Scaling Stratified Stochastic Gradient Descent for Distributed Matrix Completion

Nabil Abubaker

Nabil Abubaker, M. Ozan Karsavuran and Cevdet Aykanat, "**Scaling Stratified Stochastic Gradient Descent for Distributed Matrix Completion**", *IEEE Transactions on Knowledge and Data Engineering, 2023*

Code: https://github.com/nfabubaker/CESSGD

Bilkent University

ETH zürich

~~Scaling Stratified Stochastic Gradient Descent for Distributed Matrix Completion~~

Communication-Efficient Stratified SGD

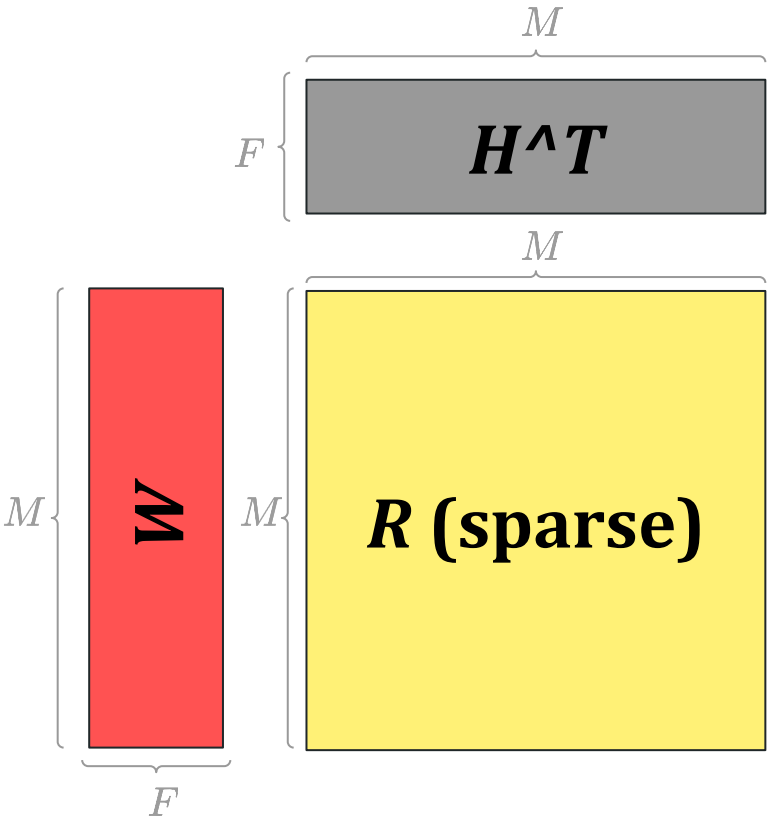
Nabil Abubaker

Nabil Abubaker, M. Ozan Karsavuran and Cevdet Aykanat, "**Scaling Stratified Stochastic Gradient Descent for Distributed Matrix Completion**", *IEEE Transactions on Knowledge and Data Engineering, 2023*

Code: https://github.com/nfabubaker/CESSGD

Bilkent University

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:

$$Loss = \sum (r_{ij} - w_i h_j^T)^2$$

By:

Step size

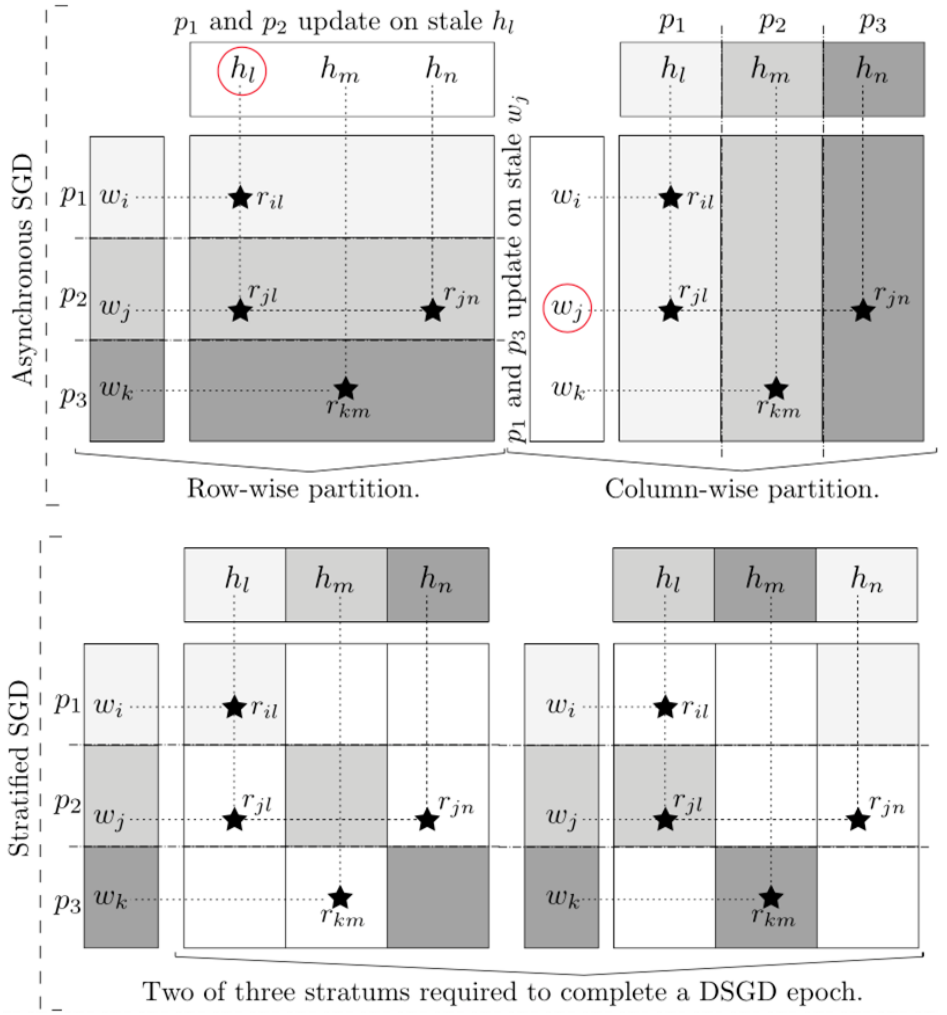
Regularization factor

$$w_i = w_i + \epsilon ((r_{ij} - w_i h_j^T) h_j + \gamma w_i)$$

$$h_j = h_j + \epsilon ((r_{ij} - w_i h_j^T) w_i + \gamma h_j)$$

Parallel SGD

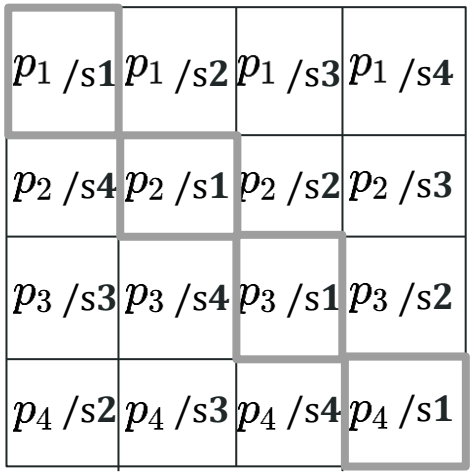
- **Asynchronous**: Allows staleness
- **Stratified (SSGD)**: Doesn't allow staleness
  - Serializable
  - Better convergence
  - Well-studied behaviour



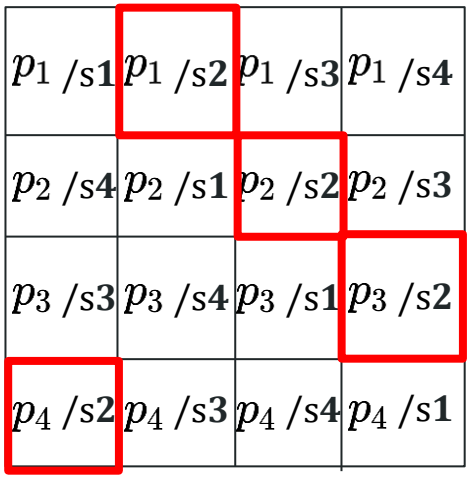
Stratified SGD proposed in:
R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 69–77, 2011.

