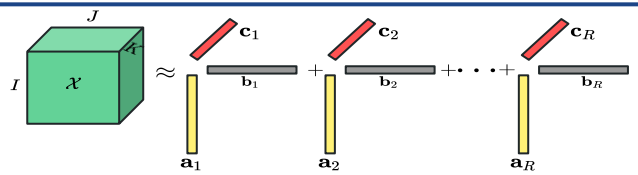
Algorithms and Models for Scaling Sparse Tensor and Matrix Factorizations

Nabil Abubaker
Advisor: Prof. Cevdet Aykanat
July 6th, 2022
Department of Computer Engineering, Bilkent University

Tensor and Matrix Factorizations



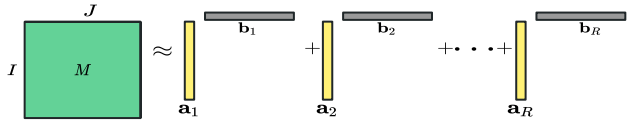
Data Modeling and Analysis

Factorizations

Principal Component Analysis
Singular Value Decomposition
Low-rank Matrix Factorization
Sparse Matrix Factorization

Matrix/Tensor Completion

Canonical Polyadic Decomposition
Tucker Decomposition
Tensor Train Decomposition



Methods

Clustering
Feature Extraction
Collaborative Filtering
Compression
Sparsification
Data Mining
Graph Embedding
.
.

Applications

Neuroscience
E.g., **fMRI analysis**
Computational Biology
E.g., **Microarray analysis**
E-commerce
E.g., **Recommender Systems**
Network Analysis
E.g., **Community Detection, Link prediction**
.
.

Why speedup Tensor and Matrix factorizations?

- **Speedup** the data-extraction and analysis process
  - Conduct experiments faster
  - Reach conclusions faster
  - Repeat failed or unsatisfactory results faster

} **Speedup scientific discovery** and **industrial advancements**

- **Less energy** consumption
- **Allow** larger frameworks/applications utilize the factorization as powerful and scalable kernel.

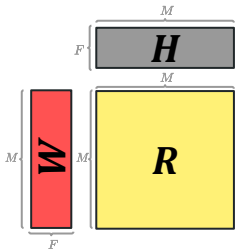
Overview

Part 1

Stratified SGD for Matrix Completion

High bandwidth overhead
→ We propose exchanging essential communication with P2P messages.

Leads to high message counts = high latency
→ A novel Hold and Combine algorithm to put an upper limit of $O(KlgK)$ on message counts per processor.

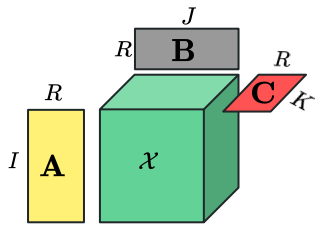


Parts 2 & 3

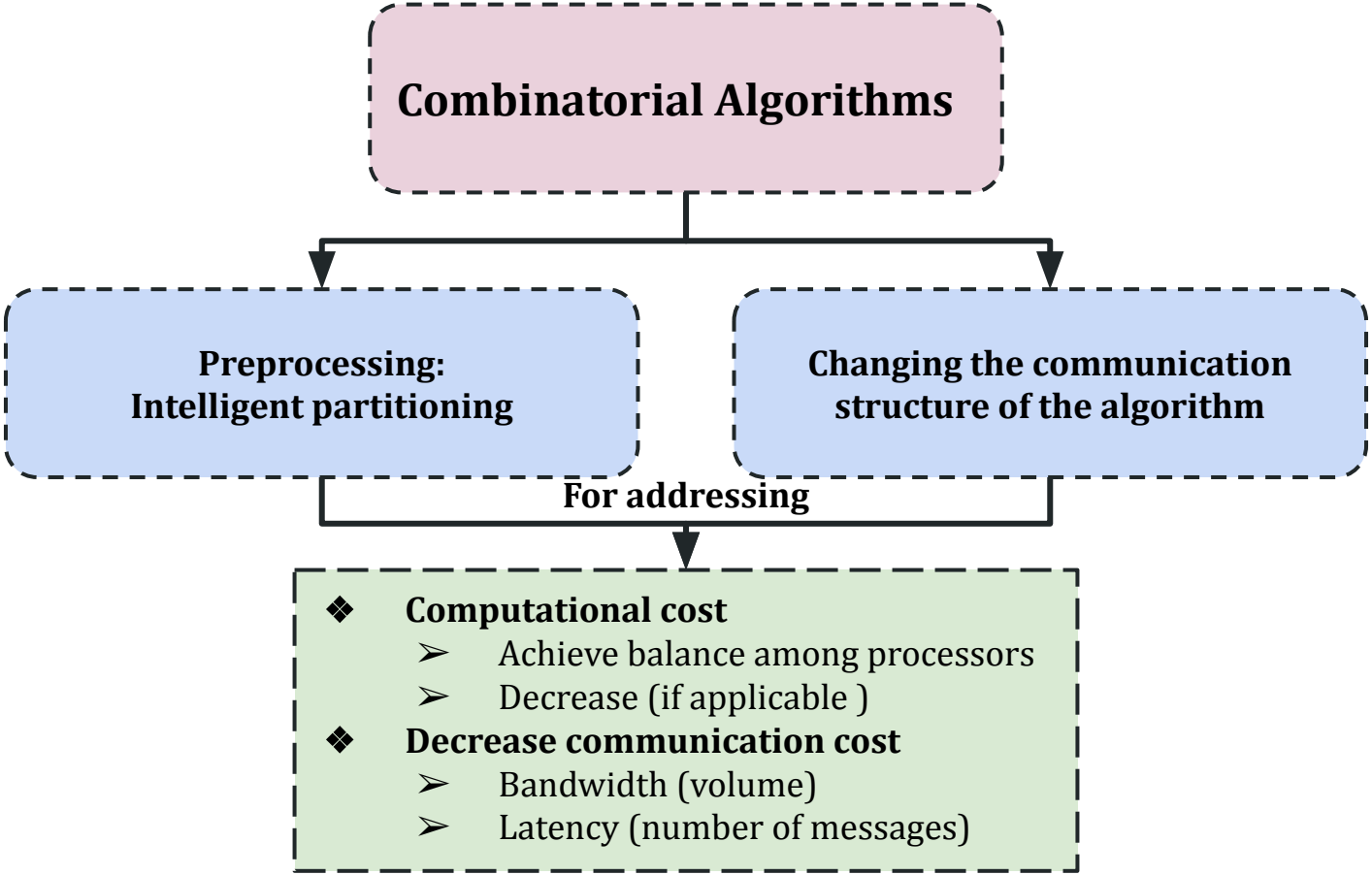
CPD-ALS for Tensor Decomposition

Load Balancing for MTTKRP is a hard when tensor is stored in CSF format
→ We propose two strategies to tackle the load balancing problem

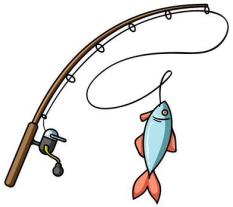
High message counts = high latency
→ We propose a framework to exchange up to lgK messages in lgK stages.



Overview


Combinatorial Algorithms
Preprocessing:
Intelligent partitioning
Changing the communication structure of the algorithm
For addressing
❖ Computational cost
➢ Achieve balance among processors
➢ Decrease (if applicable)
❖ Decrease communication cost
➢ Bandwidth (volume)
➢ Latency (number of messages)

Background: Hypergraph Partitioning



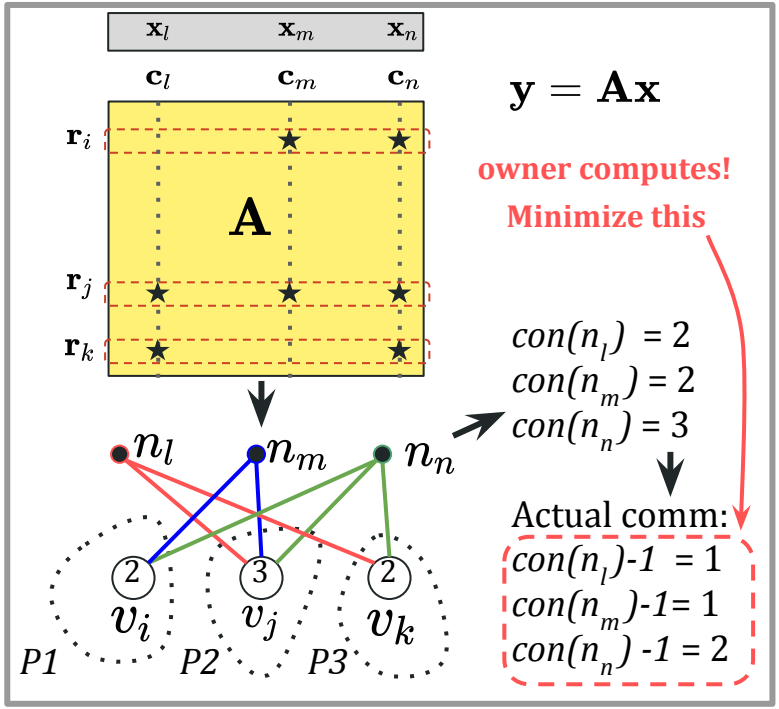
A **hypergraph** consists of a set of vertices and nets.

Nets (also called **hyperedges**) → generalization of edges in Graphs
Edges connect **two vertices**
Nets connect **two or more vertices**

***K*-way Hypergraph partitioning:**

- Split the vertices into *K* **non-intersecting** sets
- Optimize an objective function → defined over ***cutsize***
- Common objectives:
  - Minimize nets on the cut (count or cost)
  - Minimize the number of parts that cut-nets connect → *commonly used to minimize volume of communication in parallel algorithms.*

Example: Column-net model for rowwise parallel SpMV

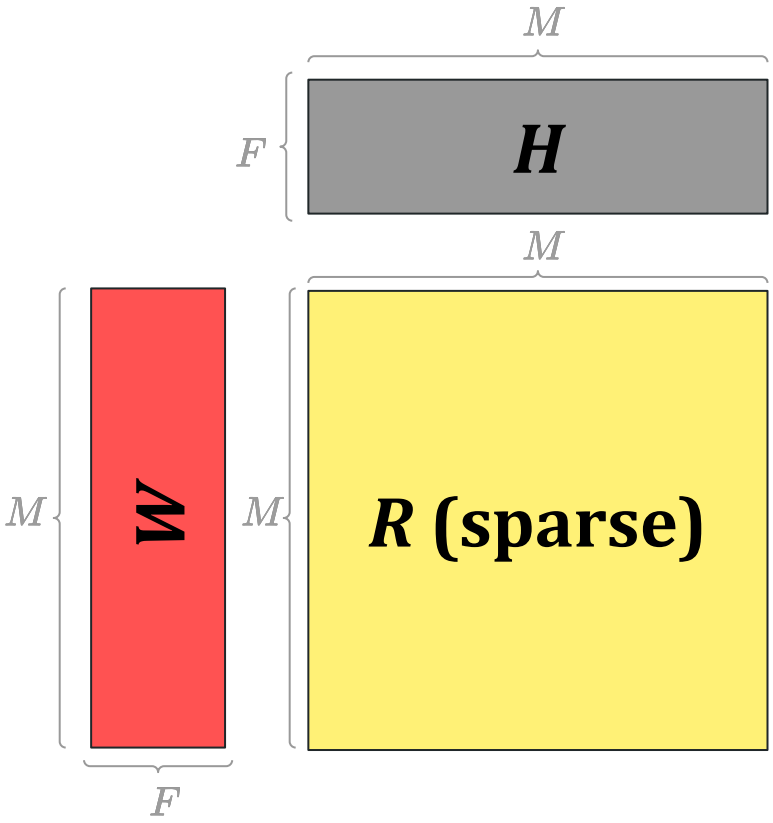


Part 1: Scaling Stratified SGD for Distributed Matrix Completion

Submitted to IEEE Transactions on Knowledge and Data Engineering as:

Nabil Abubaker, M. Ozan Karsavuran and Cevdet Aykanat, "**Scaling Stratified Stochastic Gradient Descent for Distributed Matrix Completion**"

SGD for Matrix Completion



Goal:

Find low-rank approximation $\boldsymbol{R} \cong \boldsymbol{W}\boldsymbol{H}^T$

A missing entry $\boldsymbol{r}_{ij}$ in $\boldsymbol{R}$ can be approximated (completed) by $\boldsymbol{r}_{ij} = \boldsymbol{w}_i\boldsymbol{h}_j^T$

How?

Using Stochastic Gradient Descent to find $\boldsymbol{W}$ and $\boldsymbol{H}$ that minimize:

$$Loss = \sum (\mathbf{r}_{ij} - \mathbf{w}_i\mathbf{h}_j^T)^2$$

By:

Step size

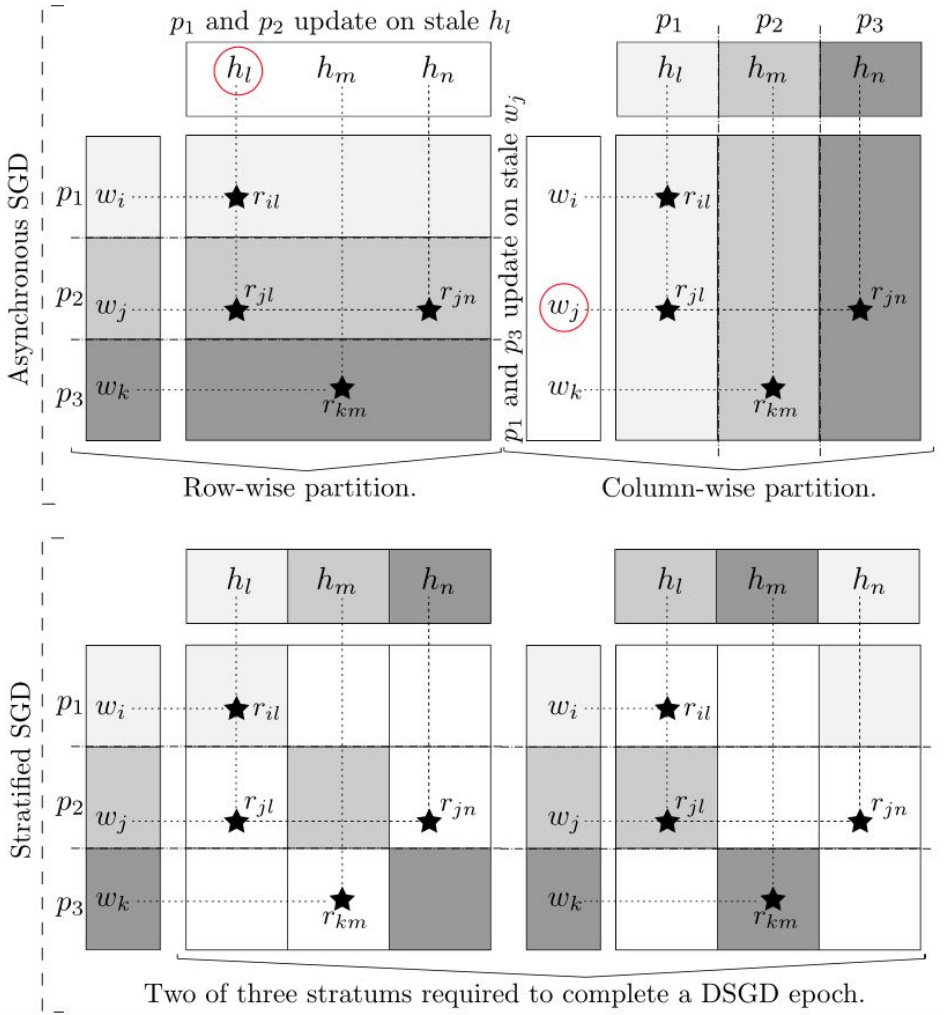
Regularization factor

$$\mathbf{w}_i = \mathbf{w}_i + \epsilon((\mathbf{r}_{ij} - \mathbf{w}_i\mathbf{h}_j^T)\mathbf{h}_j + \gamma\,\mathbf{w}_i)$$

$$\mathbf{h}_j = \mathbf{h}_j + \epsilon((\mathbf{r}_{ij} - \mathbf{w}_i\mathbf{h}_j^T)\mathbf{w}_i + \gamma\,\mathbf{h}_j)$$

Parallel SGD

- **Asynchronous**: Allows staleness
- **Stratified (SSGD)**: Doesn't allow staleness
  - Serializable
  - Better convergence
  - Well-studied behaviour



Stratified SGD proposed in:
R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 69–77, 2011.

