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

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

Communication-Efficient Stratified SGD

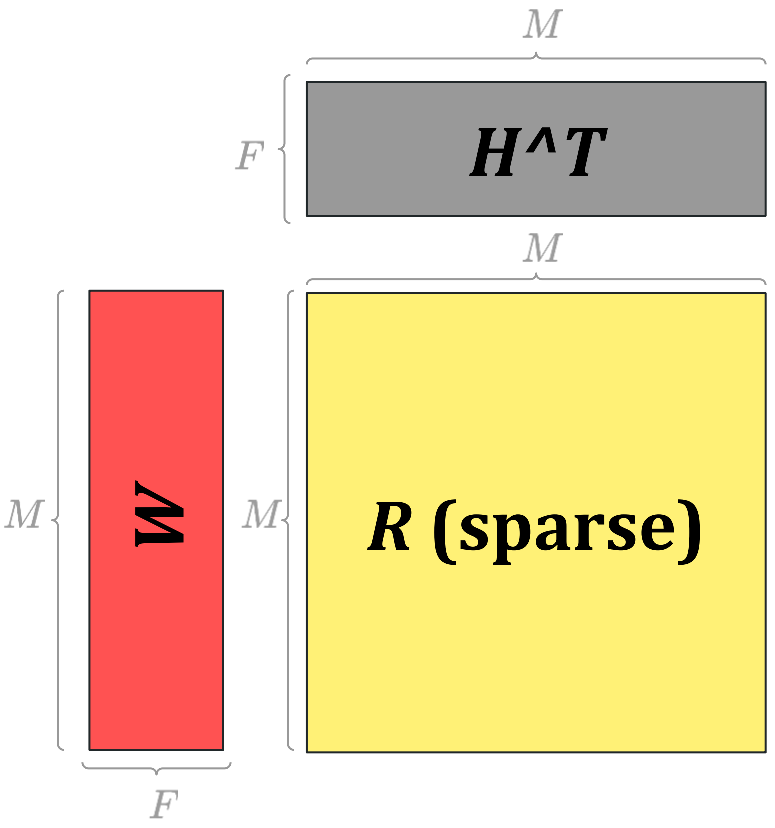
Nabil Abubaker

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 $\boldsymbol{R} \cong \boldsymbol{W}\boldsymbol{H}^{\mathrm{T}}$

A missing entry $\boldsymbol{r_{ij}}$ in $\boldsymbol{R}$ can be approximated (completed) by $\boldsymbol{r_{ij} = w_i h_j}^{\mathrm{T}}$

How?

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

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

By:

Step size

Regularization factor

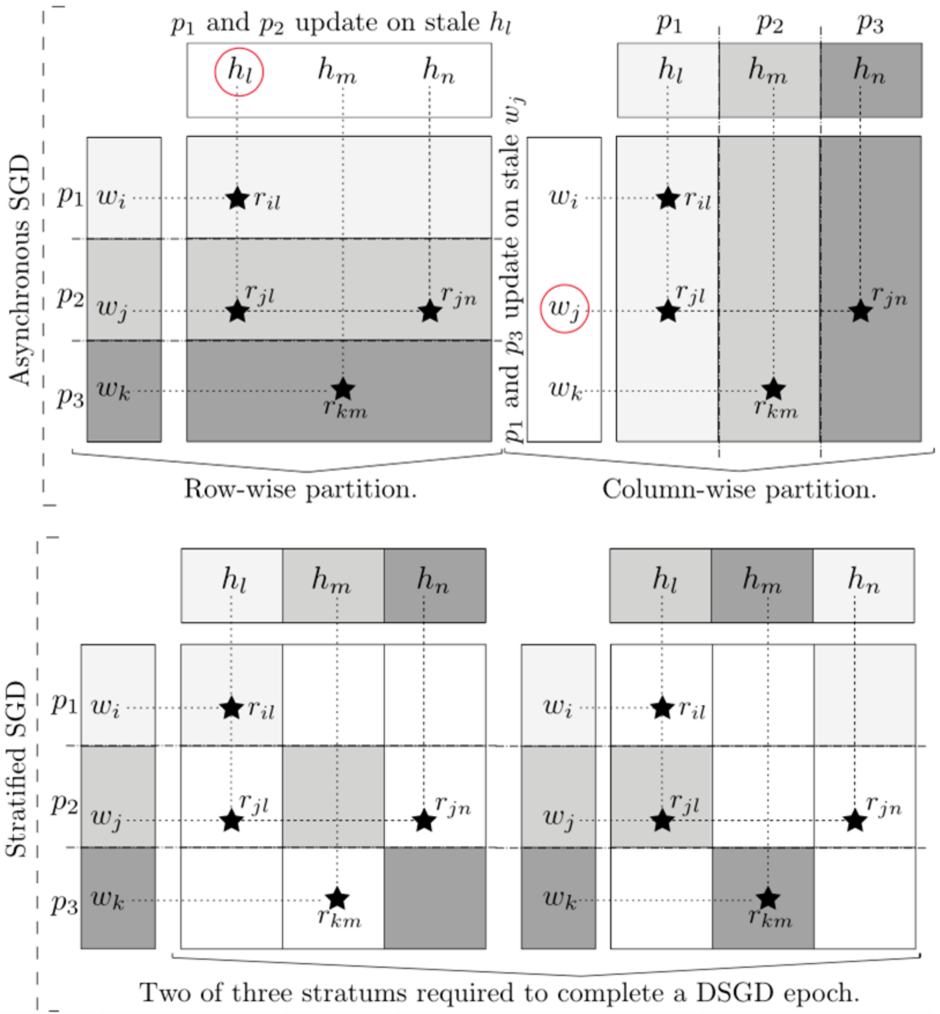
$$\mathbf{w}_i = \mathbf{w}_i + \epsilon\, ((\mathbf{r}_{ij} - \mathbf{w}_i \mathbf{h}_j^{\mathrm{T}})\mathbf{h}_j + \gamma\, \mathbf{w}_i)$$

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

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.