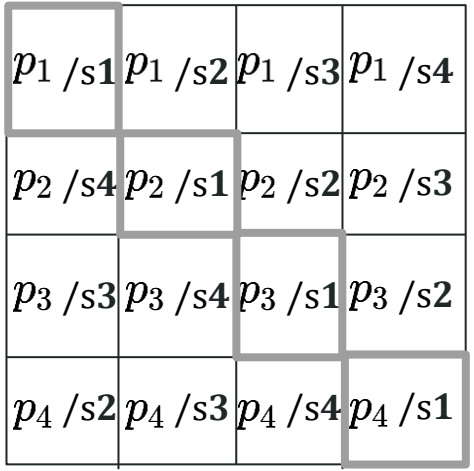
s1

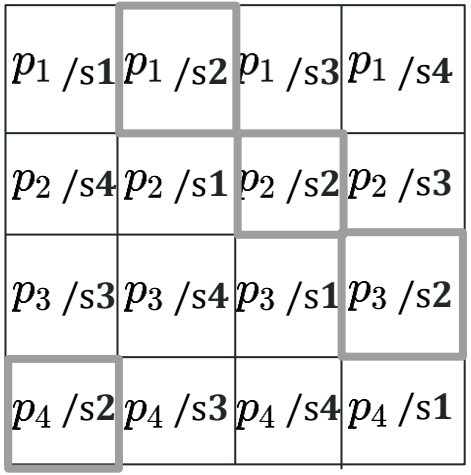
p_1 /s1	p_1 /s2	p_1 /s3	p_1 /s4
p_2 /s4	p_2 /s1	p_2 /s2	p_2 /s3
p_3 /s3	p_3 /s4	p_3 /s1	p_3 /s2
p_4 /s2	p_4 /s3	p_4 /s4	p_4 /s1


p_1 /s1 p_1 /s2 p_1 /s3 p_1 /s4
p_2 /s4 p_2 /s1 p_2 /s2 p_2 /s3
p_3 /s3 p_3 /s4 p_3 /s1 p_3 /s2
p_4 /s2 p_4 /s3 p_4 /s4 p_4 /s1

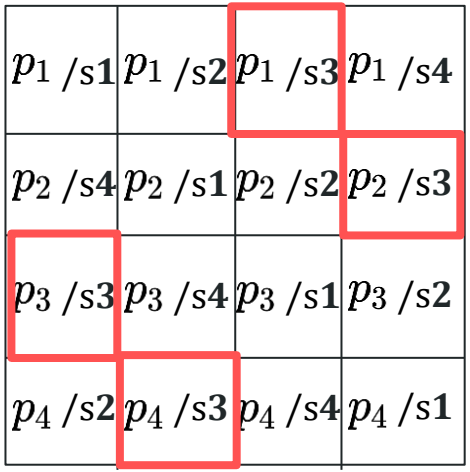
s2

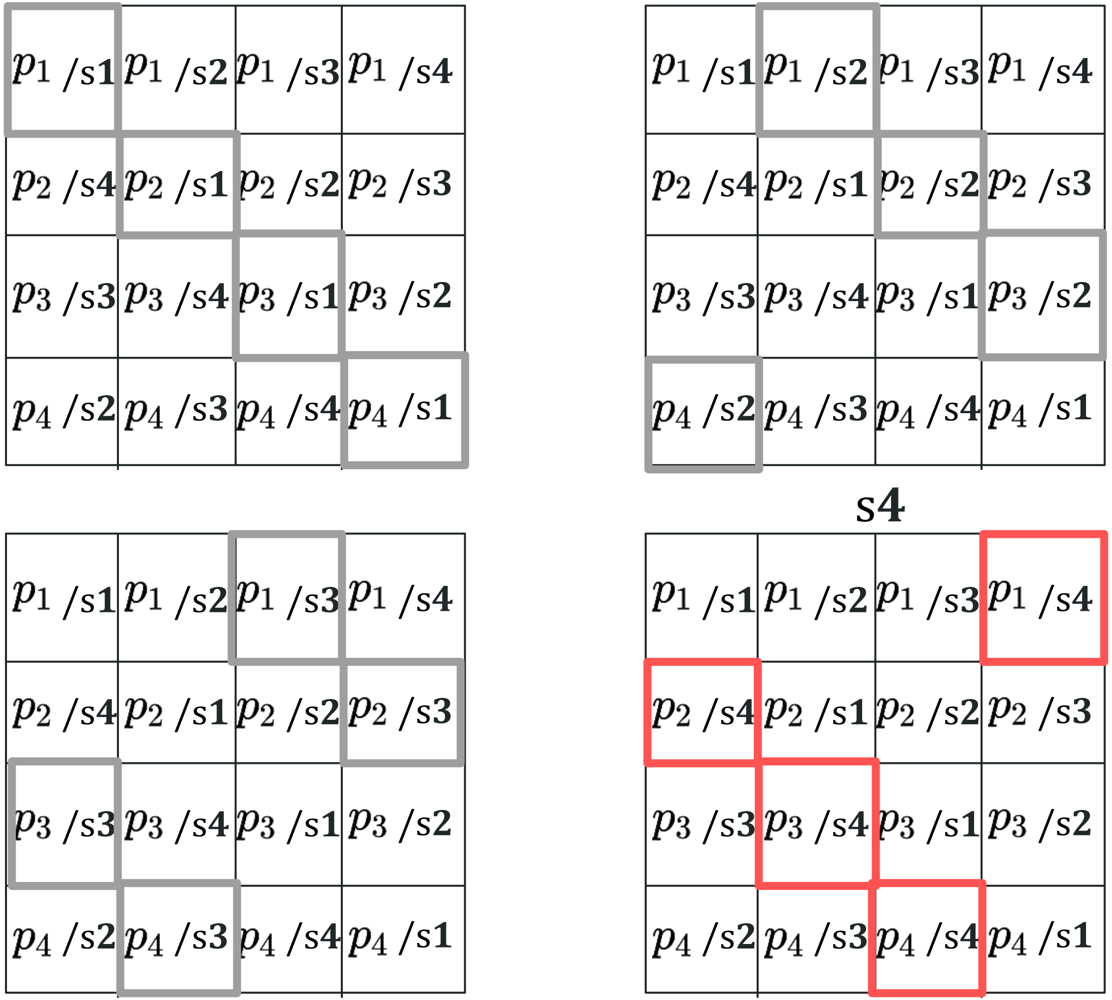

p_1 /s1 p_1 /s2 p_1 /s3 p_1 /s4
p_2 /s4 p_2 /s1 p_2 /s2 p_2 /s3
p_3 /s3 p_3 /s4 p_3 /s1 p_3 /s2
p_4 /s2 p_4 /s3 p_4 /s4 p_4 /s1


p_1 /s1 p_1 /s2 p_1 /s3 p_1 /s4
p_2 /s4 p_2 /s1 p_2 /s2 p_2 /s3
p_3 /s3 p_3 /s4 p_3 /s1 p_3 /s2
p_4 /s2 p_4 /s3 p_4 /s4 p_4 /s1


p_1 /s1 p_1 /s2 p_1 /s3 p_1 /s4
p_2 /s4 p_2 /s1 p_2 /s2 p_2 /s3
p_3 /s3 p_3 /s4 p_3 /s1 p_3 /s2
p_4 /s2 p_4 /s3 p_4 /s4 p_4 /s1

s3


p_1 /s1 p_1 /s2 p_1 /s3 p_1 /s4
p_2 /s4 p_2 /s1 p_2 /s2 p_2 /s3
p_3 /s3 p_3 /s4 p_3 /s1 p_3 /s2
p_4 /s2 p_4 /s3 p_4 /s4 p_4 /s1

