Communication-Avoiding Sparse Inverse Covariance Matrix Estimation

Penporn Koanantakool
CS Division, University of California, Berkeley

Joint work with:
Sang-Yun Oh, Dmitriy Morozov, Aydin Buluc,
Leonid Oliker, Katherine Yelick

SIAM AN17
July 14, 2017

In this talk, you WON'T see

lower bounds

proofs

"optimality"

You WILL see

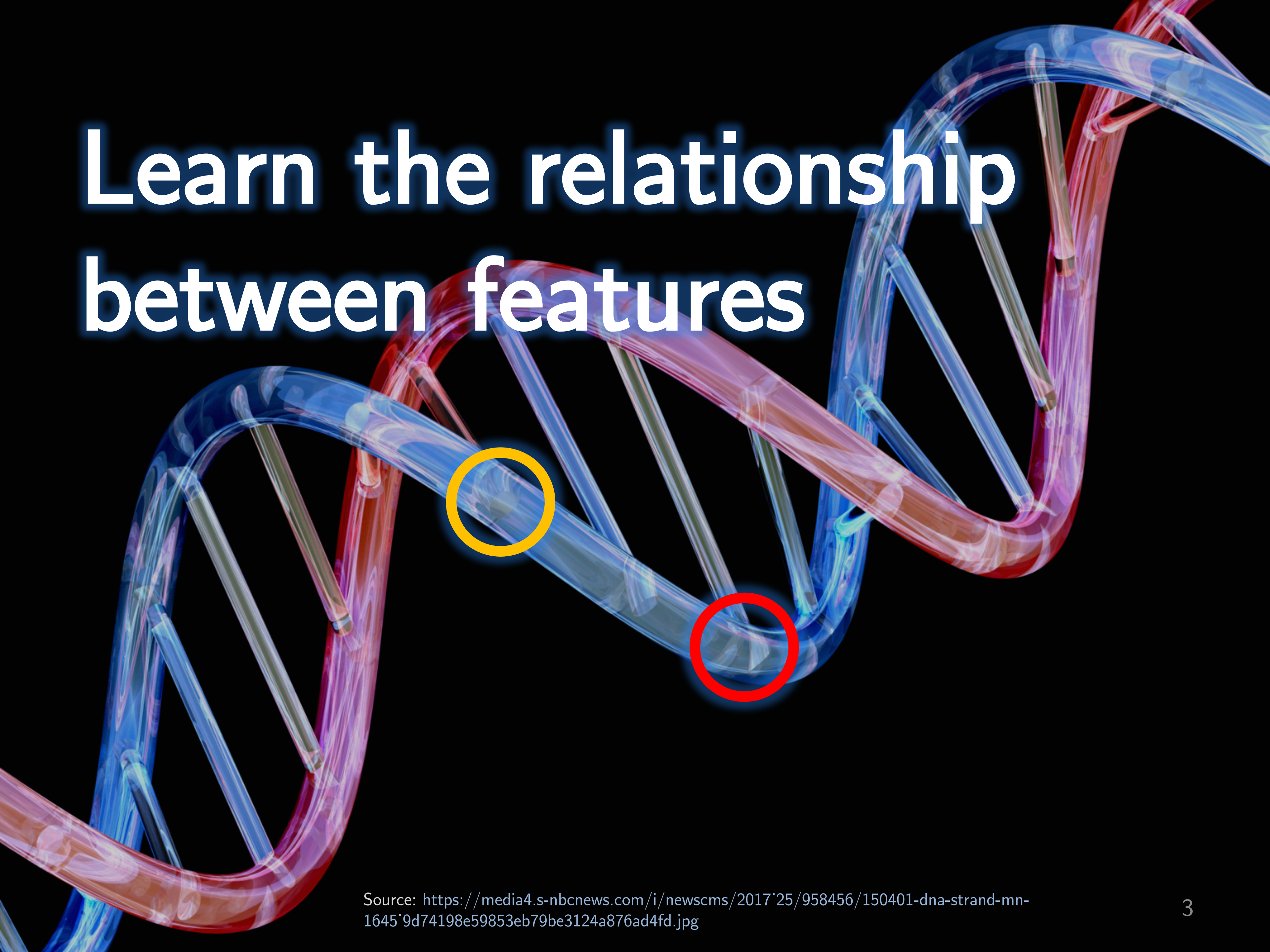
matmuls

$10\times$ speedup

comparisons

software release

Learn the relationship between features



Source: https://media4.s-nbcnews.com/i/newscms/2017˙25/958456/150401-dna-strand-mn-1645˙9d74198e59853eb79be3124a876ad4fd.jpg

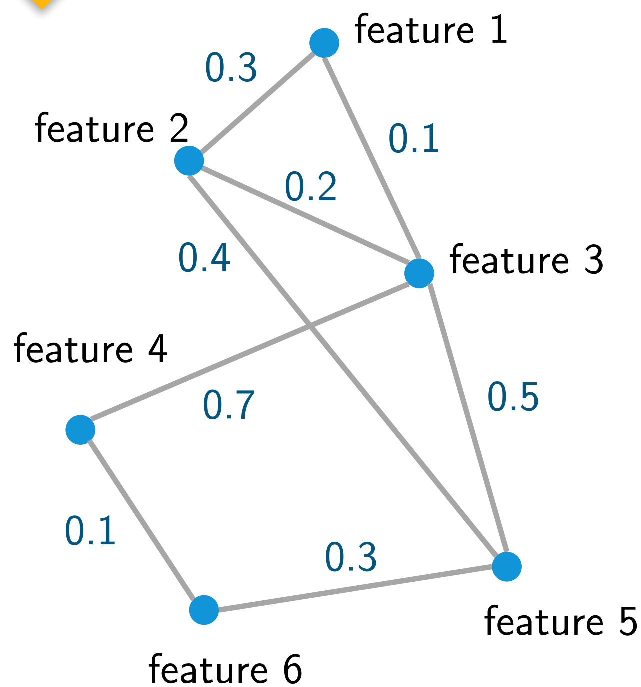
3D brain fMRI

Source: http://sites.cardiff.ac.uk/events/files/formidable/3d-brain-render.jpg

Inverse Covariance Matrix (Σ^{-1})(ICM)

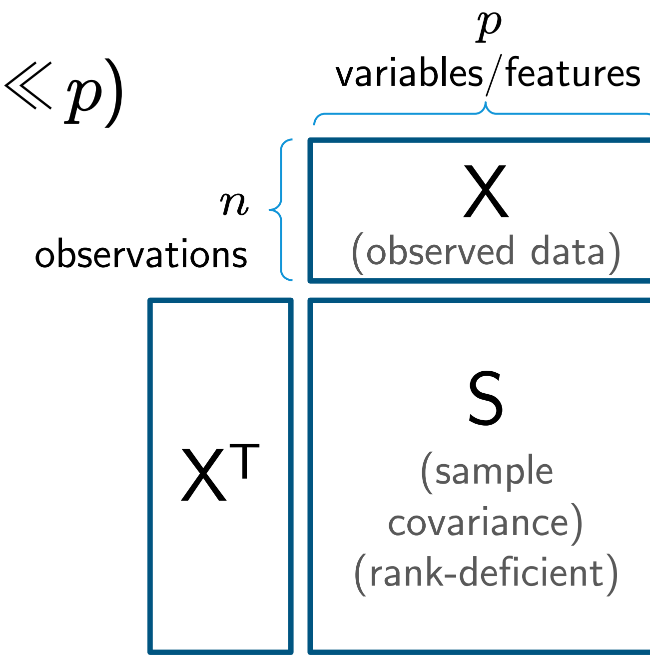
encodes relationships accounting for other variables (distinguishes direct vs indirect association)

Graphical Model



Sparse ICM Estimation

- Takes n observations of p features ($n \ll p$)
- Forms $S = X^T X$.
- Estimates S^{-1} (poor man's $\mathbf{\Sigma}^{-1}$).
  - Defines an objective function and finds the optimal matrix.
- Enforces sparsity.
  - Saves time.
  - Improves stability.



- Most implementations cannot handle > 20k variables.
- BigQUIC [Hsieh et al. 2013]
  - Up to 1M variables, but chain graph (degree $=$ 3).
  - Shared memory.

CONCORD-ISTA

- **CONCORD** objective function

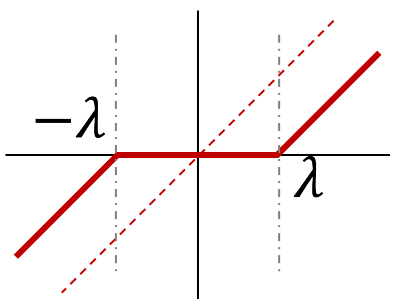
non-smooth

$$Q(\Omega) = \frac{n}{2}\left[-\log\det\Omega_{\mathrm{D}}^2 + \mathrm{tr}(S\Omega^2) + \lambda\|\Omega_{\mathrm{X}}\|_1\right]$$

Ω : estimated sparse inverse of S

Ω_D : diagonal elements
Ω_X : off-diagonal elements

- Proximal gradient method
  - **Smooth:** Gradient descent
  - **Non-smooth:** Soft-thresholding
    (**ISTA**: Iterative Soft-Thresholding Algorithm)



$$\mathcal{S}_{\Lambda}(\Omega) = \mathtt{sign}(\Omega)\max\{|\Omega| - \Lambda, 0\}$$

[Oh et al. 2014]

CONCORD-ISTA

Input: Observation $X^{n\times p}$, penalty $\Lambda^{p\times p}$.

Input: Penalty λ, convergence ϵ (constants).

Output: Estimated sparse ICM $\Omega^{p\times p}$

1: $S \leftarrow X^T X/n$ // Forms $p{\times}p$ sample covariance matrix.

2: $\Omega_0 \leftarrow I^{p\times p}$ // Initial guess.

3: $h_0 \leftarrow -\log\det(\Omega_0)_D + \frac{1}{2}\,\mathrm{tr}(\Omega_0 S \Omega_0) + \lambda\|\Omega_0\|_F^2$ // Objective function.

CONCORD-ISTA

Input: Observation $X^{n \times p}$, penalty $\Lambda^{p \times p}$.

Input: Penalty λ, convergence ϵ (constants).

Output: Estimated sparse ICM $\Omega^{p \times p}$

$S \leftarrow X^T X / n$ // Forms $p \times p$ sample covariance matrix.

$\Omega_0 \leftarrow I^{p \times p}$ // Initial guess.

$h_0 \leftarrow -\log\det(\Omega_0)_D + \frac{1}{2}\operatorname{tr}(\Omega_0 S \Omega_0) + \lambda \|\Omega_0\|_F^2$ // Objective function.

repeat $k \leftarrow 0, 1, 2, \ldots$

  $G \leftarrow -((\Omega_k)_D)^{-1} + \frac{1}{2}(S\Omega_k + \Omega_k S) + 2\lambda\Omega_k$ // Gradient.

until $\max|\Omega_{k+1} - \Omega_k| < \epsilon$

CONCORD-ISTA

Input: Observation $X^{n \times p}$, penalty $\Lambda^{p \times p}$.

Input: Penalty λ, convergence ϵ (constants).

Output: Estimated sparse ICM $\Omega^{p \times p}$

$S \leftarrow X^T X / n$ // Forms $p{\times}p$ sample covariance matrix.

$\Omega_0 \leftarrow I^{p \times p}$ // Initial guess.

$h_0 \leftarrow -\log\det(\Omega_0)_D + \frac{1}{2}\operatorname{tr}(\Omega_0 S \Omega_0) + \lambda \|\Omega_0\|_F^2$ // Objective function.

repeat $k \leftarrow 0, 1, 2, \ldots$

    $G \leftarrow -((\Omega_k)_D)^{-1} + \frac{1}{2}(S\Omega_k + \Omega_k S) + 2\lambda\Omega_k$ // Gradient.

    repeat $\tau \leftarrow 1, \frac{1}{2}, \frac{1}{4}, \ldots$ // Line-searches for appropriate step size.

        $\Omega_{k+1} \leftarrow \mathcal{S}_{\tau\Lambda}(\Omega_k - \tau G)$ // Computes new Ω.

        $h_{k+1} \leftarrow -\log\det(\Omega_{k+1})_D + \frac{1}{2}\operatorname{tr}(\Omega_{k+1} S \Omega_{k+1}) + \lambda \|\Omega_{k+1}\|_F^2$

        $q \leftarrow h_k + \operatorname{tr}((\Omega_{k+1} - \Omega_k)^T G) + \frac{1}{2\tau}\|\Omega_{k+1} - \Omega_k\|_F^2$

    until $h_{k+1} \leq q$ // Until descent condition satisfied.