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

$\boxed{p_1\ /s1}$	$p_1\ /s2$	$p_1\ /s3$	$p_1\ /s4$
$p_2\ /s4$	$\boxed{p_2\ /s1}$	$p_2\ /s2$	$p_2\ /s3$
$p_3\ /s3$	$p_3\ /s4$	$\boxed{p_3\ /s1}$	$p_3\ /s2$
$p_4\ /s2$	$p_4\ /s3$	$p_4\ /s4$	$\boxed{p_4\ /s1}$

s2

$p_1\ /s1$	$\boxed{p_1\ /s2}$	$p_1\ /s3$	$p_1\ /s4$
$p_2\ /s4$	$p_2\ /s1$	$\boxed{p_2\ /s2}$	$p_2\ /s3$
$p_3\ /s3$	$p_3\ /s4$	$p_3\ /s1$	$\boxed{p_3\ /s2}$
$\boxed{p_4\ /s2}$	$p_4\ /s3$	$p_4\ /s4$	$p_4\ /s1$

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

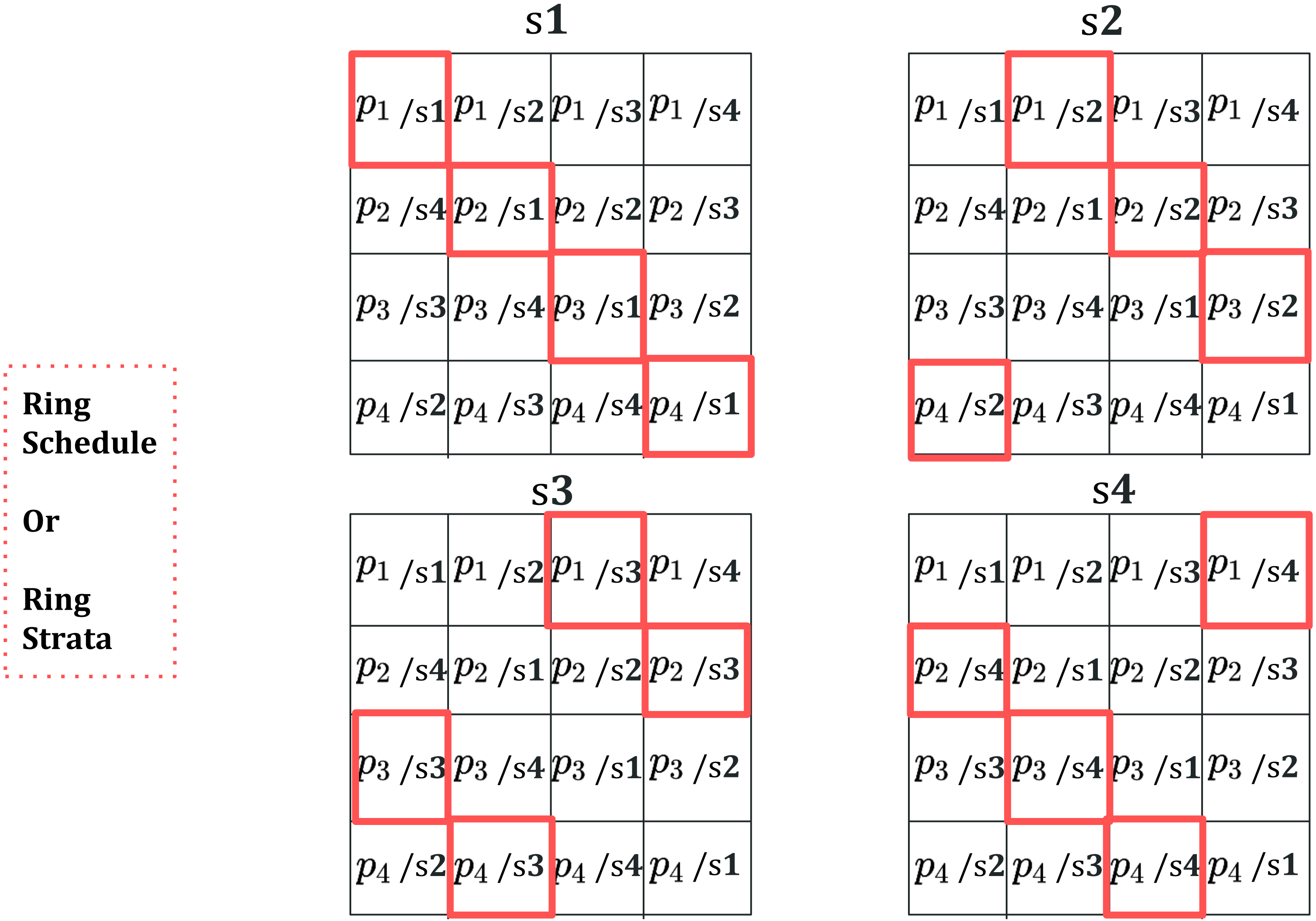
p_1 /s1	p_1 /s2	p_1 /s3	p_1 /s4
p_2 /s4	p_2 /s1	p_2 /s2	p_2 /s3
p_3 /s3	p_3 /s4	p_3 /s1	p_3 /s2
p_4 /s2	p_4 /s3	p_4 /s4	p_4 /s1

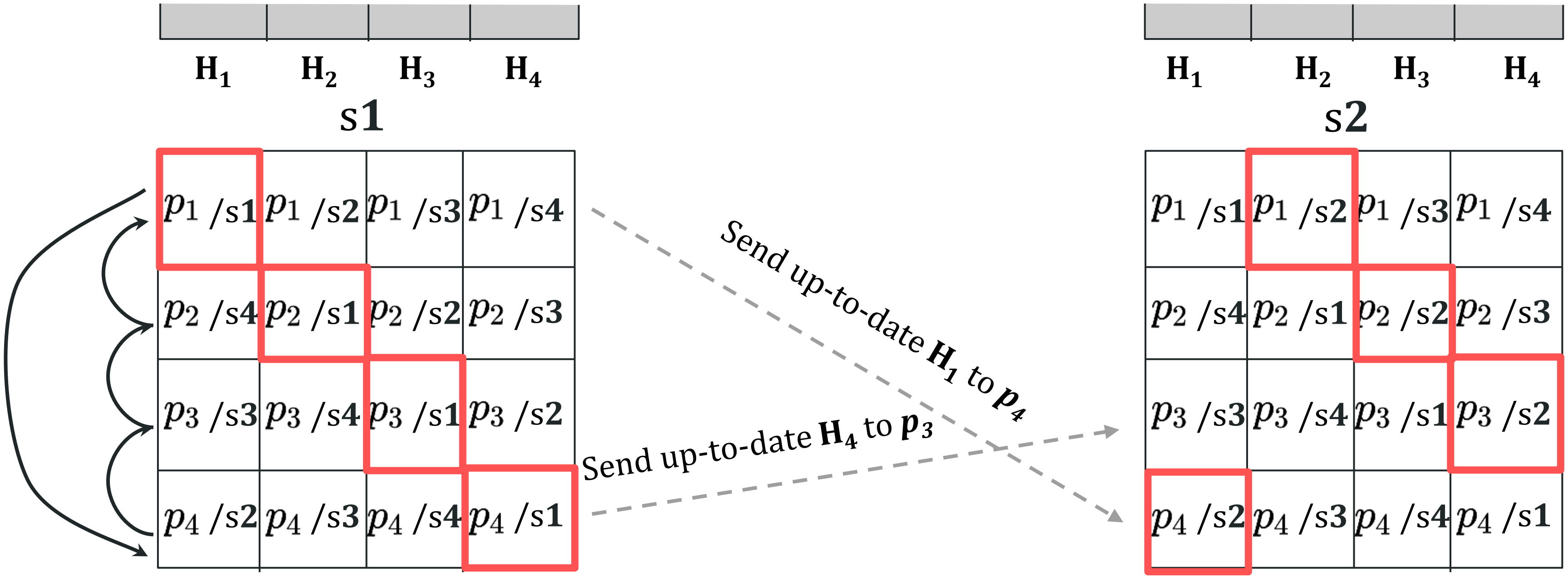
p_1 /s1	p_1 /s2	p_1 /s3	p_1 /s4
p_2 /s4	p_2 /s1	p_2 /s2	p_2 /s3
p_3 /s3	p_3 /s4	p_3 /s1	p_3 /s2
p_4 /s2	p_4 /s3	p_4 /s4	p_4 /s1