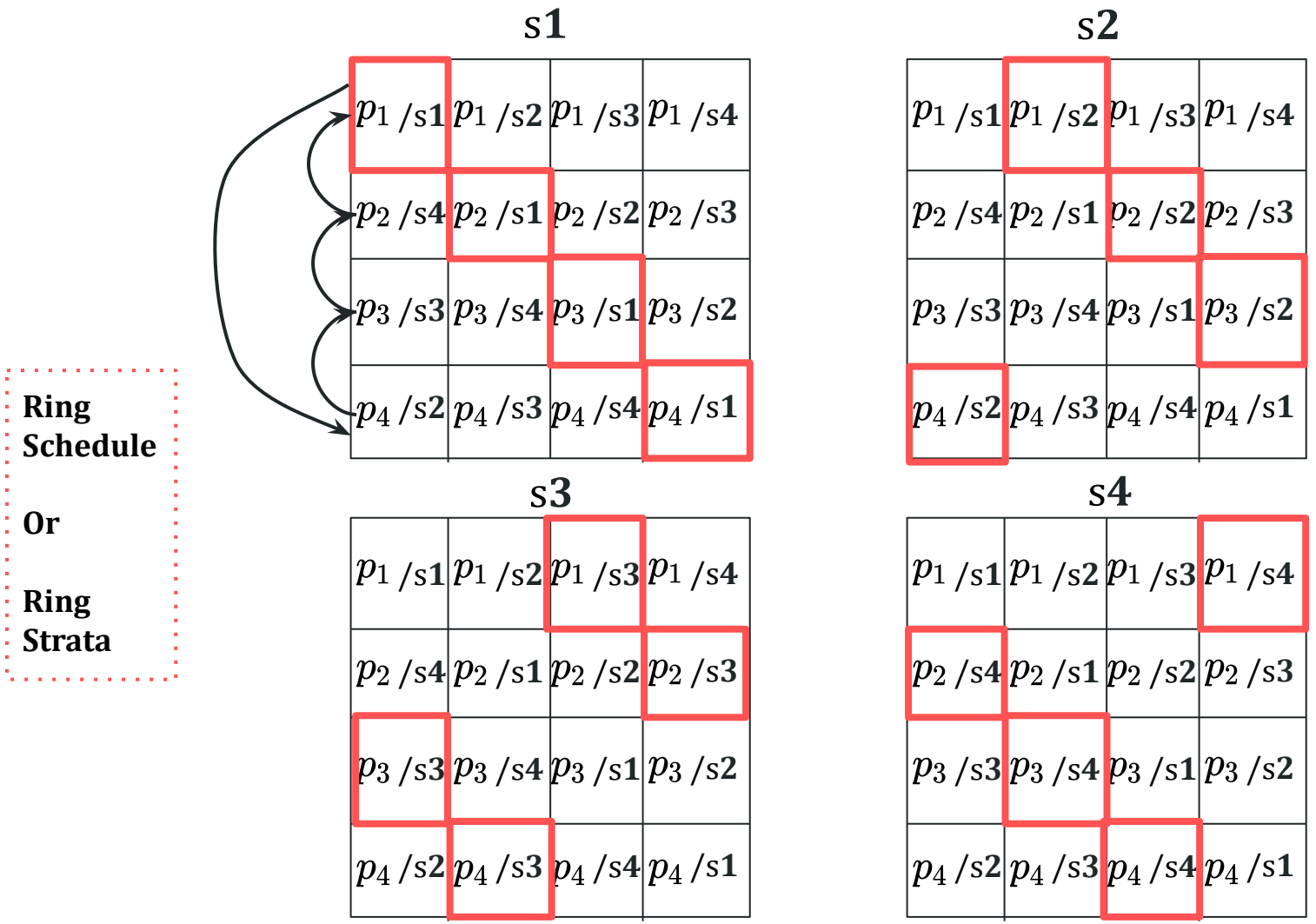
Problem with distributed SSGD:

Existing implementations communicate **blocks** of latent factors:

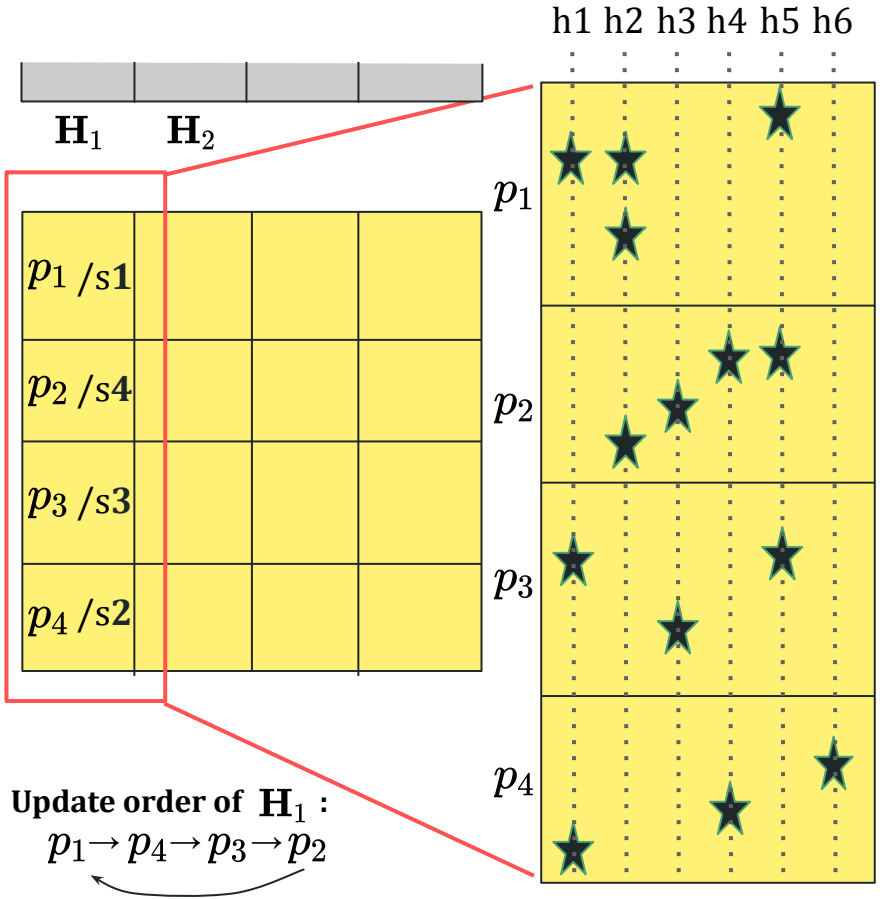
- p_1 updates block $\mathbf{H}_j$
- p_1 sends **all rows in** $\mathbf{H}_j$ to the processor that updates it next
- Data is **sparse** → extra **unnecessary** data movement

Proposal:

Communicate only latent factors that are **essential** for the correctness of the SGD algorithm **using P2P messages**


s1
s2
s3
s4
Ring Schedule
Or
Ring Strata

Finding essential communication

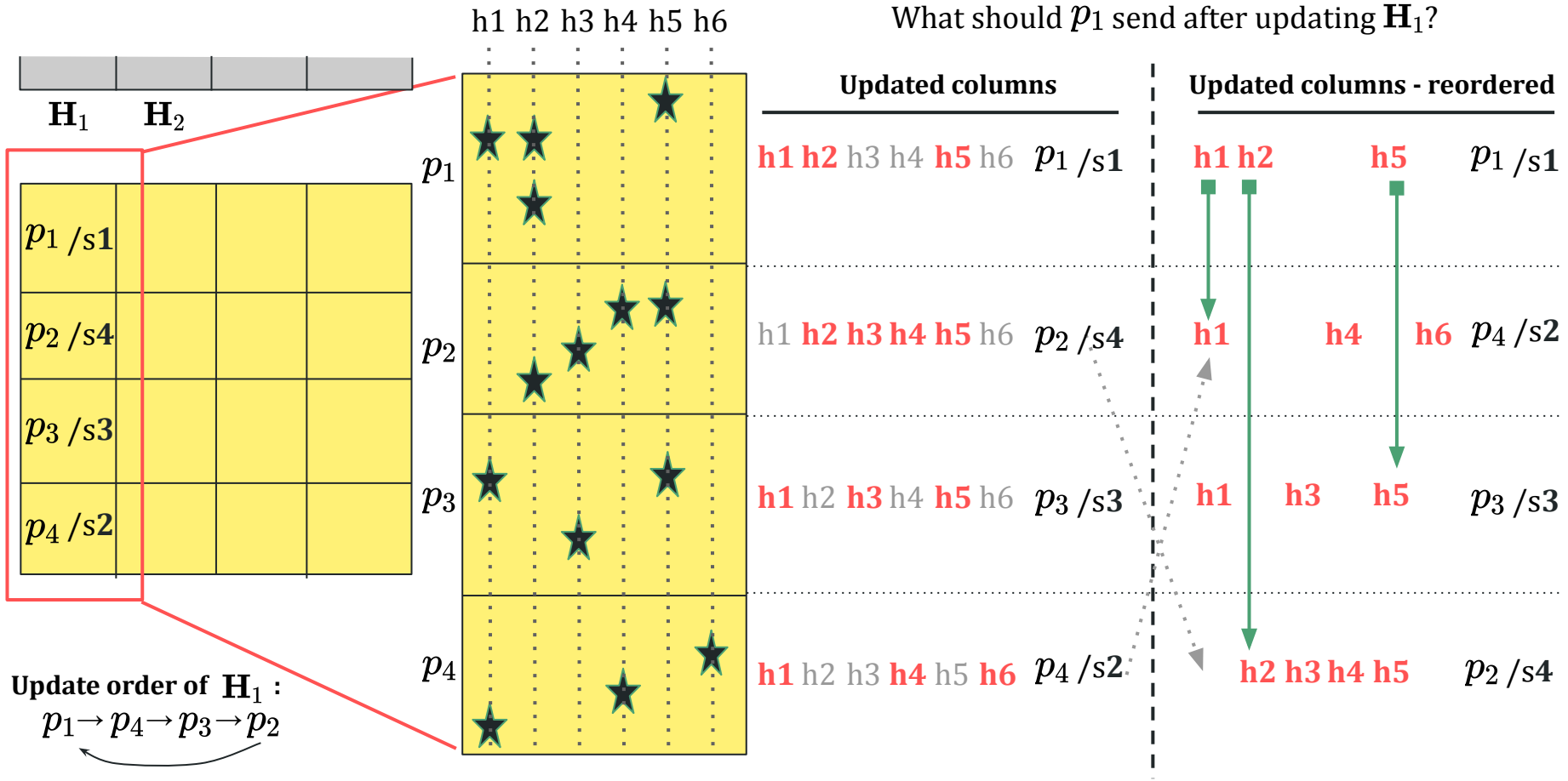


$\mathbf{h_i}$ is sent from p_x to p_y if :

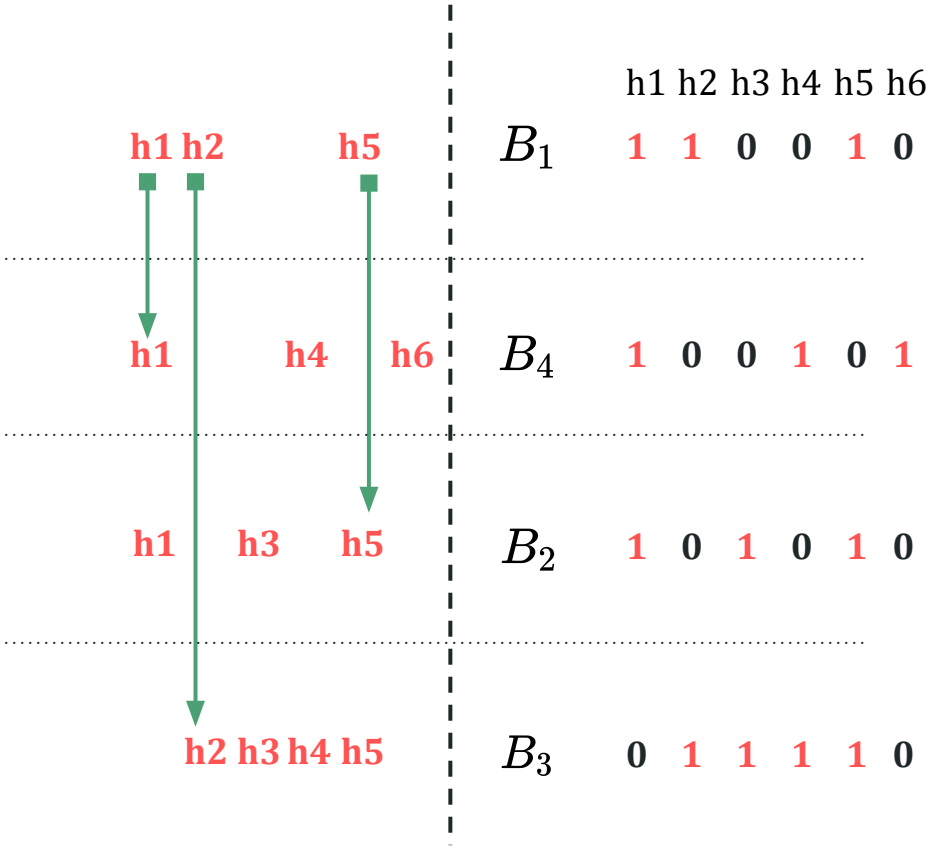
→ both p_x and p_y update $\mathbf{h_i}$ and
→ **no** processor **in between** does.

