This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2021-6213C

Exceptional service in the national interest

Distributed Generalized Canonical Polyadic Decomposition

With a plan for asynchrony

Cannada Lewis, Eric Phipps, Tamara Kolda

May 21, 2021

SIAM Conference on Applied Linear Algebra

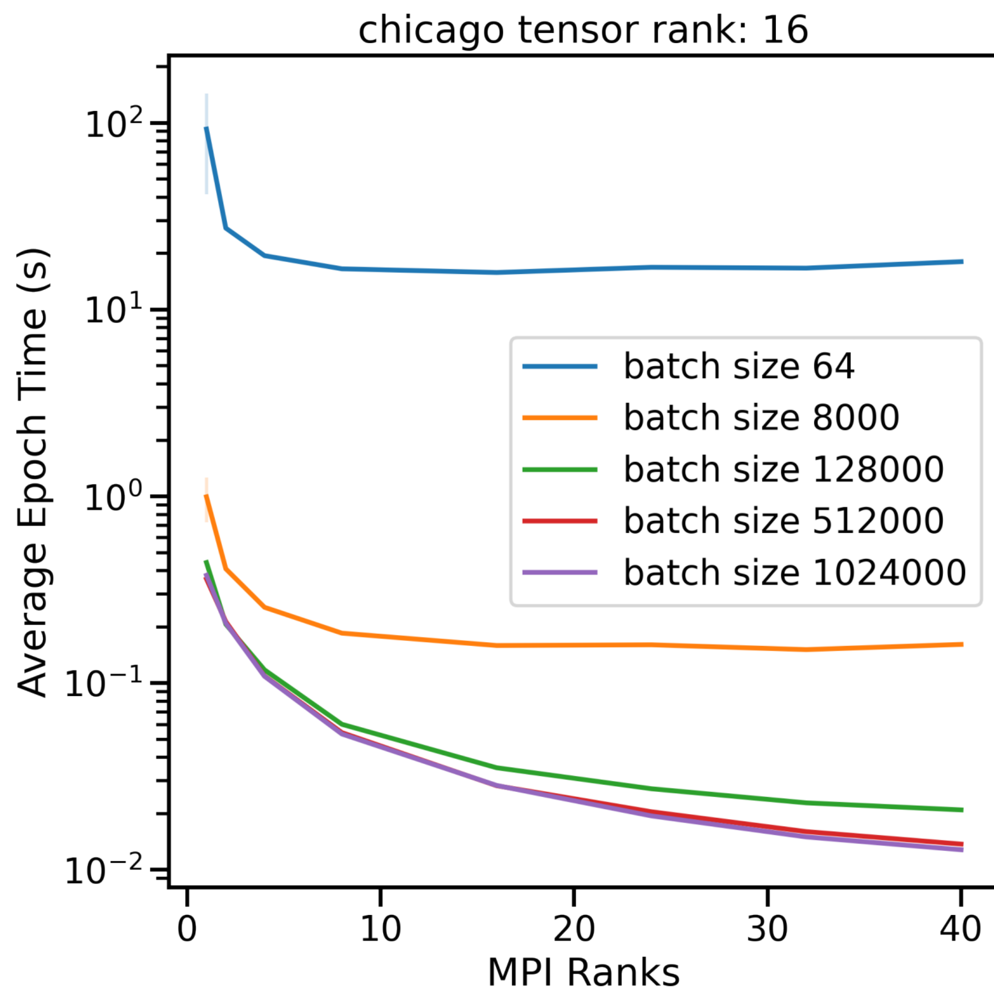
Work Funded By: DOD Advanced Computing Initiative

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

U.S. DEPARTMENT OF ENERGY

NNSA

Take Home Message



We achieved good strong scaling with up to 20X resources starting from a single node