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

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


Ring
Schedule
Or
Ring
Strata
s1
s2
s3
s4
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



$\mathbf{H_1}$ $\mathbf{H_2}$ $\mathbf{H_3}$ $\mathbf{H_4}$
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
Send up-to-date $\mathbf{H_1}$ to $\boldsymbol{p_4}$
Send up-to-date $\mathbf{H_4}$ to $\boldsymbol{p_3}$
$\mathbf{H_1}$ $\mathbf{H_2}$ $\mathbf{H_3}$ $\mathbf{H_4}$
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

Problem with distributed SSGD:

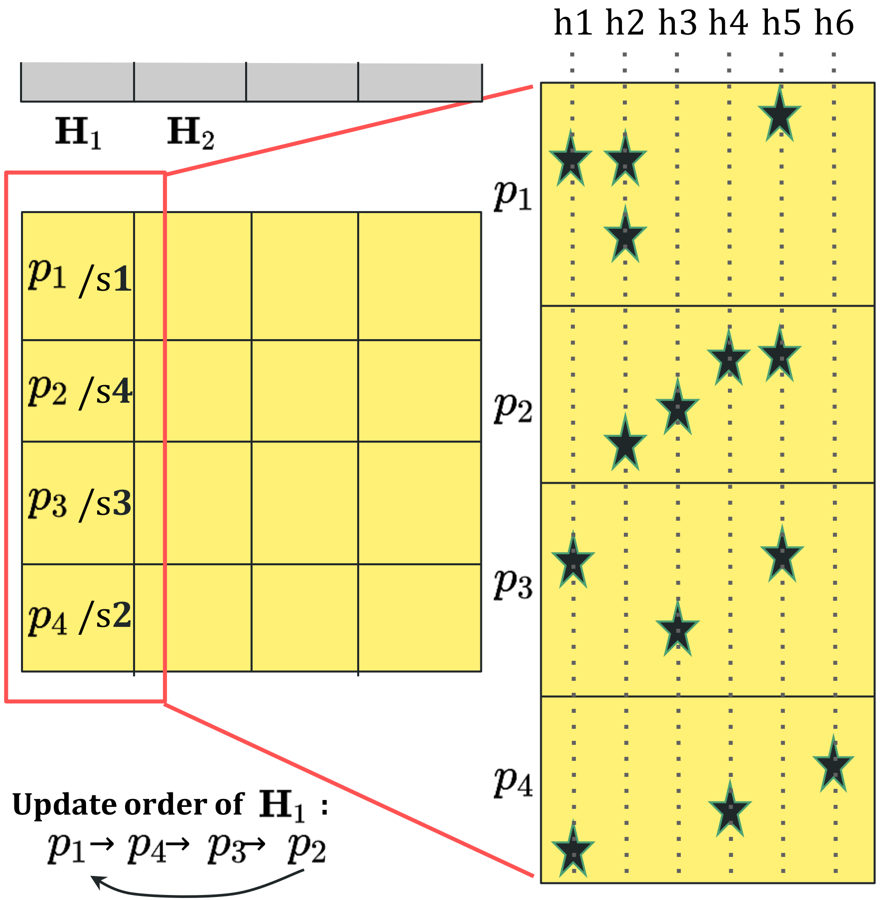
Existing implementations communicate blocks of factor matix rows:

- ➢ 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

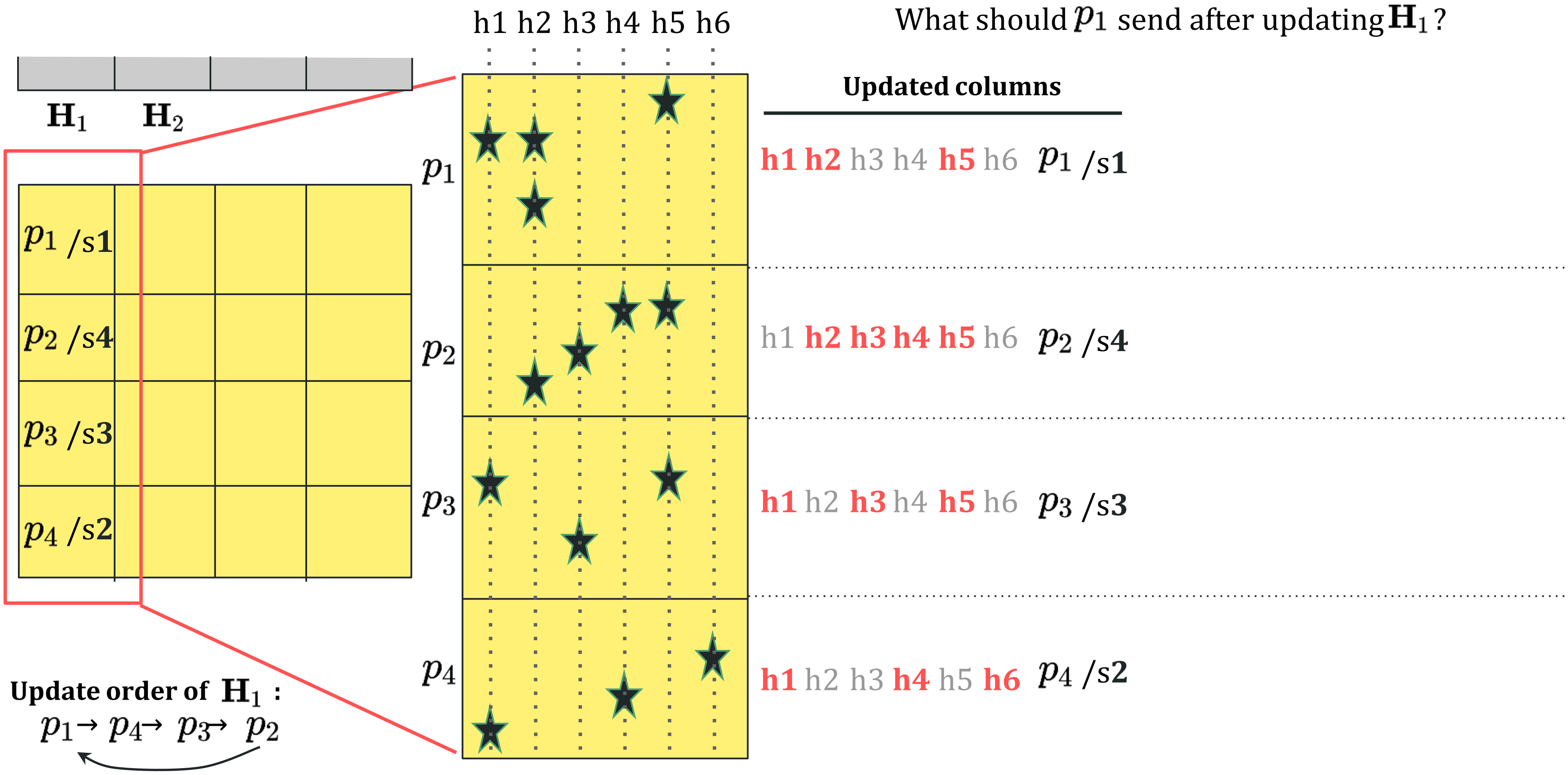


$\mathbf{h}_\mathrm{i}$ is sent from $\boldsymbol{p_x}$ to $\boldsymbol{p_y}$ if :

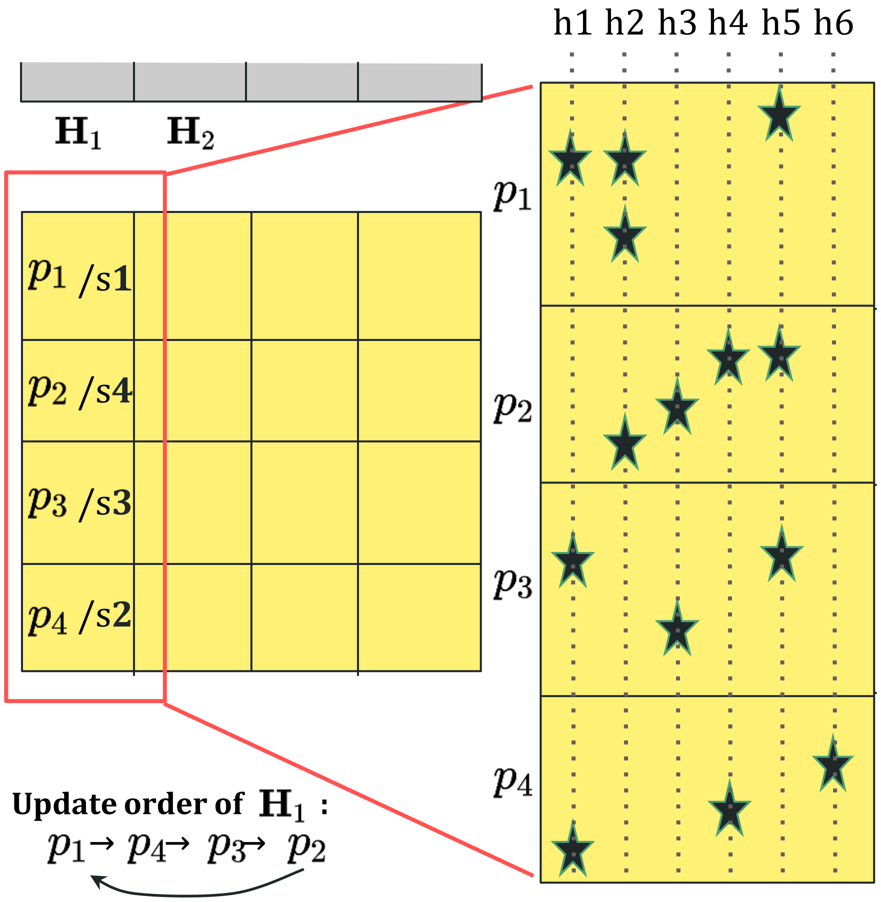
→ both $\boldsymbol{p_x}$ and $\boldsymbol{p_y}$ update $\mathbf{h}_\mathrm{i}$ and
→ **no** processor **in between** does.