According to the ring order

Finding essential communication



Efficiently finding essential communication



Updated by p_x **AND** p_y

but **not**

any processor in between

$$B_x \wedge B_y \oplus (B_x \wedge B_y \wedge (B_{x+1} \vee B_{x+2} \cdots B_{y-1}))$$

Built up incrementally

So far:

- Finding essential communication can be used to send P2P messages.
- **Invaluable** for reducing the volume of communication

- **Problem:** Number of messages significantly increase compared to block-wise communication:
  - **With block-wise:** each processor sends **1** msg per sub-epoch; **total of *K*** messages per processor.
  - **With P2P,** each processor sends up to ***K-1*** msgs per sub-epoch; **total of *O(K^2)*** messages per processor.

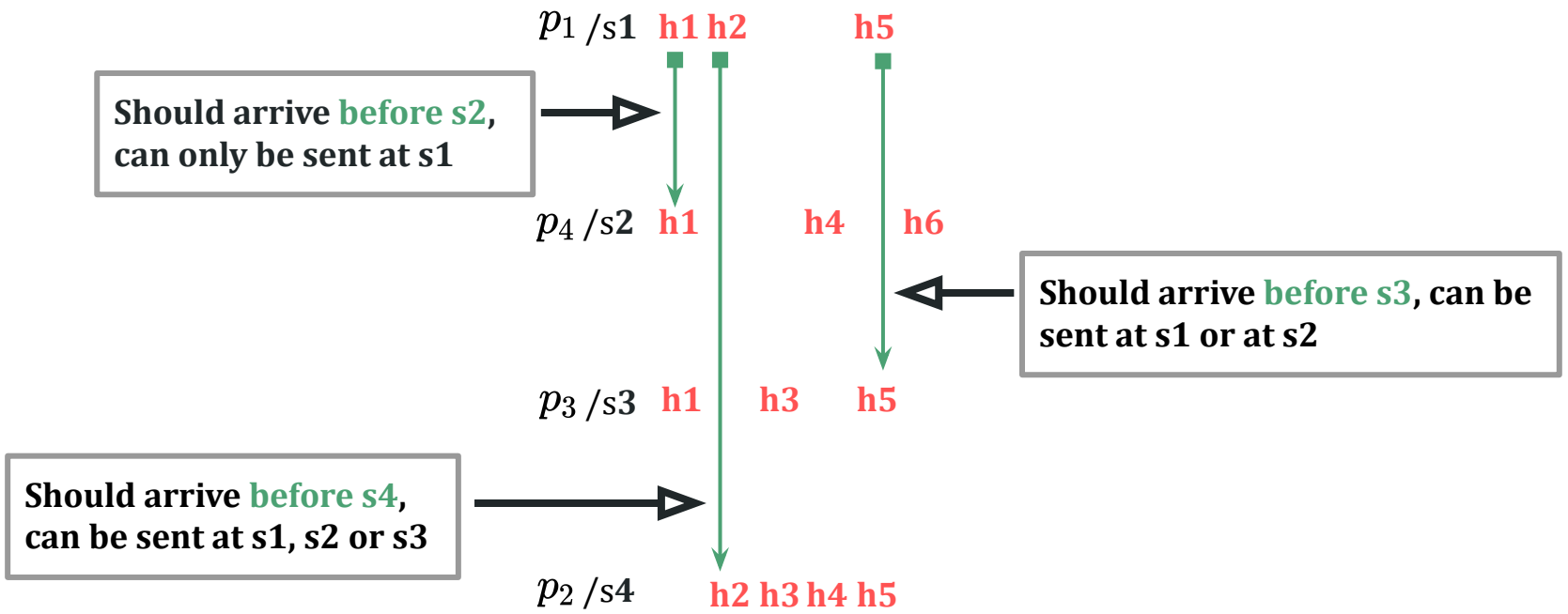
Research Question

Is it possible to exchange the same essential communication with message count asymptotically less than ***O(K^2)***

Key Observation

Latent factors send via P2P messages are not always immediately needed in the next sub-epoch.

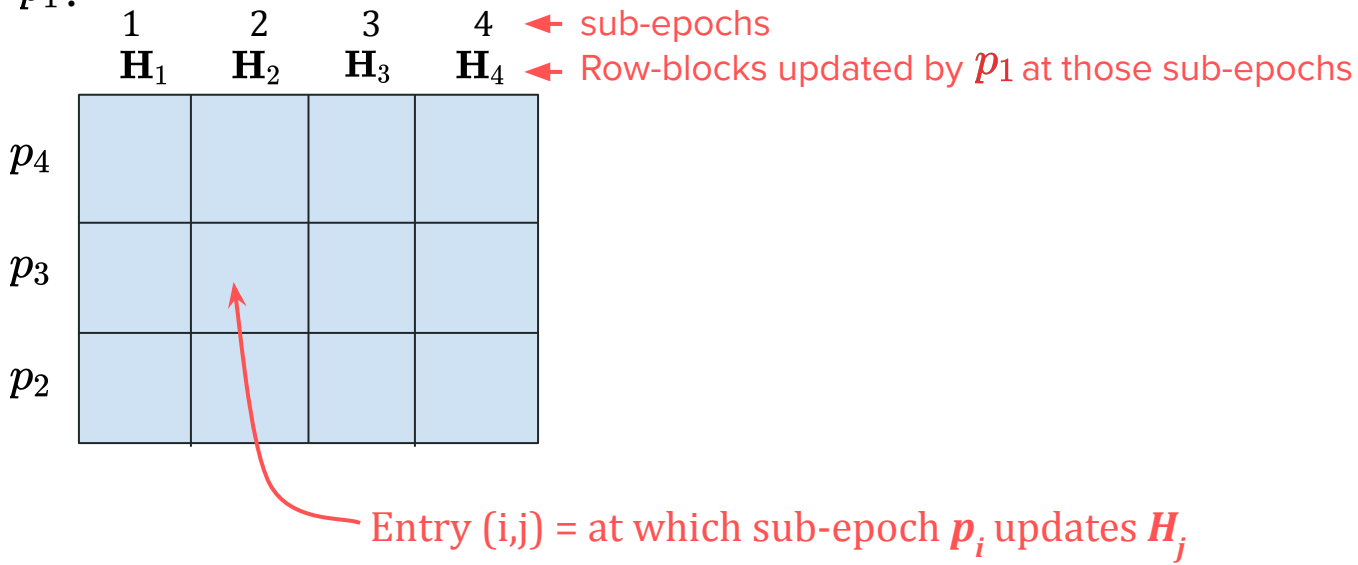
Key Observation: A closer look



The Hold & Combine Algorithm

Key idea: **Hold latent factors to be sent to the same processor at different sub-epochs and combine/send them in one message**

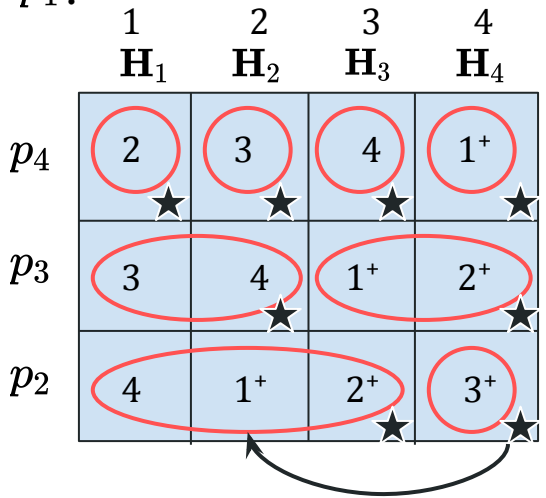
Schedule of p_1 :



The Hold & Combine Algorithm

Key idea: **Hold latent factors to be sent to the same processor at different sub-epochs and combine/send them in one message**

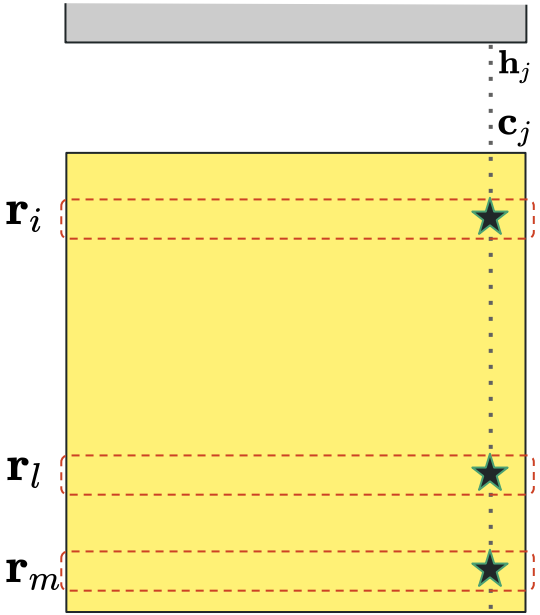
Schedule of p_1 **:**



$$\sum_{i=1}^{\frac{K}{2}} 2 + \sum_{i=1}^{\frac{K-1}{2}} \frac{K}{i}$$

No H&C: up to 12 messages (N * N-1)
With H&C: up to 8 messages (N * lgN)

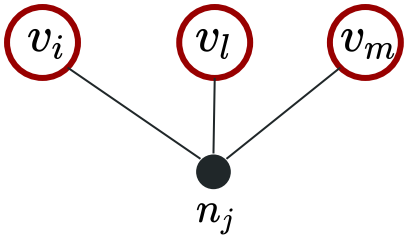
An HP-based method for P2P Communication

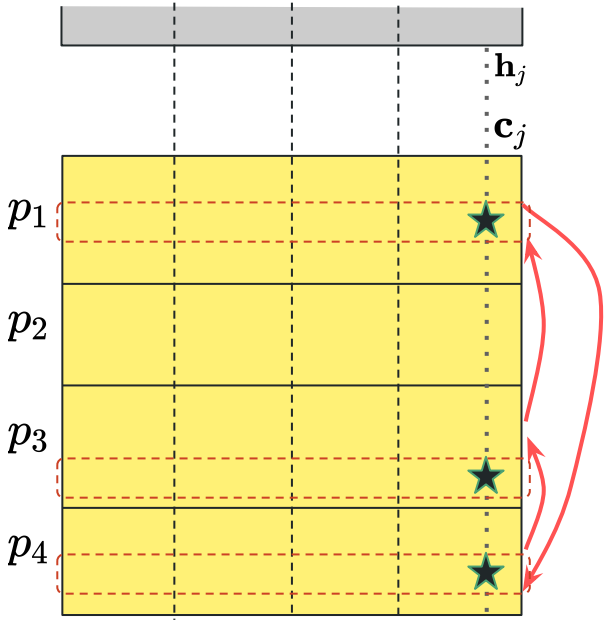


A **row** is represented by a **vertex**

A **column** is represented by a **net**

A cut-net n_j with connectivity $con(n_j)$ means the corresponding h_j will be communicated $con(n_j)$ times during an SGD epoch





Topologically similar to the **column-net model,** but **different *cutize*** metric: minimize ***con*** instead of ***con*-1**

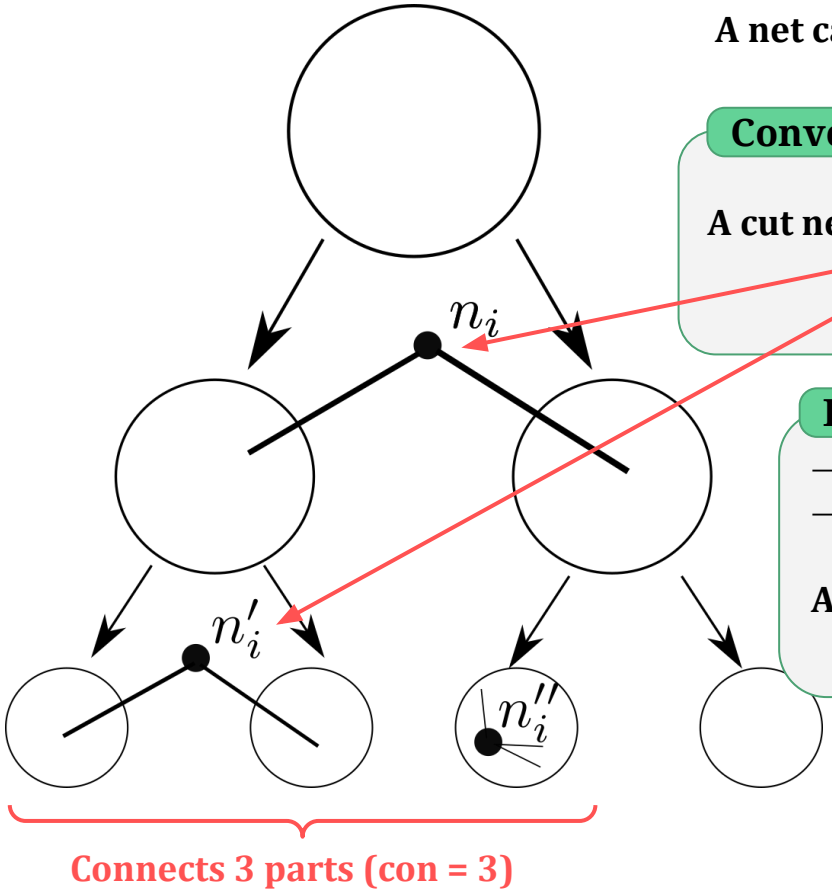
Problem:

The ***con*** metric, also known as the sum of external degrees (SOED), is supported by **only a few** hypergraph partitioning tools.

Proposal:

A **recursive bipartitioning (RB)** framework to encode the SOED metric that can utilize **any** hypergraph partitioning tool/algorithm.