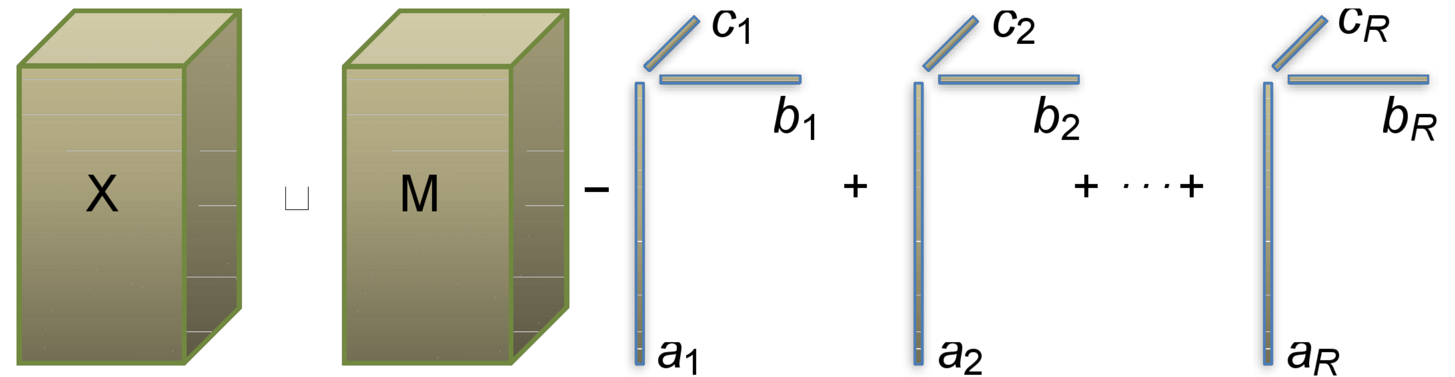
AGENDA

- What is the CP decomposition?
- Introduction of Generalized CP.
- Introduction of Genten/GentenMPI
- Our distribution strategy and motivations.
- Results.

The Canonical Polyadic (CP) Decomposition

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

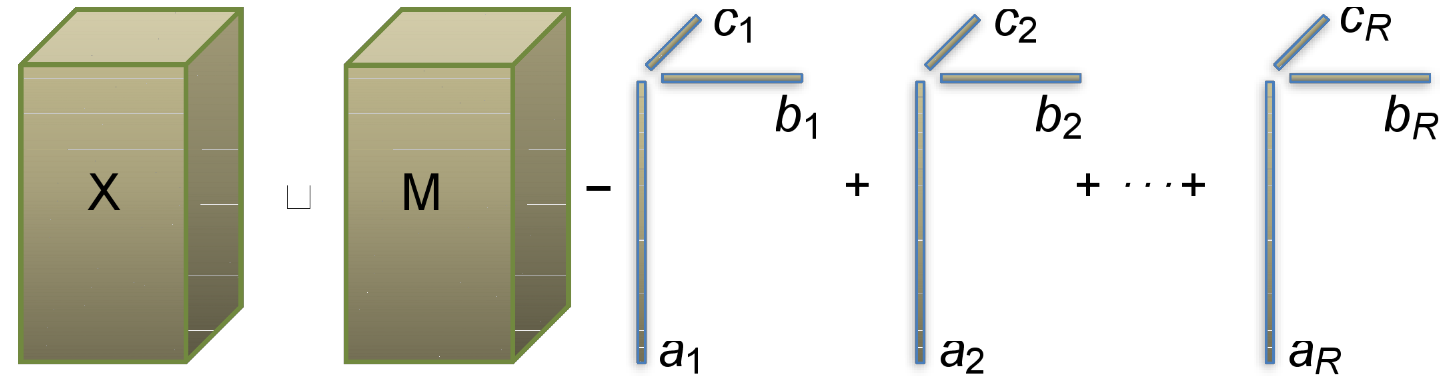
Traditionally it has used the Gaussian metric to determine the best fit.



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

$$\text{s.t.} \quad \mathcal{M} = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!] = \boldsymbol{a}_1 \circ \boldsymbol{b}_1 \circ \boldsymbol{c}_1 + \cdots + \boldsymbol{a}_R \circ \boldsymbol{b}_R \circ \boldsymbol{c}_R$$

The Generalized Canonical Polyadic* (GCP) Decomposition



The GCP decomposition provides a framework for using many different loss functions, that may yield better decompositions for certain tensors, for example tensors based on count data.

Unfortunately GCP does not allow for the use of the traditional CP-ALS algorithm for minimization.