According to the update order

Finding essential communication



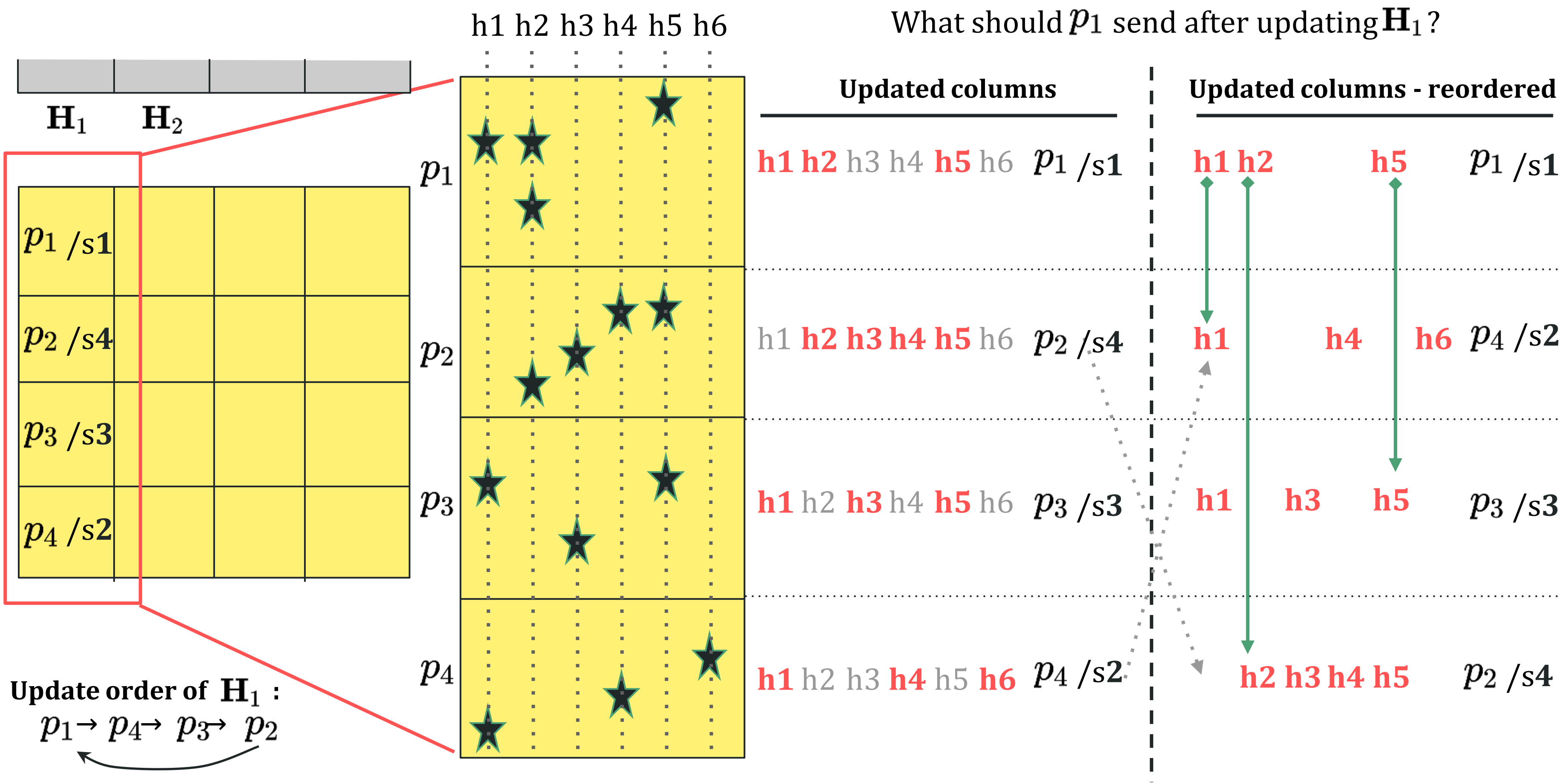
Finding essential communication



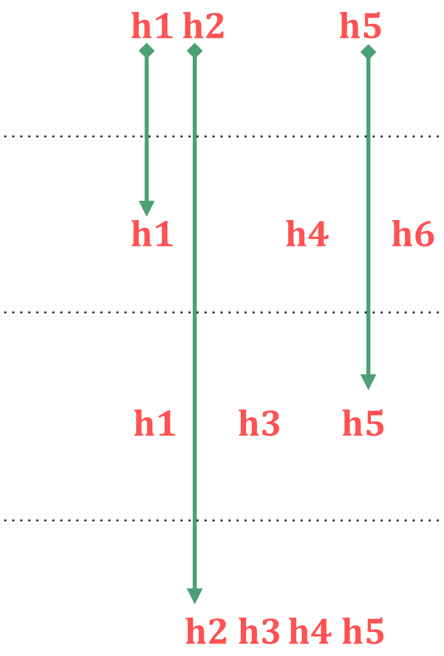
What should p_1 send after updating $\mathbf{H}_1$?

Updated columns		Updated columns - reordered	
h1 h2 h3 h4 h5 h6	p_1 /s1	h1 h2 h5	p_1 /s1
h1 h2 h3 h4 h5 h6	p_2 /s4	h1 h4 h6	p_4 /s2
h1 h2 h3 h4 h5 h6	p_3 /s3	h1 h3 h5	p_3 /s3
h1 h2 h3 h4 h5 h6	p_4 /s2	h2 h3 h4 h5	p_2 /s4

Finding essential communication



Efficiently finding essential communication



	h1	h2	h3	h4	h5	h6
B_1	1	1	0	0	1	0
B_4	1	0	0	1	0	1
B_2	1	0	1	0	1	0
B_3	0	1	1	1	1	0

$$\overbrace{B_x \wedge B_y}^{\text{Updated by } p_x \textbf{ AND } p_y} \overbrace{\oplus}^{\text{but } \textbf{not}} \overbrace{(B_x \wedge B_y \wedge \underbrace{(B_{x+1} \vee B_{x+2} \cdots B_{y-1})}_{\text{Built up incrementally}})}^{\text{any processor in between}}$$

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.