An RB framework for encoding the SOED metric



A net can be cut at most *K*-1 times to connect *K* parts

Conventional net splitting:

A cut net contributes its cost *con*-1 times to the cutsize:

$$\text{cost}(n_i) * (\textit{con}(n_i)\text{-}1)$$

Proposal:

→ Initially assign each net with twice its cost
→ The first time it becomes cut, reset the cost to original

A cut net contributes its cost *con* times to the cutsize:

$$\text{cost}(n_i) + (\text{cost}(n_i) * (\textit{con}(n_i)\text{-}1)) = \text{cost}(n_i) * \textit{con}(n_i)$$

Experiments and Key Results

6 real-world sparse **rating** matrices. 5M < nnz < 475M nonzeros

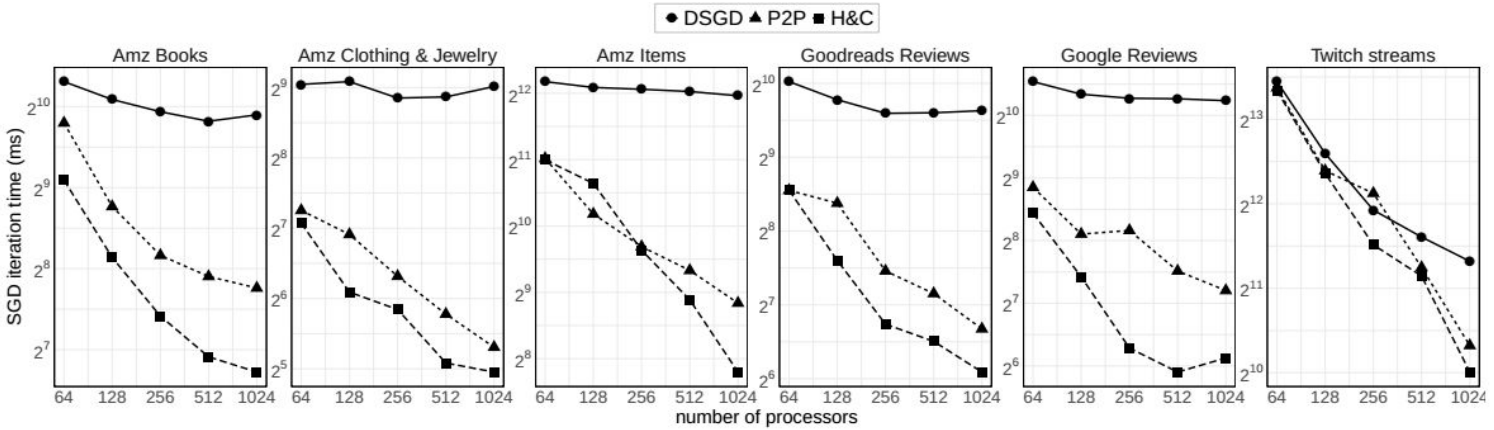
In terms of bandwidth:

- Using P2P **reduces** total volume by **10x - 120x** (P2P has the same volume with or w/o H&C)
- **HP-based** distribution **reduces** P2P volume by **1.4x - 5x** compared to random distribution

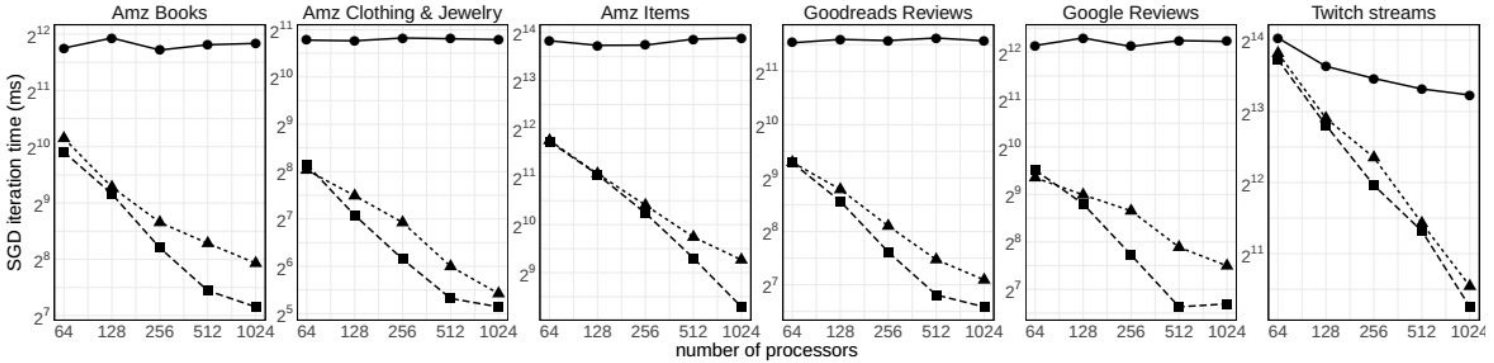
In terms of latency:

- Using random-based P2P **increases** total messages by **3.5x - 57x** (and **1.8x - 25x** for **HP-based**)
- Using random-based **H&C increases** total messages by **3x - 8x** (and **1.4x - 5x** for **HP-based**)

Experiments and Key Results - Cont'd



(a) $F = 16$

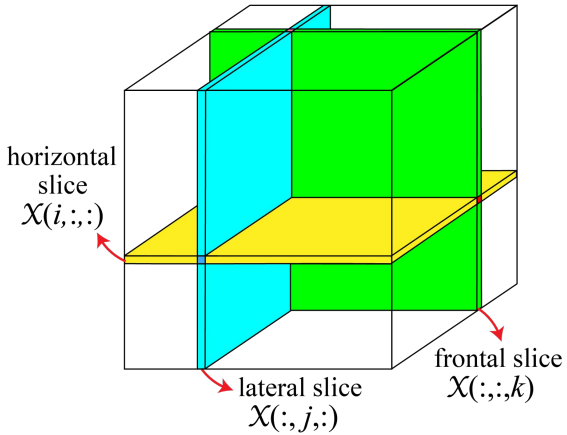


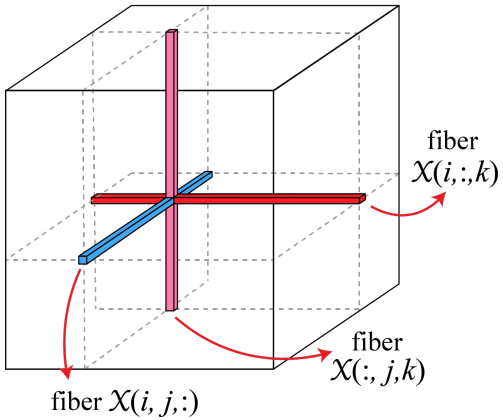
(b) $F = 64$

Parts 2 & 3: Scaling CPD-ALS for Distributed Tensor Decomposition

Background: Tensors

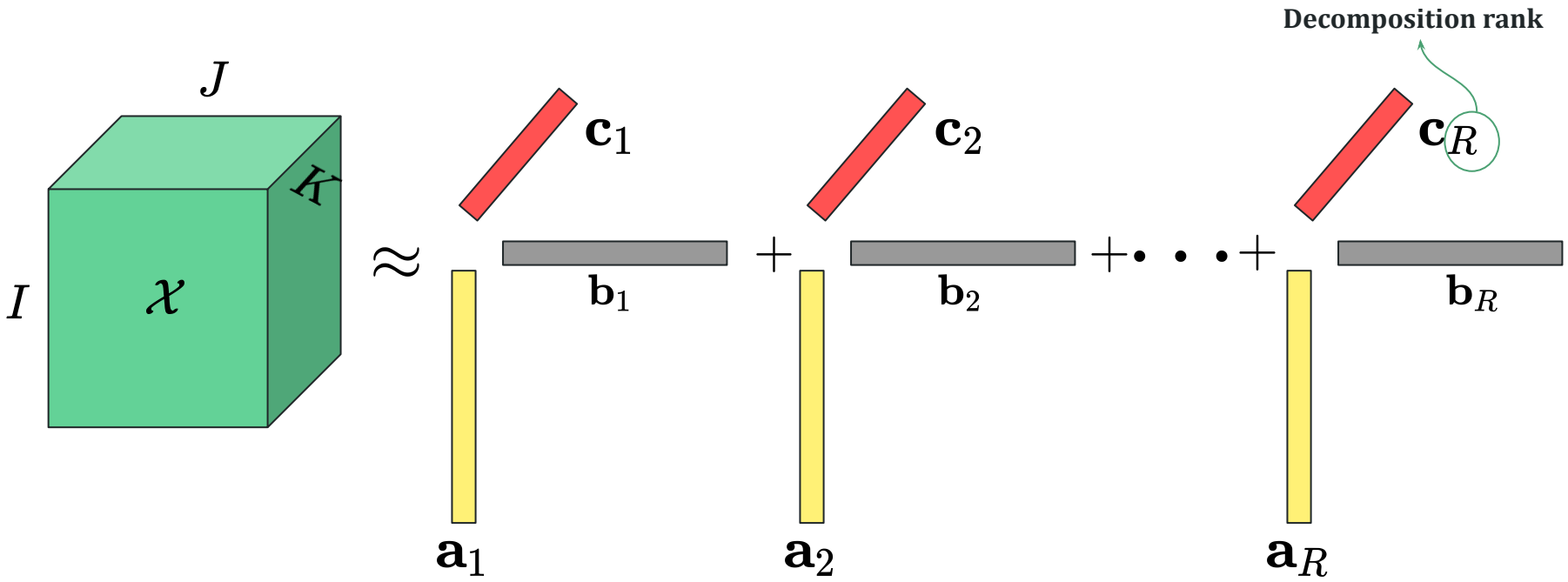
- Multi-dimensional arrays. (N-way/N-mode tensors)
- Tenors naturally represent multi-way data.

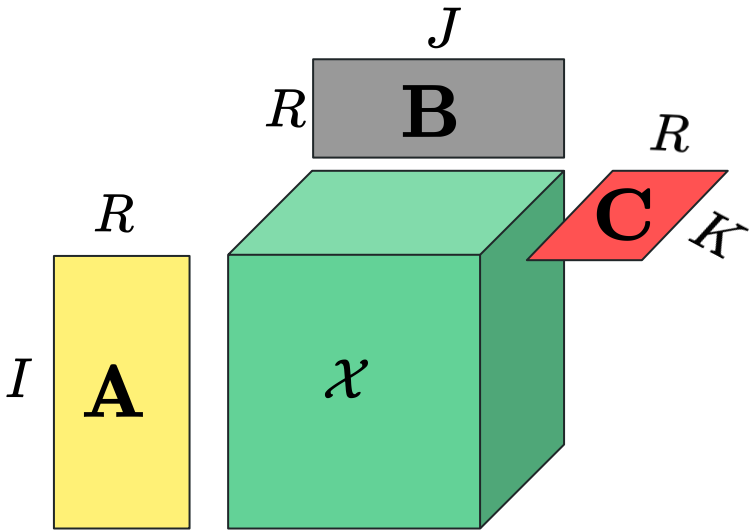




Background: The Canonical Polyadic (CP) Decomposition

aka CANDECOMP/PARAFAC

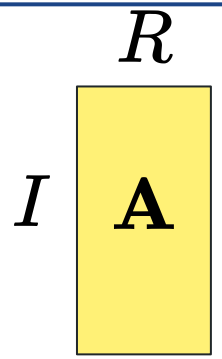



J
R
B
R
C
K
R
I
A
$\mathcal{X}$

Background: The CPD-ALS algorithm (per-mode)

$$\mathbf{A} = \underbrace{\mathbf{X}_{(1)}}_{I \times JK}\underbrace{(\mathbf{C} \odot \mathbf{B})}_{JK \times R}\underbrace{(\mathbf{C}^\top\mathbf{C} * \mathbf{B}^\top\mathbf{B})^{-1}}_{R \times R}$$

Matricized Tensor Times Khatri-Rao Product (MTTKRP)



Column Normalization

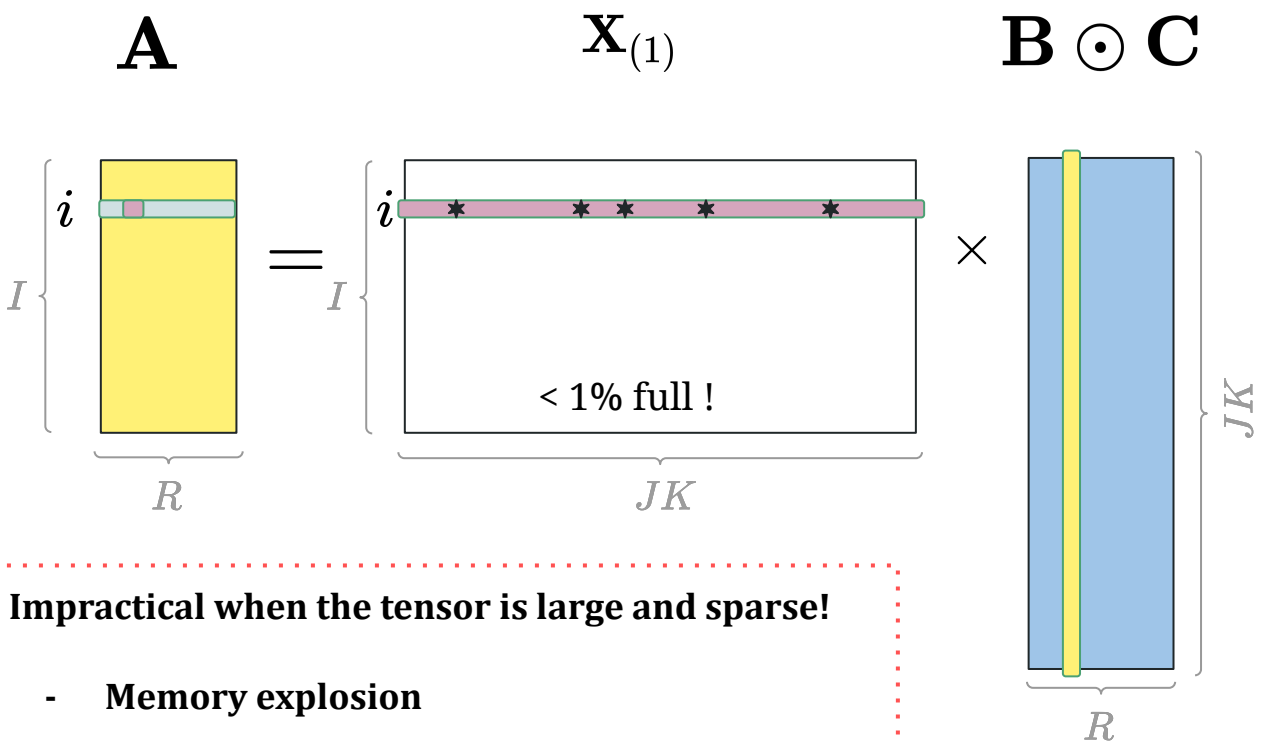
$$\mathbf{a}_r = \mathbf{a}_r / \lambda_r \;\; s.t. \;\; \lambda_r = \|\mathbf{a}_r\| \;\; \text{for } r = 1, \ldots, R$$

Part 2: True Load Balancing for MTTKRP

Published as:

Nabil Abubaker, Seher Acer and Cevdet Aykanat, "**True Load Balancing for Matricized Tensor Times Khatri-Rao Product**", **IEEE Transactions on Parallel and Distributed Systems**, 32 (8), 1974-1986, 2021.