until $\max|\Omega_{k+1} - \Omega_k| < \epsilon$

ONE recurring matmul

$S = X^T X / n$

for k = 0, 1, ... $\times 100$

  Transpose W

  for τ = 1, 0.5, ... $\times 10$

    $W = \Omega_{k+1} S$

TWO recurring matmuls

$\Omega S \rightarrow \Omega(X^T X) \rightarrow (\Omega X^T) X$

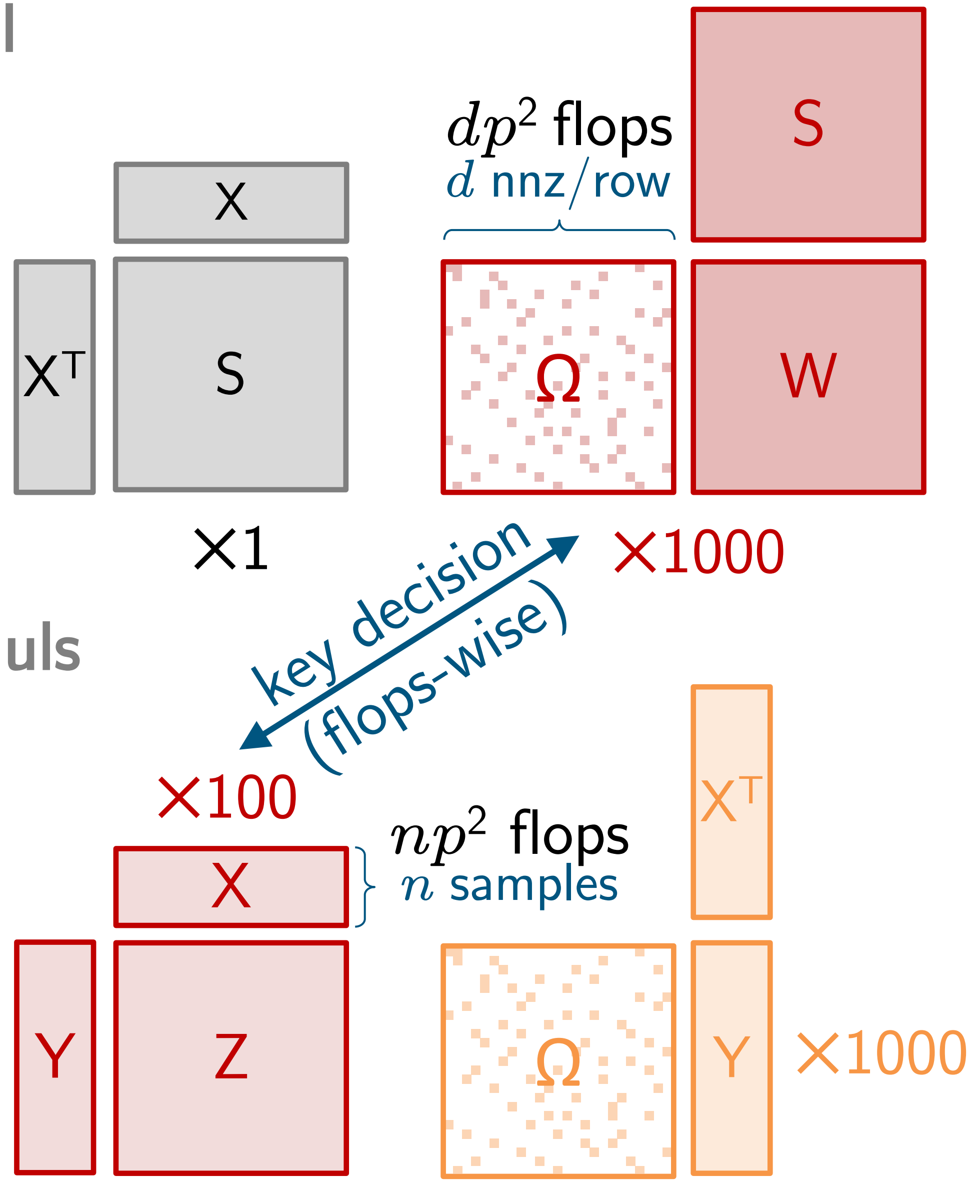
for k = 0, 1, ... $\times 100$

  $Z = YX$

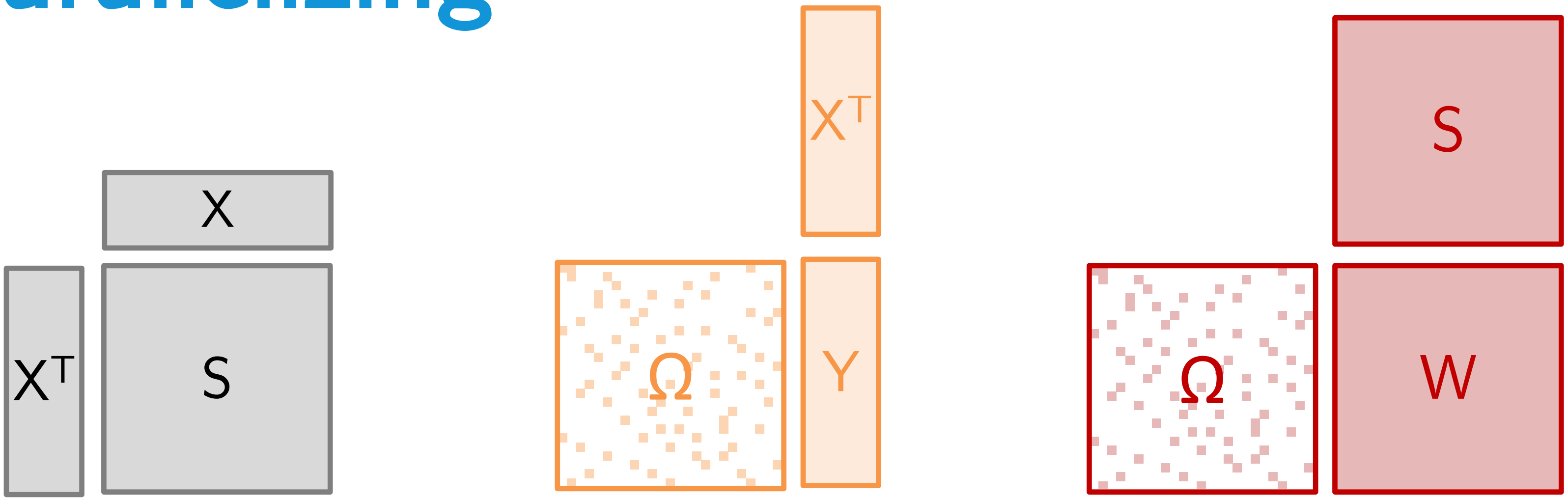
  Transpose Z

  for τ = 1, 0.5, ... $\times 10$

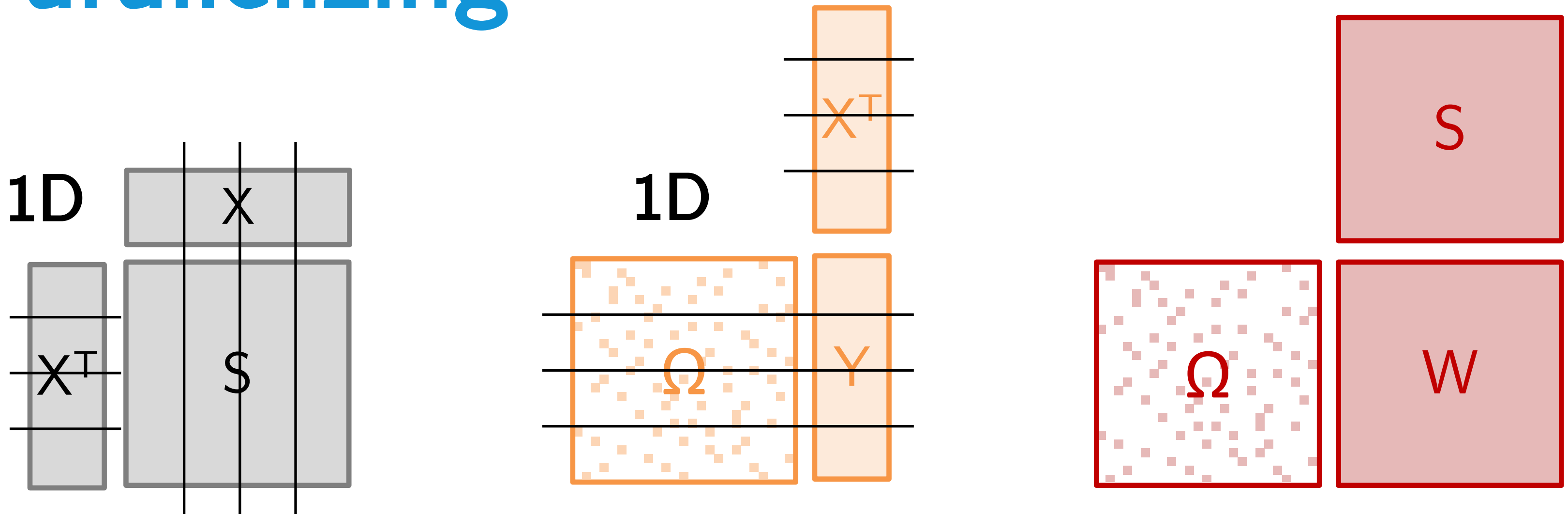
    $Y = \Omega_{k+1} X^T$



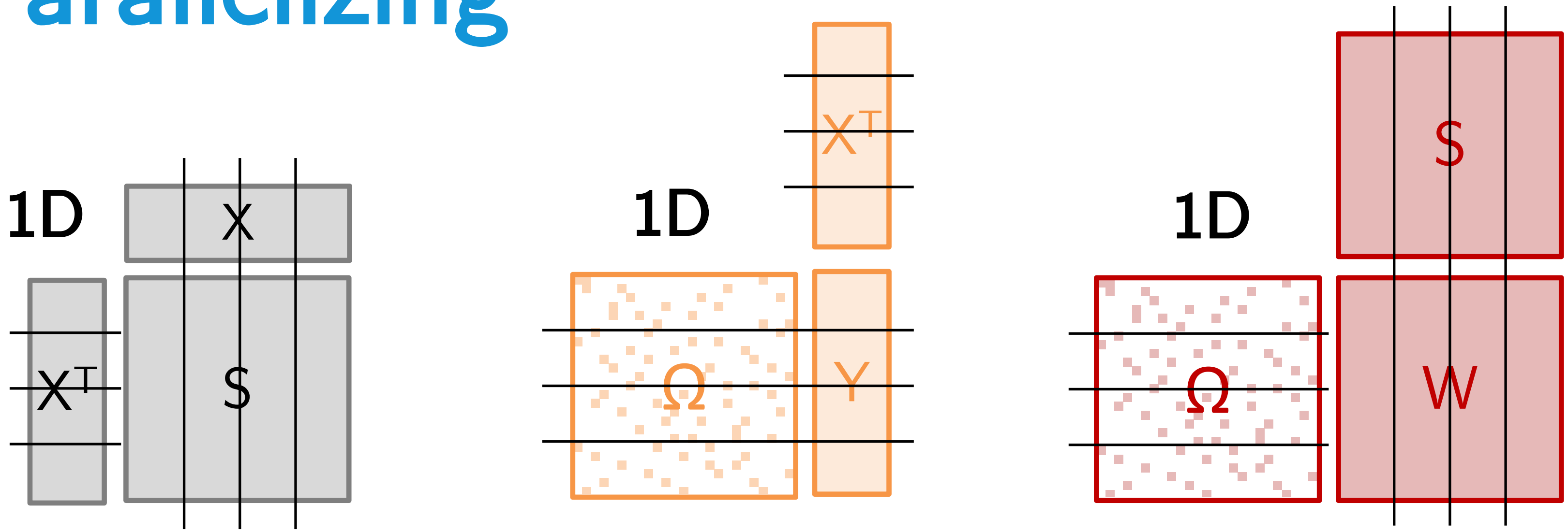
Parallelizing



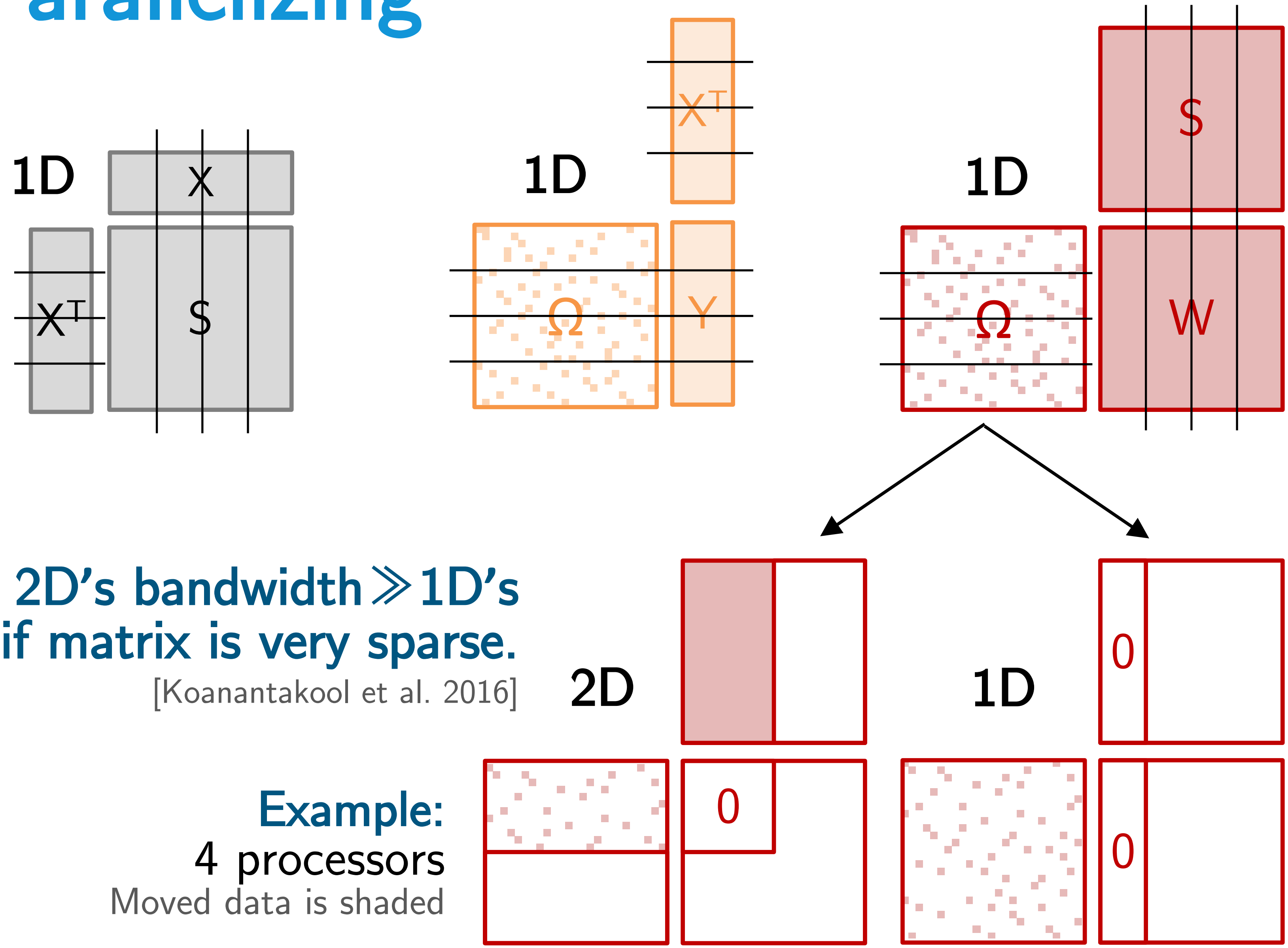
Parallelizing



Parallelizing



Parallelizing

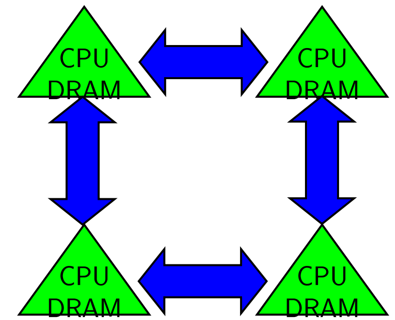


2D's bandwidth ≫ 1D's if matrix is very sparse.
[Koanantakool et al. 2016]

Example:
4 processors
Moved data is shaded

Parallel Algorithm Cost Model

Image courtesy of: James Demmel



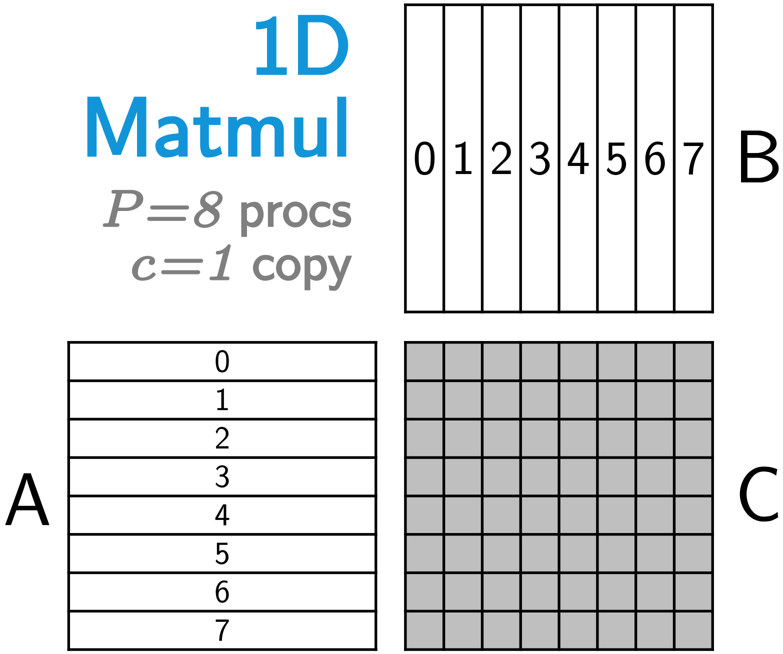
- P distributed, homogenous processors connected through network
- **Per-processor** costs along **critical path**

$$\text{time} = \underbrace{\#\text{flops} \cdot t_{\text{flop}}}_{\text{Computation}} + \underbrace{\#\text{messages} \cdot t_{\text{message}} + \#\text{words} \cdot t_{\text{word}}}_{\text{Communication}}$$

- Assume t_{flop}, t_{message}, t_{word} are machine-specific constants.
- Minimize #messages and #words