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

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 try for GCP is just Gradient Descent.
  - But even for sparse tensors gradient descent yields dense intermediates (**Y**), while SGD allows for sparse intermediates
  - Empirical evidence in deep learning suggests that SGD may exhibit better convergence properties than gradient descent
- Other methods are possible and may offer improved convergence. For example Teresa's talk.

SGD randomly samples a subset of indices and often uses special step functions, doing less work per update and allowing for a sparse MTTKRP

Gradient Descent Terms

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

Then factor gradients are:

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

MTTKRP!

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) = step(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, in principle AMD/Intel GPUs, but not yet tested
  - Intel Xeon Phi

kokkos

Previous Distribution Work Genten MPI

- An MPI implementation of Genten based on Trilinos/Tpetra
  - Authors Karen Devine and Grey Ballard
  - Details at: https://www.osti.gov/biblio/1656940

Example Scaling and Performance

- Why is this work not based on GentenMPI?
  - GentenMPI reimplanted methods in Genten in Tpetra, but we would like to get more reuse of the genten code if possible
  - Trillinos/Tpetra are heavy weight dependencies
  - Code in this work can be unified with GentenMPI in the future.

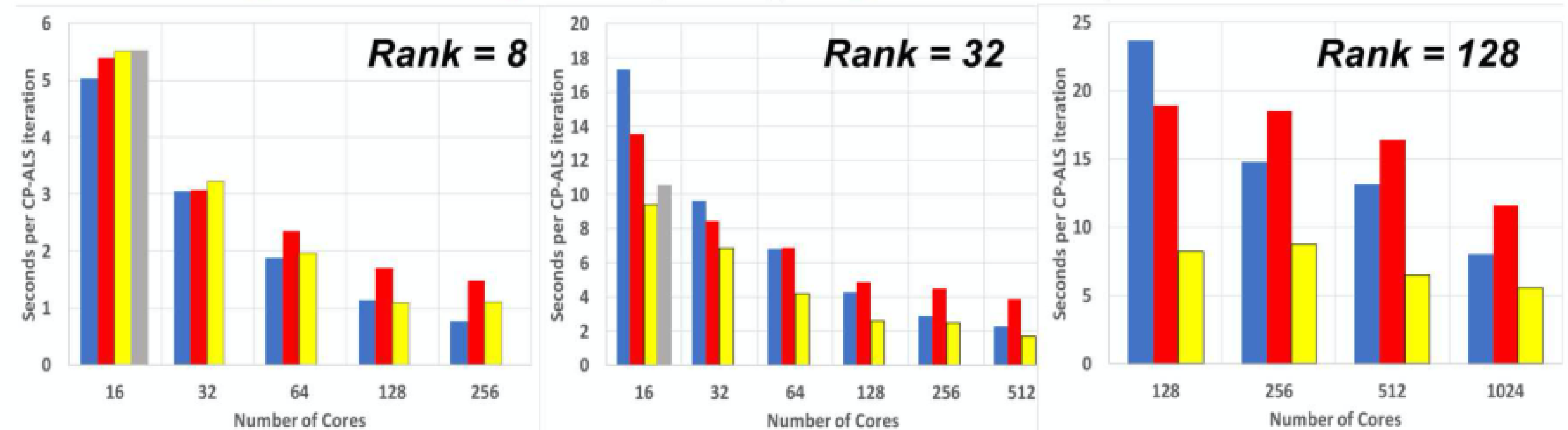


Figure 5-3. Strong scaling of CP-ALS on the *delicious-4d* tensor from the FROSTT [20] collection. GentenMPI times per CP-ALS iteration are compared with those from Genten [19] and SPLATT [21] with MPI-only and MPI+OpenMP.

Locality of Tensor Data and Tensor Gradient Data In SGD for GCP

Let's look at the gradient calculation for a matrix block distributed over 9 blocks

The gradient tensor for block **0** depends only on block **A** from **F**(0) and block **D** from **F**(1)

Gradient Tensor Block 0 can then be used to generate partial contributions to the factor gradients of **F**(0) block **A** and **F**(1) block **D**.