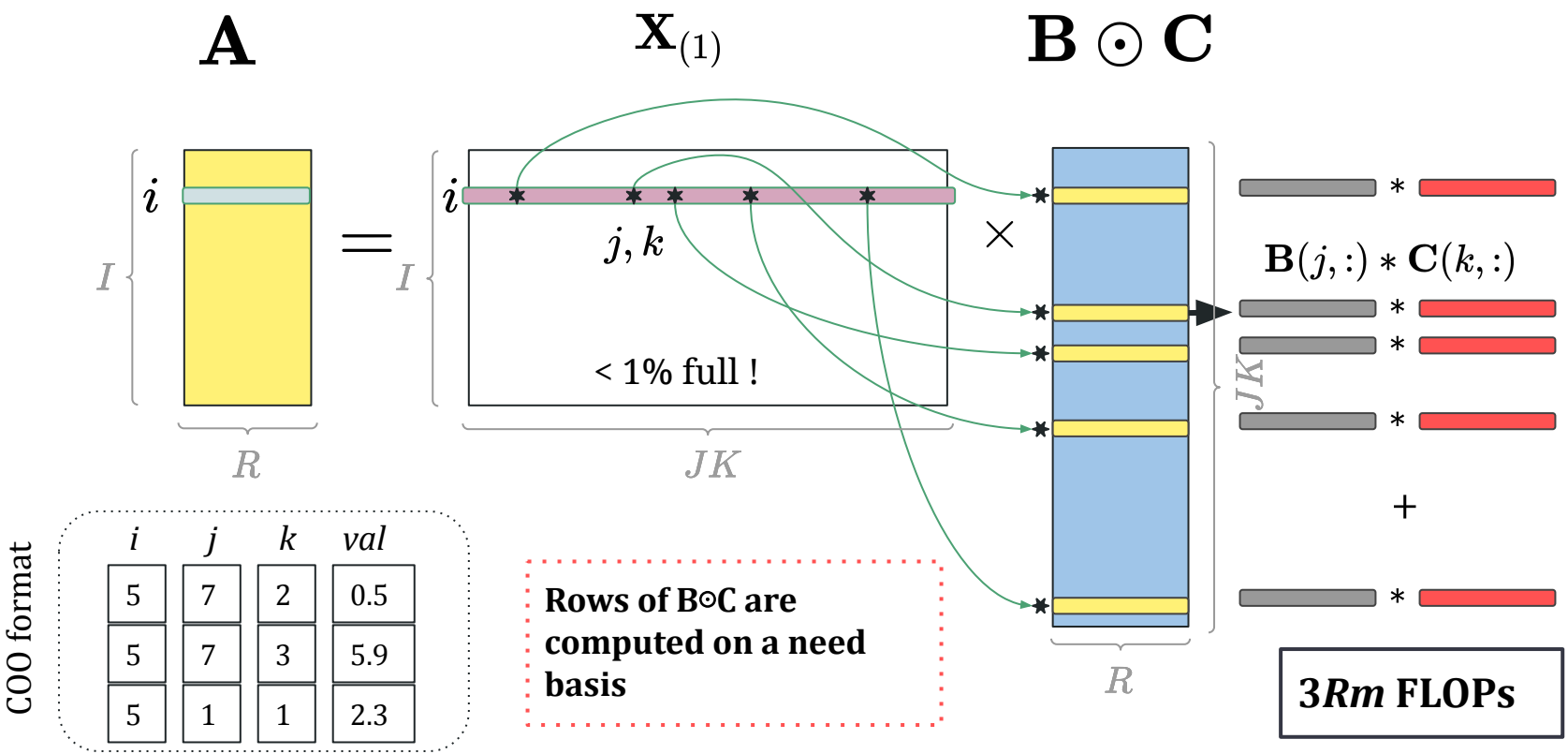
Computing the MTTKRP



Impractical when the tensor is large and sparse!

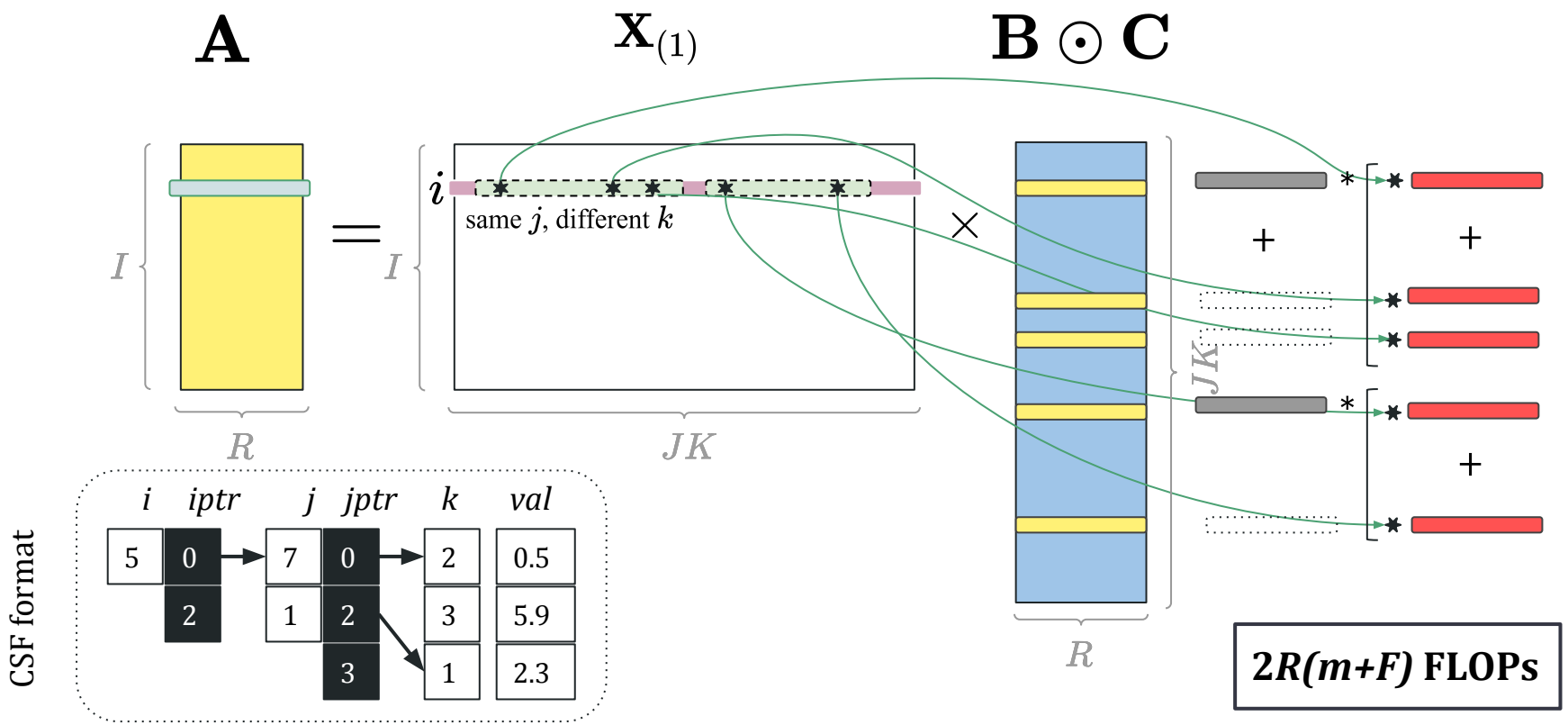
- **Memory explosion**
- **Not all rows of B⊙C are used**

COO-oriented MTTKRP



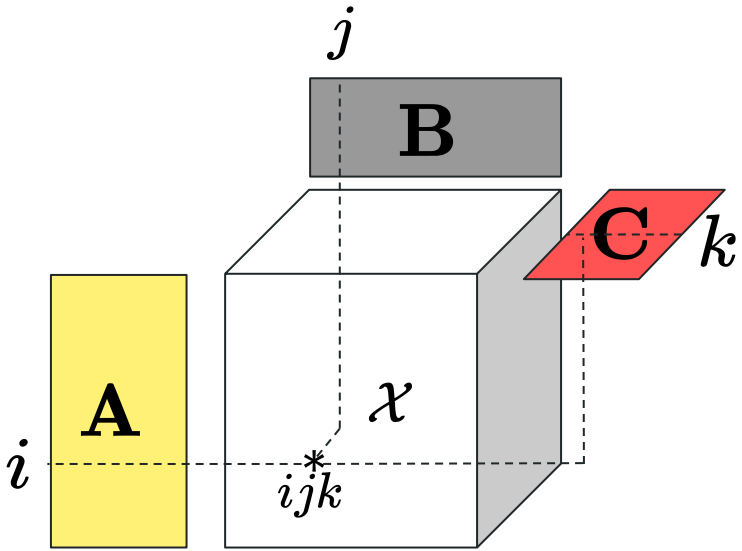
Bader, Brett W., and Tamara G. Kolda. "Efficient MATLAB computations with sparse and factored tensors." SIAM Journal on Scientific Computing 30.1 (2008): 205-231.

CSF-oriented MTTKRP



S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, SPLATT: Efficient and parallel sparse tensor-matrix multiplication, in IPDPS 2015: IEEE International Parallel and Distributed Processing Symposium, IEEE, 2015, pp. 61--7

COO-Based Fine-grain task definition of MTTKRP



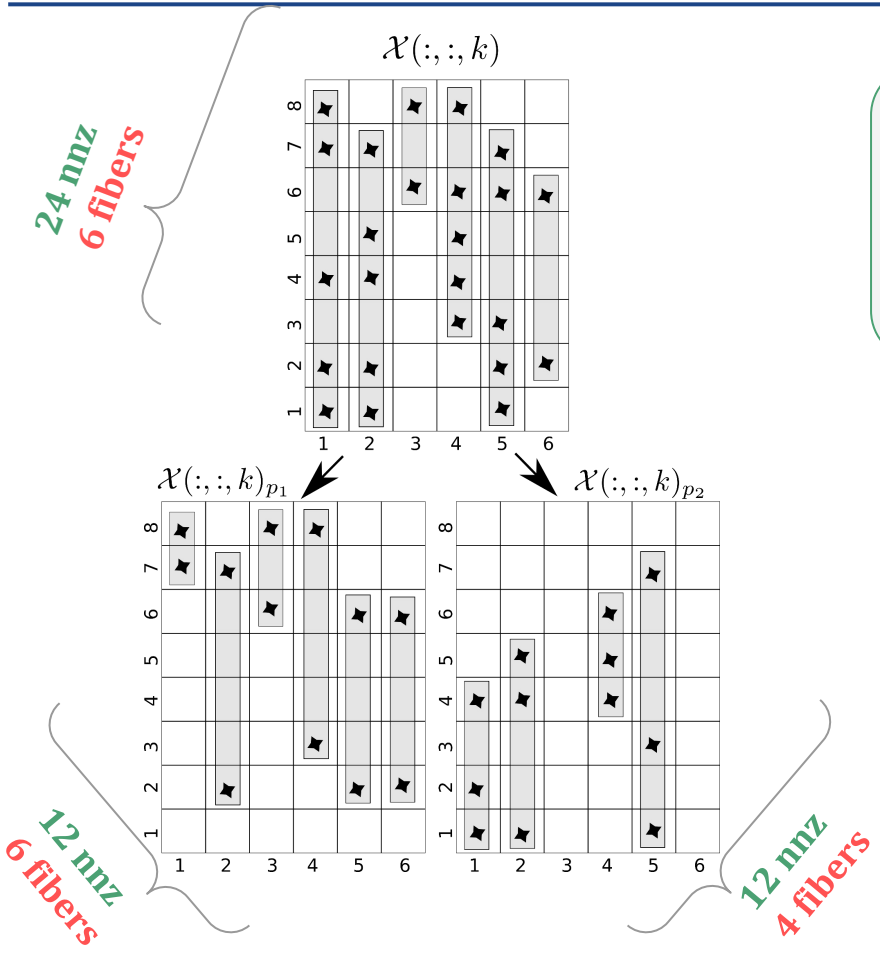
A nonzero ***ijk*** is associated with the partial computation of :

***i*^th^** row of **A**

***j*^th^** row of **B**

***k*^th^** row of **C**

The load balancing problem when using CSF-oriented MTTKRP



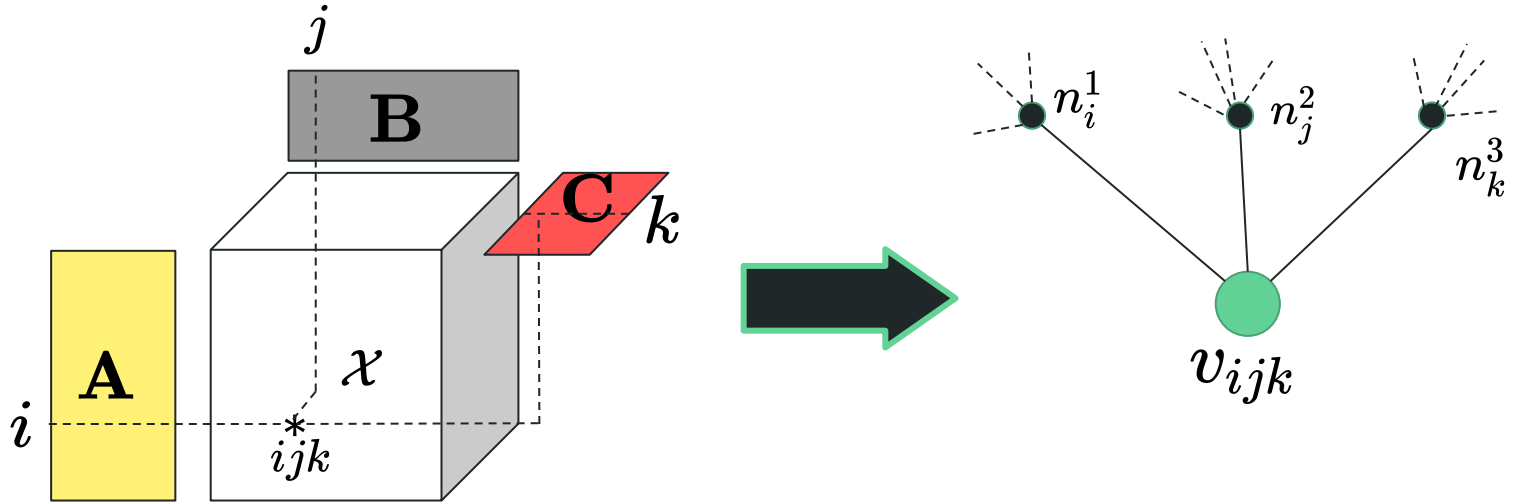
COO-oriented MTTKRP:

- **Same** FLOPs: 3x12R each
- Total FLOPs equal to pre-distribution: 72 total FLOPs before and after (**no increase**)

CSF-oriented MTTKRP:

- **Different** FLOPs:
  - p1: 2R(12 + 6)
  - p2: 2R(12 + 4)
- Total FLOPs **increased** compared to pre-partition: 60 before vs 68 after
- Still **better than COO**, but the increase need to be controlled.