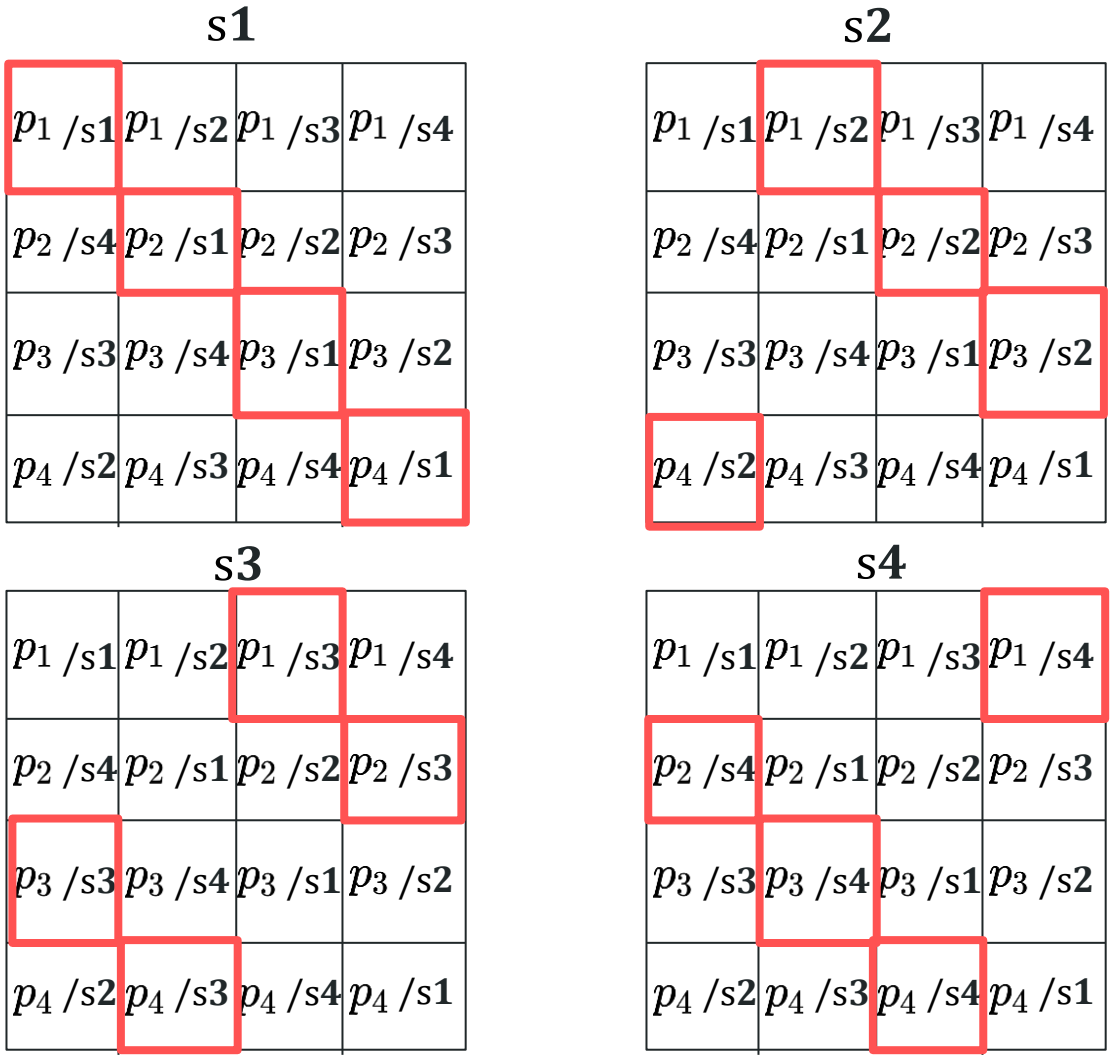


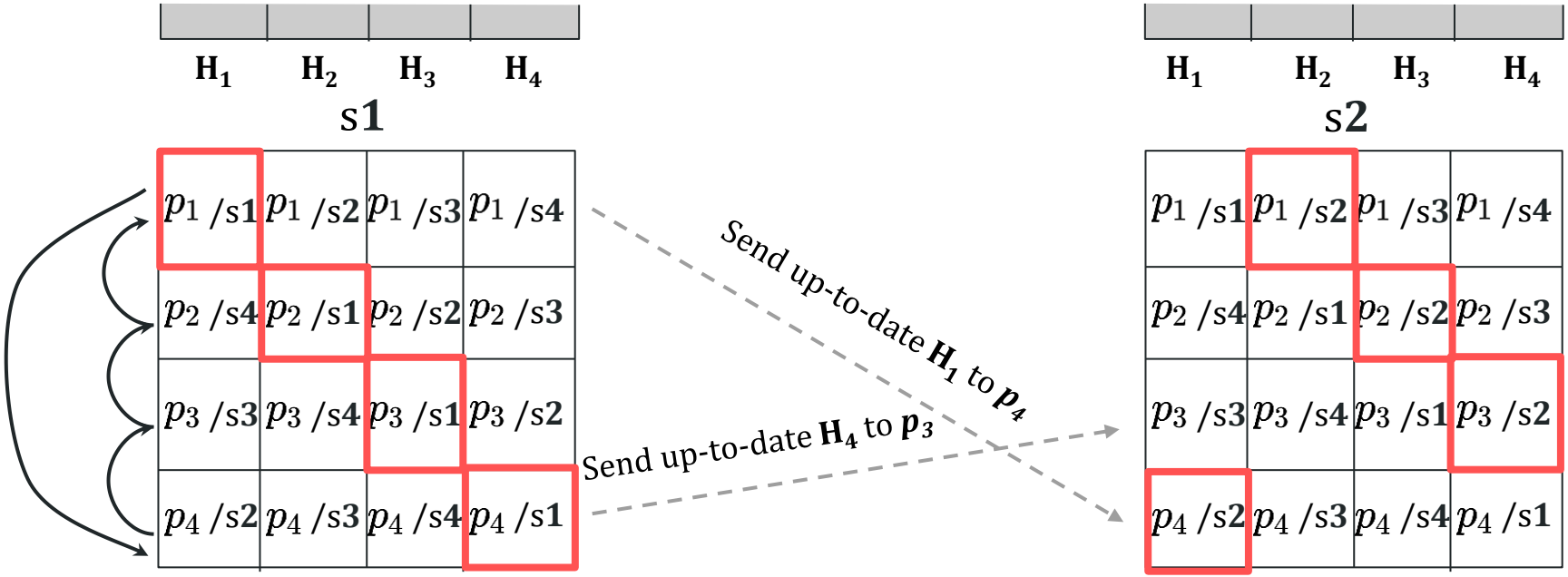
p_1 /s1 p_1 /s2 p_1 /s3 p_1 /s4
p_2 /s4 p_2 /s1 p_2 /s2 p_2 /s3
p_3 /s3 p_3 /s4 p_3 /s1 p_3 /s2
p_4 /s2 p_4 /s3 p_4 /s4 p_4 /s1
s4

Ring Schedule

Or

Ring Strata




H1
H2
H3
H4
s1
p1 /s1
p1 /s2
p1 /s3
p1 /s4
p2 /s4
p2 /s1
p2 /s2
p2 /s3
p3 /s3
p3 /s4
p3 /s1
p3 /s2
p4 /s2
p4 /s3
p4 /s4
p4 /s1
Send up-to-date H1 to p4
Send up-to-date H4 to p3
H1
H2
H3
H4
s2
p1 /s1
p1 /s2
p1 /s3
p1 /s4
p2 /s4
p2 /s1
p2 /s2
p2 /s3
p3 /s3
p3 /s4
p3 /s1
p3 /s2
p4 /s2
p4 /s3
p4 /s4
p4 /s1

Problem with distributed SSGD:

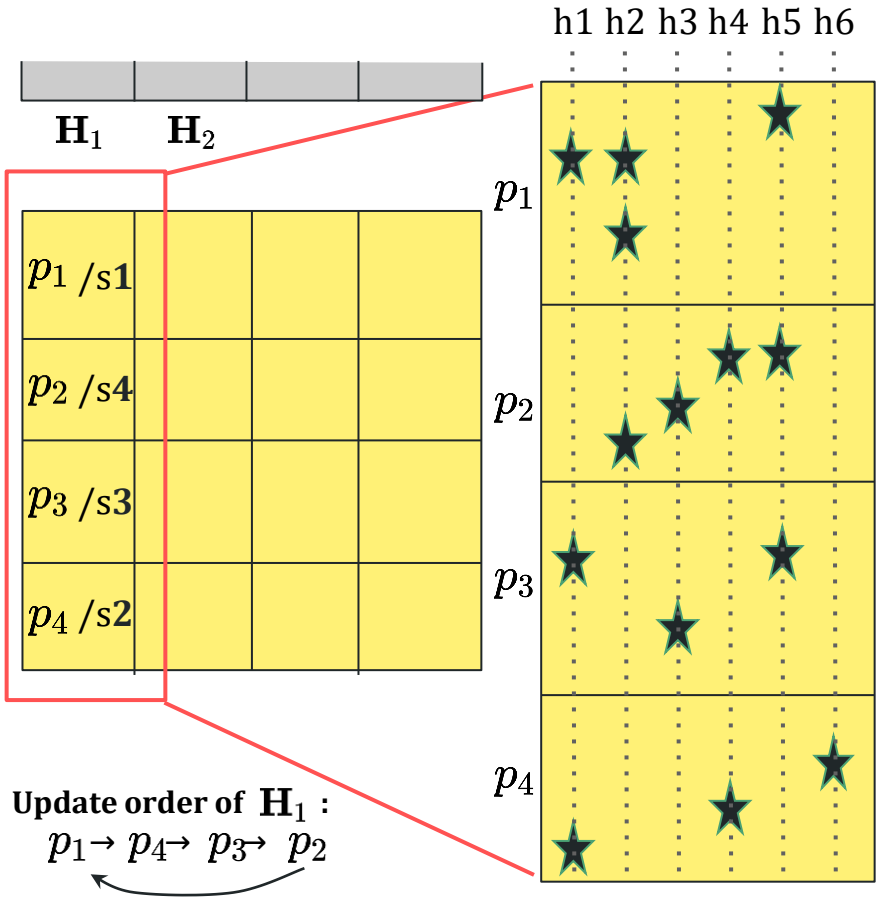
Existing implementations communicate **blocks** of factor matix rows:

- 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 factor matrix rows that are **essential** for the correctness of the SGD algorithm **using P2P messages**

Finding essential communication

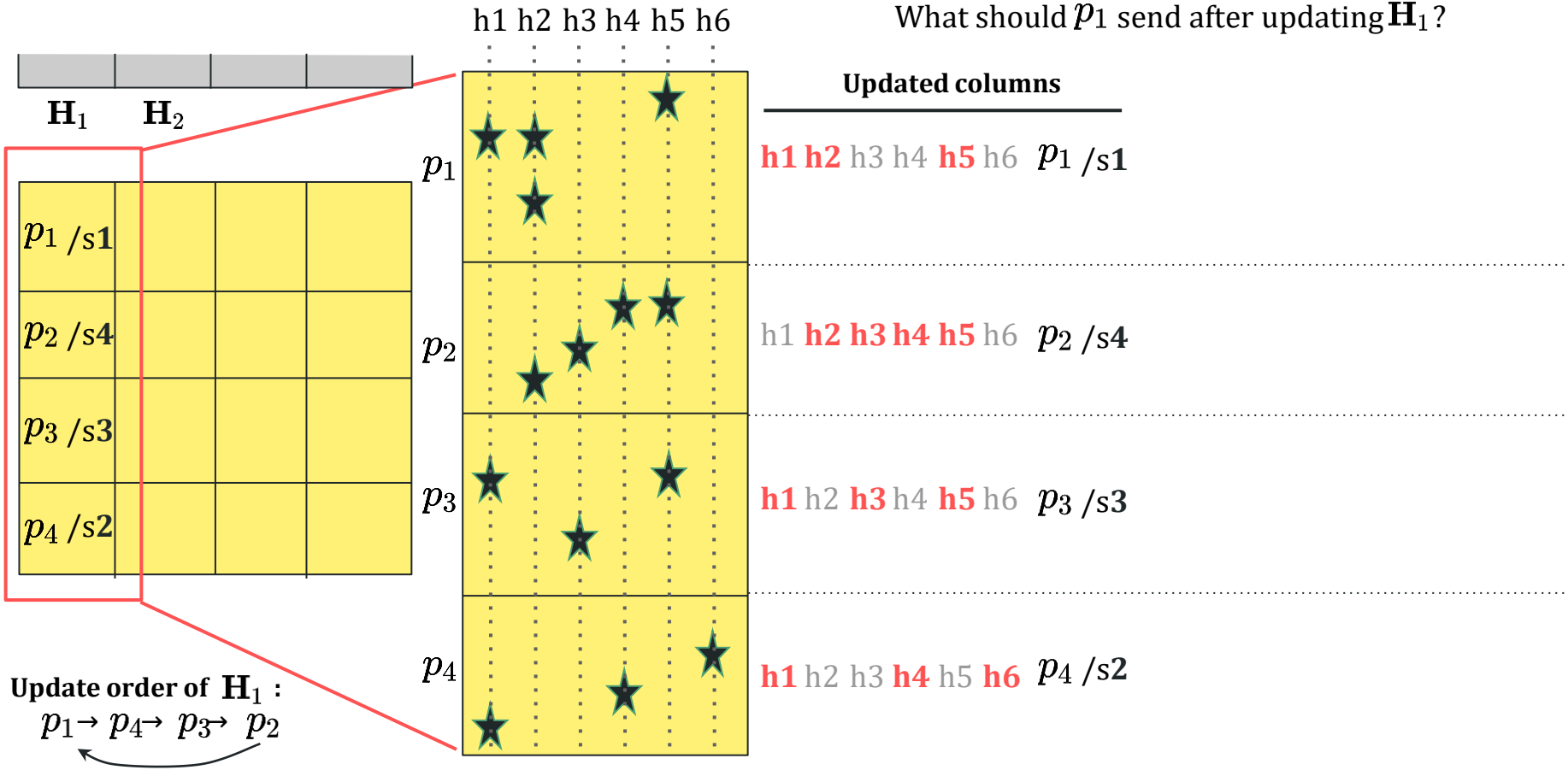


h_i is sent from p_x to p_y if :

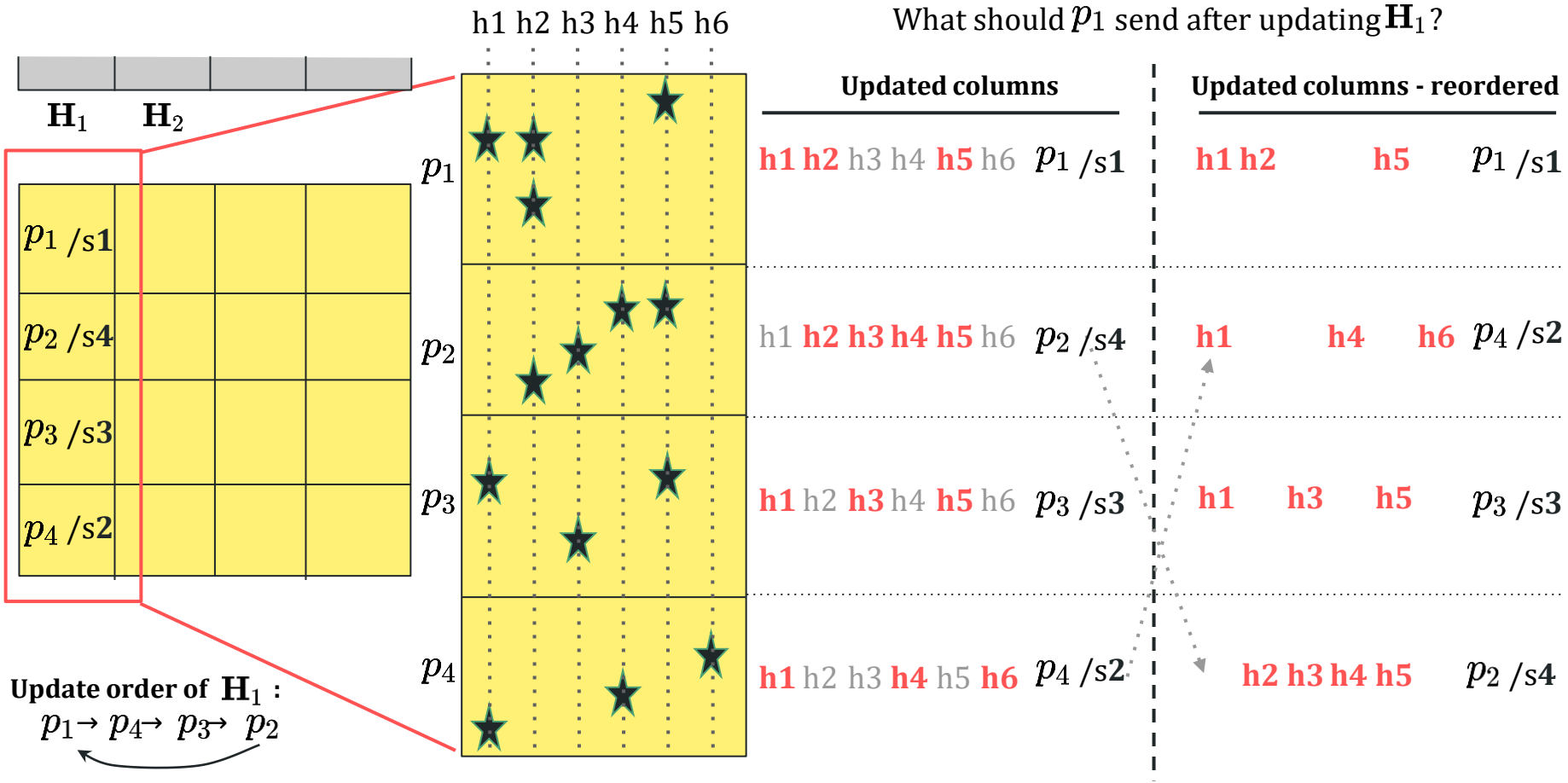
→ both p_x and p_y update **h_i** and
→ **no** processor **in between** does.