Locality of Tensor Data and Tensor Gradient Data In SGD for GCP

What each block can produce

$F^{(0)}$ (rows: A, B, C) and $F^{(1)}$ (columns: D, E, F)

$F^{(0)}$ \ $F^{(1)}$	D	E	F
A: $G_A = G_{A(0)} + G_{A(1)} + G_{A(2)}$	**Block 0** Inputs: T(0), A, D Outputs: G(0), $G_{A(0)}$, $G_{D(0)}$	**Block 1** Inputs: T(1), A, E Outputs: G(1), $G_{A(1)}$, $G_{E(0)}$	**Block 2** Inputs: T(2), A, F Outputs: G(2), $G_{A(2)}$, $G_{F(0)}$
B: $G_B = G_{B(0)} + G_{B(1)} + G_{B(2)}$	**Block 3** Inputs: T(3), B, D Outputs: G(3), $G_{B(0)}$, $G_{D(1)}$	**Block 4** Inputs: T(4), B, E Outputs: G(4), $G_{B(1)}$, $G_{E(1)}$	**Block 5** Inputs: T(5), B, F Outputs: G(5), $G_{B(2)}$, $G_{F(1)}$
C: $G_C = G_{C(0)} + G_{C(1)} + G_{C(2)}$	**Block 6** Inputs: T(6), C, D Outputs: G(6), $G_{C(0)}$, $G_{D(2)}$	**Block 7** Inputs: T(7), C, E Outputs: G(7), $G_{C(1)}$, $G_{E(2)}$	**Block 8** Inputs: T(8), C, F Outputs: G(8), $G_{C(2)}$, $G_{F(2)}$

Distribution Goals and Strategy

Goals:

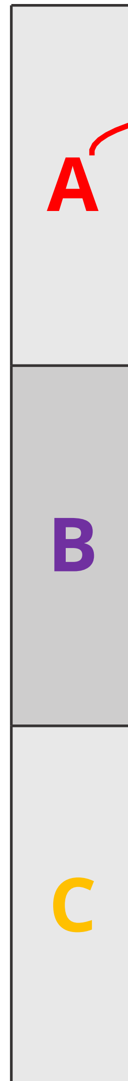
- Minimize storage (within reason) to run on a small number of fat nodes or devices.
- Be reasonably communication efficient while reusing as much code as possible.

Plan:

- Distribute tensor with 1 block per node
- Replicate factors on subgrids using MPI cartesian topologies
- AllReduce the gradients across the subgrids to keep factors up to date

Replication of Factors on Sub Grids, The Allreduces Follow The Same Pattern

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



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

D	E	F
Block 0 T(0), A, D	**Block 1** T(1), A, E	**Block 2** T(2), A, F
Block 3 T(3), B, D	**Block 4** T(4), B, E	**Block 5** T(5), B, F
Block 6 T(6), C, D	**Block 7** T(7), C, E	**Block 8** T(8), C, F

Attempt at Less Synchronization

- Instead of Allreducing the Gradient at every step we can average the factors after every N^{th} iteration, this breaks the step dependence of SGD, but appears to work okay in practice.

- Alternatively we can use a method similar to Elastic Averaging* which allows the factors to differ, but tethers them to a center variable. This method provides a way to do work asynchronously via MPI_IAllReduce (shown at the end if there is time)

Algorithm Similar to Elastic AVG SGD (in pseudocode):

```
// Given Local Tensor T, Local Factors F,
// Local Gradients G, and
// a synchronized center variable

iter = 1
C = F // Center starts out equal to F
while(!converged):
      if iter % N == 0:
            for d in ndims:
                  diff = F(d) – C(d)
                  F(d) -= alpha * diff
                  AllReduce(diff)
                  C(d) += alpha * diff
      ++iter

      // ... Same as single node SGD after
      // this point only C is synchronized
```

*Zhang, Sixin and Choromanska, Anna E and LeCun, Yann, “Deep learning with Elastic Averaging SGD”, Advances in Neural Information Processing Systems 2015

