T.indices():  
        Y(idx) = delta_f(T, F, idx)
```

```
    for d in ndims:  
        G(d) = MTTKRP(Y, F, d)  
        F(d) = Update(F(d), G(d), ...)
```

```
    converged = test(T, F)
```



Genten

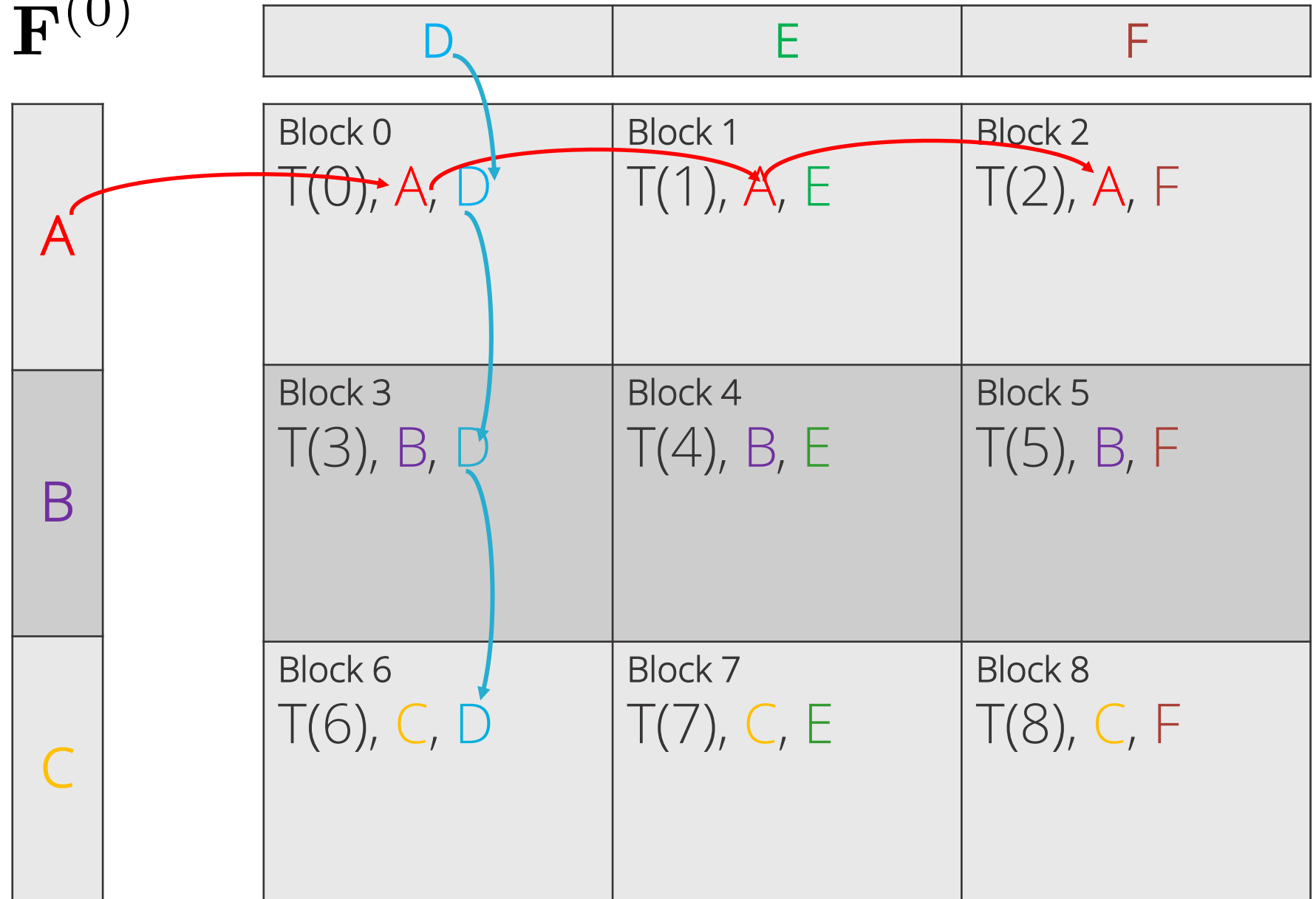
- GCP Software developed at Sandia
 - With contributions from Eric Phipps, Tamara Kolda, Daniel Dunlavy, Grey Ballard, and others
 - A C++/**Kokkos** port of the Matlab Tensor Toolbox
 - Publicly available at <https://gitlab.com/tensors/genten>
 - Implements full CP-ALS algorithm for sparse (and dense) tensors, as well as GCP algorithm for sparse tensors
- Uses the **Kokkos** C++ performance portability layer for shared memory parallelism
 - Multicore CPU: OpenMP, pThreads, ...
 - GPUs: Cuda, HIP, ...



Tensor Distribution and Replication of Factors on Sub Grids

$\mathbf{F}^{(0)}$

$\mathbf{F}^{(1)}$



Blocking chosen to minimize factor replication.



Distribute Via Allreduce

Every node generates local gradient tensors, computes local factor updates and then allreduces each gradient.

Things to be aware of:

- Random sampling is only over local data
- No effort was made for each rank to have the same number of non-zeros
- Objective function is scored as a sum of local objectives

