# Sparsity-Aware Distributed Tensor Decomposition

Zheng Miao[1], Jon C. Calhoun (advisor)[1], Rong Ge (advisor)[1], Jiajia Li (advisor)[2]

[1]Clemson University [2]Pacific Northwest National Laboratory
{zmiao, jonccal, rge}@clemson.edu Jiajia.Li@pnnl.gov

## Problem

Tensor decomposition is widely used in data analytics, machine learning, quantum chemistry, and other domains. Sparse CPD, a popular low-rank tensor decomposition has been studied in distributed memory environment to obtain performance improvement. The state-of-the-art work [1] has three existing major challenges, especially for sparse tensors with irregular shape and nonzero distribution:

- *First, how to configure MPI processes for a sparse CPD as it is hard to measure its nonzero distribution and features with simple parameters?*
- *Second, computation cost is not always dominated by sparse tensors computation, but could also be other computations. How to balance the trade-off between them?*
- *Third, no execution overlap between computation and communication. Pursuing execution overlap needs to deeply investigate and split some of the stages.*

## Background

**CPD**: CANDECOMP/PARAFAC decomposition (CPD) factorizes a tensor into a sum of component rank-one tensors.

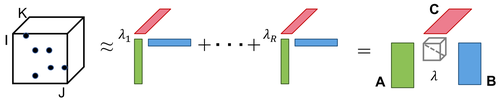


Figure 1:CPD for a third-order sparse tensor $X \times J \times K$.

**Data decomposition and distribution.** The medium-grained algorithm decomposes the tensor $\mathcal{X}$ and each factor matrix $\mathbf{A}^{(n)}$ in a non-overlapping fashion and distributes to processes.

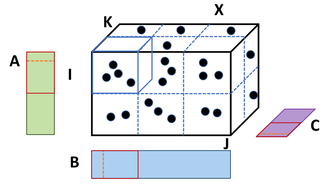


Figure 2:Tensor and matrix distribution over a $2 \times 3 \times 2$ process grid. The tensor is partitioned into $2 \times 3 \times 2$ subtensors, each mapped to a process. Each factor matrix is first partitioned by the layers affiliated with tensor partition and then evenly split among the corresponding process subgrid.

## Contributions

- Our work investigates the state-of-the-art implementation and observes five performance issues.
- We propose three strategies to solve the process configuration challenge, data distribution options and overlapping computation and communication.
- Our method scales well in the experiments when using up to 64 24-core nodes and obtains up to 2.3× performance improvement over distributed SPLATT library [1].

## Observations

**Observation 1**: *Both computation and communication have non-negligible costs and the dominance varies with tensors.*

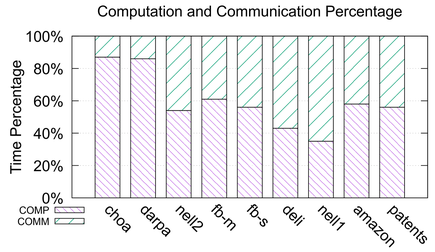


Figure 3:Computation and communication percentage for CPD using 64 nodes.

**Observation 2**: *Computation cost is not always dominated by sparse tensor computation, but also dense matrix computations.*

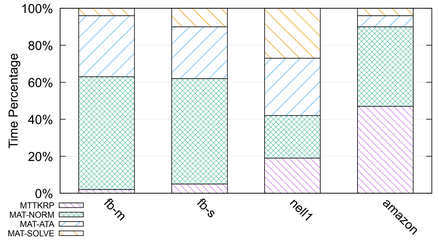


Figure 4:Time percentage of main computational kernels for CPD using 64 nodes.

**Observation 3**: *The communication bottleneck is mostly transforming matrices.*

**Observation 4**: *Different load imbalance factors influence computation and communication overhead.*

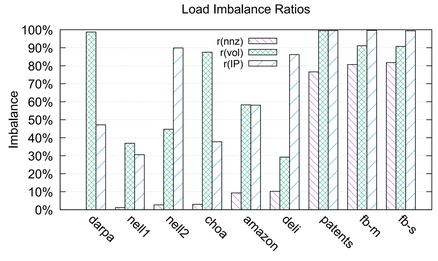


Figure 5:Load imbalance ratios ($r_{nnz}$, $r_{vol}$, and $r_{Ip}$) for sparse tensors on 64 nodes.

**Observation 5**: *There is no execution overlap between computation and communication.*

## Acknowledgements

This research was funded by the Laboratory Directed Research and Development program at PNNL under contract No. ND8577. Pacific Northwest National Laboratory (PNNL) is a multiprogram national laboratory operated for DOE by Battelle Memorial Institute under Contract DE-AC05-76RL01830.

## Optimizations

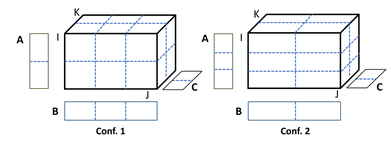


Figure 6:Two example grid configurations for 12 processes.

**Issue 1**: The performance difference could be up to 3.1× on 32 processes among different grids. The state-of-the-art work [1] developed a prediction algorithm by considering tensor dimension sizes but not the sparsity nor the matrix distribution.