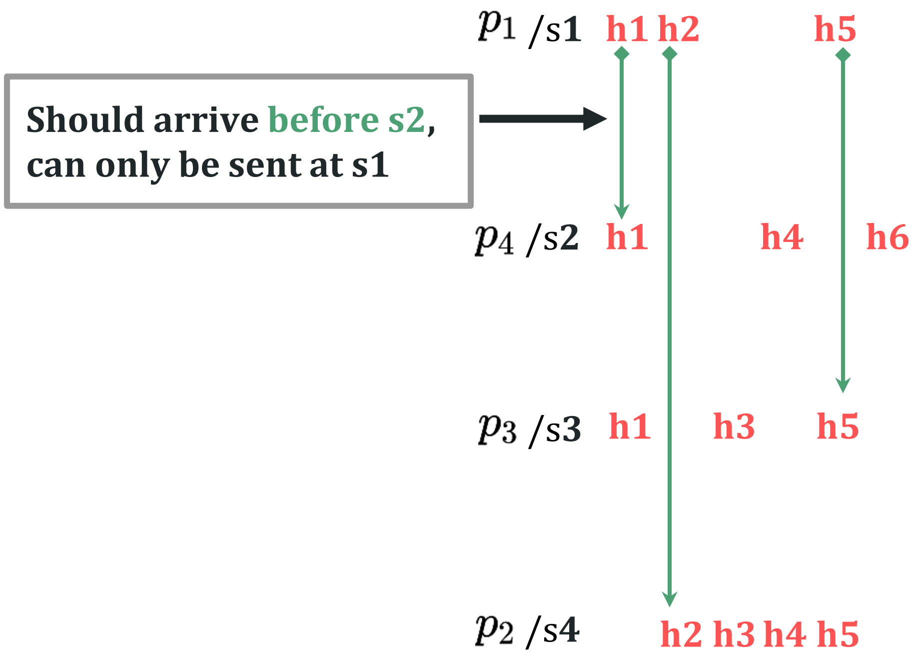
Research Question

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

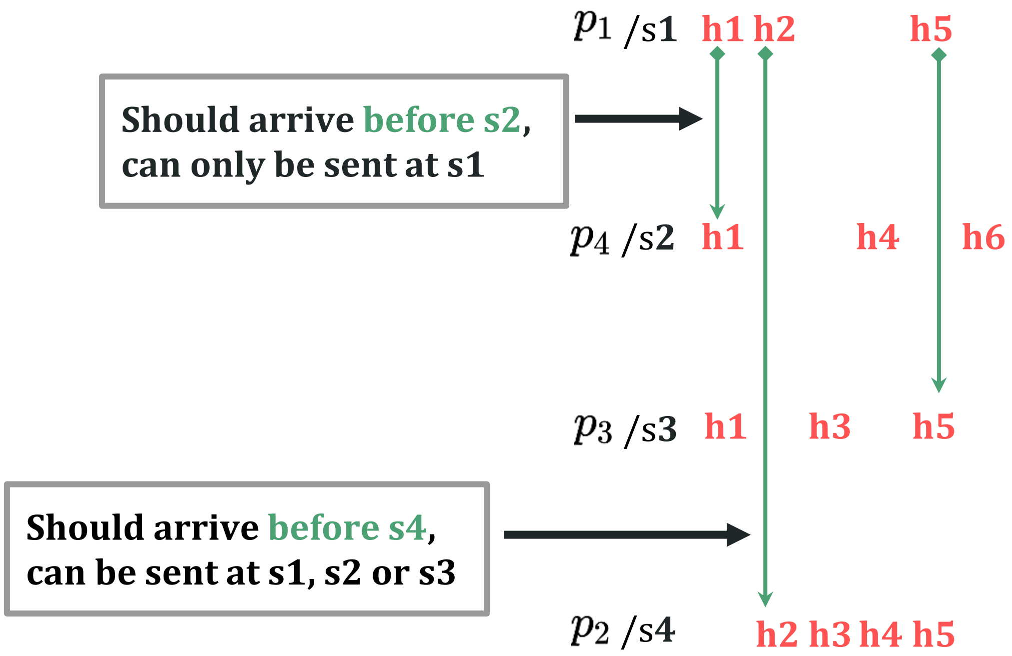
Key Observation

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

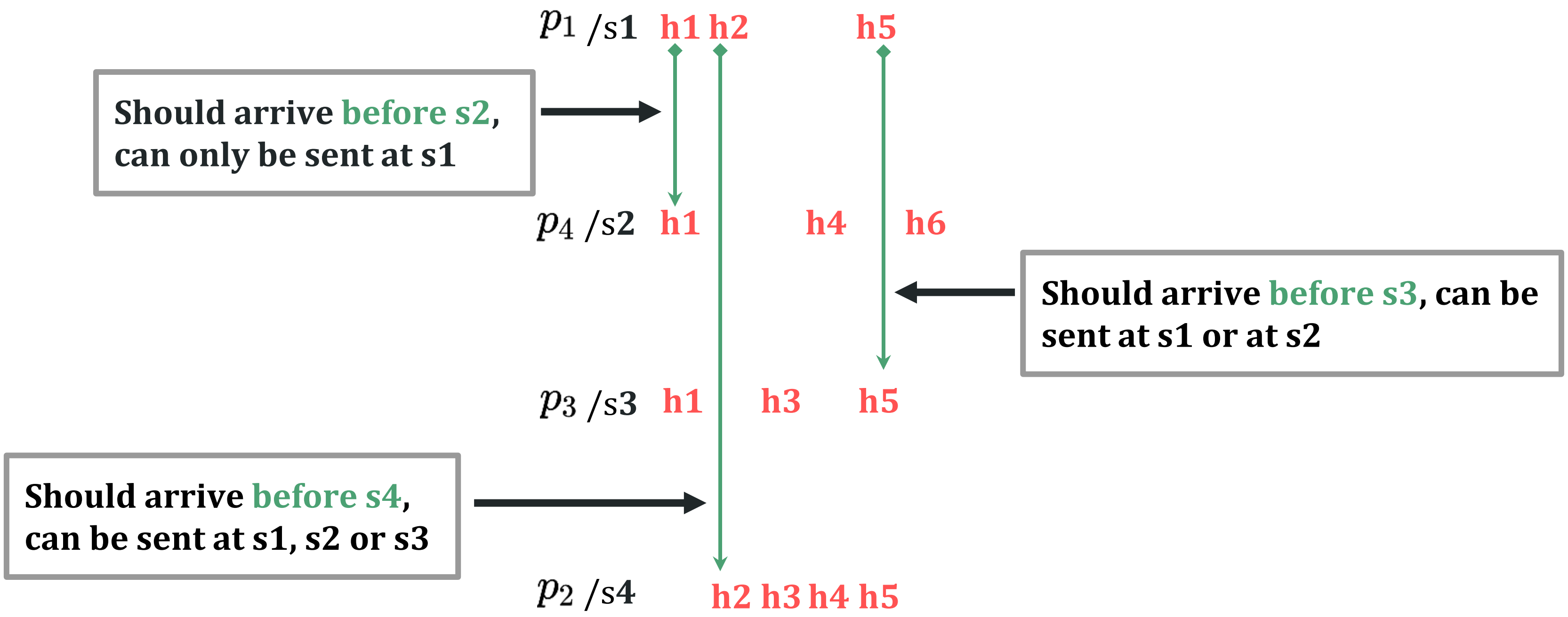
Key Observation: A closer look



Key Observation: A closer look



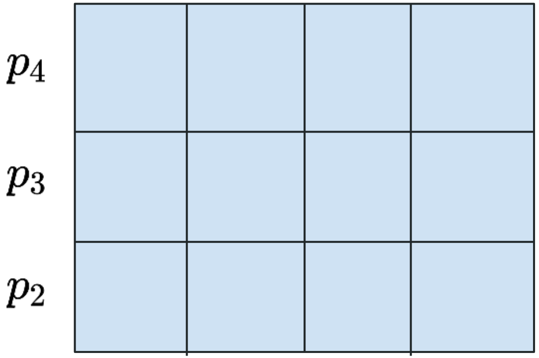
Key Observation: A closer look



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 :