Results

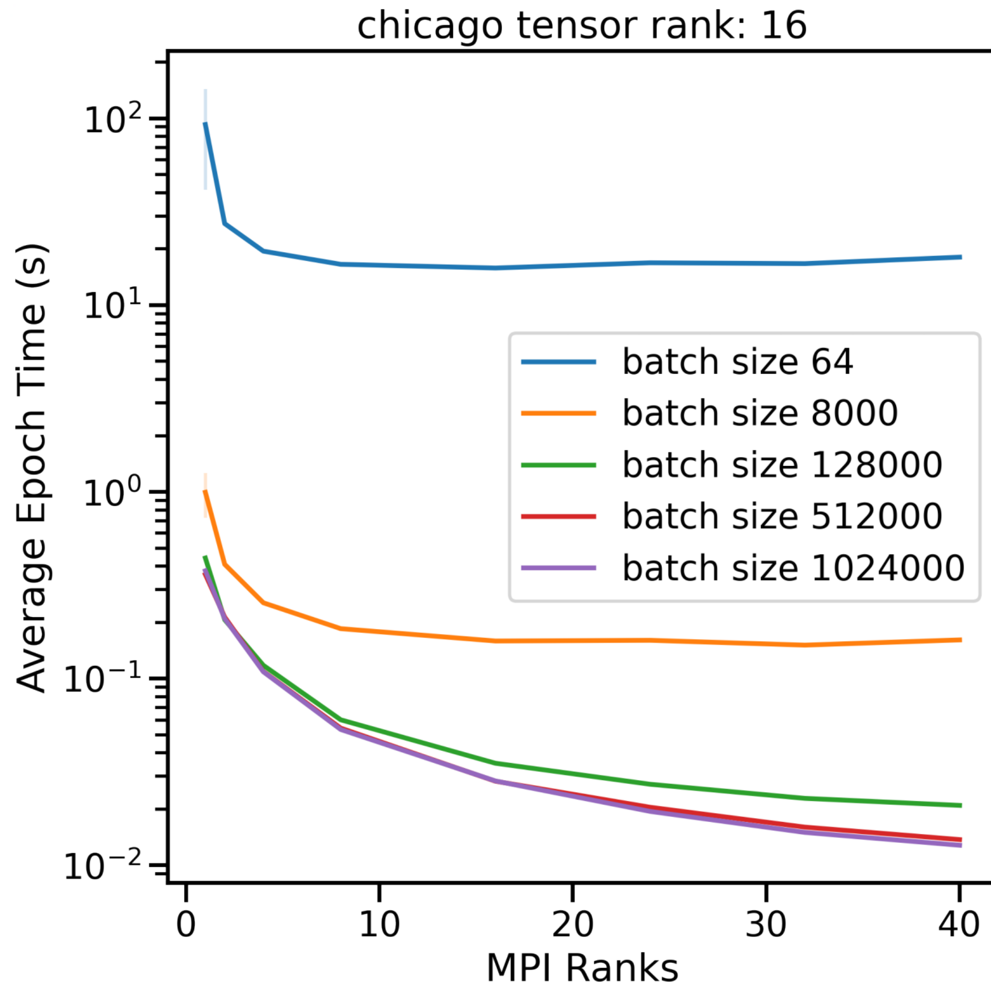
Terminology And Details

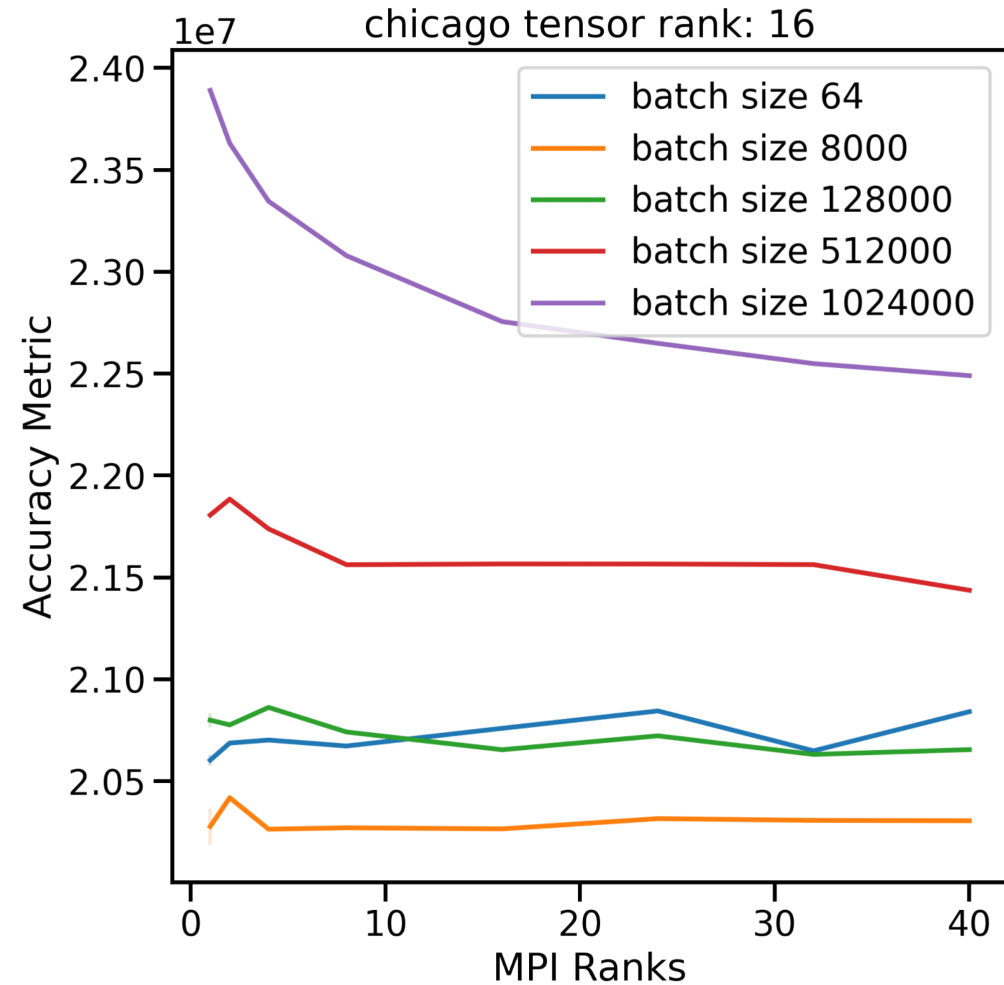
- **Epoch**: A set of stochastic gradient steps, can be arbitrarily sized, but in this work we coupled it with the number of iterations and batch size to sample a large fraction of the non-zeros (between 50% and ~100%)
- **Batch Size**: Number of samples used in a specific stochastic gradient update step, larger batch sizes increase the parallelism available in a given iteration, but using a very large batch size can cause poor convergence and may do unnecessary work.
- **Other**: We use a sampling strategy that samples approximately the same number of zeros and non-zeros. See “Stochastic Gradients for Large-Scale Tensor Decomposition”* for exact details
- Accuracy metric score (fest) is the sum of the loss function over 100000 zeros and 100000 non-zeros all randomly sampled
- Hardware was dual socket (24 core/socket) Xeon Skylake Processors with avx-512. MPI was run in a configuration of 1 MPI rank per socket and 24 OpenMP threads per process.
- Blocking was chosen to minimize the storage required for the factor matrices
- Each job was run for 50 epochs, Chicago Crime used enough iterations/epoch to sample approximately 100% of non-zeros and Vast 2015 used enough iterations/epoch to sample approximately 50% of non-zeros. Both tensors were obtained from frostt.io**