**Solution 1**: We consider sparsity into the prediction by an accurate nonzero imbalance prediction and choosing from several close-to-optimal candidates.

**Algorithm 1** Our grid configuration strategy with $n_p = 1$.

**Require:** Number of processes $P$, tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$;
**Ensure:** Grid configuration $G$;
▷ Ordered from largest to smallest
$primes$ = getPrimes(P);
avgl = $(I_1 + I_2 + I_3)/3$
**for** $pr$ in $primes[0:-1]$ **do**
  Assign $pr$ to the largest mode $I_n$
  $I_n -=$ avgl
**end for**
Assign $primes[-1]$ to every mode, generate three grid candidates.
Predict $r_{nnz}$ of three candidates under different tensor distribution strategies
$G$ = candidates with min($r_{nnz}$)
**Return** $G$;

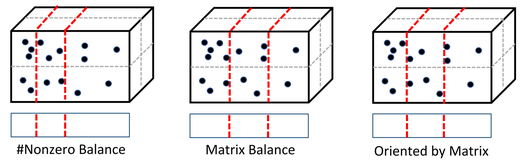


Figure 7:Tensor partitionings in the second mode with 12 processes as 2x3x2. All layer boundaries in gray are fixed and only boundaries in red are adjusted.

**Issue 2**: The first strategy targeting nonzero balance does not consider layer and matrix sizes, while the second strategy targeting matrix balance does not consider nonzero distribution.

**Solution 2**: We introduce a new strategy based on balanced matrix sizes and also consider nonzero distribution.

**Algorithm 2** Our proposed tensor distribution oriented by matrix size strategy in tensor mode $n$.

**Require:** Number of processes $p_n$ in mode-$n$, tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$;
**Ensure:** Layer configuration $L$;
**for** $i$ in $p_n$ **do**
  $I_{L_i} = I_n/p_n$; ▷ Initial layer size
**end for**
**if** Ordered adjustment **then**
  **for** $i$ in $p_n$ **do**
    $m_i$ = #nonzeros in layer $I_{L_i}$
    ▷ $c$ is constant parameter
    $I_{L_i}$ -= $(m_i - M)/p_n/S_i/c$;
  **end for**
**else if** Max to min adjustment **then**
  Sort $I_L$ by #nonzeros from Max to Min
  **repeat**
    $Max_L - = (Max_n - Min_n)/S$
    $Min_L + = (Max_n - Min_n)/S$
  **until** traverse all $I_L$
**end if**
**Return** $L$;

## Experiments & Results

We perform experiments on a Linux-based Intel Xeon CPU E5-2670 v3 multicore server platform with 24 physical cores distributed on two sockets, each with 2.3 GHz frequency.

Table 1:Description of sparse tensors.

| Tensors | Dimensions | #Nnzs | Density |
|---|---|---|---|
| *choa* | $712K \times 10K \times 767$ | 27M | $5.0 \times 10^{-6}$ |
| *darpa* | $22K \times 22K \times 24M$ | 28M | $2.4 \times 10^{-9}$ |
| *nell2* | $12K \times 9K \times 29K$ | 77M | $2.4 \times 10^{-5}$ |
| *fb-m* | $23M \times 23M \times 166$ | 100M | $1.1 \times 10^{-9}$ |
| *fb-s* | $39M \times 39M \times 532$ | 140M | $1.7 \times 10^{-10}$ |
| *deli* | $533K \times 17M \times 2.5M$ | 140M | $6.1 \times 10^{-12}$ |
| *nell1* | $2.9M \times 2.1M \times 25M$ | 144M | $9.1 \times 10^{-13}$ |
| *amazon* | $4.8M \times 1.8M \times 1.8M$ | 1742M | $1.1 \times 10^{-10}$ |
| *patents* | $46 \times 239K \times 239K$ | 3597M | $1.4 \times 10^{-3}$ |

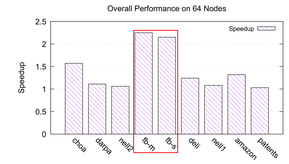


Figure 8:Overall performance speedup of our method for CPD using 64 nodes. Our approach obtains higher speedup for irregular tensors (*fb-m*, *fb-s*) whose $r_{nnz}$, $r_{vol}$, and $r_{Ip}$ are large as Figure 5 shows.

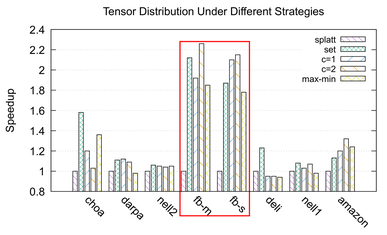


Figure 9:Tensor distribution under different strategies using 64 nodes.

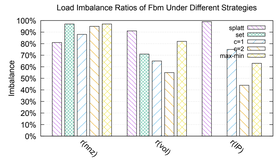


Figure 10:Load imbalance ratios of fbm under different strategies.

## Conclusions

Future , we will apply our optimizations to more sparse tensor formats and other tensor decomposition algorithms.

## References

[1] S. Smith and G. Karypis, "A medium-grained algorithm for distributed sparse tensor factorization," in *Parallel and Distributed Processing Symposium (IPDPS), 2016 IEEE International*, IEEE, 2016.