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

← sub-epochs

← Row-blocks updated by p_1 at those sub-epochs

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

← sub-epochs

← Row-blocks updated by p_1 at those sub-epochs

Entry (i,j) = at which sub-epoch $\boldsymbol{p_i}$ updates $\boldsymbol{H_j}$

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**

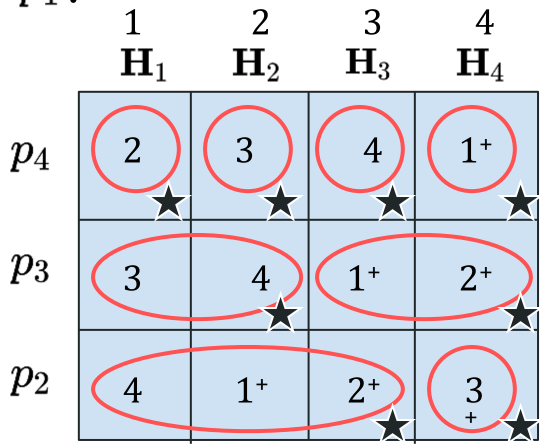
Schedule of p_1 **:**

	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**

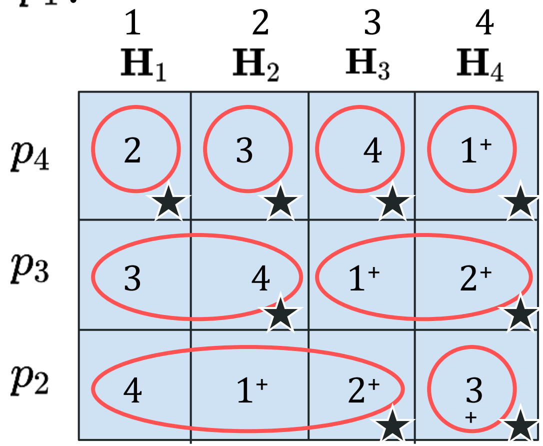
Schedule of p_1 :



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 :

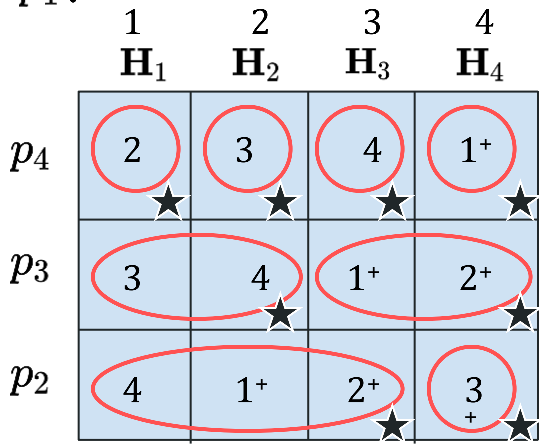


$$\sum_{i=1}^{\frac{K}{2}} 2 + \sum_{i=1}^{\frac{K-1}{2}} \frac{K}{i}$$

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 :



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

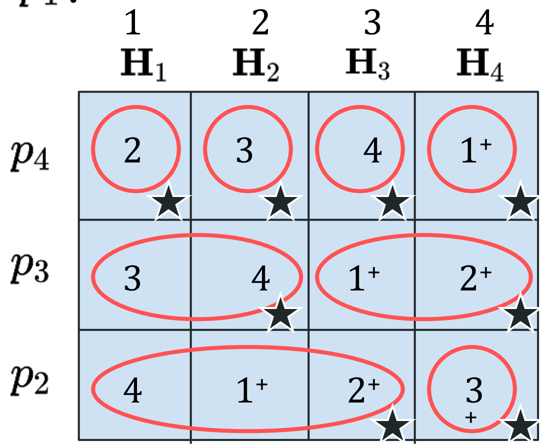
No H&C: up to 12 messages (N * N-1)

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 :



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

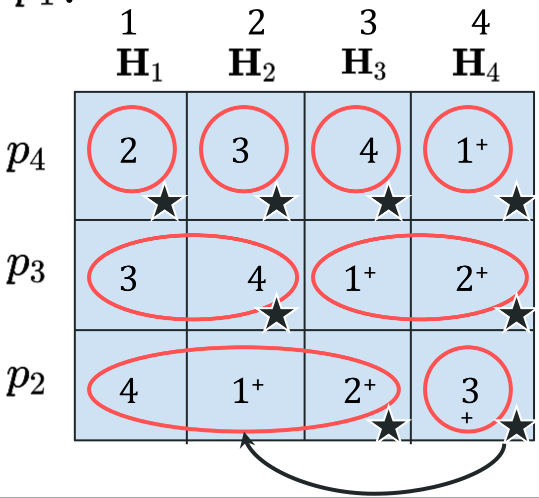
No H&C: up to 12 messages (N * N-1)

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

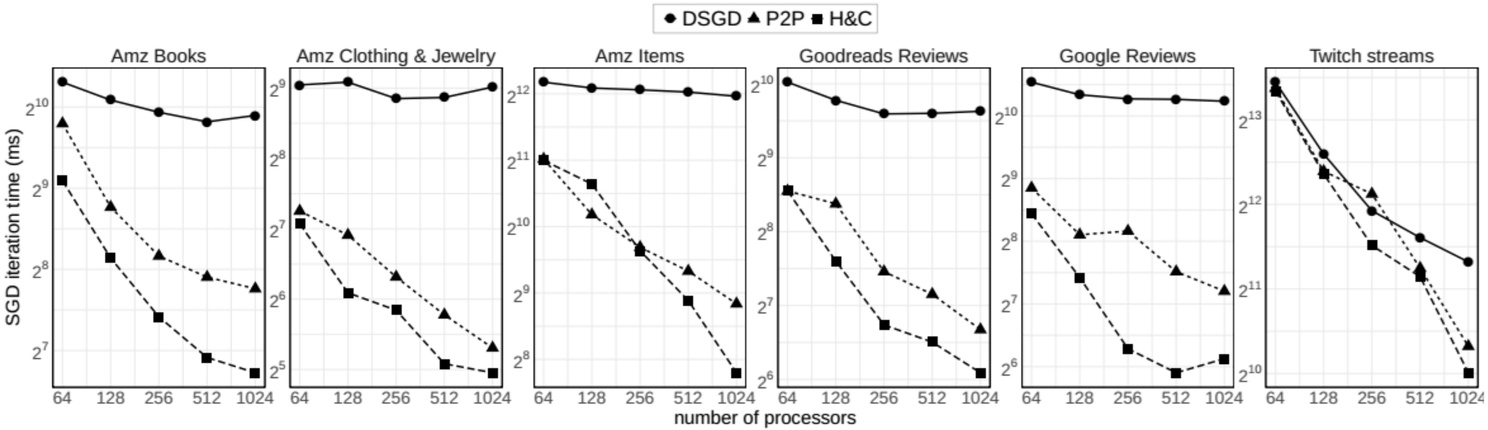
In terms of bandwidth:

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

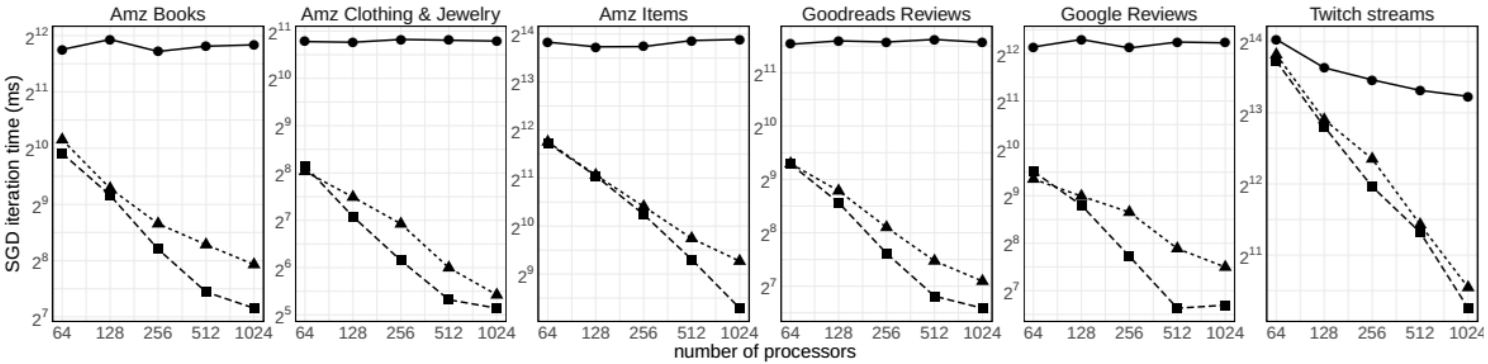
In terms of latency:

- ❖ 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

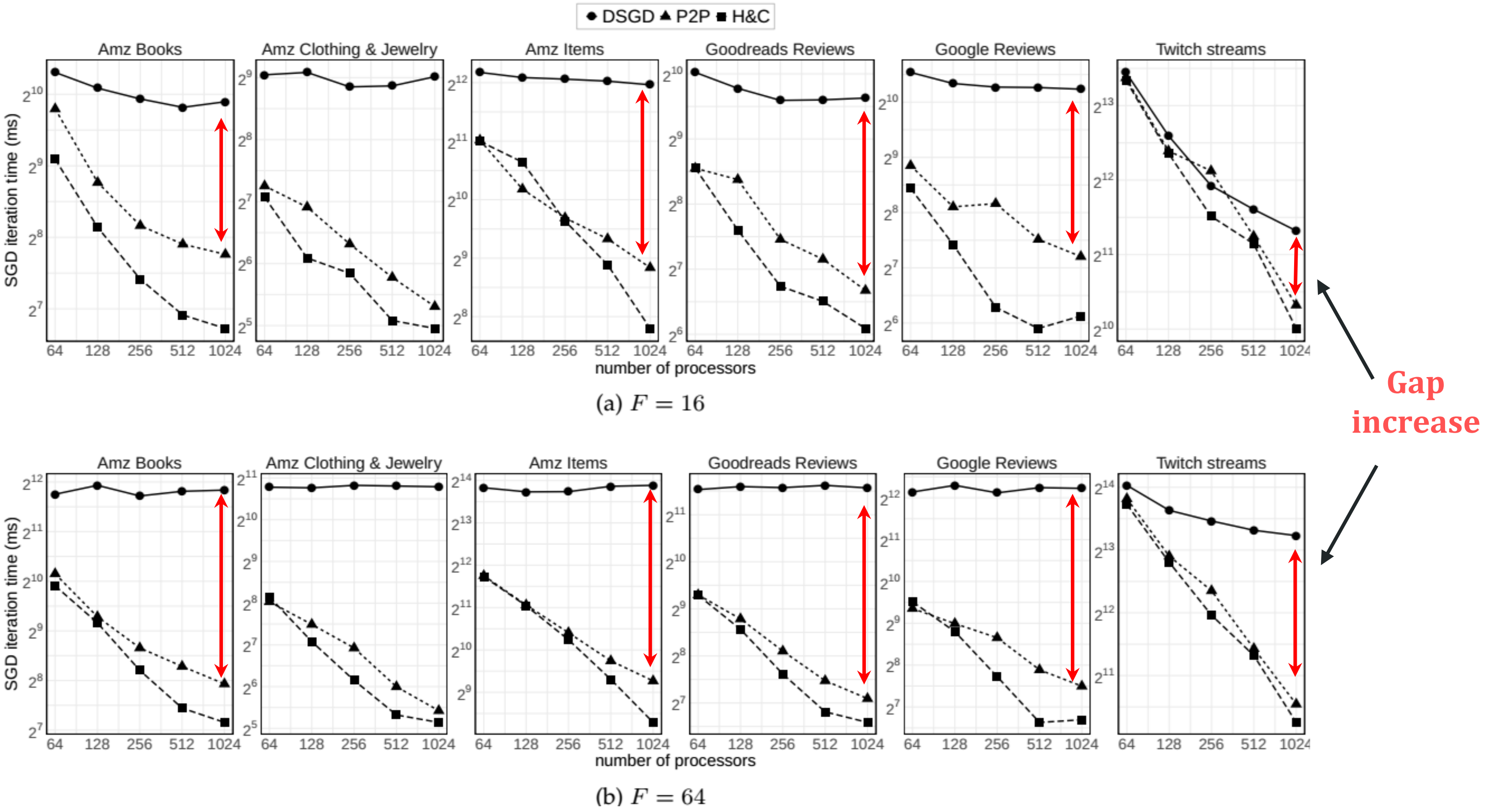


(a) $F = 16$



(b) $F = 64$

Experiments and Key Results - Cont'd



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