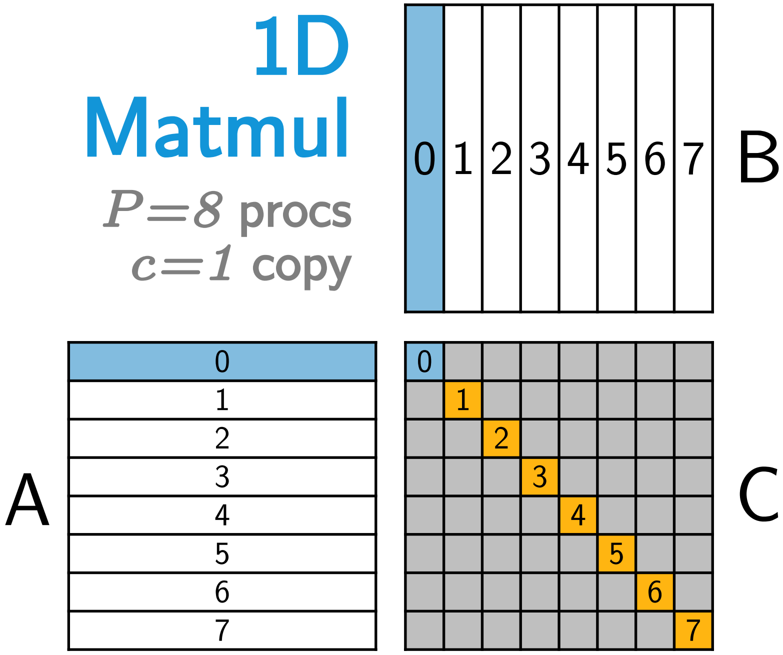
1D Matmul

$P = 8$ procs
$c = 1$ copy



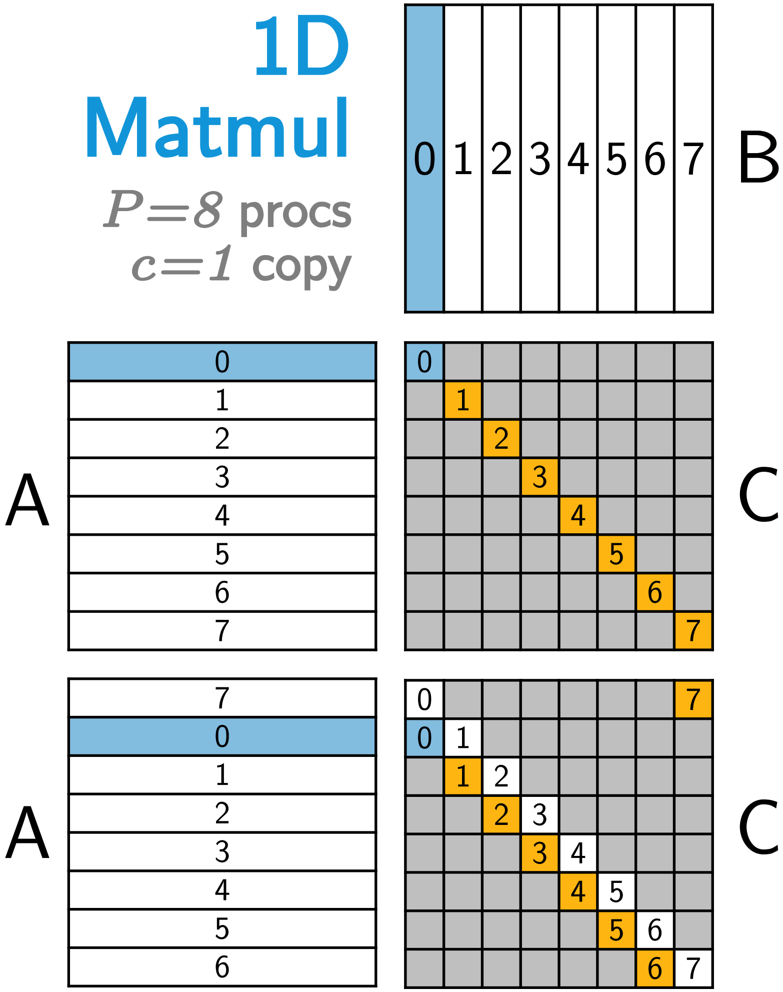
1D Matmul

P=8 procs
c=1 copy



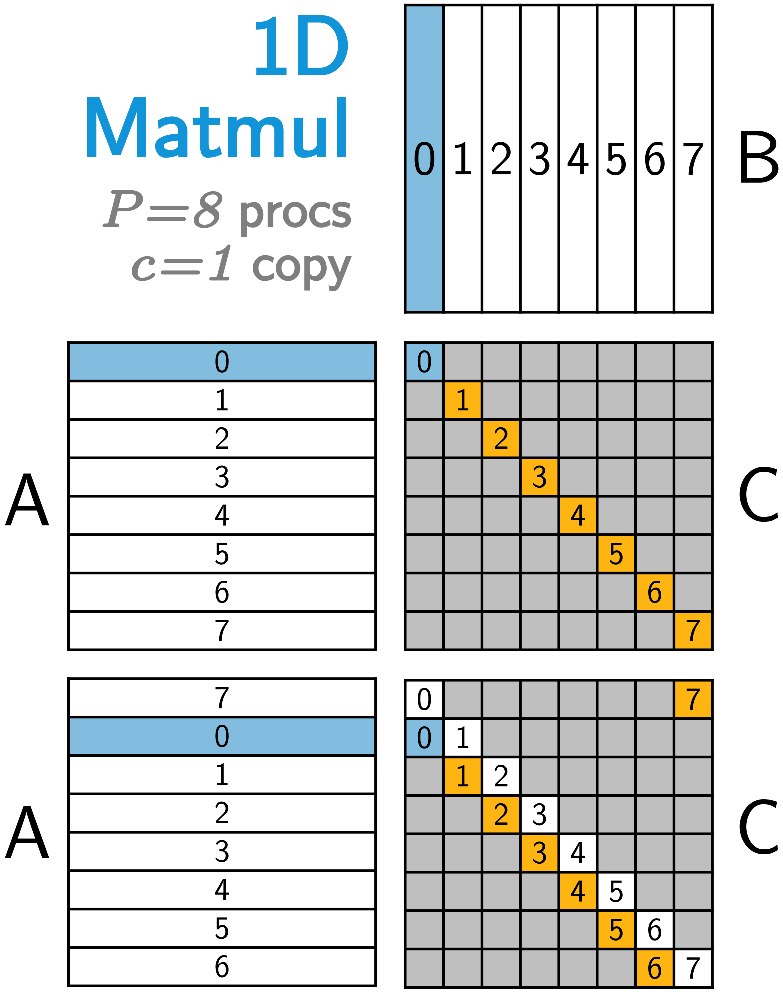
1D Matmul

$P=8$ procs
$c=1$ copy



1D Matmul

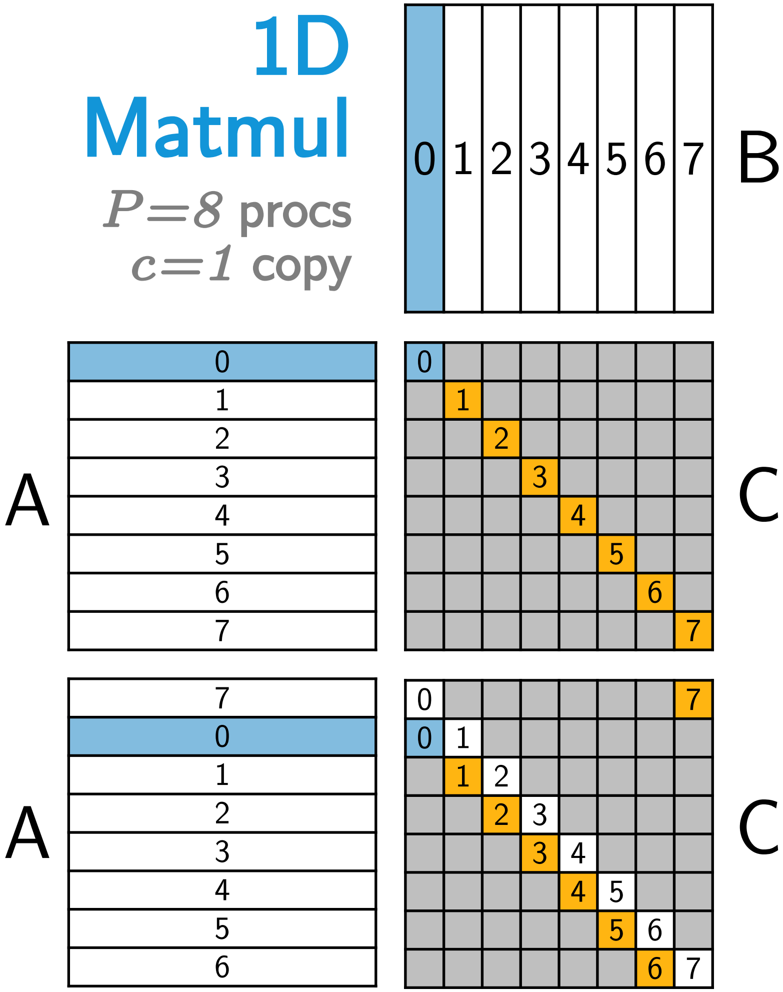
P=8 procs
c=1 copy



	1D
message size	nnz(A)/P
#messages	P
#words	nnz(A)

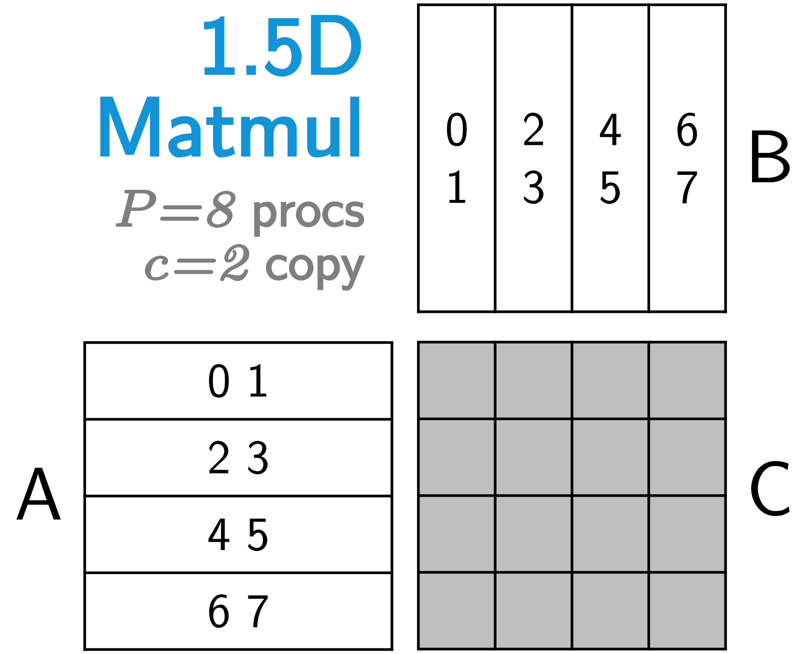
1D Matmul

P=8 procs
c=1 copy



1.5D Matmul

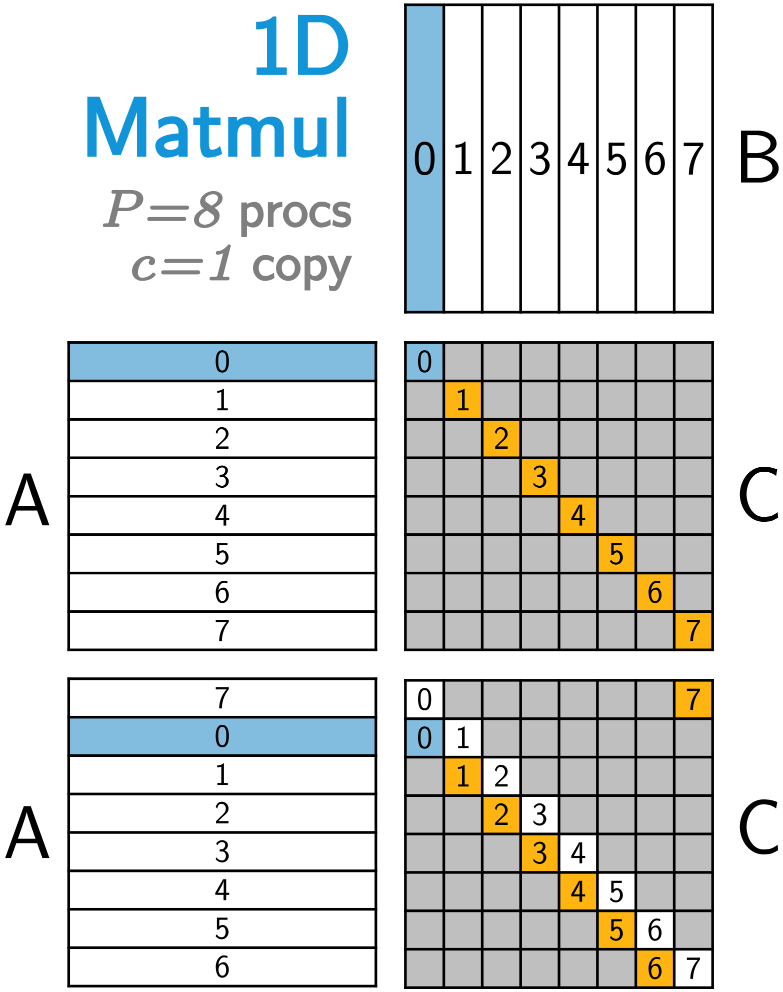
P=8 procs
c=2 copy



	1D
message size	nnz(A)/P
#messages	P
#words	nnz(A)

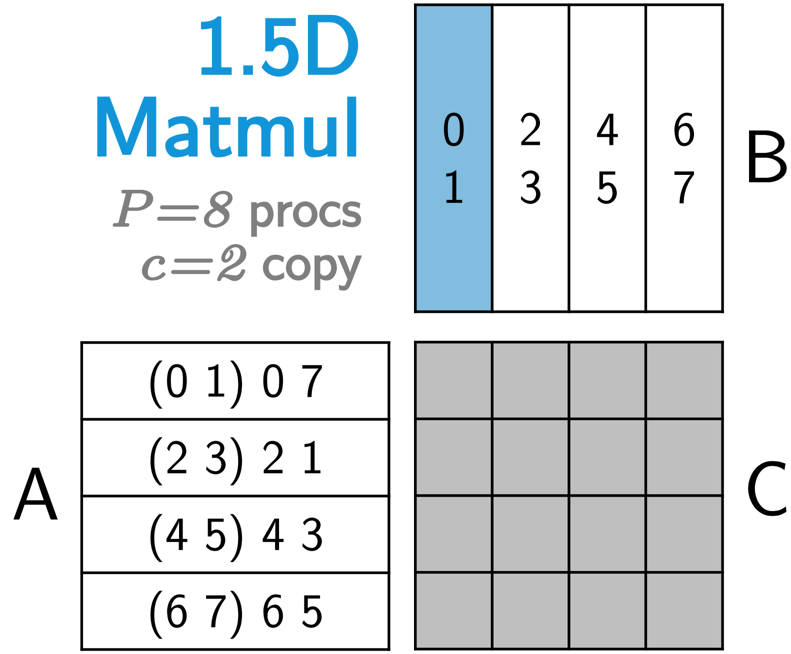
1D Matmul

$P{=}8$ procs
$c{=}1$ copy



1.5D Matmul

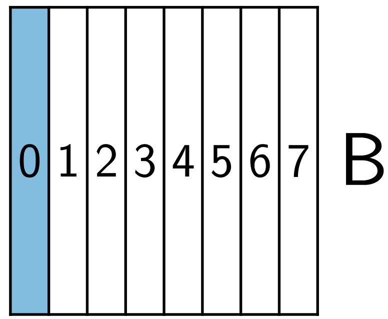
$P{=}8$ procs
$c{=}2$ copy

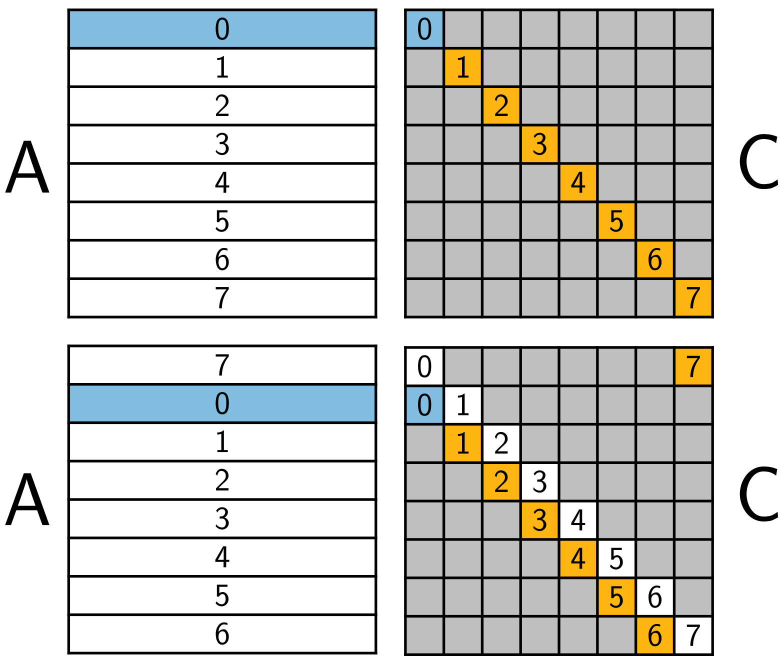


	1D
message size	nnz(A)/P
#messages	P
#words	nnz(A)