```
Algorithm GD/SGD (in pseudocode):
// Given Tensor T, Factors F, and Gradients G
while(!converged):
    Y = Tensor(T.shape())

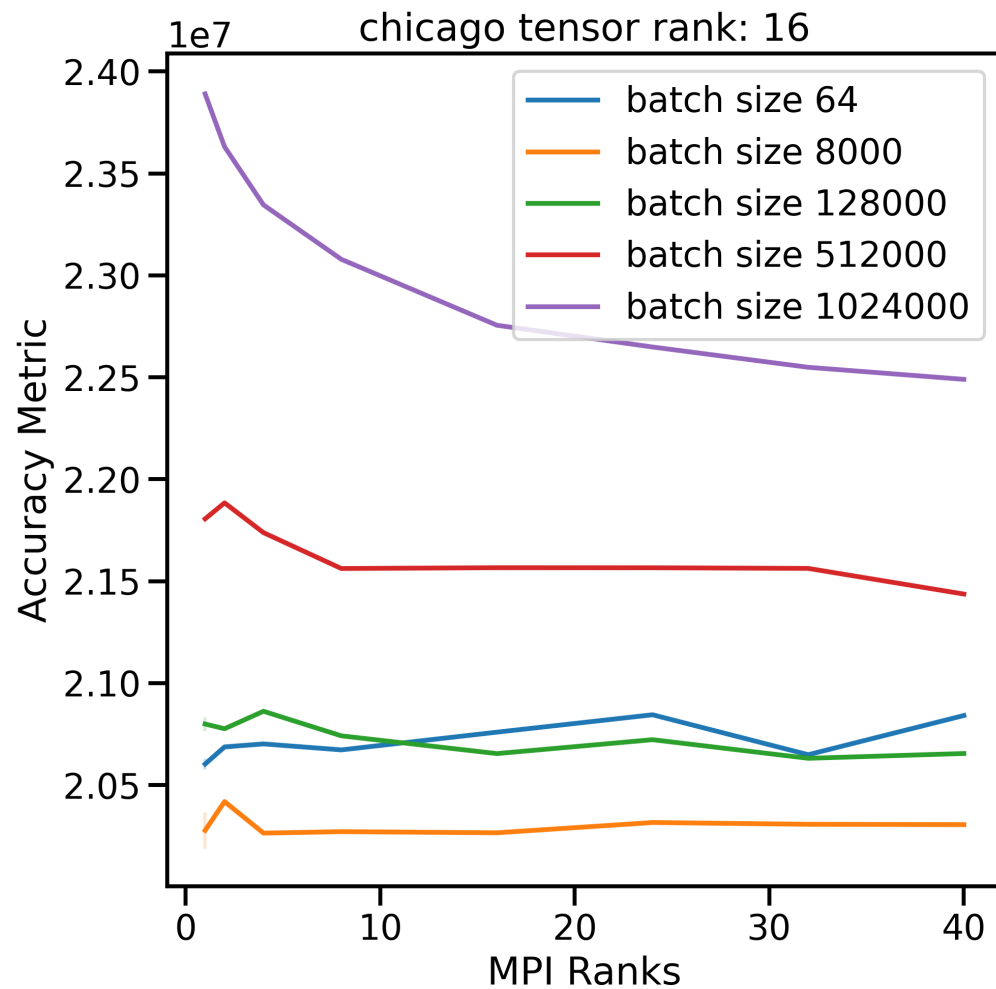
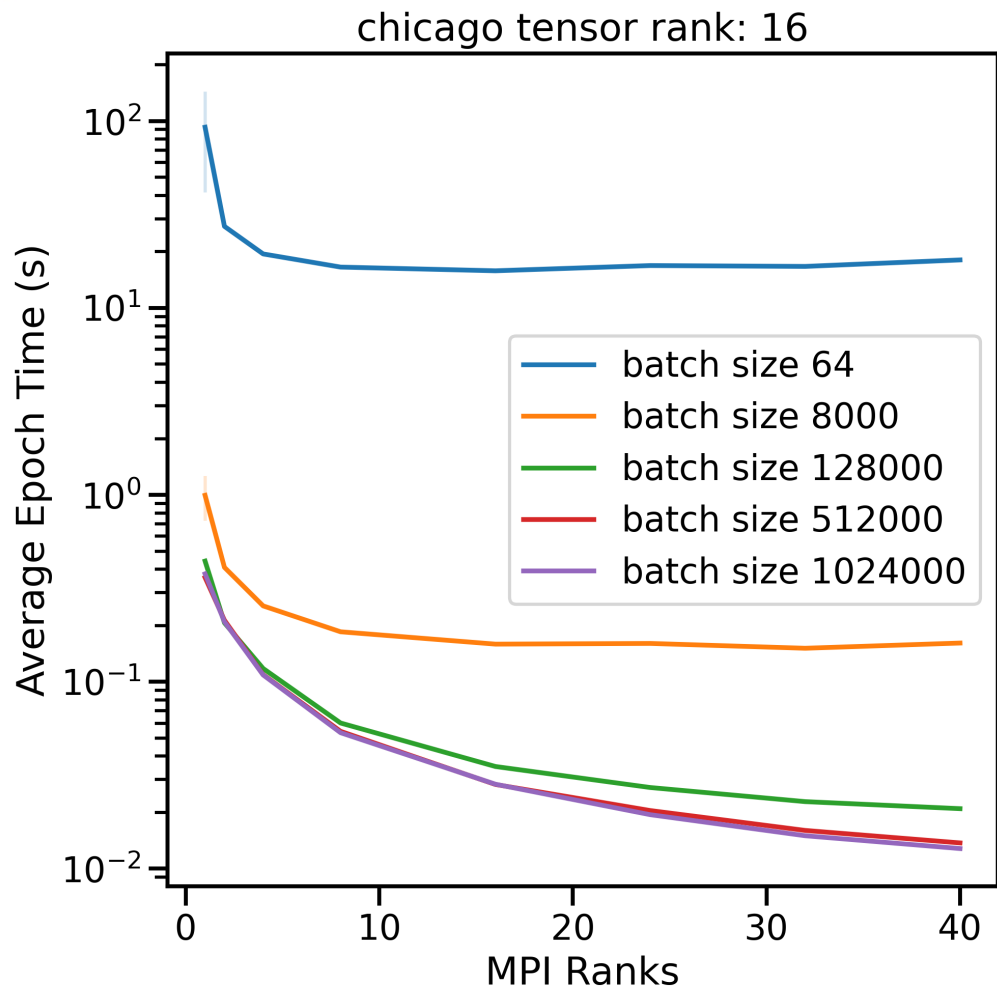
    for idx in T.local_indices():
        Y(idx) = delta_f(T, F, idx)