*Tamara Kolda and David Hong SIAM J. MATH. DATA SCI. No. 4, pp. 1066-1095, 2020

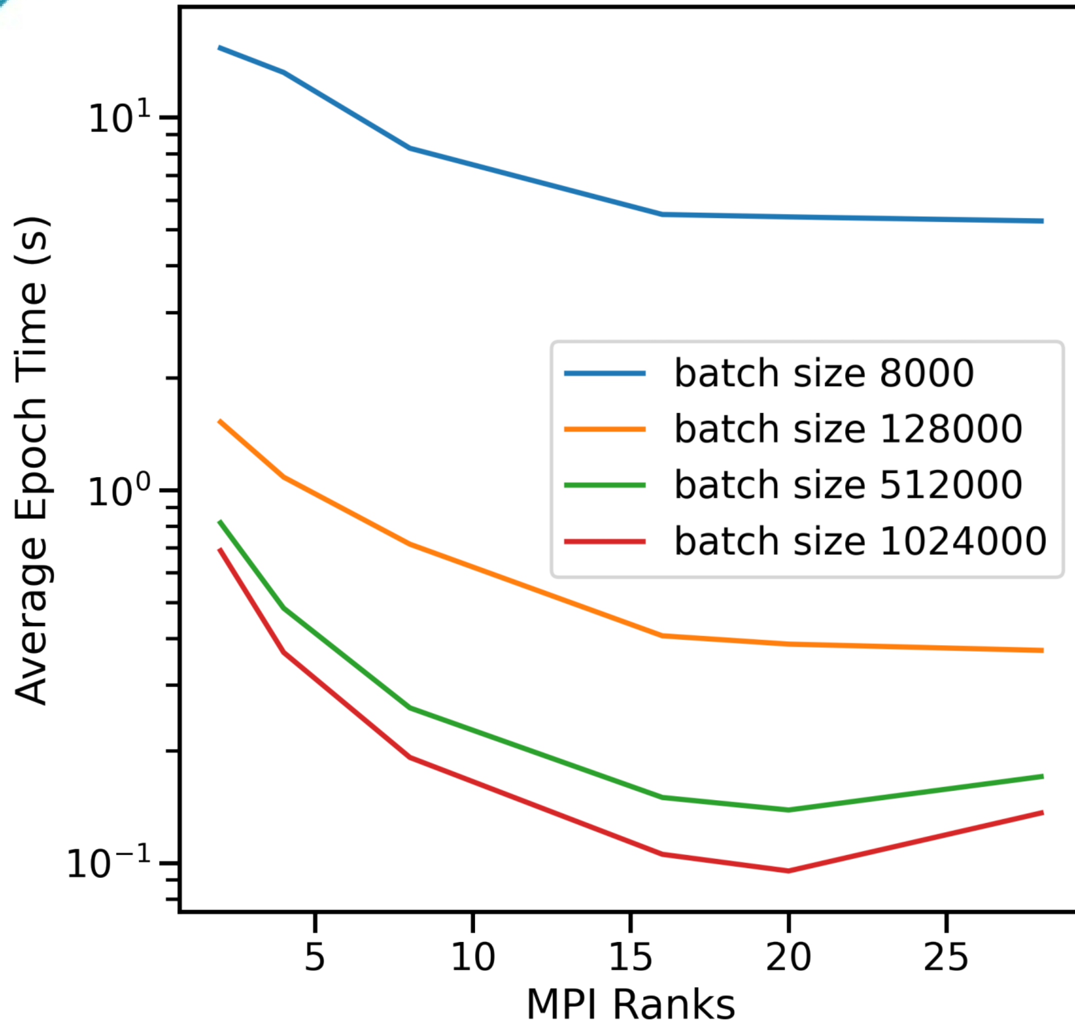
**The Formidable Repository of Open Sparse Tensors and Tools, 2017