According to the update order

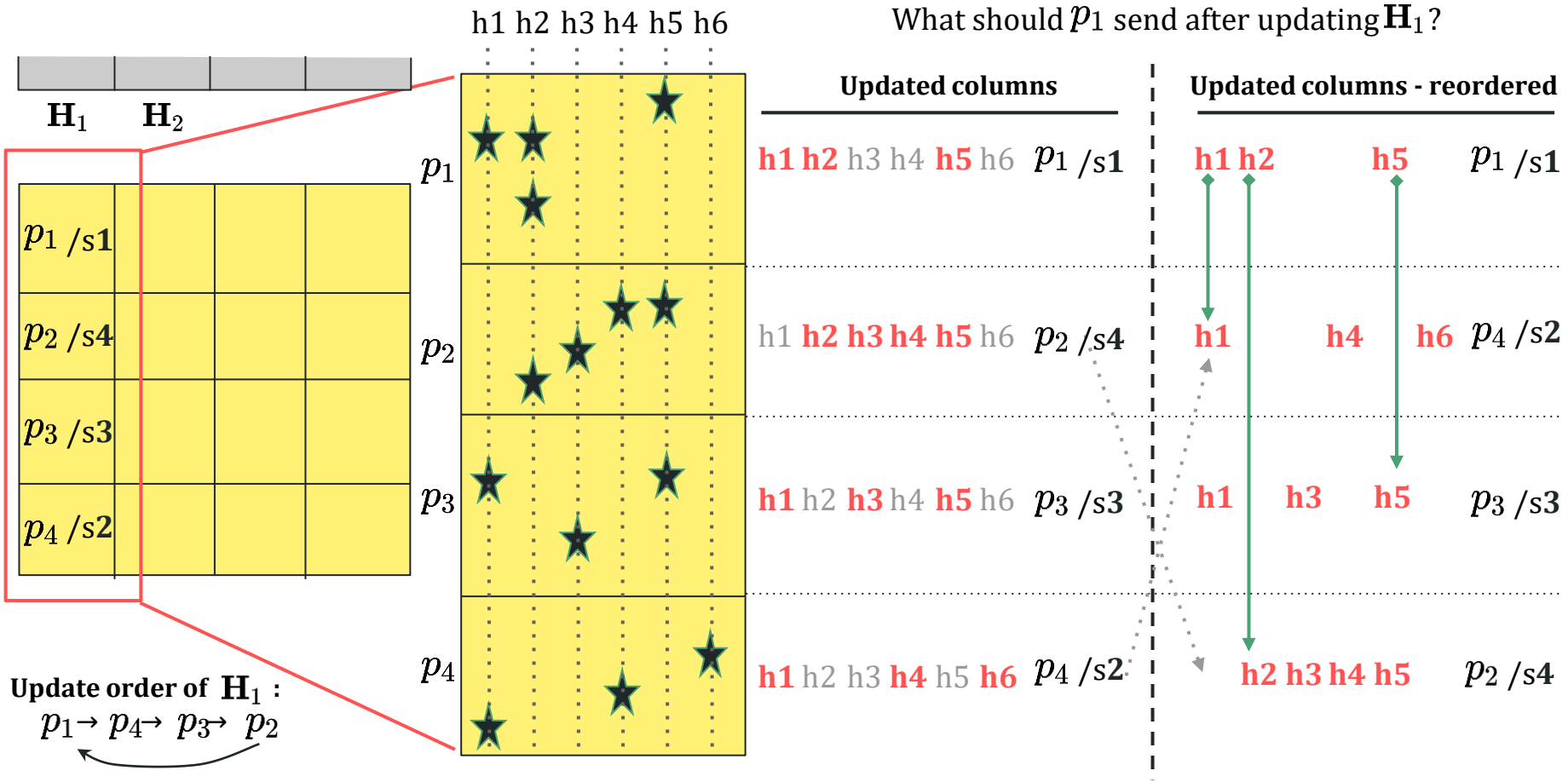
Finding essential communication



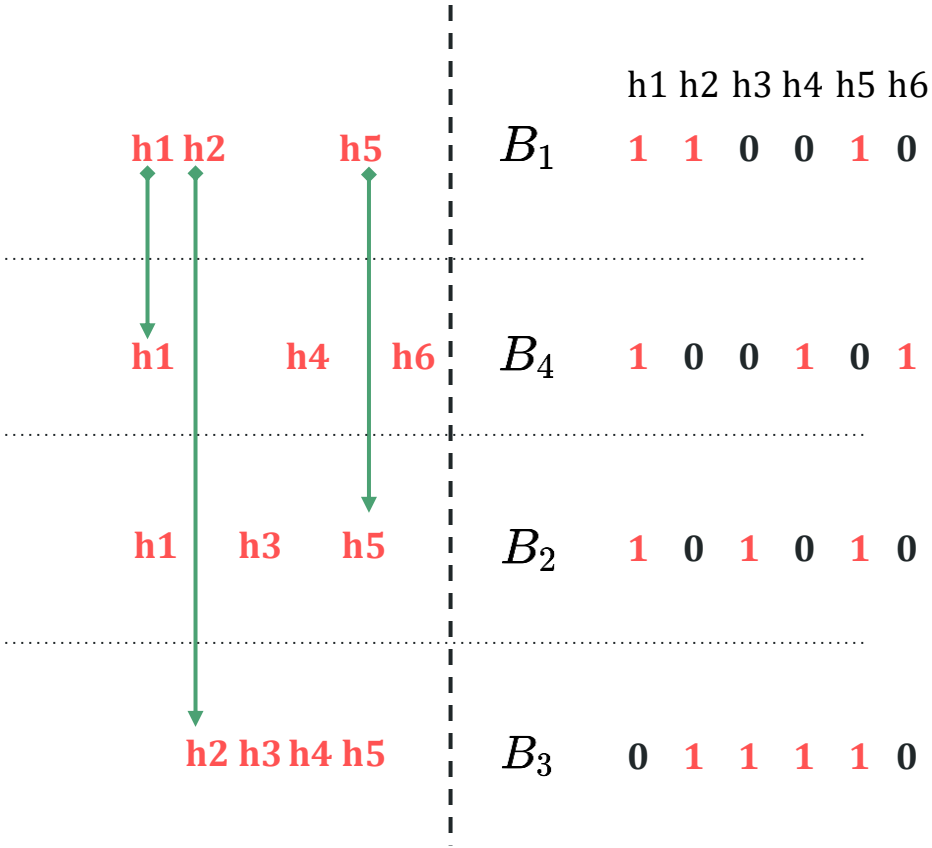
Finding essential communication



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.

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 $O(K^2)$

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.