1D Matmul

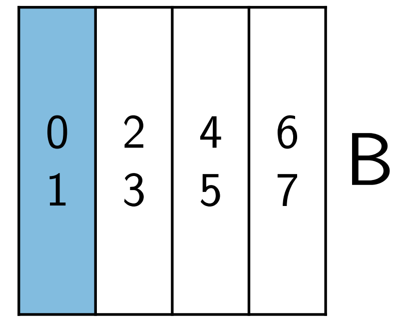
P=8 procs
c=1 copy

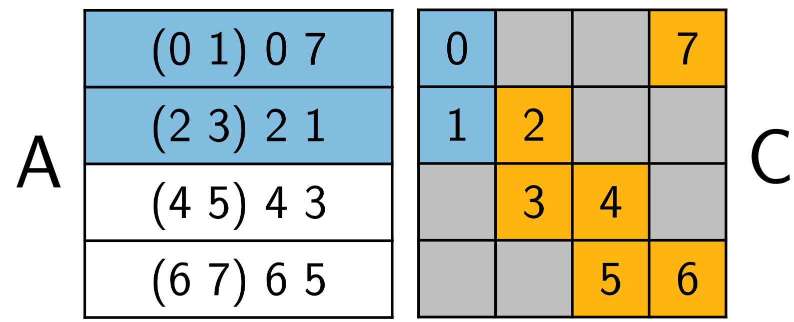




1.5D Matmul

P=8 procs
c=2 copy

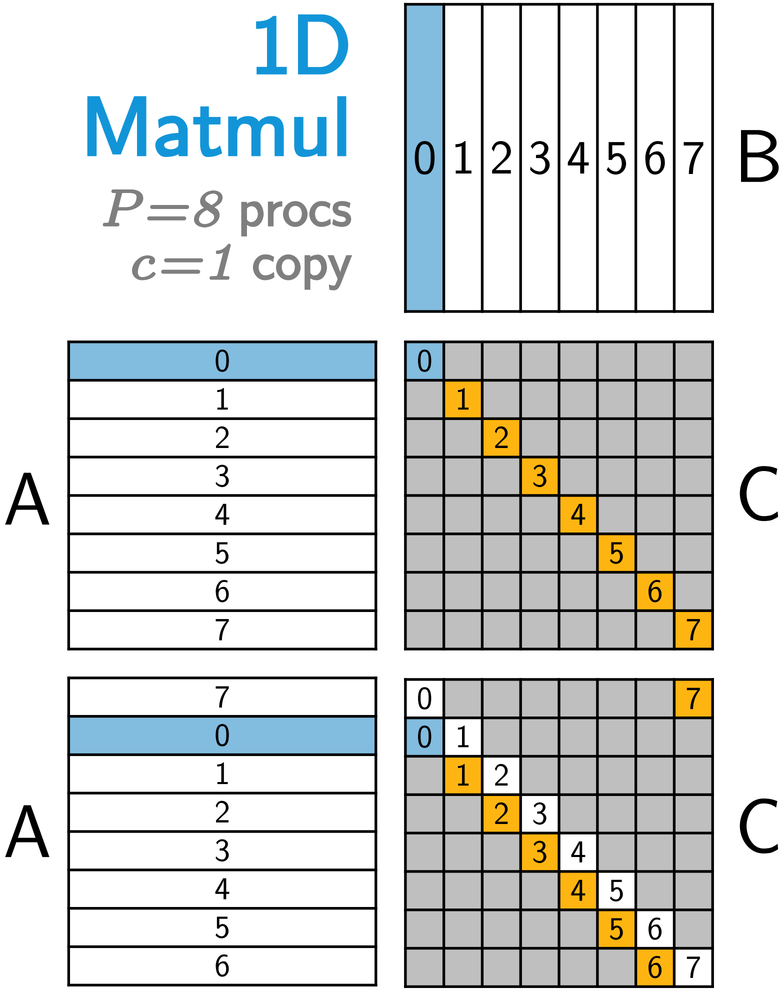




	1D
message size	nnz(A)/P
#messages	P
#words	nnz(A)

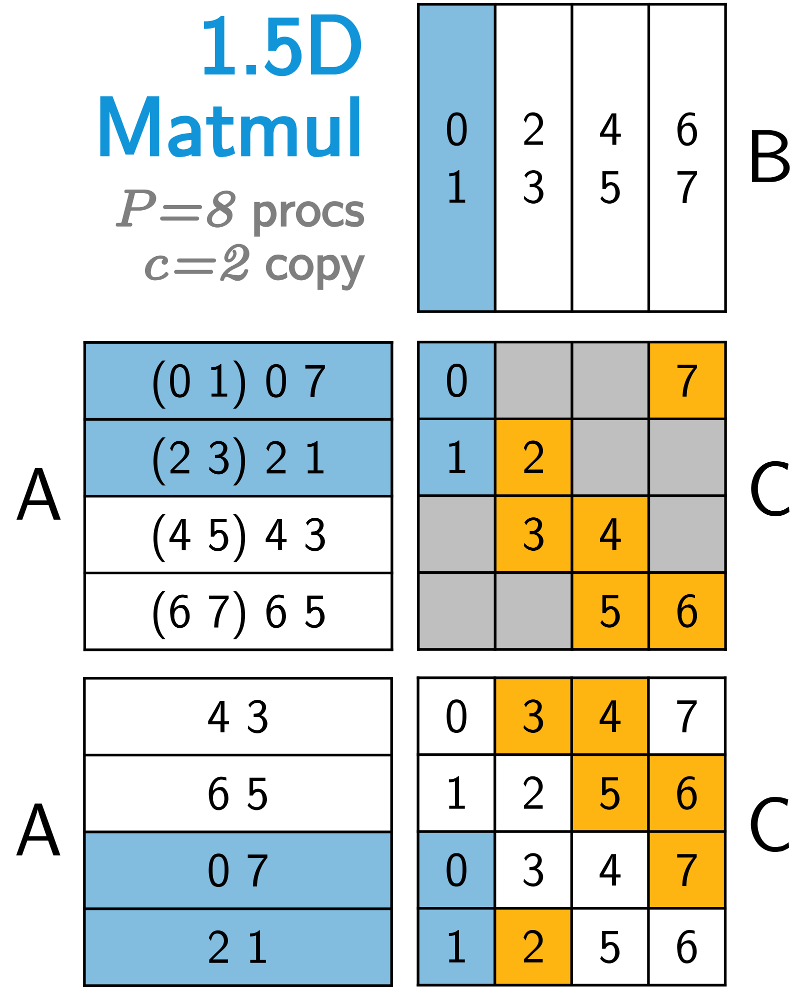
1D Matmul

P=8 procs
c=1 copy



1.5D Matmul

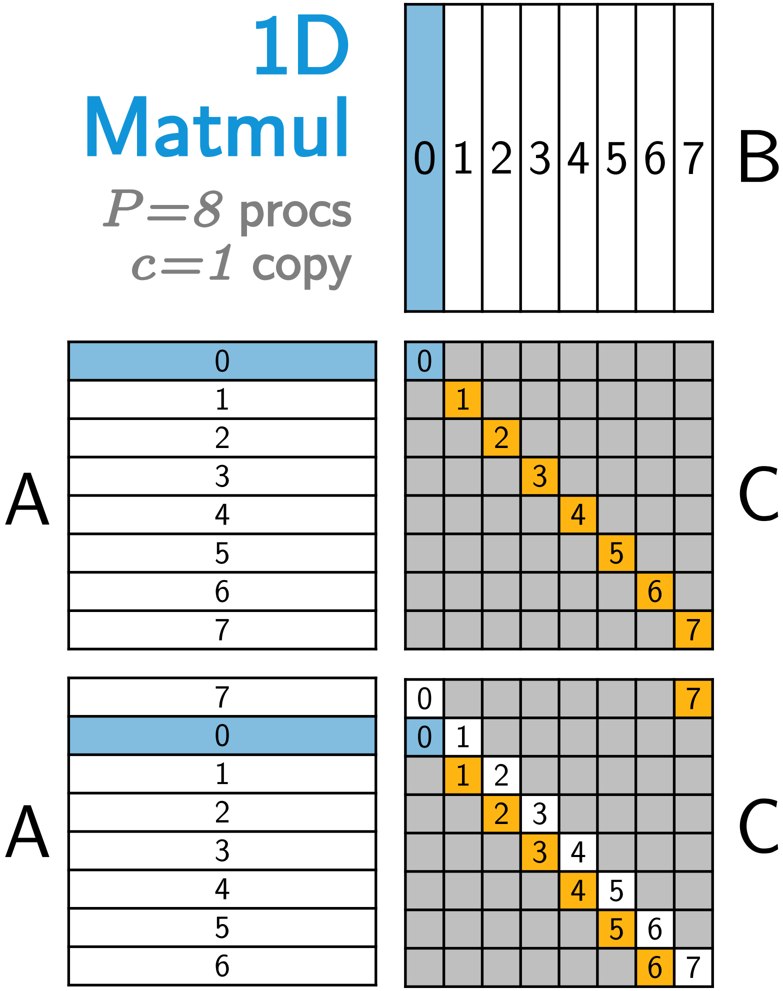
P=8 procs
c=2 copy



	1D
message size	nnz(A)/P
#messages	P
#words	nnz(A)

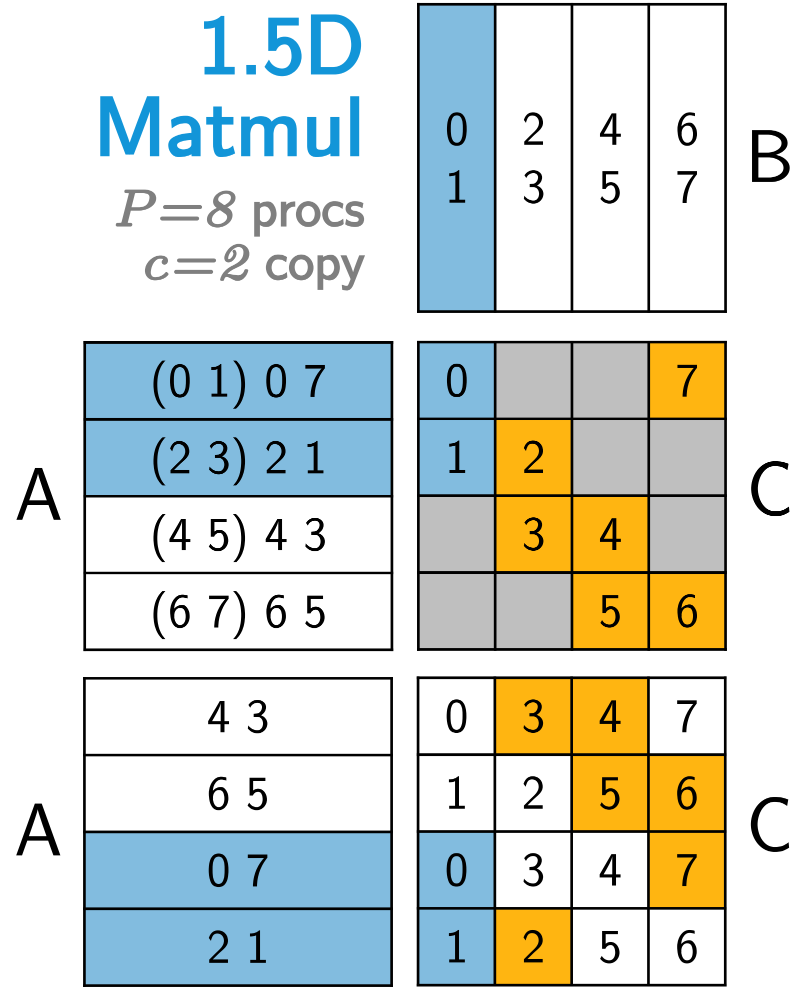
1D Matmul

P=8 procs
c=1 copy



1.5D Matmul

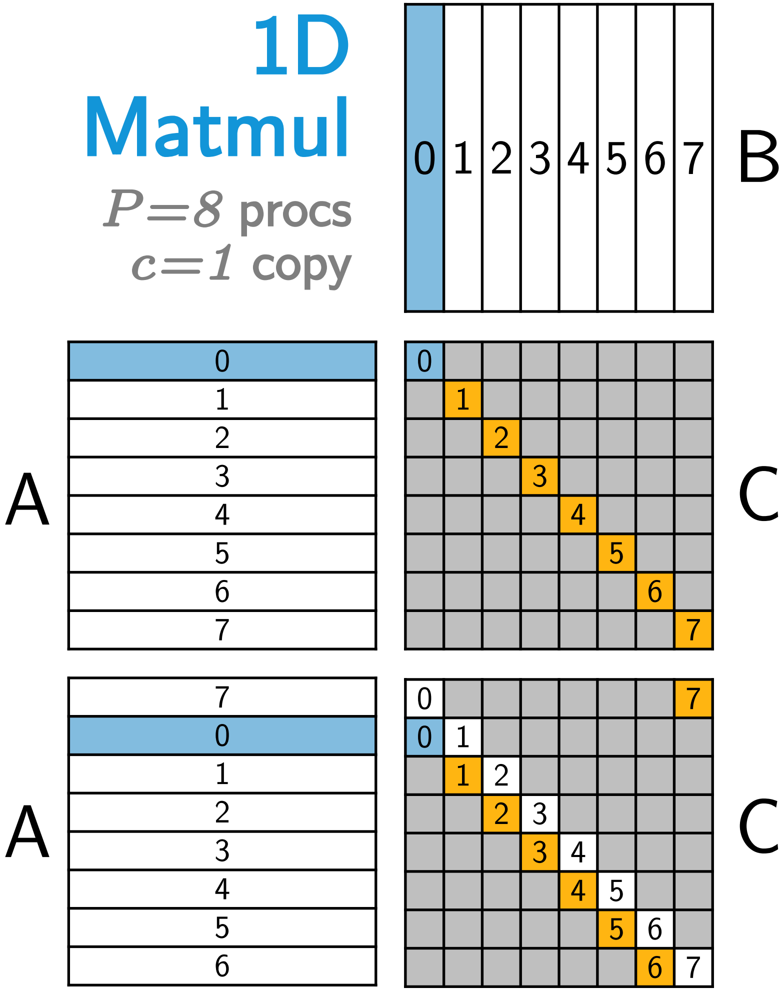
P=8 procs
c=2 copy



	1D	1.5D
message size	nnz(A)/P	c · nnz(A)/P
#messages	P	P/c^2
#words	nnz(A)	nnz(A)/c

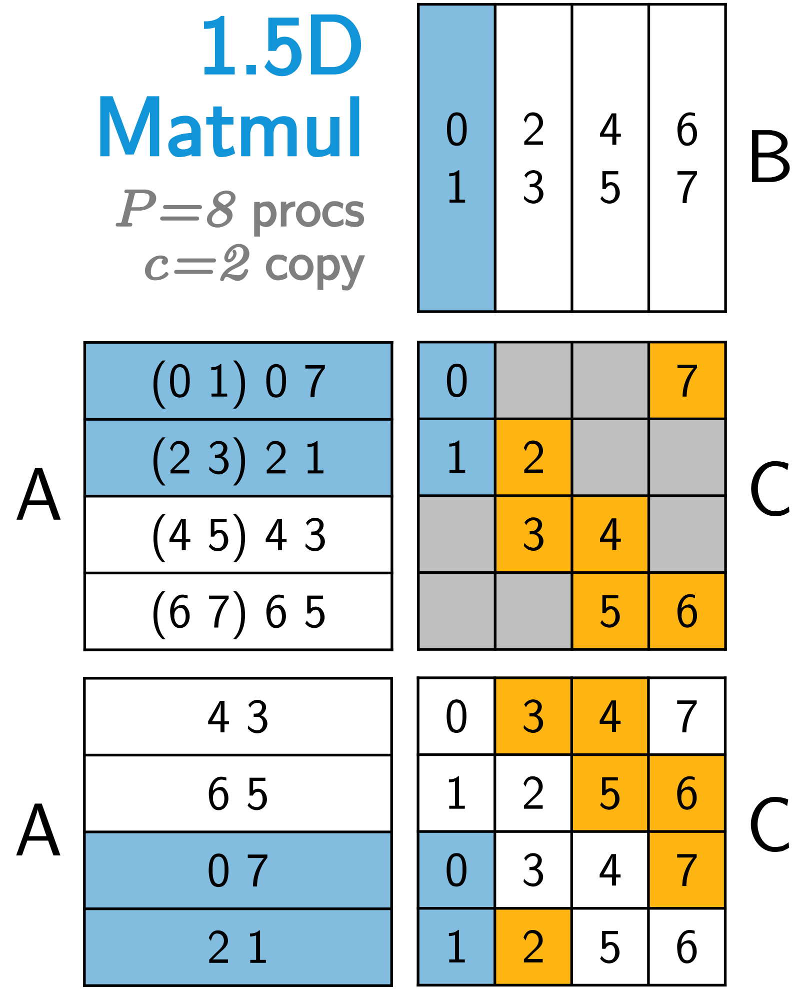
1D Matmul

P=8 procs
c=1 copy



1.5D Matmul

P=8 procs
c=2 copy

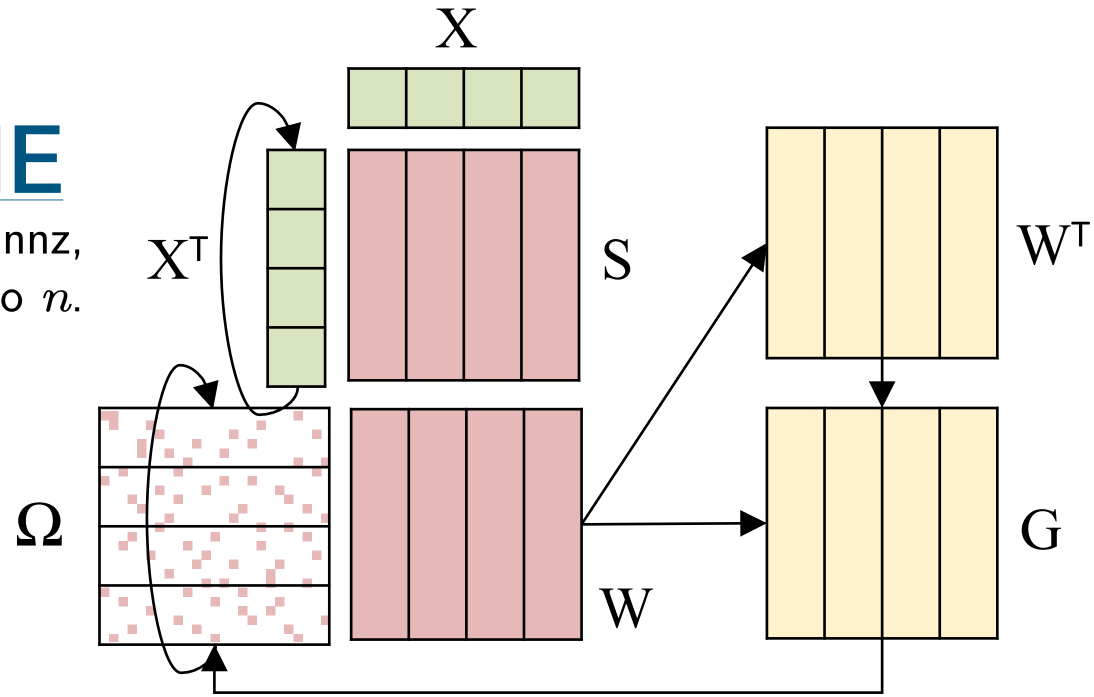


	1D	1.5D	1.5D (c_A, c_B)
message size	nnz(A)/P	$c \cdot$ nnz(A)/P	$c_A \cdot$ nnz(A)/P
#messages	P	P/c^2	P/($c_A c_B$)
#words	nnz(A)	nnz(A)/c	nnz(A)/c_B