Fine-grain hypergraph model for the COO-Based MTTKRP



n_i^1 connects the vertices (atomic tasks) that contribute to the computation of row $\mathbf{A}(i,:)$

> Encodes reduce and expand communications

Partitioning the hypergraph:

n_i^1 is cut $\rightarrow$ incurs $2R(con(n_i^1)-1)$ words of communication volume

O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1-11, doi: 10.1145/2807591.2807624.

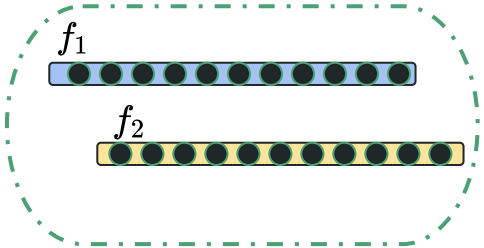
How to tackle the load balancing problem of MTTKRP

Two Key ingredients:

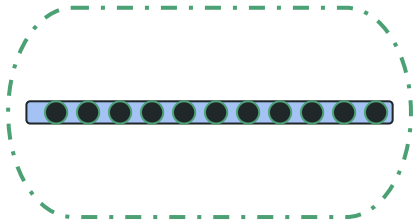
- A **weighting scheme** that considers the cost of fibers
- A **partitioning objective** of reducing the increase in total FLOPs

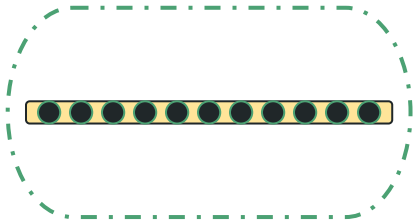
The Inverse-Fiber-Size (IFS) weighting scheme

$$w(v) = 2R(1 + \frac{1}{nnz(f_1)})$$



$$w(v) = 2R(1 + \frac{1}{nnz(f_2)})$$



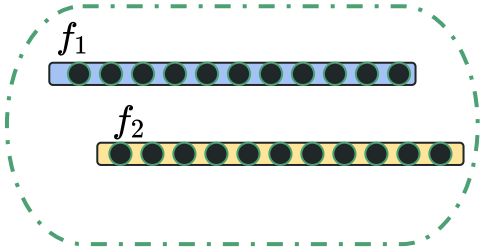


part weight $= 2R(nnz(f_1) + 1)$ ✅

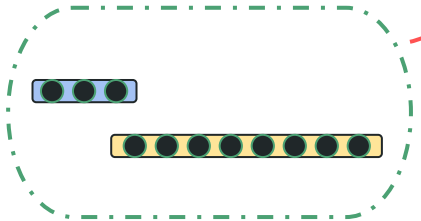
part weight $= 2R(nnz(f_2) + 1)$ ✅

The Inverse-Fiber-Size (IFS) weighting scheme

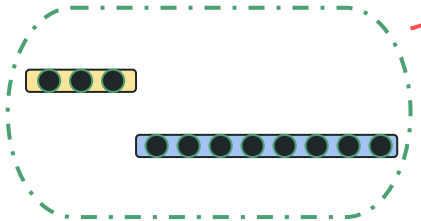
$$w(v) = 2R(1 + \frac{1}{nnz(f_1)})$$



$$w(v) = 2R(1 + \frac{1}{nnz(f_2)})$$



Re-compute to correct the weights



Re-compute to correct the weights

part weight $= 2R(3 + \frac{3}{11} + 8 + \frac{8}{11})$ ❌

part weight $= 2R(3 + \frac{3}{3} + 8 + \frac{8}{8})$ ✅

part weight $= 2R(3 + \frac{3}{11} + 8 + \frac{8}{11})$ ❌

part weight $= 2R(3 + \frac{3}{3} + 8 + \frac{8}{8})$ ✅

Key Observation:

Decreasing fiber fragmentation would:

1- **Increase** the accuracy of the IFS weighting scheme. (better load balancing)

2- **Decrease** the number of FLOPs

Proposal:

The partitioning objective should encapsulate reducing fiber fragmentation.

Augment the hypergraph model with **Fiber nets.**

Augmenting fine-grain HP model

For each fiber, add a net that connects the vertices corresponding to the constituent nonzeros of that fiber

- Two types of cut nets with connectivity `con`:
  - **Volume cut net**: a communication of **$2R*(con-1)$ words** will be incurred (reduce and expand, each of size R)
  - **Fiber cut net:** a computational increase of **$2R*(con-1)$ FLOPs** will be incurred (associated with the new fibers)
- We add a constant α for the tradeoff between communication and computation.

Key Results

6 real-world 3-mode and 4-mode tensors. 15M < nnz < 190M

Improvement % over FG on 512 processors

	Max FLOPs	Total FLOPs
IFS only	8%	4%
IFS + fiber-nets augmentation (impFG)	23%	18%

Improvement % of impFG over FG on 512 processors

	MTTKRP runtime	Total (CPD-ALS) runtime
R=32	21%	14%
R=64	22%	16%
R=128	23%	18%

Part 3: A Framework for Latency Hiding in CPD-ALS

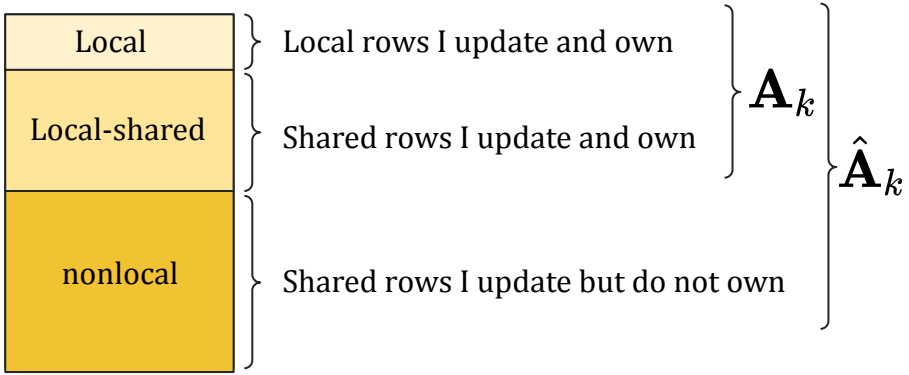
Published as:

Nabil Abubaker, M. Ozan Karsavuran and Cevdet Aykanat, "**Scalable Unsupervised ML: Latency Hiding in Distributed Sparse Tensor Decomposition**", **IEEE Transactions on Parallel and Distributed Systems**, 33(11), 3028 - 3040, 2022

Communication in parallel CPD-ALS

$$\mathbf{A}' = \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^\top \mathbf{C} * \mathbf{B}^\top \mathbf{B})^{-1}$$

Local $\mathbf{A}$ in p_k



Sparse REDUCE: send nonlocal rows, recv local-shared rows

Dual

Sparse EXPAND: recv nonlocal rows, send local-shared rows

$$\hat{\mathbf{V}}'_k = \mathbf{X}^{(1)}_k(\hat{\mathbf{C}}_k \odot \hat{\mathbf{B}}_k)$$

Sparse REDUCE on $\hat{\mathbf{V}}'_k$

$$\mathbf{A}'_k = \mathbf{V}_k(\mathbf{C}^\top \mathbf{C} * \mathbf{B}^\top \mathbf{B})^{-1}$$

$$\lambda'_k = \langle \mathbf{A}'_k(:, c), \mathbf{A}'_k(:, c) \rangle \; \forall c \in \{1 \dots R\}$$

ALL-REDUCE to compute λ'

$$\mathbf{A}_k = \mathbf{A}'_k(:, c)/\sqrt{\lambda_c} \; \forall c \in \{1 \dots R\}$$

Sparse EXPAND on $\hat{\mathbf{A}}_k$

ALL-REDUCE to compute $\mathbf{A}^\top \mathbf{A}$

Communication in parallel CPD-ALS

Problem:

- Sparse reduce/expand operations incur **high** message counts (for large *K*)
- CPD-ALS becomes **latency-bound**

Proposed Solution:

For each mode, there are **two** sparse reduce/expand operations and **two** ALL-REDUCE:

Embed each sparse reduce/expand into an ALL-REDUCE

$$\hat{\mathbf{V}}_k' = \mathbf{X}_k^{(1)}(\hat{\mathbf{C}}_k \odot \hat{\mathbf{B}}_k)$$

Sparse REDUCE on $\hat{\mathbf{V}}_k'$

$$\mathbf{A}_k' = \mathbf{V}_k(\mathbf{C}^\top\mathbf{C} * \mathbf{B}^\top\mathbf{B})^{-1}$$

$$\lambda_k' = \langle \mathbf{A}_k'(:,c), \mathbf{A}_k'(:,c) \rangle \; \forall c \in \{1 \dots R\}$$

ALL-REDUCE to compute λ'

$$\mathbf{A}_k = \mathbf{A}_k'(:,c)/\sqrt{\lambda_c} \; \forall c \in \{1 \dots R\}$$

Sparse EXPAND on $\hat{\mathbf{A}}_k$

ALL-REDUCE to compute $\mathbf{A}^\top\mathbf{A}$

The embedding Framework: Overview

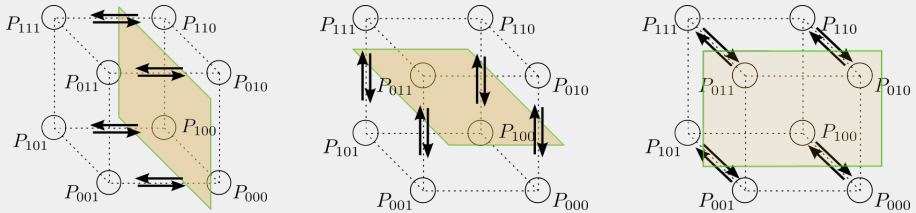
Computation/ communication **reordering scheme**

(Allow pairing each sparse op with an ALL-REDUCE)

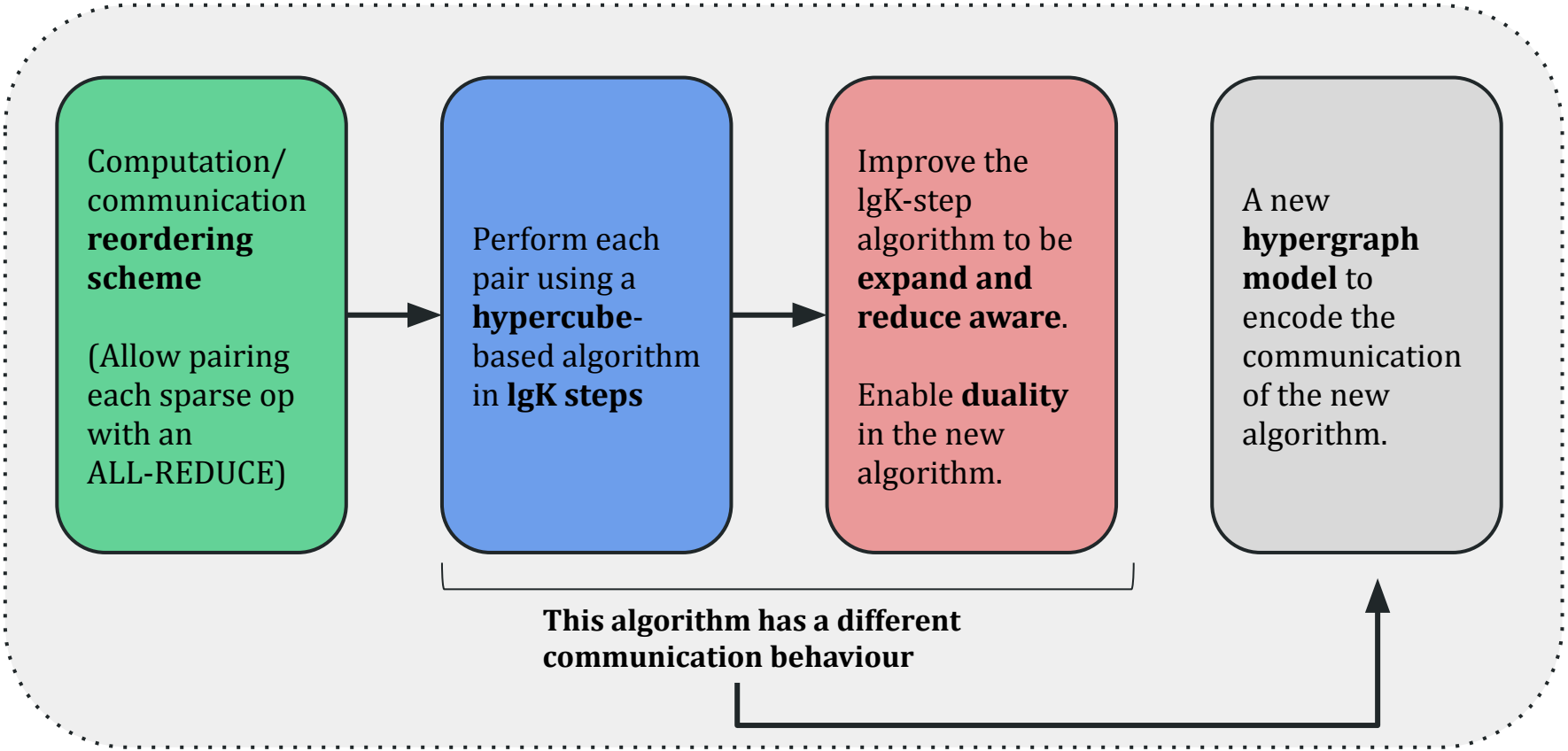
Perform each pair using a **hypercube**-based algorithm in **lgK steps**

Improve the lgK-step algorithm to be **expand and reduce aware**.

Enable **duality** in the new algorithm.