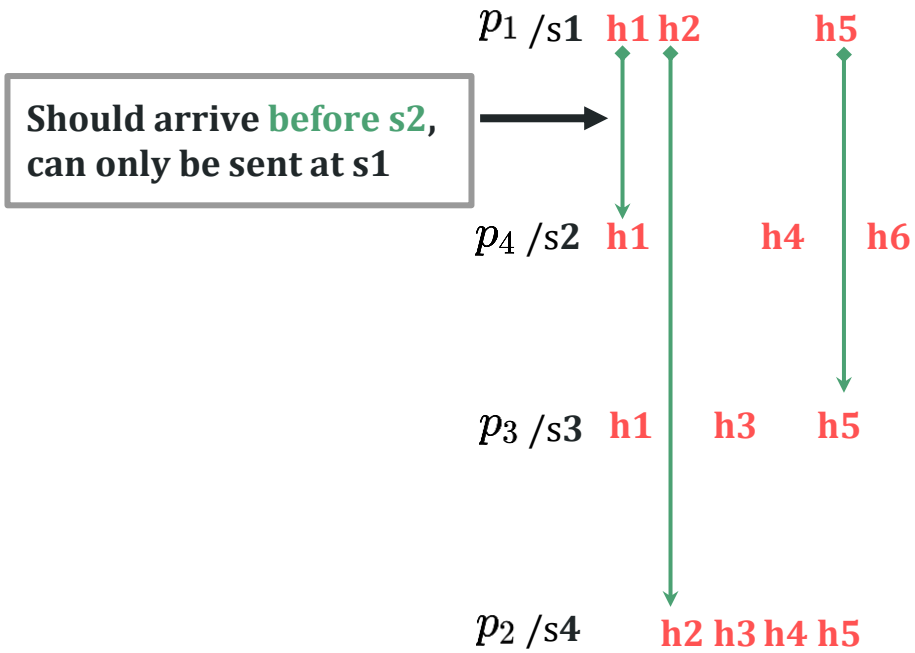
Research Question

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

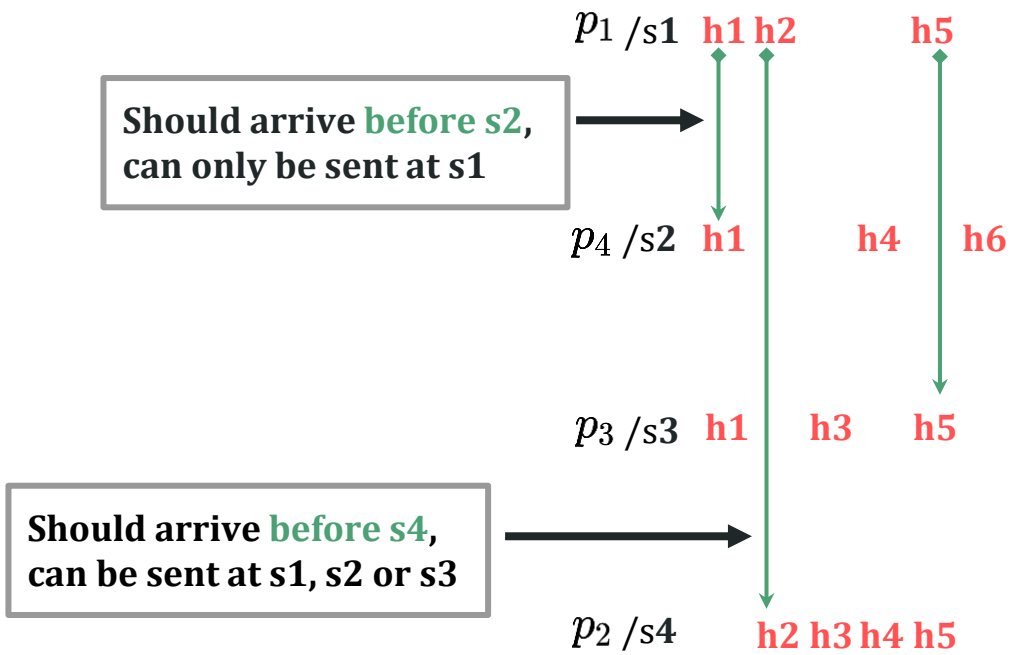
Key Observation

H-matrix rows sent via P2P messages are not always immediately needed in the next sub-epoch.