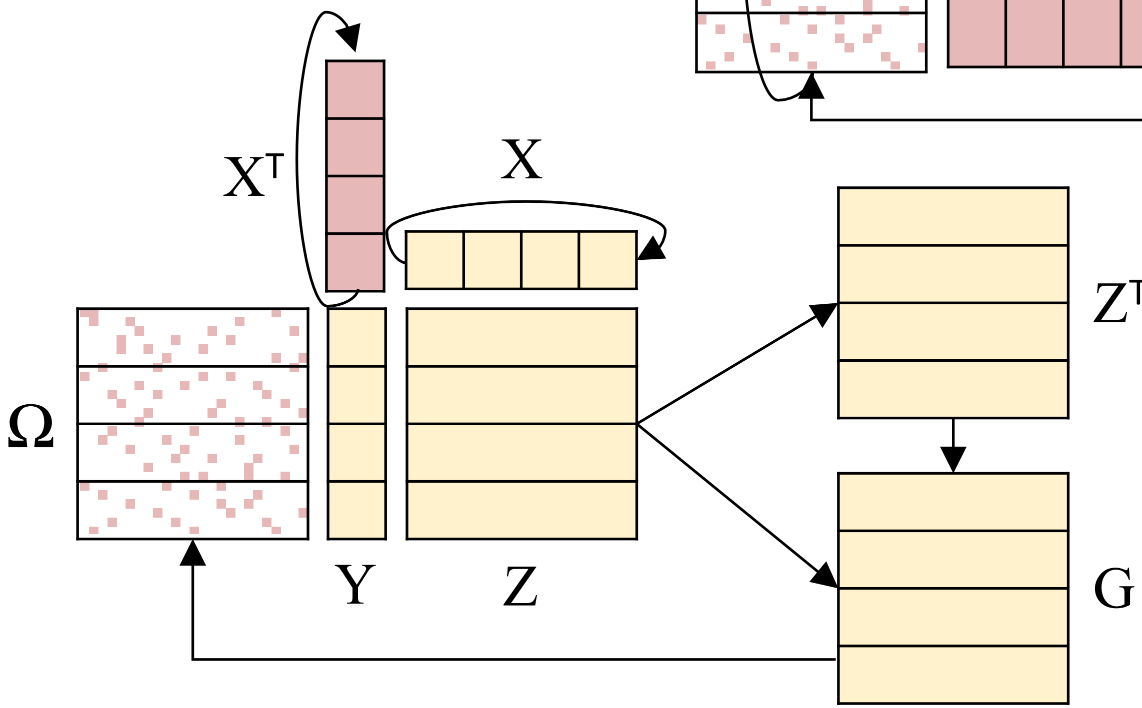
Summary

ONE

#flops varies with p and %nnz,
independent to n.



Green: once.
Yellow: every outer iteration.
Red: every inner iteration.



TWO

#flops varies with p and n,
not so much with %nnz.

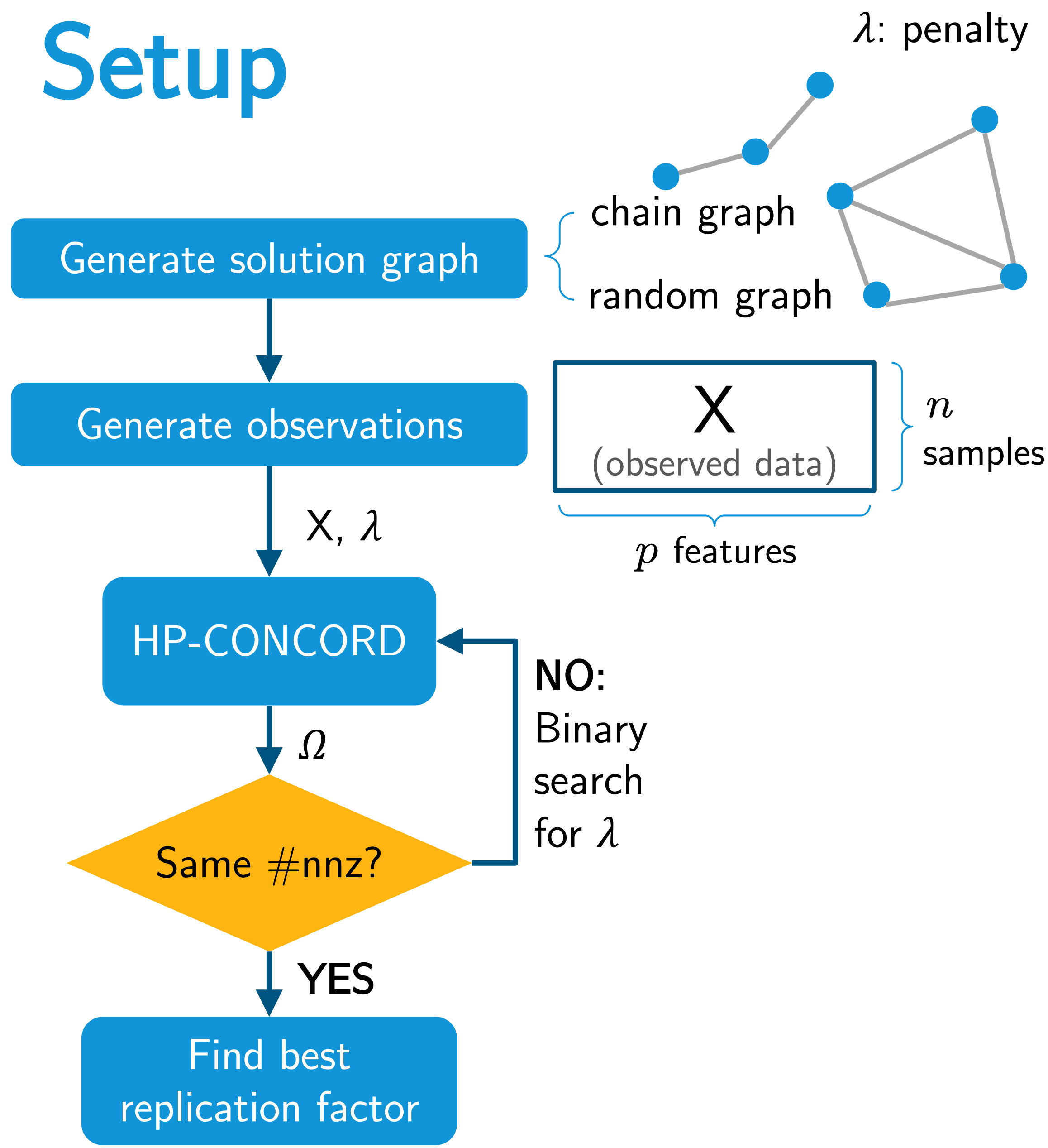
Performance Results

Experimental Setup

- C++
- MPI/OpenMP
- 12 threads/proc

- MKL for local matmuls

- **Edison@NERSC**
  - Cray XC30
  - 12-core Intel Ivy Bridge@2.4GHz 2 sockets/node



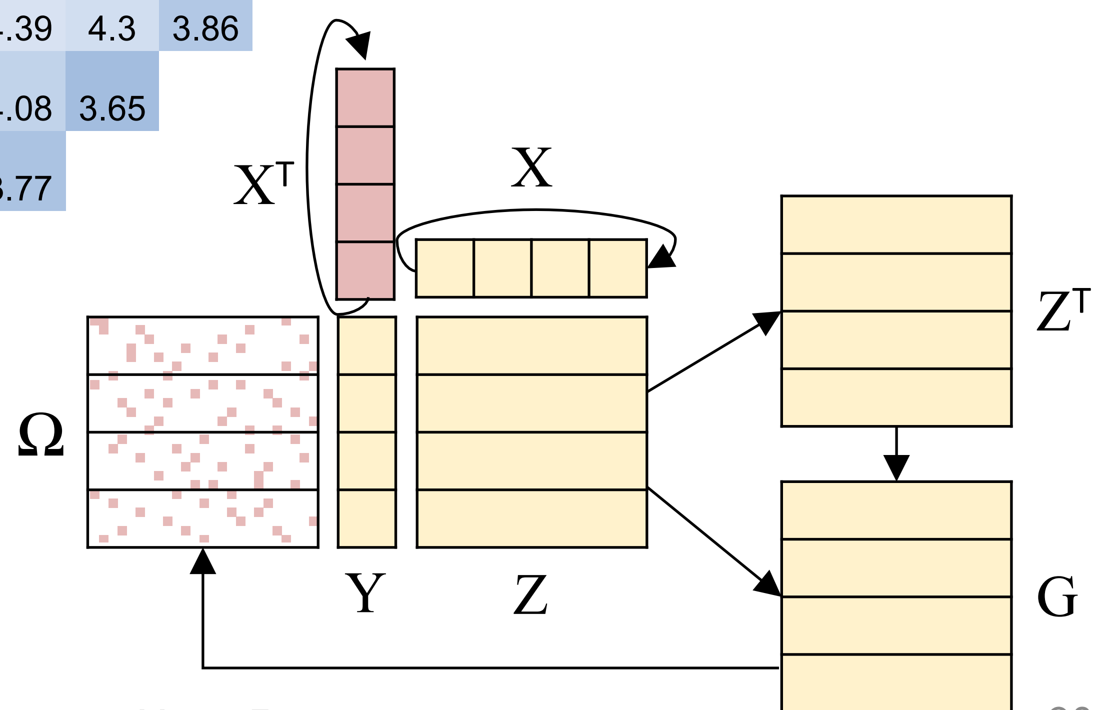
Replication effects

Total running time (seconds). Chain graph (3 nnz/row), **TWO**, 256 nodes (P=512), n=100 samples, p=40k features

c: X^T, X

c_Ω \ c	1	2	4	8	16	32	64	128	256	512
1	13.15	9.69	8.73	7.34	7.09	7.01	7.17	6.56	7	5.01
2	9.35	5.75	5.78	7.1	5.42	5.42	5.78	5.13	3.9	
4	5.28	3.94	3.42	5.08	4.44	4.39	4.3	3.86		
8	4.08	3.45	3.07	2.84	4.19	4.08	3.65			
16	4	3.55	3.2	**2.63**	3.83	3.77				
32	4	6.06	3.43	3.38	3.26					
64	6.13	4.77	4.7	4.42						
128	7.8	6.59	5.87							
256	10.97	9.47								

Good → Bad



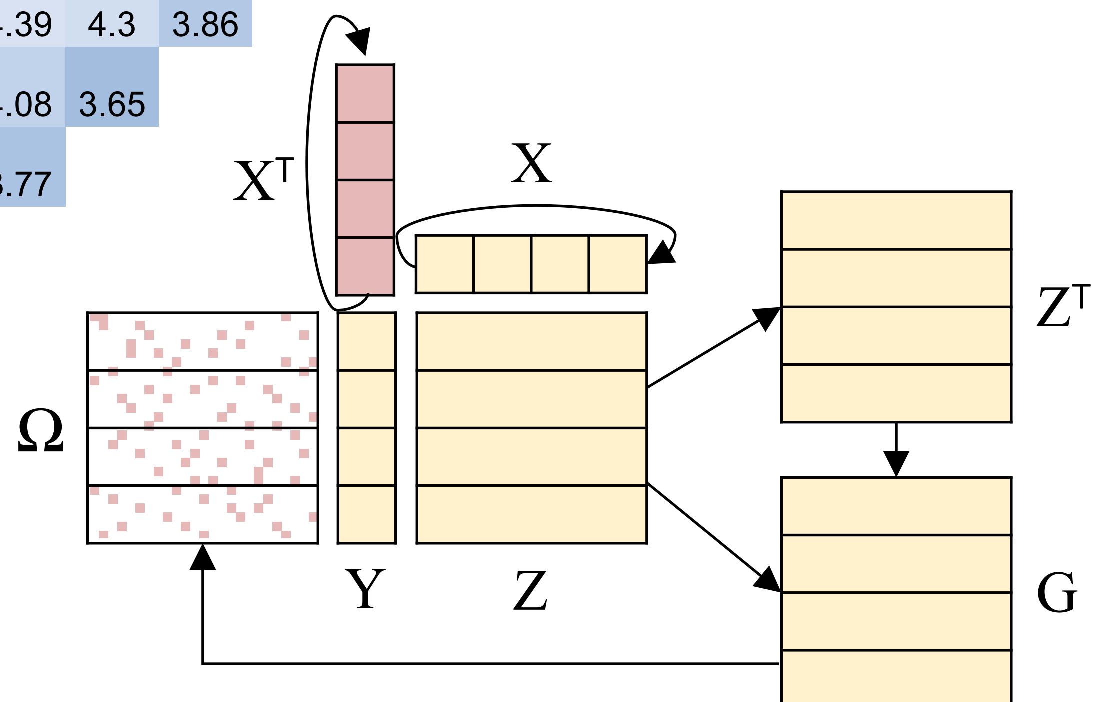
Replication effects

Total running time (seconds). Chain graph (3 nnz/row), **TWO**, 256 nodes (P=512), n=100 samples, p=40k features