    for d in ndims:
        G(d) = MTTKRP(Y, F, d)
        AllReduce(G(d), SubComm)
        F(d) = Update(F(d), G(d), ...)

converged = test(T, F)
```



Strong Scaling Chicago Crime ([6186, 24, 77, 32] NNZ: 5330673)





Communication Avoiding Algorithms

Things to be aware of:

- Random sampling is only over local data
- No effort was made for each rank to have the same number of non-zeros
- Objective function is scored as a sum of local objectives

Algorithm GD/SGD (in pseudocode):

```
// Given Tensor T, Factors F, Gradients G, Iteration I
while(!converged):
    Y = Tensor(T.shape())

    for idx in T.local_indices():
        Y(idx) = delta_f(T, F, idx)

    for d in ndims:
        G(d) = MTTKRP(Y, F, d)
        F(d) = Update(F(d), G(d), ...)

        if(I % Nsync == 0):
            Sync(F)

converged = test(T, F)
```



Local SGD¹ (FedAvg²)

```
1 for (auto i = 0; i < iters; ++i) {  
2   do_iter(gradient, factors);  
3   if ((i + 1) % sync_iters == 0) {  
4     auto average = true;  
5     allReduceKT(factors, average);  
6   }  
7 }
```

Calls either `MPI_Accumulate` or `MPI_Get_Accumulate`.

Elastic Averaging³