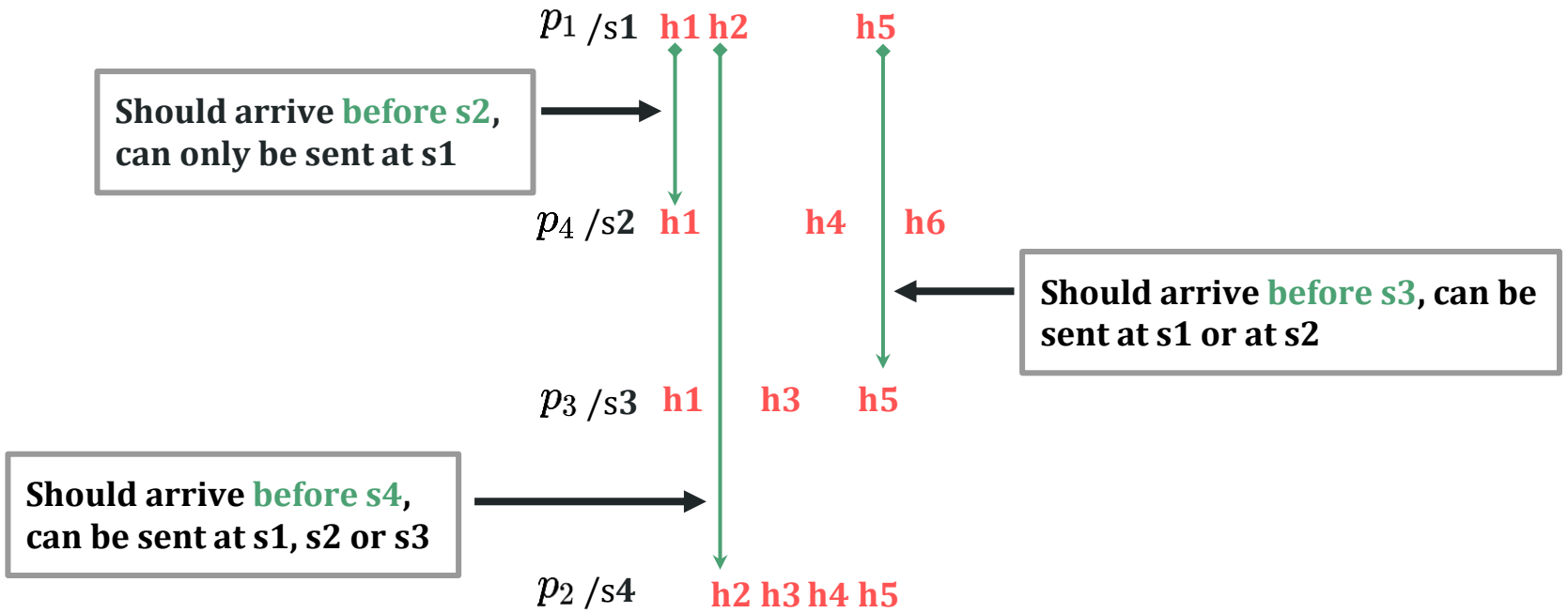
Key Observation: A closer look



Key Observation: A closer look



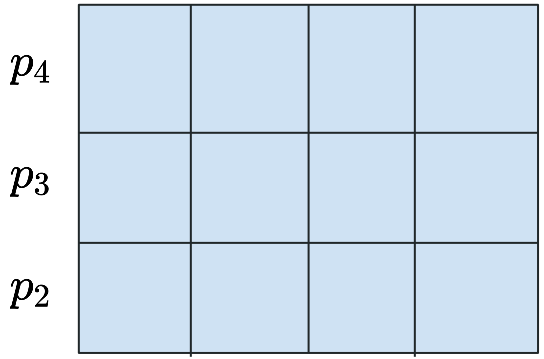
Key Observation: A closer look



The Hold & Combine Algorithm

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

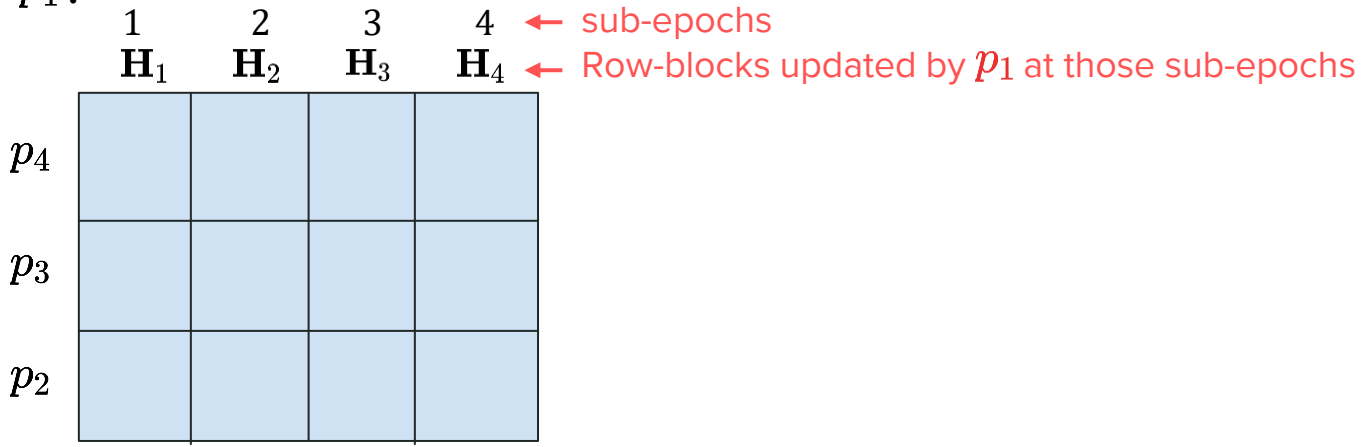
Schedule of p_1 **:**



The Hold & Combine Algorithm

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

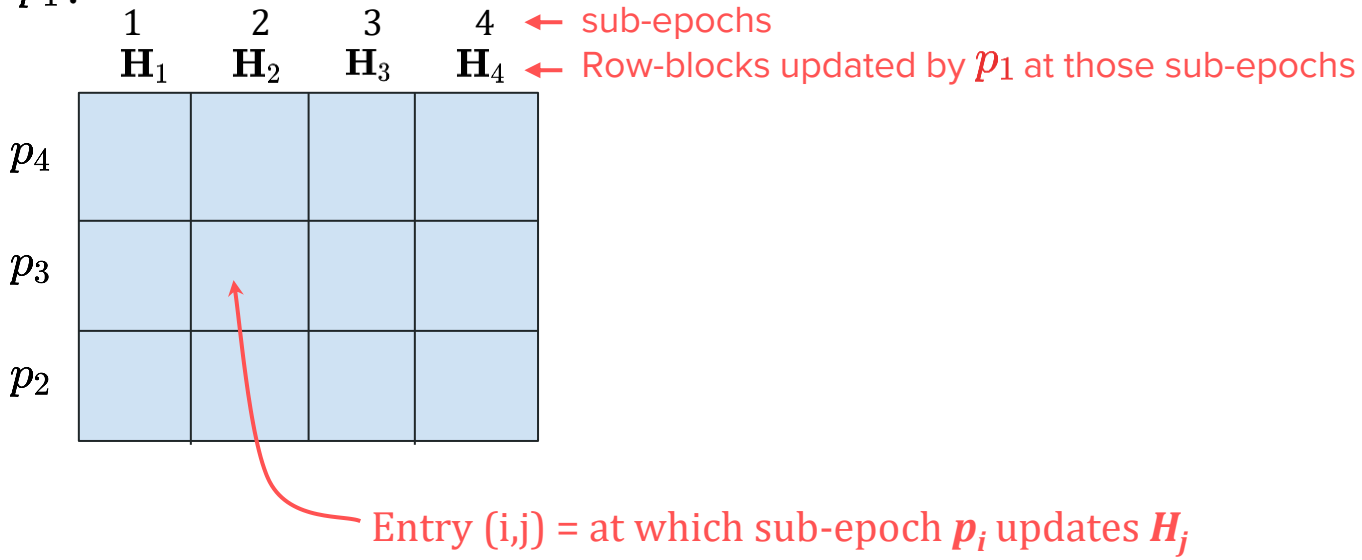
Schedule of p_1 :



The Hold & Combine Algorithm

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

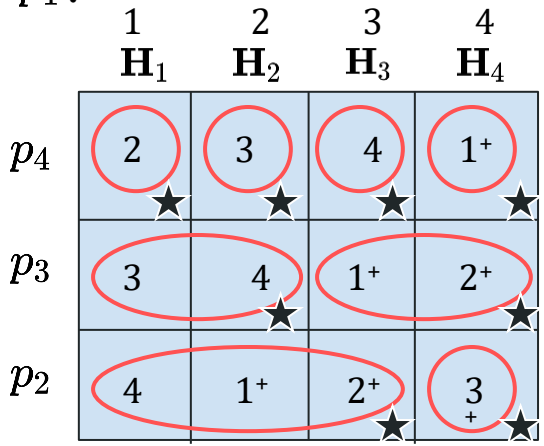
Schedule of p_1 :

	1 $\mathbf{H}_1$	2 $\mathbf{H}_2$	3 $\mathbf{H}_3$	4 $\mathbf{H}_4$
p_4	2	3	4	1^+
p_3	3	4	1^+	2^+
p_2	4	1^+	2^+	3^+

The Hold & Combine Algorithm

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

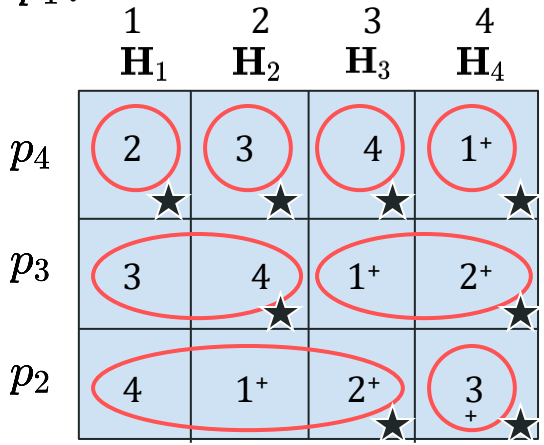
Schedule of p_1 :



The Hold & Combine Algorithm

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

Schedule of p_1 :

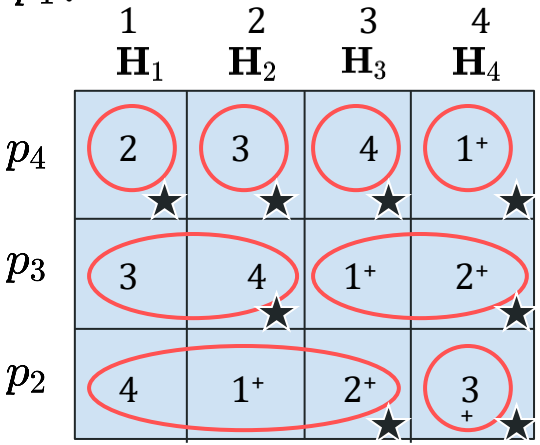


$$\sum_{i=1}^{\frac{K}{2}} 2 + \sum_{i=1}^{\frac{K-1}{2}} \frac{K}{i}$$

The Hold & Combine Algorithm

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

Schedule of p_1 :



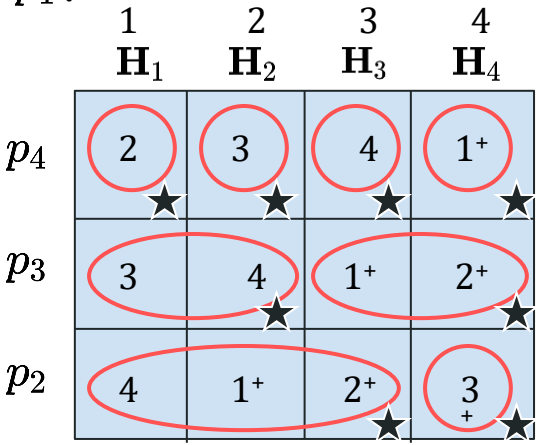
$$\sum_{i=1}^{\frac{K}{2}} 2 + \sum_{i=1}^{\frac{K-1}{2}} \frac{K}{i}$$

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

The Hold & Combine Algorithm

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

Schedule of p_1 :



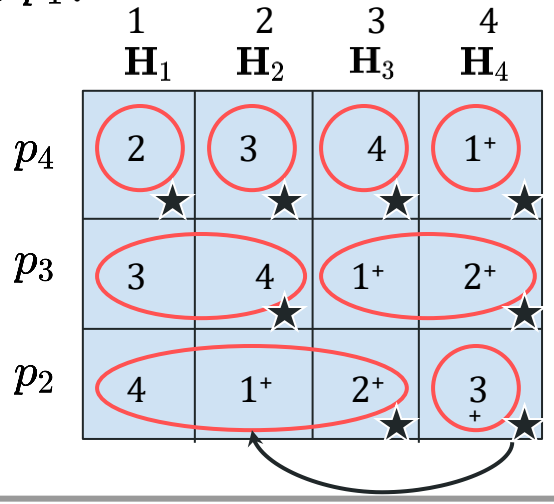
$$\sum_{i=1}^{\frac{K}{2}} 2 + \sum_{i=1}^{\frac{K-1}{2}} \frac{K}{i}$$

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

The Hold & Combine Algorithm

Key idea: Hold H-matrix rows 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)

Greedy choice: Send each message in the iteration just before it is required → leads to balanced distribution of messages among sub-epochs