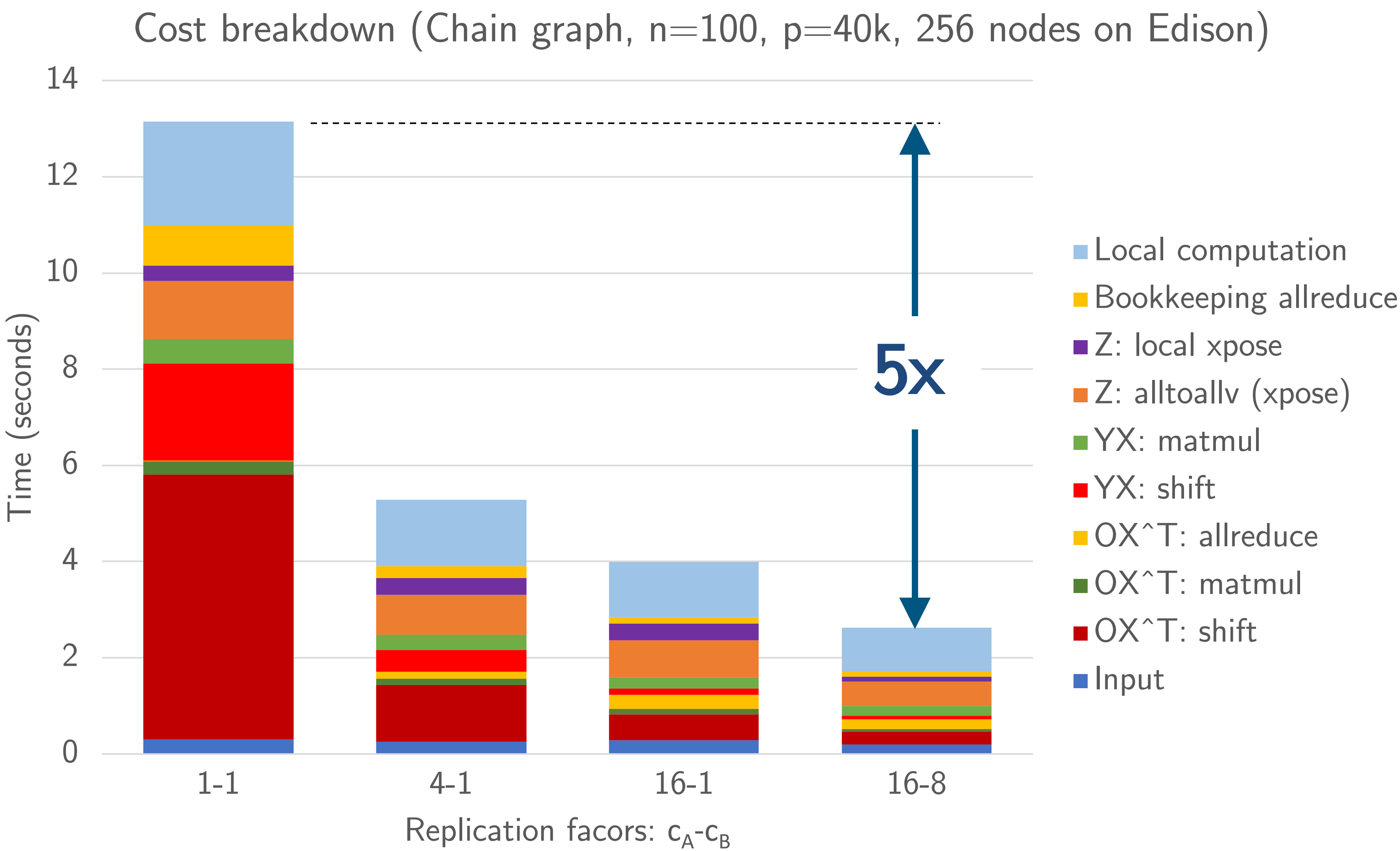
c: X^T, X

c_Ω \ c	1	2	4	8	16	32	64	128	256	512
1	13.15	9.69	8.73	7.34	7.09	7.01	7.17	6.56	7	5.01
2	9.35	5.75	5.78	7.1	5.42	5.42	5.78	5.13	3.9	
4	5.28	3.94	3.42	5.08	4.44	4.39	4.3	3.86		
8	4.08	3.45	3.07	2.84	4.19	4.08	3.65			
16	4	3.55	3.2	**2.63**	3.83	3.77				
32	4	6.06	3.43	3.38	3.26					
64	6.13	4.77	4.7	4.42						
128	7.8	6.59	5.87							
256	10.97	9.47								

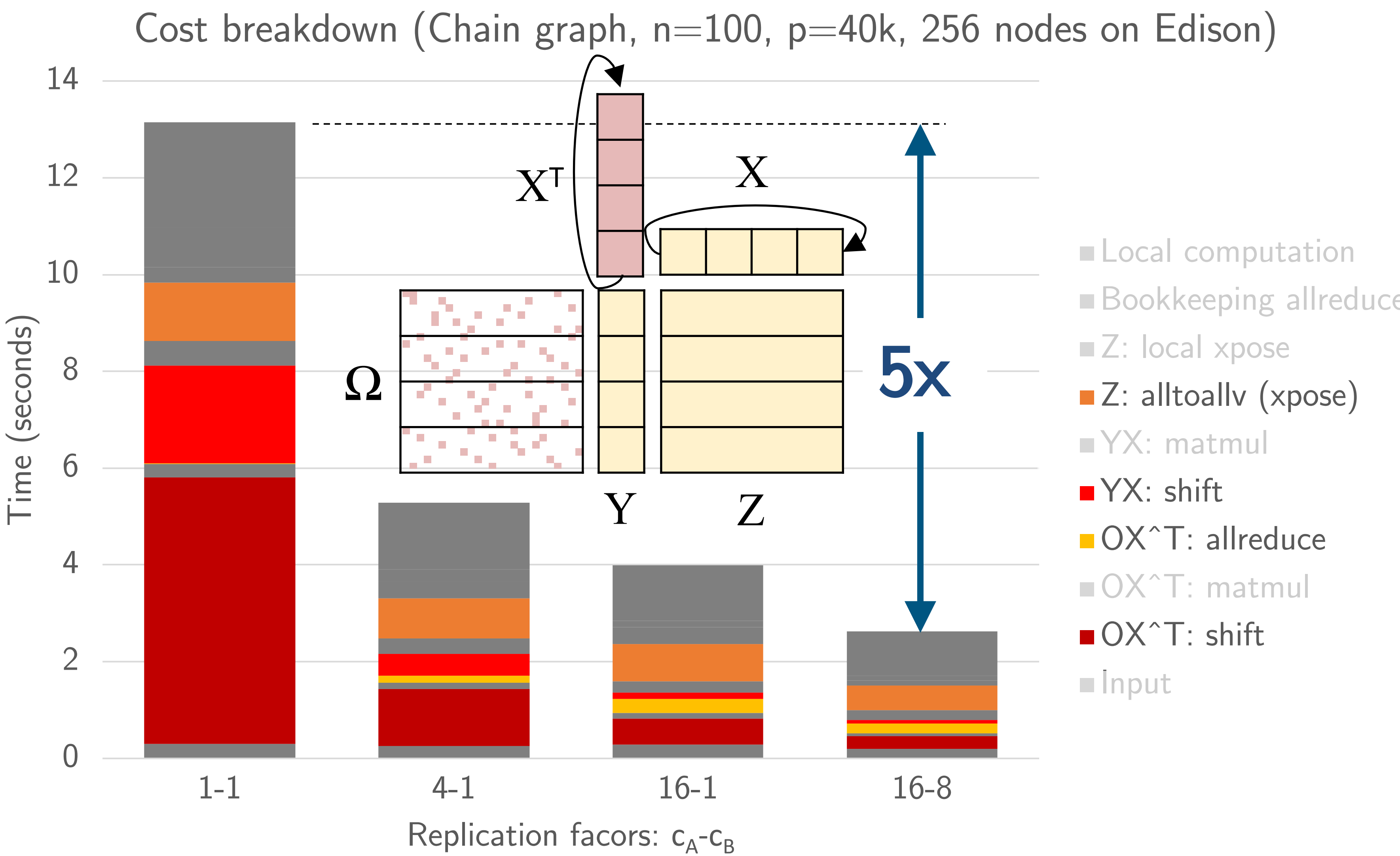
Good → Bad



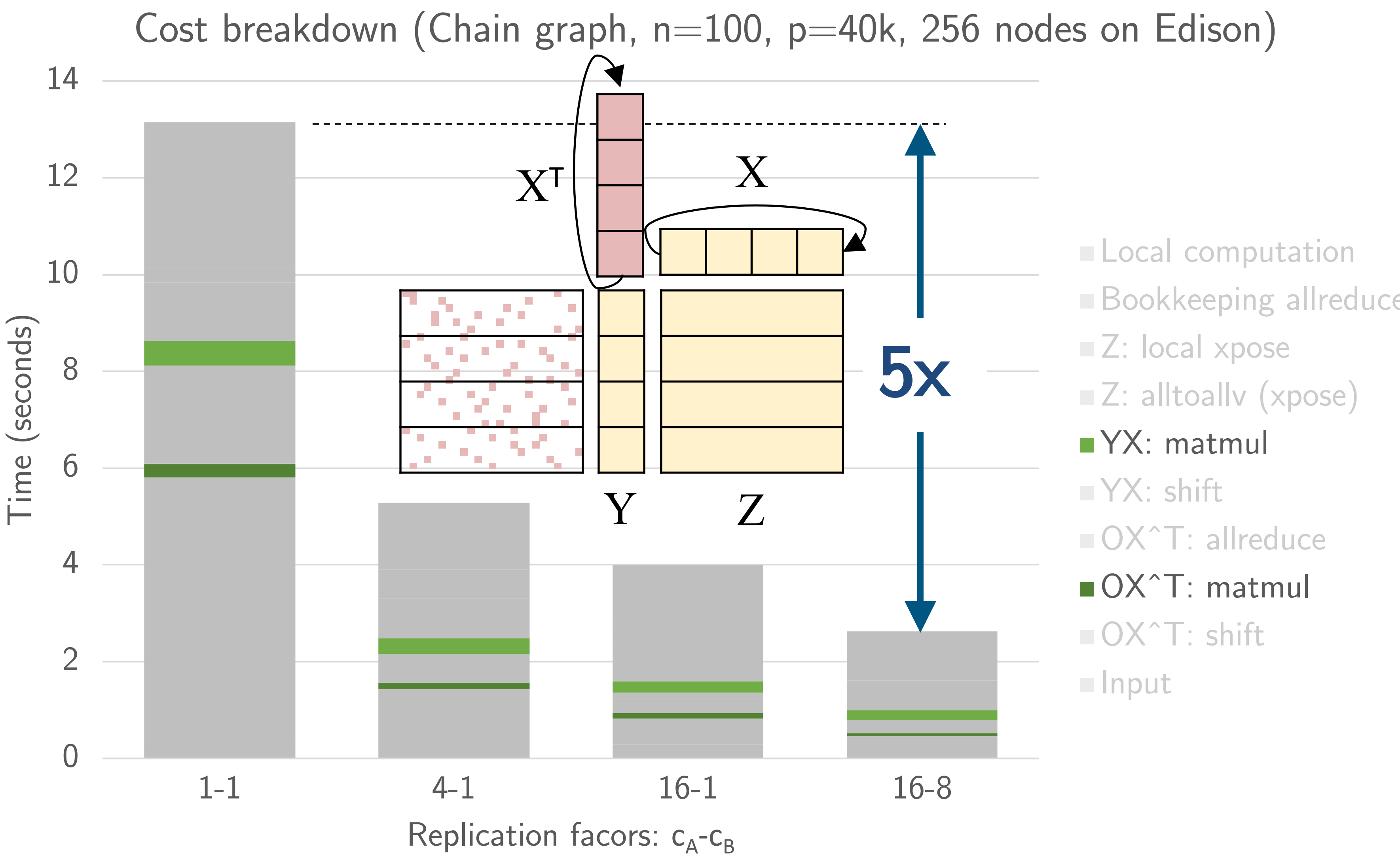
Cost breakdown



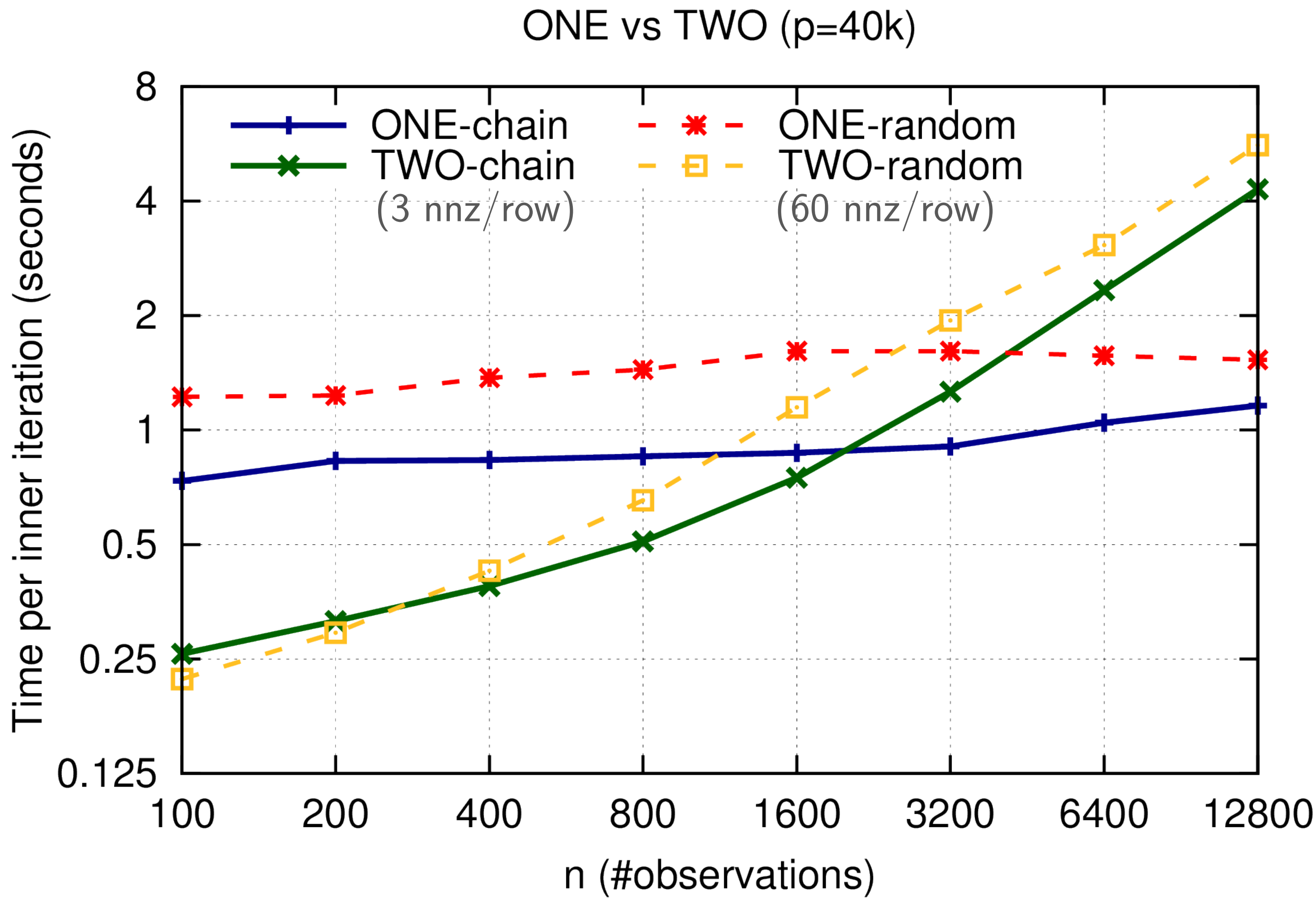
Reduced communication costs



Bonus: cheaper computation



When to use ONE/TWO?