The embedding Framework: Overview



Comp/Comm Rearrangements

Is the embedding possible?

$$\hat{\mathbf{V}}_k' = \mathbf{X}_k^{(1)}(\hat{\mathbf{C}}_k \odot \hat{\mathbf{B}}_k)$$

Sparse REDUCE on $\hat{\mathbf{V}}_k'$ ❌

$$\mathbf{A}_k' = \mathbf{V}_k(\mathbf{C}^\top\mathbf{C} * \mathbf{B}^\top\mathbf{B})^{-1}$$

$$\lambda_k' = \langle \mathbf{A}_k'(:,c), \mathbf{A}_k'(:,c) \rangle \; \forall c \in \{1 \dots R\}$$

ALL-REDUCE to compute λ'

$$\mathbf{A}_k = \mathbf{A}_k'(:,c)/\sqrt{\lambda_c} \; \forall c \in \{1 \dots R\}$$

Sparse EXPAND on $\hat{\mathbf{A}}_k$

ALL-REDUCE to compute $\mathbf{A}^\top\mathbf{A}$ ✅

Comp/Comm Rearrangements

Is the embedding possible?

$$\hat{\mathbf{V}}'_k = \mathbf{X}^{(1)}_k (\hat{\mathbf{C}}_k \odot \hat{\mathbf{B}}_k)$$

Sparse REDUCE on $\hat{\mathbf{V}}'_k$ ✅

$$\mathbf{A}'_k = \mathbf{V}_k (\mathbf{C}^\top \mathbf{C} * \mathbf{B}^\top \mathbf{B})^{-1}$$

$$\lambda'_k = \langle \mathbf{A}'_k(:, c), \mathbf{A}'_k(:, c) \rangle \; \forall c \in \{1 \dots R\}$$

ALL-REDUCE to compute λ' ❌

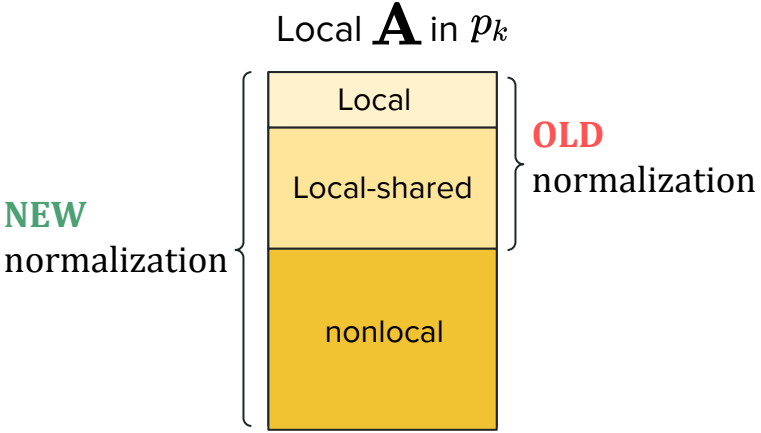
$$\mathbf{A}_k = \mathbf{A}'_k(:, c) / \sqrt{\lambda_c} \; \forall c \in \{1 \dots R\}$$

Sparse EXPAND on $\hat{\mathbf{A}}_k$

⋮

ALL-REDUCE to compute $\mathbf{C}^\top \mathbf{C}$

Comp/Comm Rearrangements



$$\hat{\mathbf{V}}_k' = \mathbf{X}_k^{(1)}(\hat{\mathbf{C}}_k \odot \hat{\mathbf{B}}_k)$$

ALL-REDUCE to compute $\mathbf{C}^\top\mathbf{C}$

Sparse REDUCE on $\hat{\mathbf{V}}_k'$

✅

$$\mathbf{A}_k' = \mathbf{V}_k(\mathbf{C}^\top\mathbf{C} * \mathbf{B}^\top\mathbf{B})^{-1}$$

$$\lambda_k' = \langle \mathbf{A}_k'(:,c), \mathbf{A}_k'(:,c)\rangle \ \forall c \in \{1\dots R\}$$

ALL-REDUCE to compute λ'

Sparse EXPAND on $\hat{\mathbf{A}}_k'$

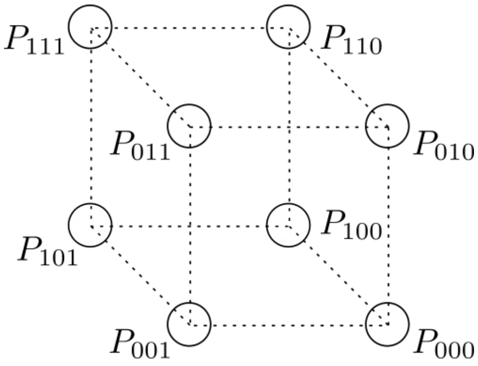
✅

$$\hat{\mathbf{A}}_k = \hat{\mathbf{A}}_k'(:,c)/\sqrt{\lambda_c} \ \forall c \in \{1\dots R\}$$

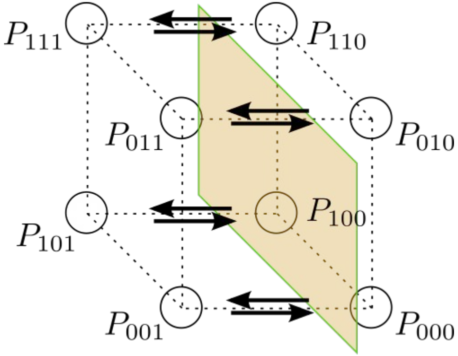
$\vdots$

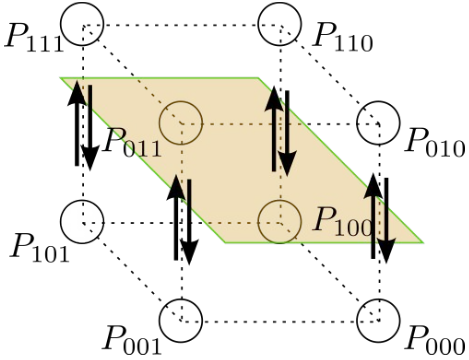
Hypercube-based ALL-REDUCE

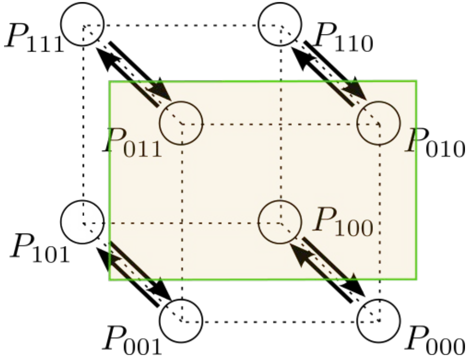
Virtual Hypercube topology



D-dimensional hypercube, ***D = lgK***





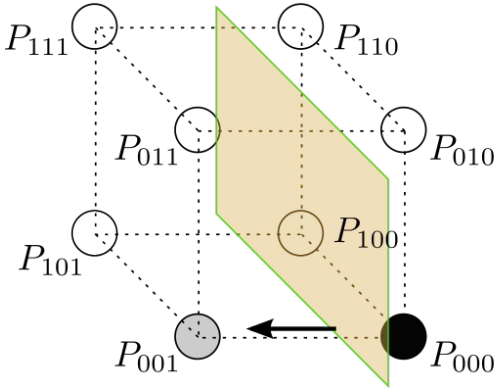


Embedding sparse expand into ALL-REDUCE

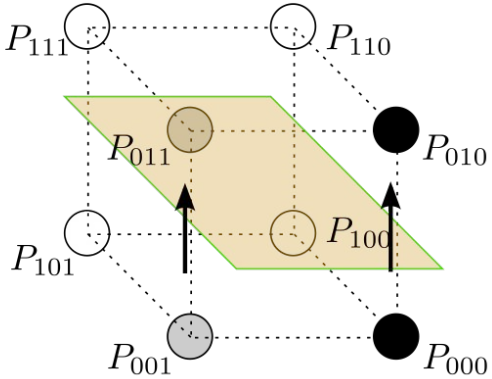
Processors that contribute to row $a_i = \{P_{000}\ (owner), P_{010}, P_{101}, P_{111}\}$

Contributing processors

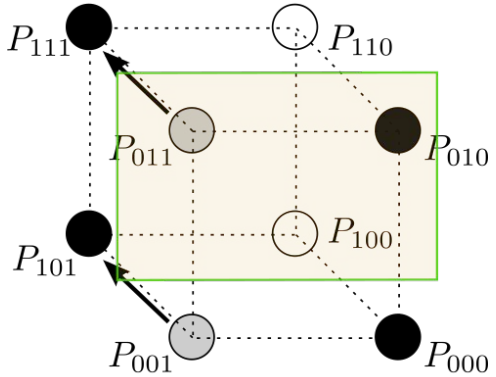
Intermediate processors



Concurrent vol = 1
Tot. vol. = 1



Concurrent vol = 1
Tot. vol. = 2



Concurrent vol = 1
Tot. vol. = 2

A factor matrix row is routed (stored and forwarded) until it reaches its destination

Expand and Reduce Aware Embedding

Problem

The routing algorithm **is not aware** of the nature of the sparse expand and reduce operations:
The same factor matrix row might appear **more than once** in the same message.

Proposed Solution

An expand-and-reduce-aware routing algorithm:

→ **In case of expand:** Remove duplicate rows from the same message
→ **In case of reduce:** Reduce factor matrix rows with the same index to one row while routing.
(previously: leave this task to the owner processor)

Enabling Duality in Embedding

Problem

Each dual sparse reduce and expand pair are to be communicated using different routing settings because the structure of communication changes during routing. This means:

→ **Seperate** store-and-forward buffers for each sparse operation (**storage and preprocessing overhead**)

→ An intelligent partitioning model specifically for that communication (or **increase the size/complexity** of the single model used)

Proposed Solution

Enabling Duality in embedding via:

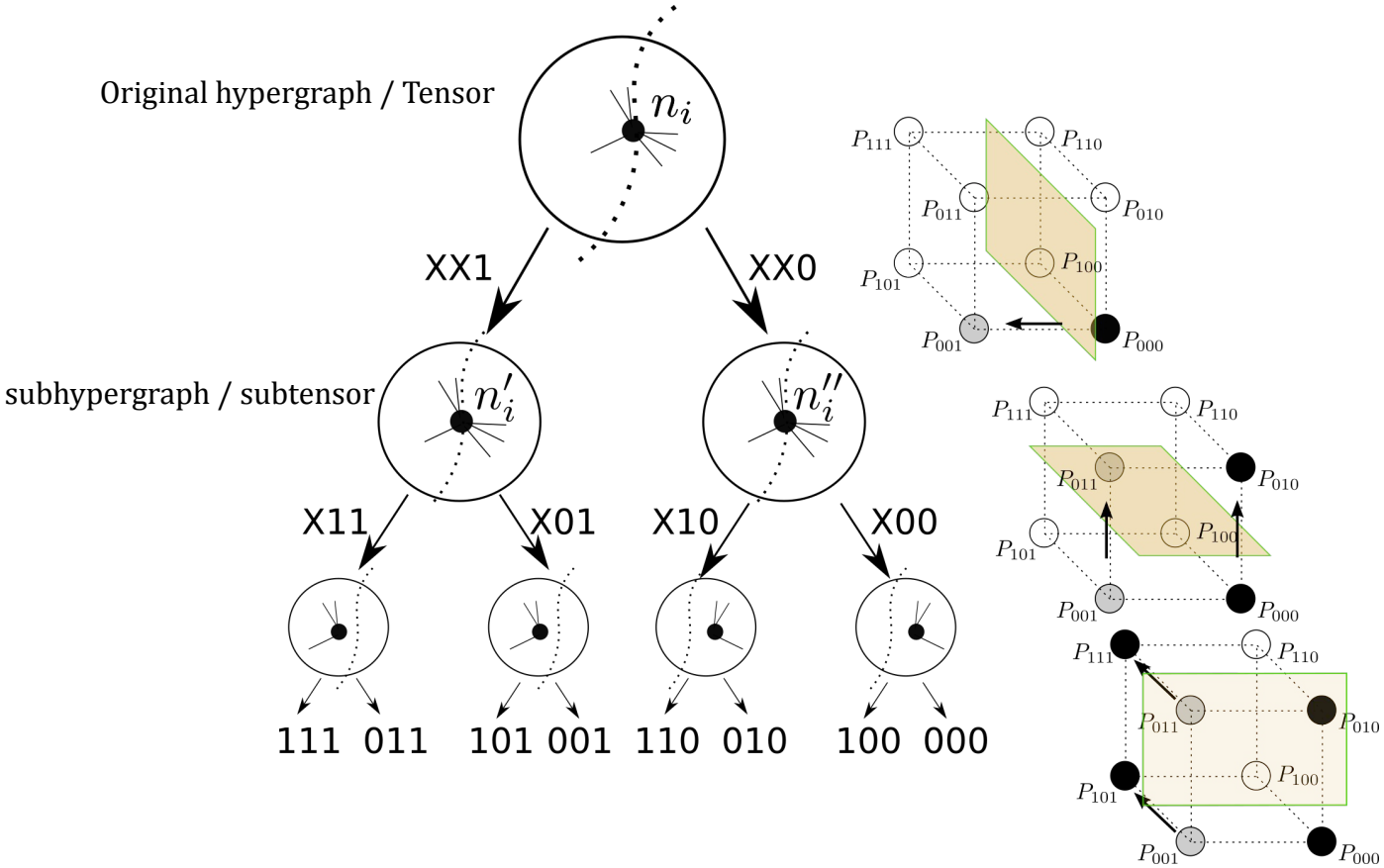
→ Performing the routing for a sparse reduce **in one order**

→ Performing the routing for the dual sparse expand in **the reverse order*.**

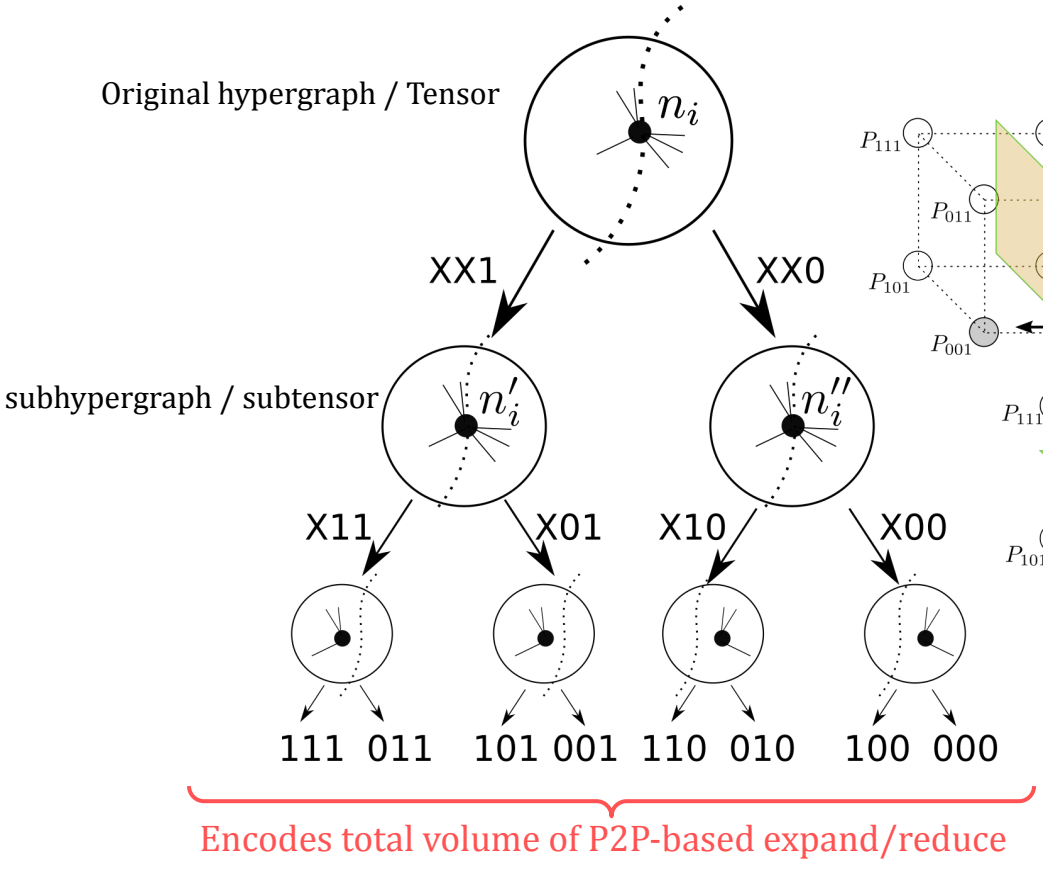
→ This would **reduce the overheads mentioned above** by using **the same** buffers and preprocessing overhead for each a dual sparse reduce and expand.