Experiments and Key Results

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

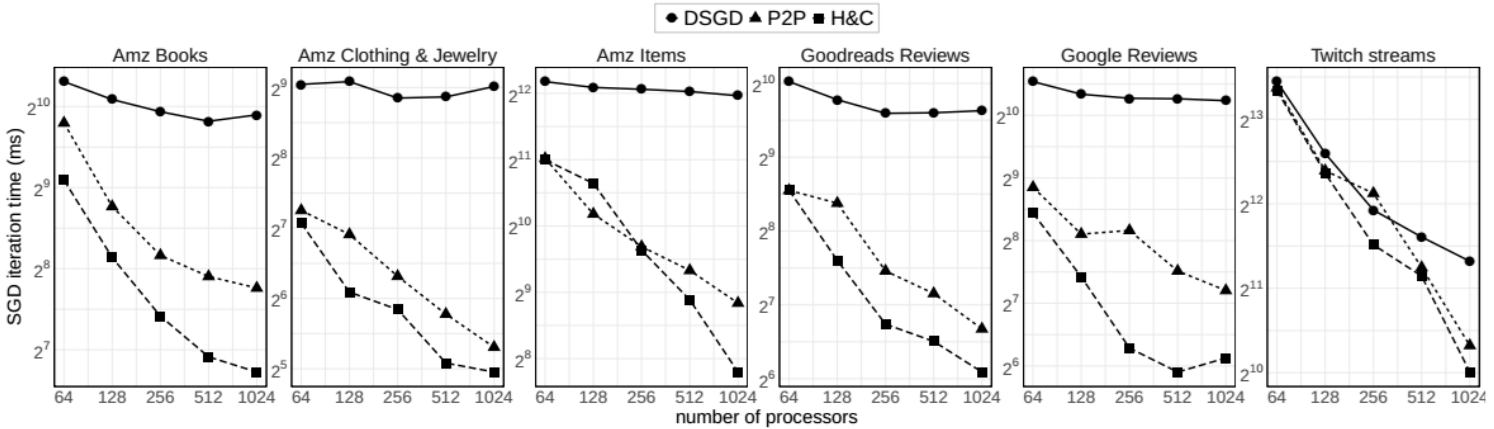
In terms of bandwidth:

- Using P2P **reduces** total volume by **10x - 120x** (P2P has the same volume with or w/o H&C)

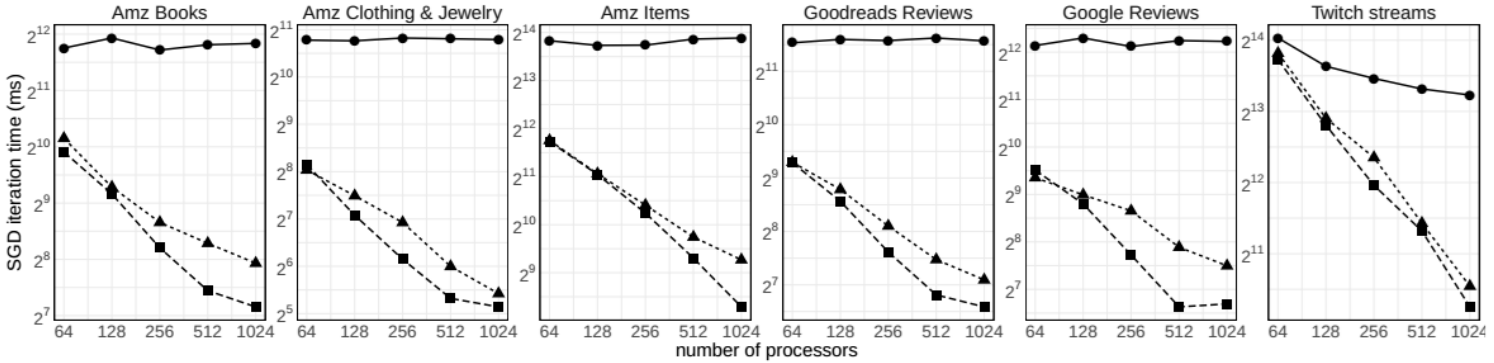
In terms of latency:

- Using random-based P2P **increases** total messages by **3.5x - 57x**
- Using **H&C increases** total messages by **3x - 8x**

Experiments and Key Results - Cont'd

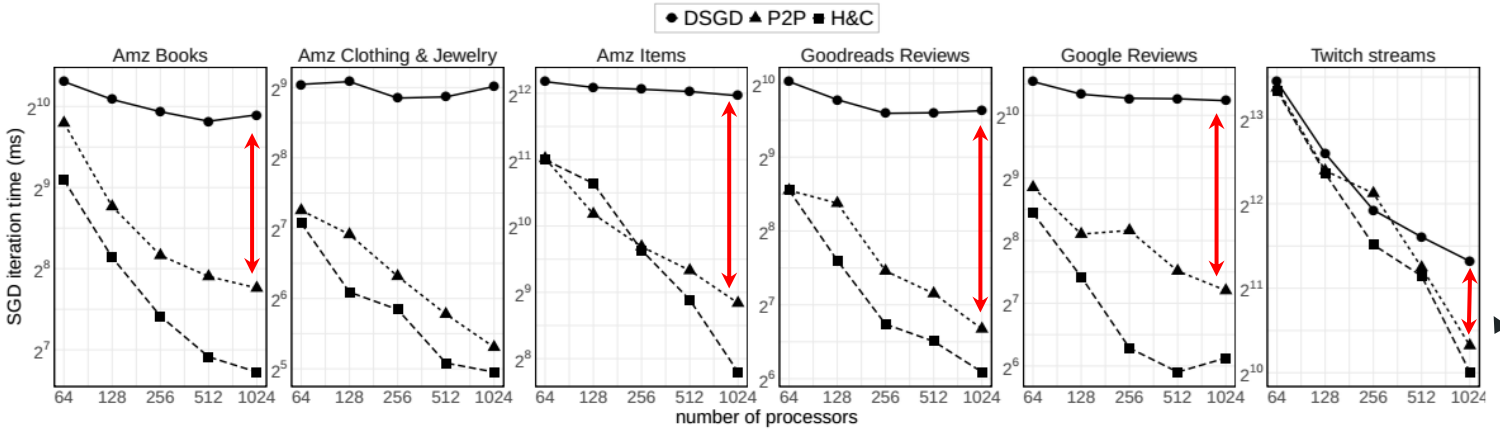


(a) $F = 16$

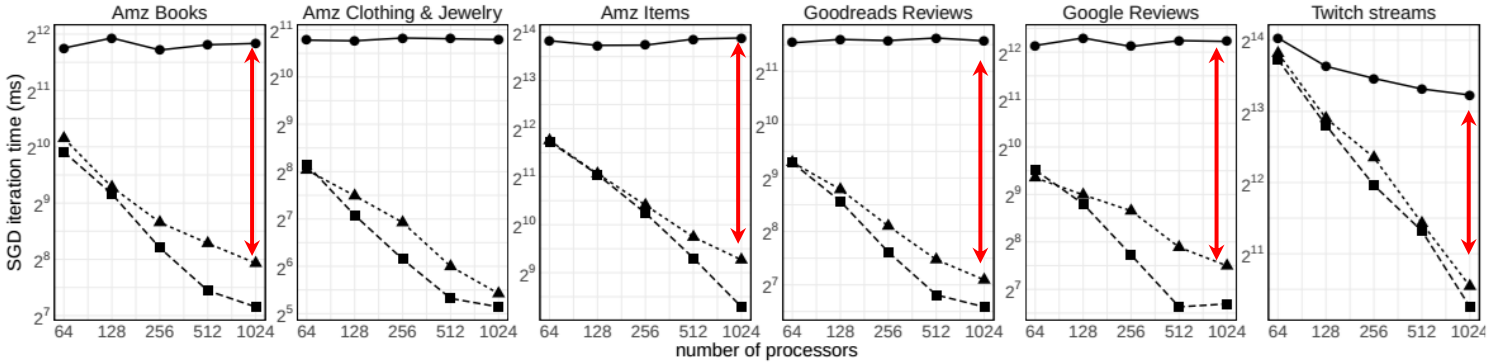


(b) $F = 64$

Experiments and Key Results - Cont'd



(a) $F = 16$



(b) $F = 64$

Gap increase

Thank you!

Questions ?

More of SPCL's research:

youtube.com/@spcl — 180+ Talks

twitter.com/spcl_eth — 1.4K+ Followers

github.com/spcl — 3.8K+ Stars

... or spcl.ethz.ch