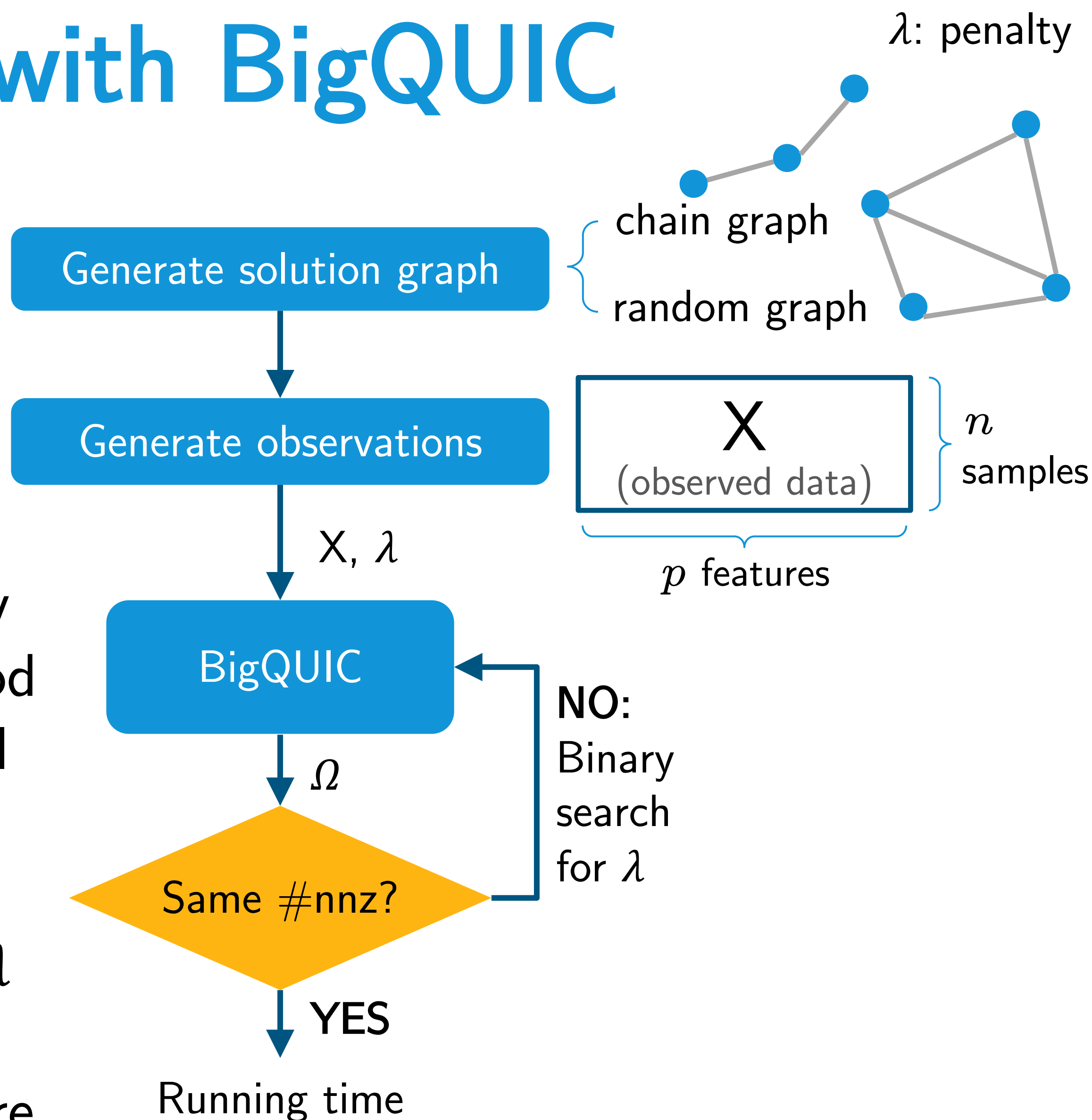


Comparison with BigQUIC

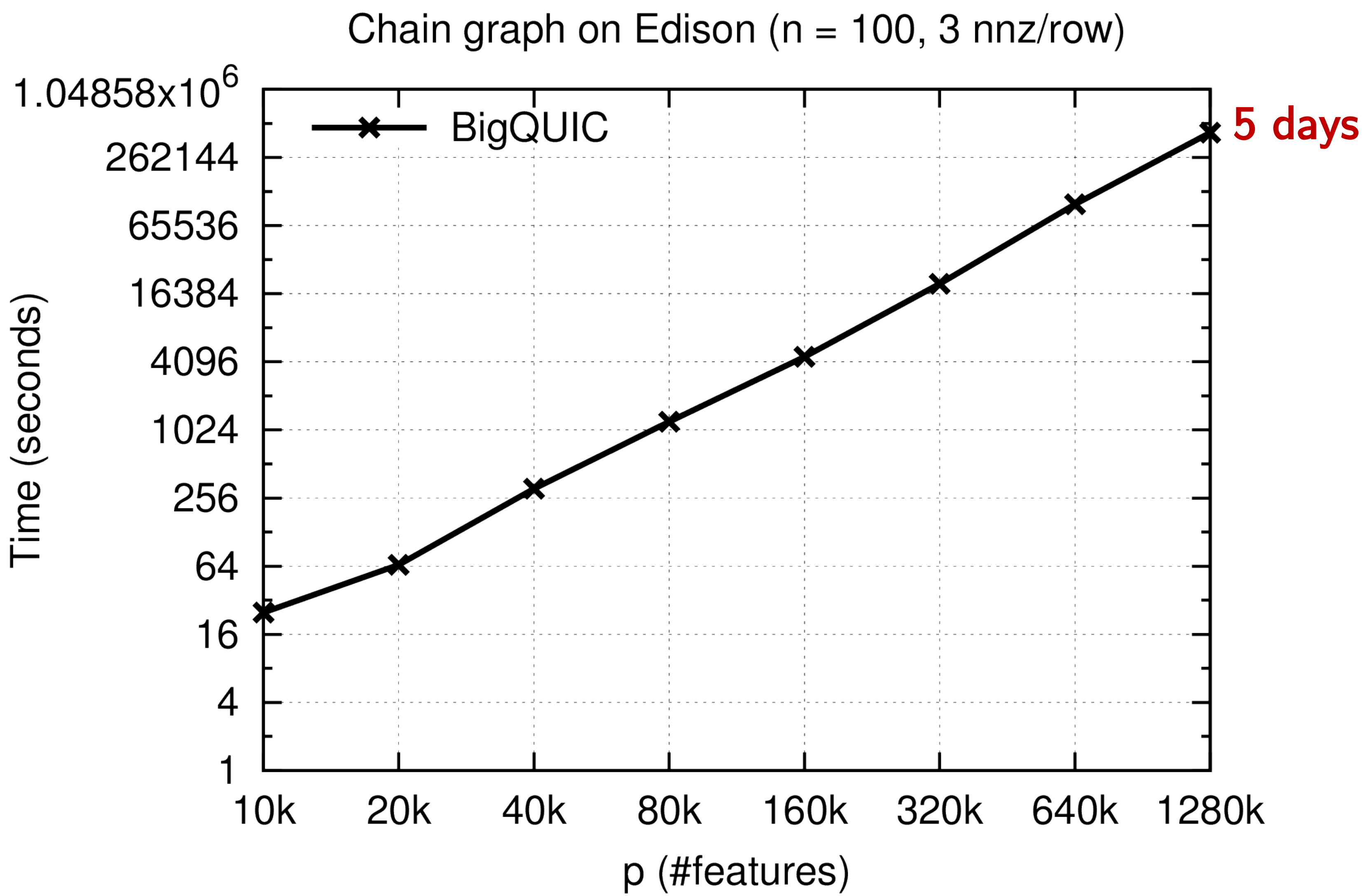
- C++
- OpenMP
- Shared-memory (single-node run)

- Different complexity
- Second-order method
- Converges in several iterations

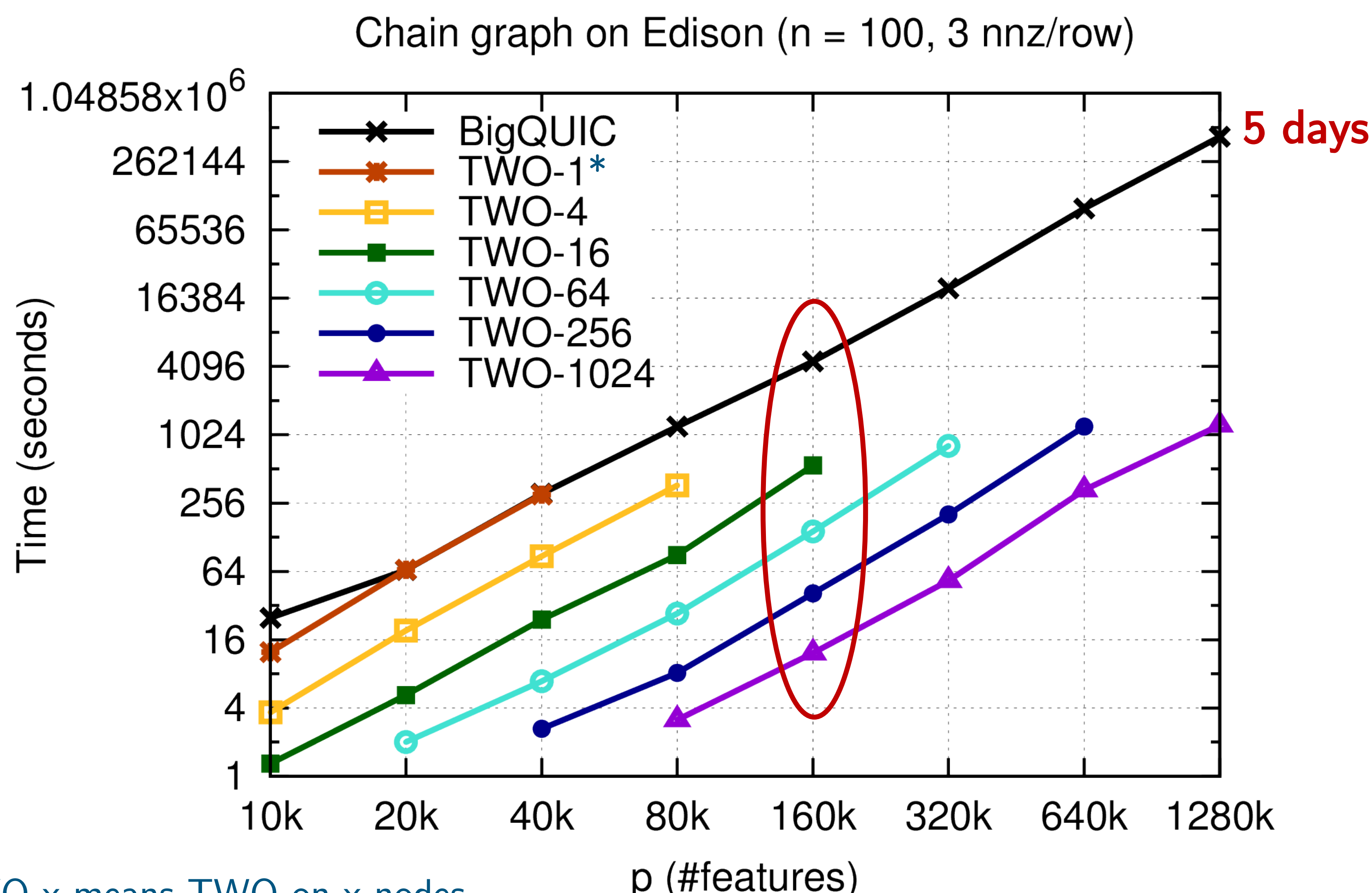
- Search for penalty λ that gives the right answer then compare running time.