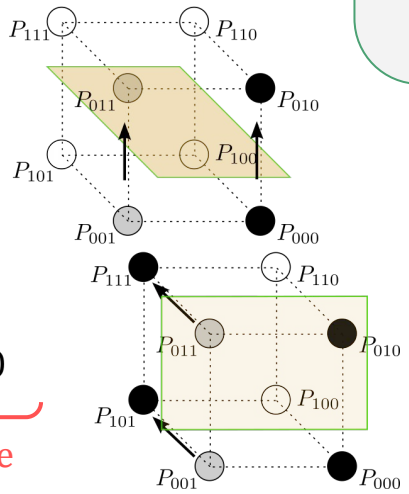
****proof provided in dissertation*.**

Recursive-Bipartitioning-based Hypergraph model



Recursive-Bipartitioning-based Hypergraph model

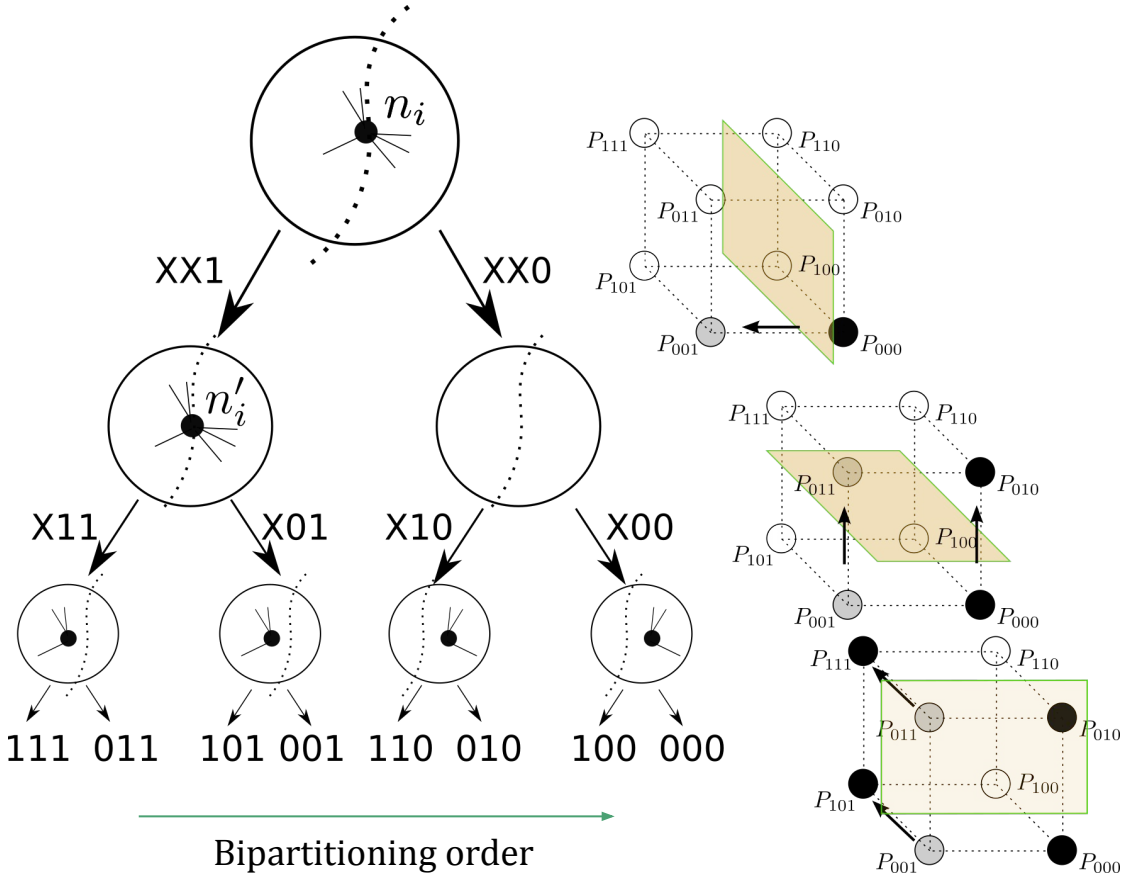




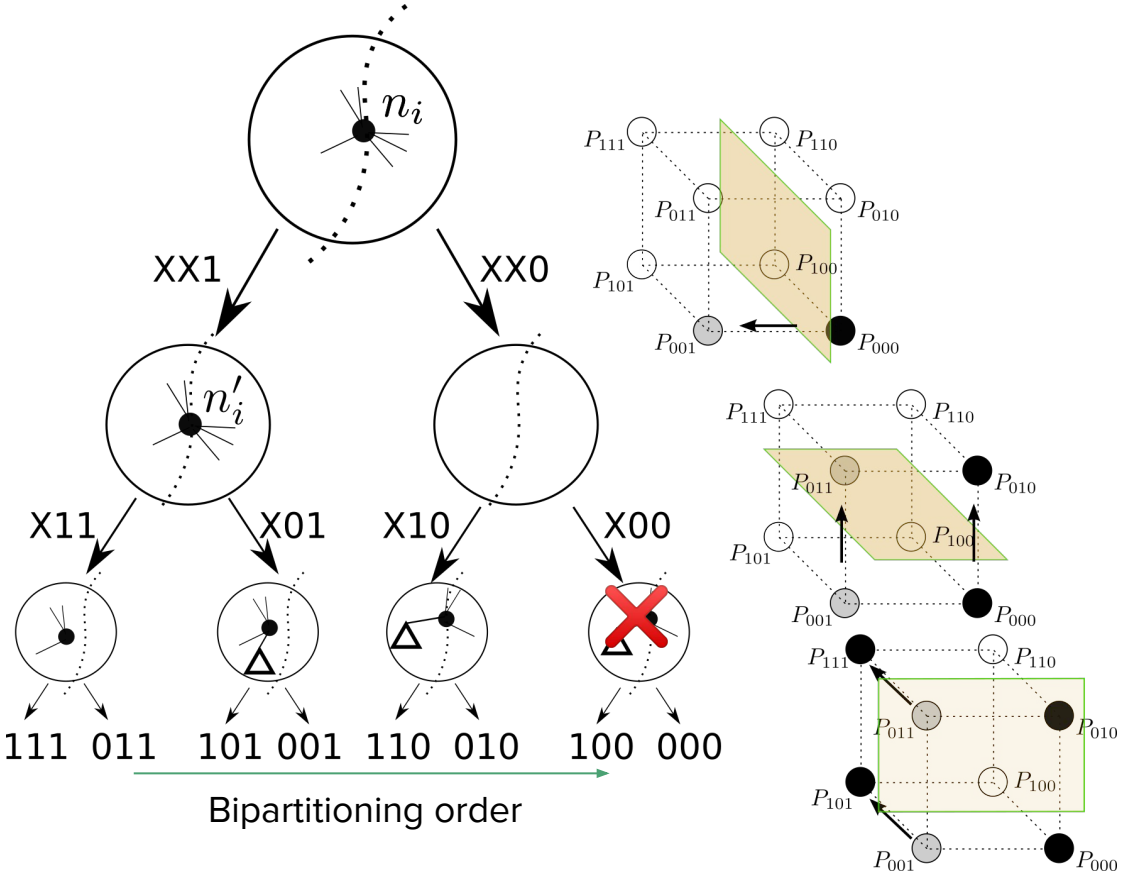
Two problems:

- Doesn't encode **forwarding**
- Doesn't encode **concurrent volume**

Encode concurrent volume: Sibling Subnet Removal



Encode forwarding: Net Anchoring



Key Results

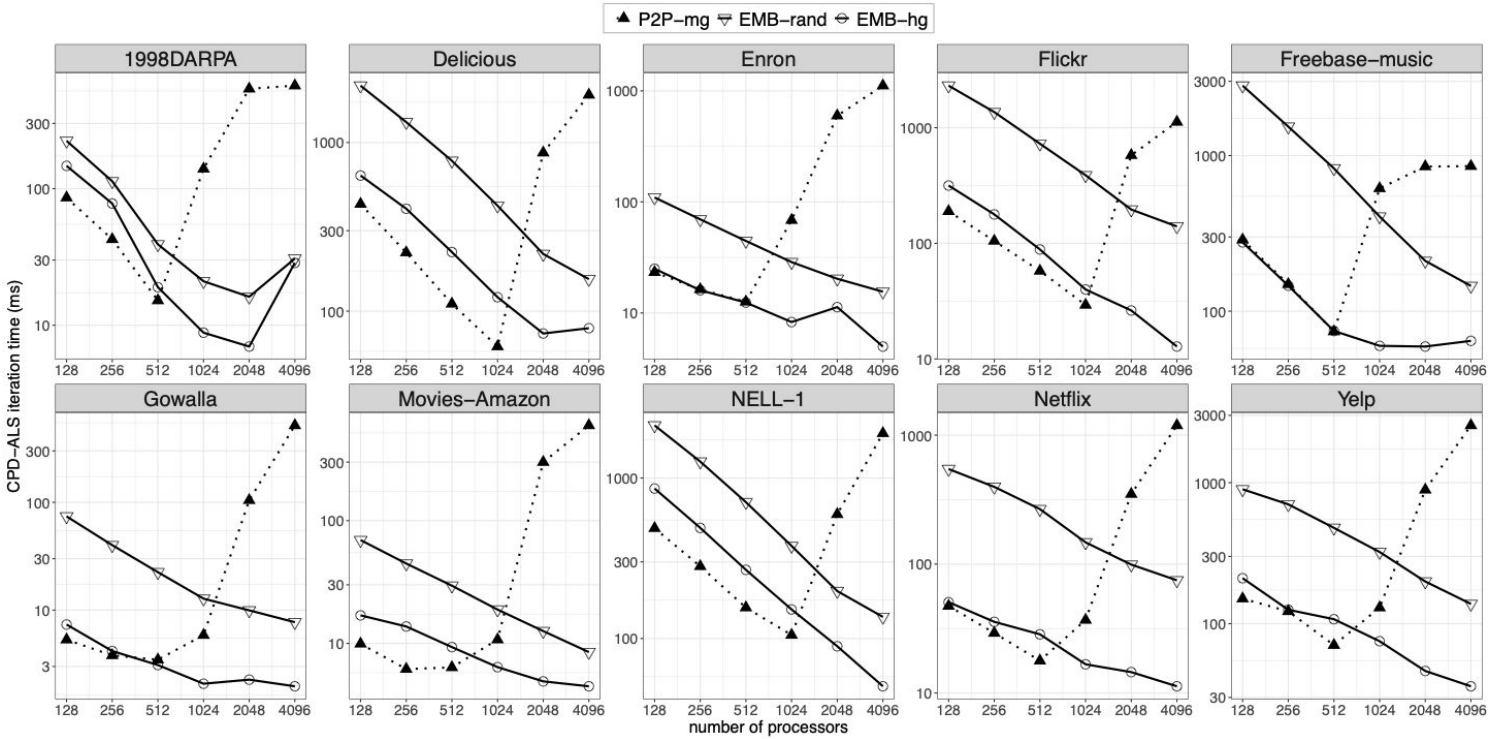
13 real-world sparse tensors. 5M < nnz < 4.6B nonzeros

The embedding scheme reduce the number of messages by **31x** on **4096** processors

The expand-and-reduce-aware embedding **improves** the runtime by **1.3x** on **4096** processors

The CPD-ALS runtime due to **HP-based** distribution is improved by **3.5x** compared to **random-based** distribution when R=32 on on **4096** processors.

Key Results



P2P-mg: M. O. Karsavuran, S. Acer and C. Aykanat, "Partitioning Models for General Medium-Grain Parallel Sparse Tensor Decomposition," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 1, pp. 147-159, 1 Jan. 2021

Conclusions

We proposed several Algorithms and Models to enable the scalability of SSGD and CPD-ALS

For SSGD, exchanging the essential communication with P2P messages is invaluable for reducing the bandwidth, and the powerful H&C algorithm makes it latency-safe.

For CPD-ALS, we shed light on the load balancing problem of MTTKR and propose two strategies to tackle it.

Furthermore, the proposed latency-hiding framework enables the algorithm to scale beyond 1024 processors.

Publications *(highlighted ones constitute the content of the dissertation)*

- ***[preprint]*** **Nabil Abubaker**, M. Ozan Karsavuran and Cevdet Aykanat, “Scaling Stratified Stochastic Gradient Descent for Distributed Matrix Completion”, *TechRxiv*.
- ***[Journal]*** **Nabil Abubaker**, M. Ozan Karsavuran and Cevdet Aykanat, “Scalable Unsupervised ML: Latency Hiding in Distributed Sparse Tensor Decomposition”, **IEEE Transactions on Parallel and Distributed Systems**, 33(11), 3028 - 3040, 2022.
- ***[Journal]*** **Nabil Abubaker**, Seher Acer and Cevdet Aykanat, “True Load Balancing for Matricized Tensor Times Khatri-Rao Product”, **IEEE Transactions on Parallel and Distributed Systems**, 32 (8), 1974-1986, 2021.
- ***[Journal]*** **Nabil Abubaker**, Kadir Akbudak and Cevdet Aykanat, “Spatiotemporal Graph and Hypergraph Partitioning Models for Sparse Matrix-Vector Multiplication on Many-Core Architectures”, **IEEE Transactions on Parallel and Distributed Systems**, 30 (2), 445-458, 2019.
- ***[Conference]*** **Nabil Abubaker**, Leonard Dervishi and Erman Ayday,”Privacy-Preserving Fog Computing Paradigm”, **IEEE Conference on Communications and Network Security (CNS)** 2017.

Algorithms and Models for Scaling Sparse Tensor and Matrix Factorizations

Nabil Abubaker
Advisor: Prof. Cevdet Aykanat
July 6th, 2022
Department of Computer Engineering, Bilkent University