```
1 const auto alpha = 0.9 / nprocs;  
2 for (auto i = 0; i < iters; ++i) {  
3   if (nprocs > 1 && (i + 1) % sync_iters == 0) {  
4     // Read the center variable  
5     doCenterOp(center, CenterOP::CenterRead);  
6  
7     factors.elastic_difference(diff, center, 1.0);  
8     diff.scale(alpha);  
9     factors_vec.plus(diff, -1.0);  
10    doCenterOp(diff, CenterOP::CenterAccumulate);  
11  }  
12  
13  do_epoch_iter();  
14 }
```

1. Stich, Sebastian U. "Local SGD converges fast and communicates little." *arXiv preprint arXiv:1805.09767* (2018).

2. McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." PMLR, 2017.

3. Zhang, Sixin, Anna Choromanska, and Yann LeCun. "Deep learning with elastic averaging SGD: 3rd, ICLR 2015



Results for single epochs

Nell2: <http://frostd.io/tensors/nell-2/>

Order: 3

Dims: 12,092 x 9,184 x 28,818

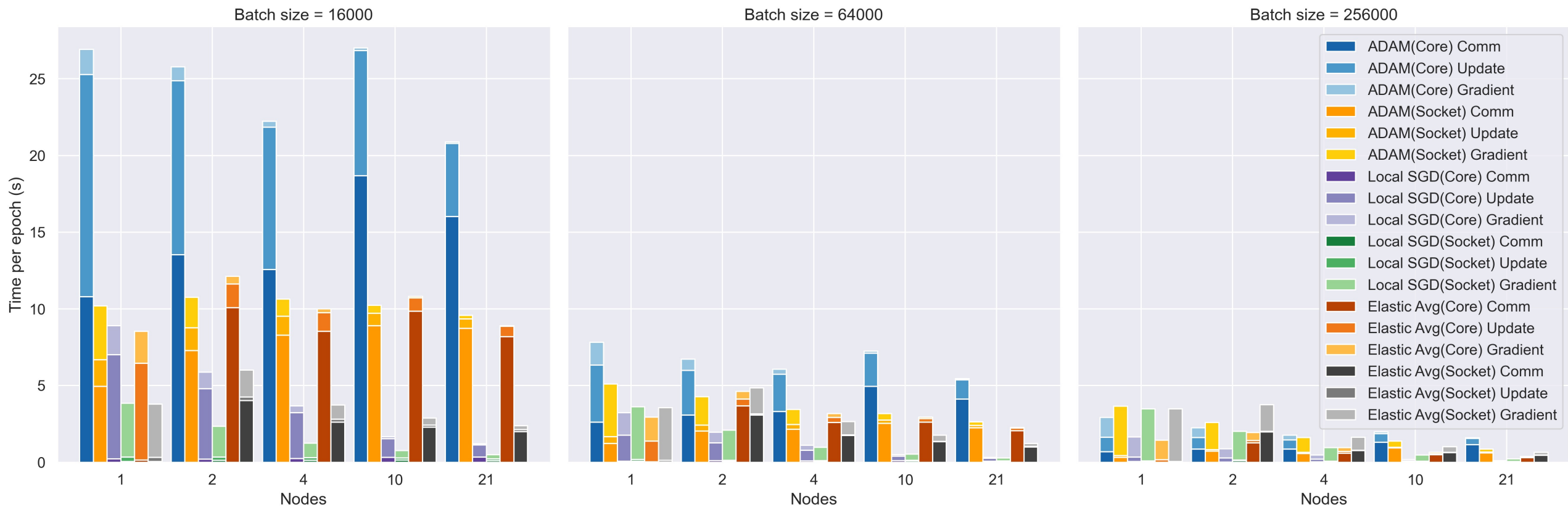
NNZ: 76,879,419

Loss: Poisson

Sync frequency: 128 batches (probably too large in retrospect)

Lessons:

1. Batch size matters A LOT
 1. Fewer updates
 2. Fewer communication events
2. Core vs Socket matters a bit
3. Infrequent communication makes epochs faster
4. Gradient calculation scales very well





Conclusions and Next Steps

Conclusions

- Created scalable version of Genten using MPI that will allow us to decompose tensors that were out of the reach of single rank Genten.

Next Steps

- Cuda aware MPI implementation
- Convergence and time to solution results for reduced communication algorithms (FedOpt¹!!)

1. Reddi, Sashank, et al. "Adaptive federated optimization." *arXiv preprint arXiv:2003.00295* (2020).