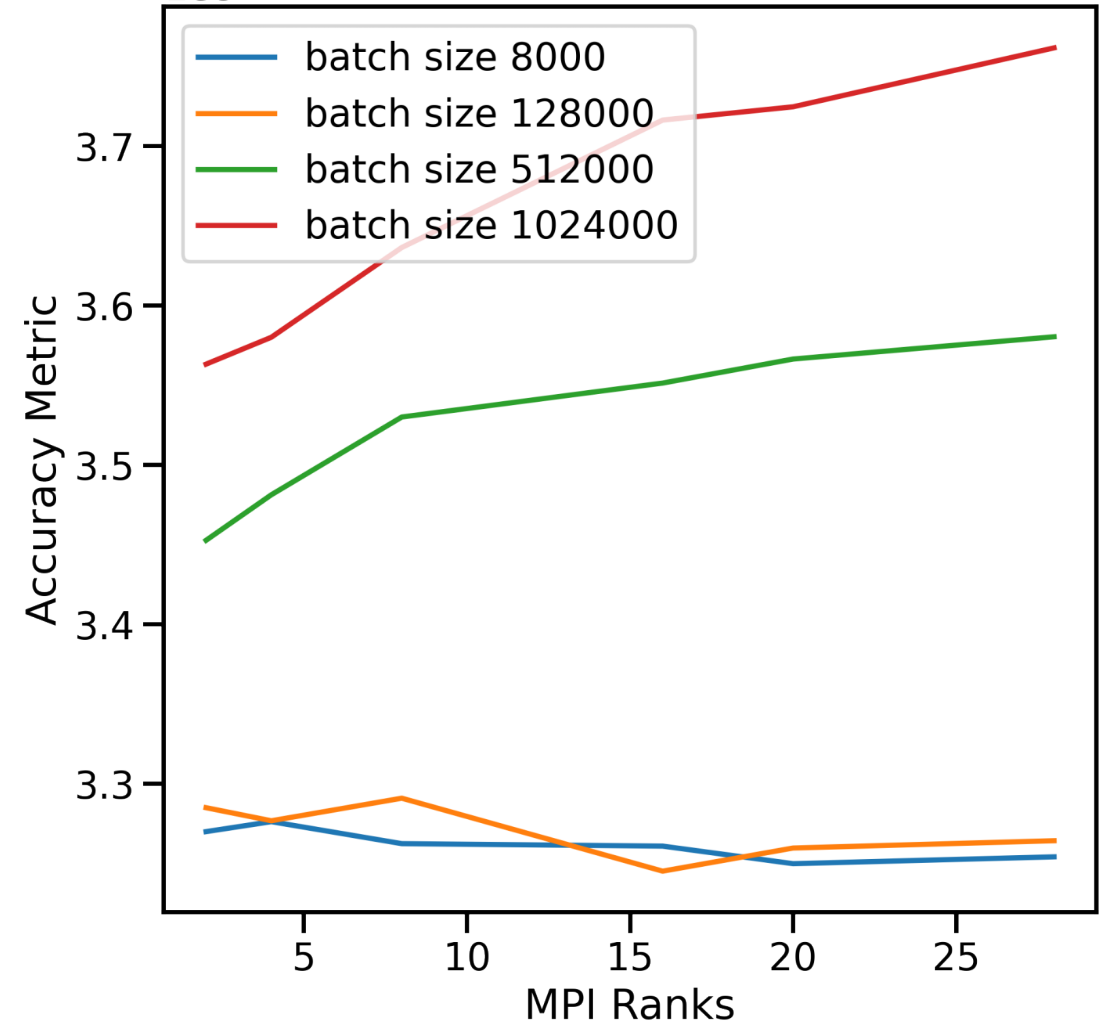
Strong Scaling Chicago Crime ([6186, 24, 77, 32] NNZ: 5330673)





Strong Scaling Vast 2015 ([165427, 11374, 2, 100, 89] NNZ: 26021945)





Reduced Synchronization Results*

*Convergence not dialed in yet

Conclusions and Next Steps

Conclusions

- Created scalable version of Genten using MPI collectives that will allow us to decompose tensors that were out of the reach of Genten before
- In the best case scenario (Chicago Crime) we can strong scale the solution to very large node counts

Next Steps

- Cuda aware MPI implementation
- Make sure MPI is configured for asynchronous progress for MPI_Iallreduce
- Create a parameter server version that only communicates changed gradient elements for comparison to the Allreduce code.
- Collect more thorough results for reduced synchronization code.

Possible Asynchrony With Algorithm Similar to Elastic AVG

Algorithm Similar to Elastic AVG SGD (in pseudocode):

```
iter = 1
first = false
C = F // Center starts out equal to F
MPI_Requests(ndims)

while(!converged):
      if iter % N == 0:
            if first:
                  first = false
            else:
                  WaitAll(MPI_Requests)
                  for d in ndims: C(d) += alpha * diff(d)

            for d in ndims:
                  diff(d) = F(d) - C(d)
                  F(d) -= alpha * diff
                  Iallreduce(diff(d), MPI_Request(d))
      ++iter
      // then same as before
```

To achieve asynchrony, replace the WaitAll with a TestAll and do more gradient updates while the Iallreduces are still completing.