Chain graph vs BigQUIC

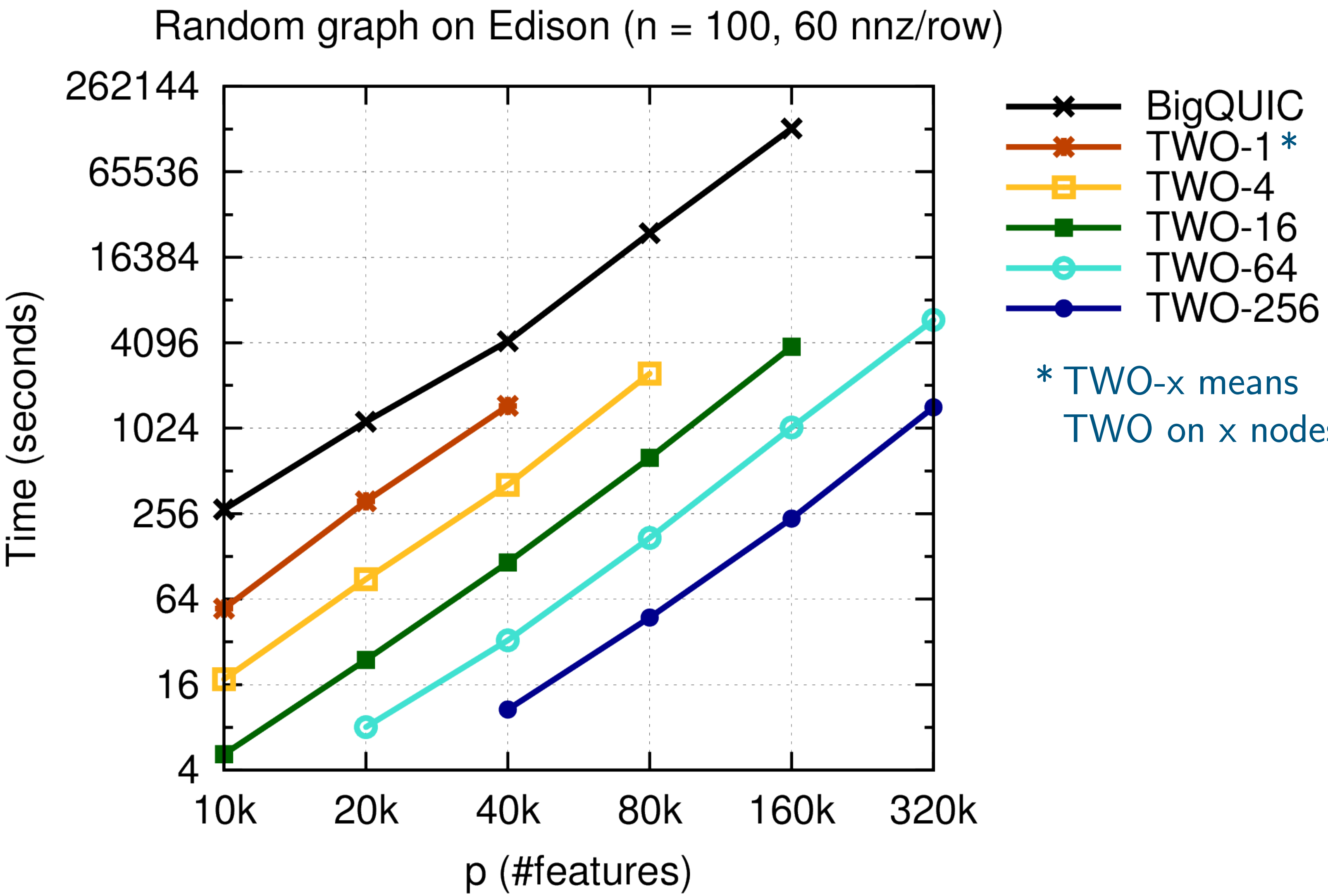


Chain graph vs BigQUIC



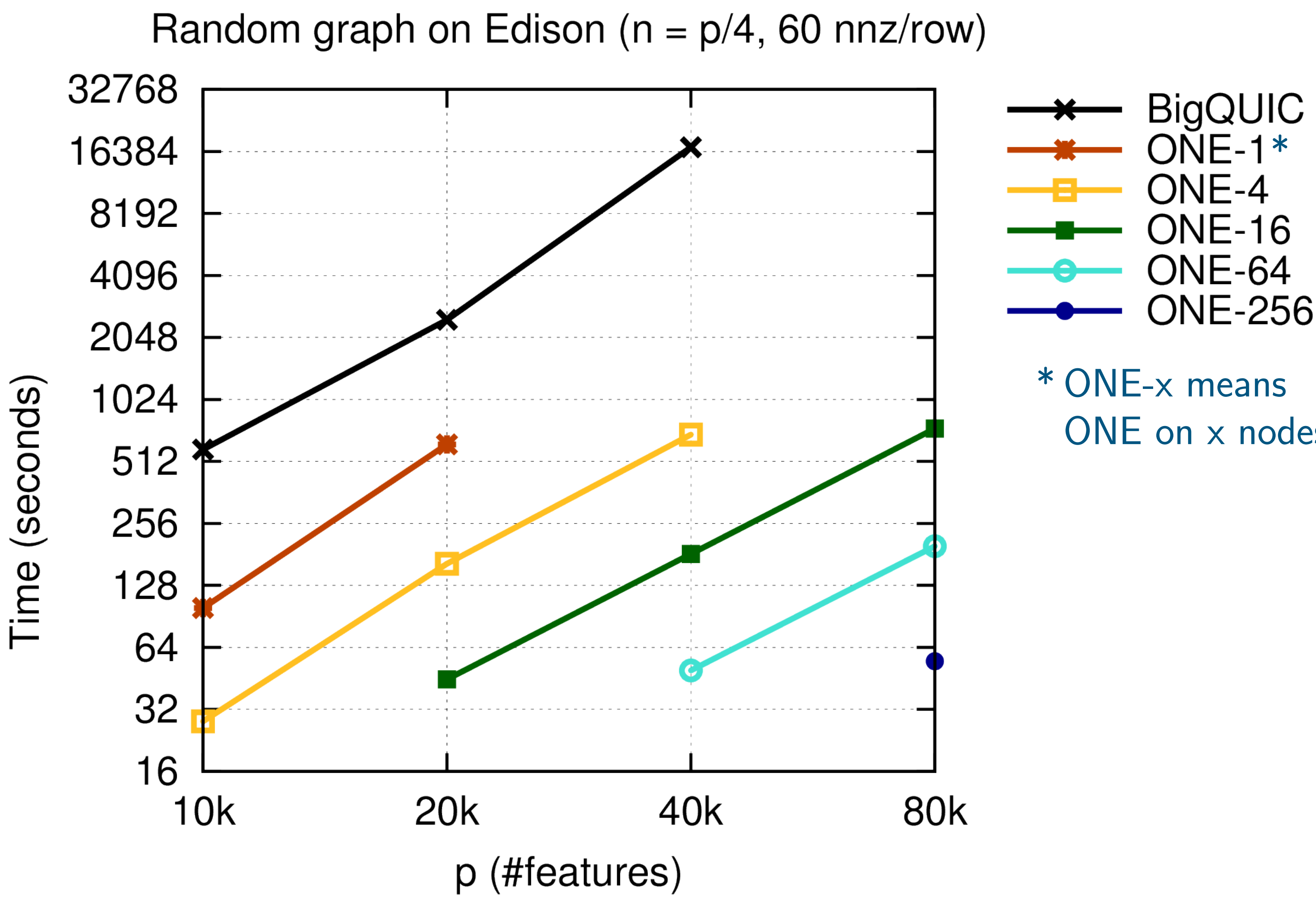
*TWO-x means TWO on x nodes

Random graph vs BigQUIC



* TWO-x means TWO on x nodes

Random graph vs BigQUIC II



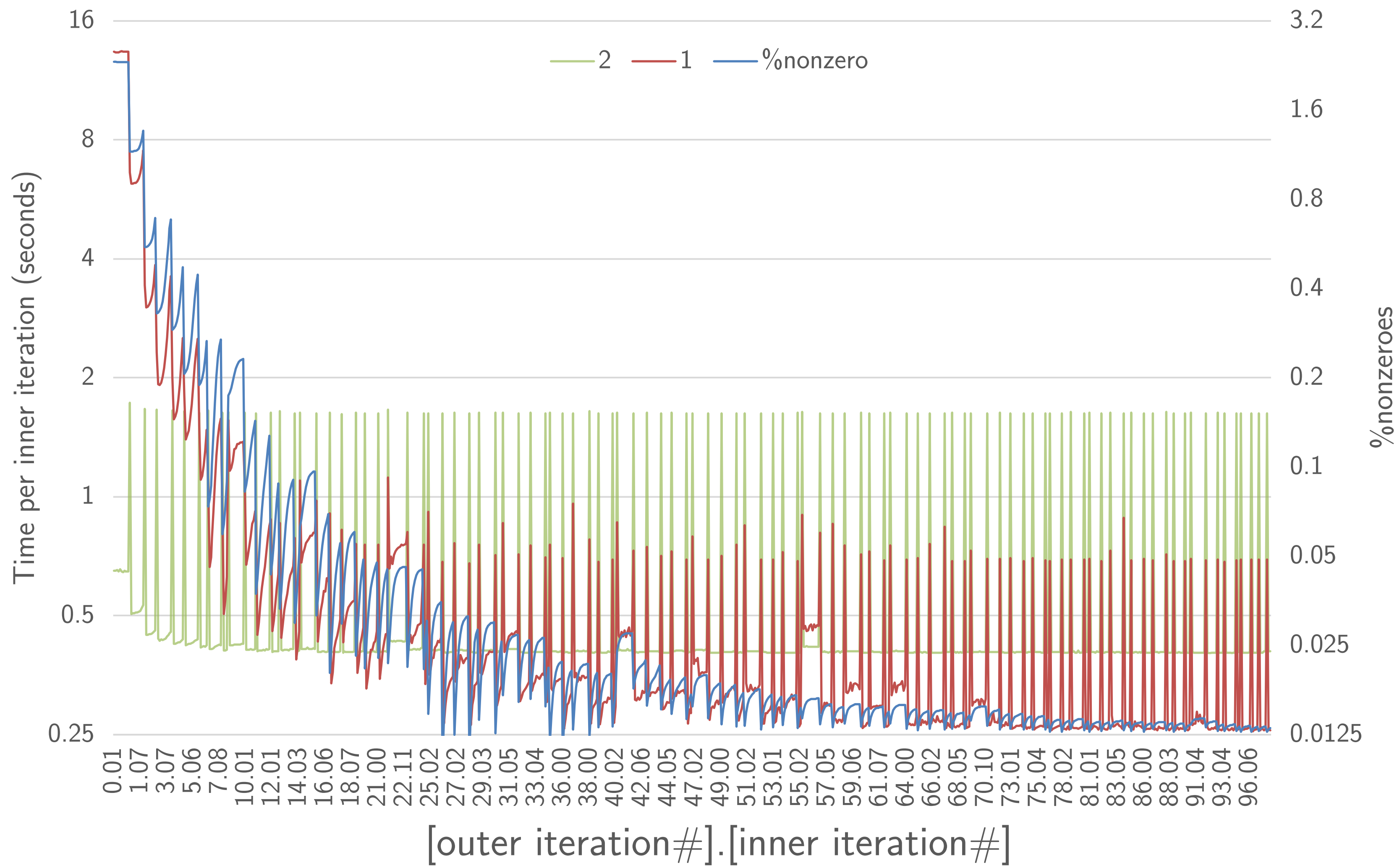
* ONE-x means ONE on x nodes

Real dataset

3D Brain fMRI scan.

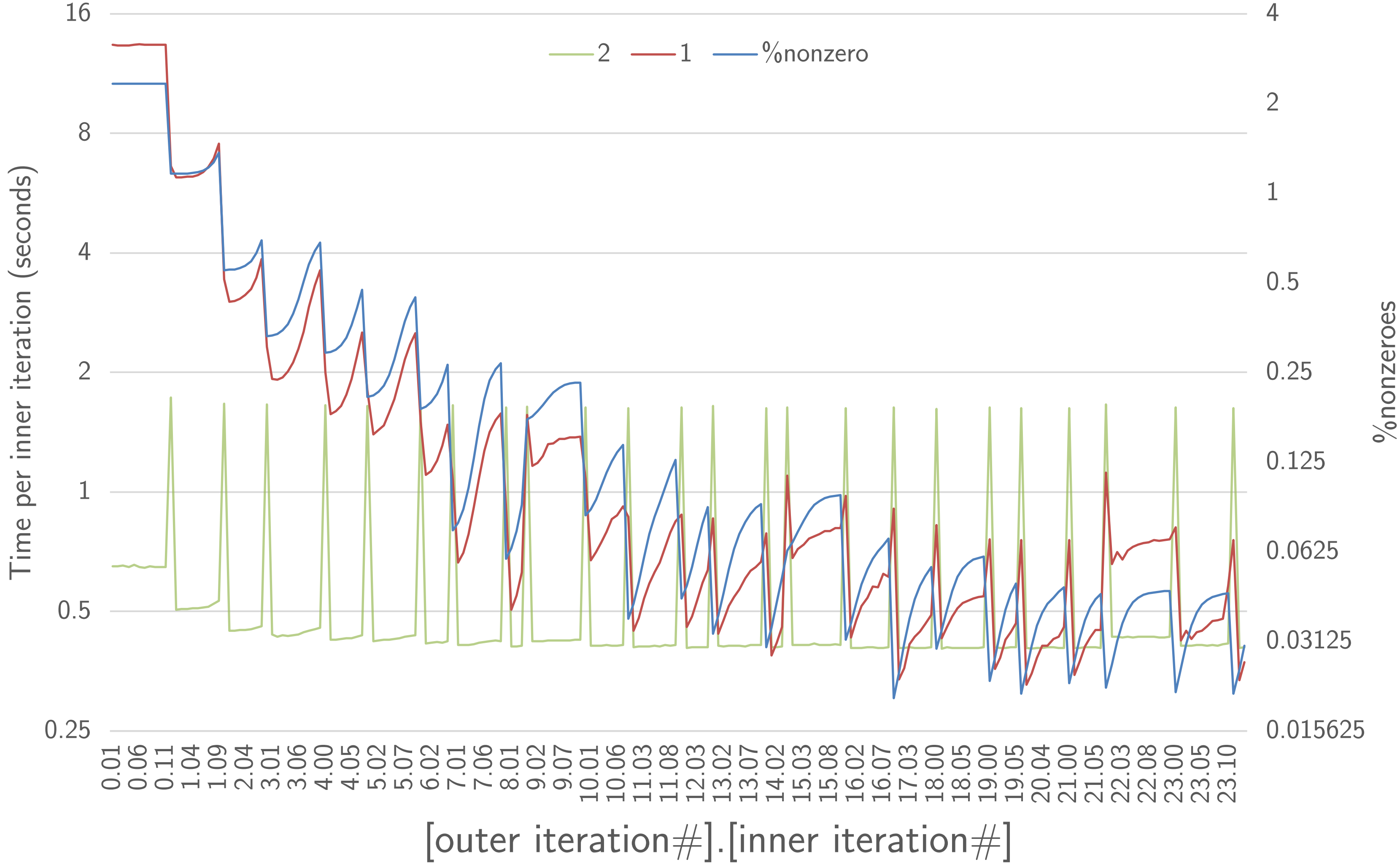
1.2k x 110k: Iteration costs

6,144 cores on Edison. 790 inner iterations. 0.095%nz (avg). 0.013%nz (final)



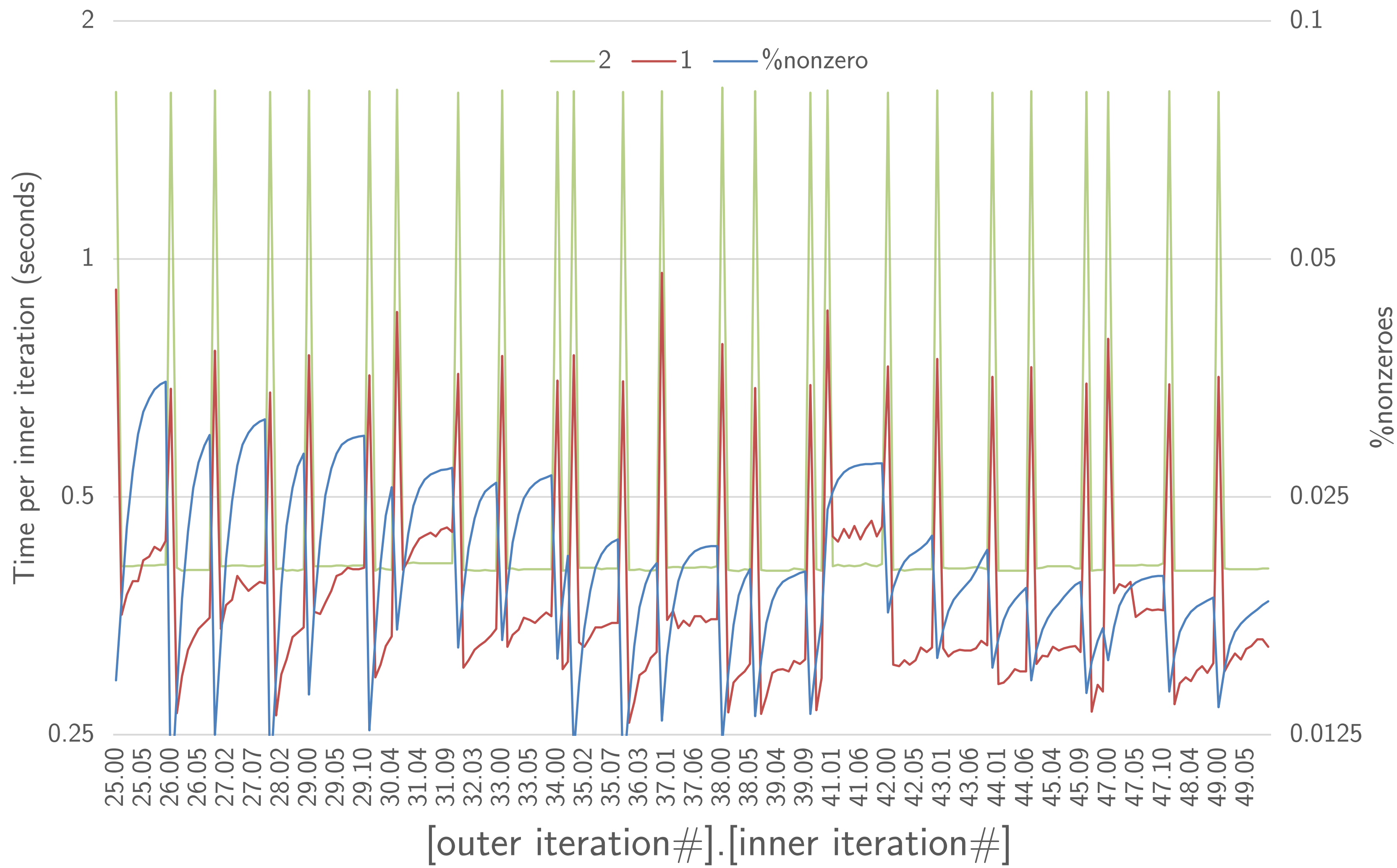
1.2k x 110k: Iterations 0–24

6,144 cores on Edison. 790 inner iterations. 0.095%nz (avg). 0.013%nz (final)



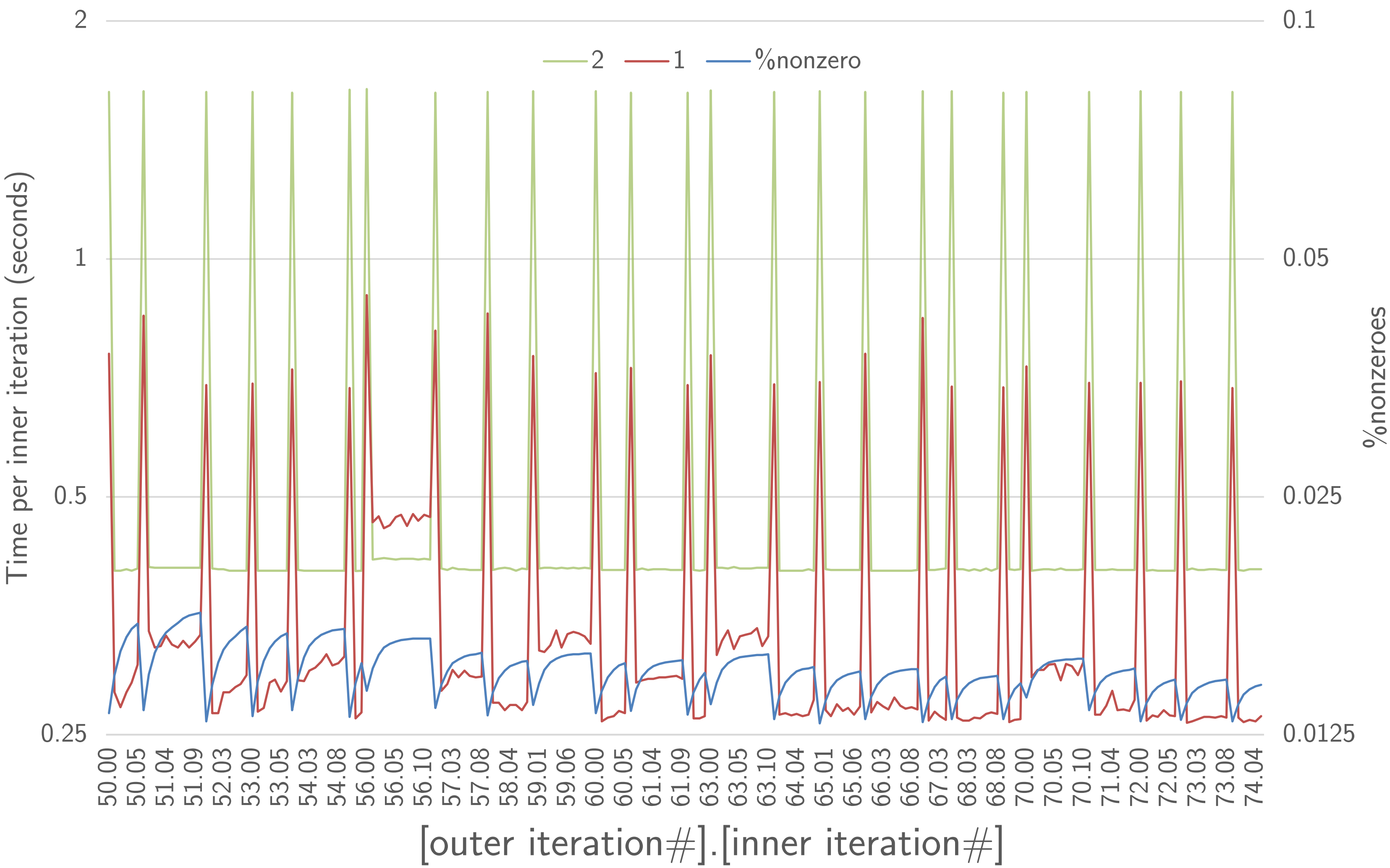
1.2k x 110k: Iterations 25-49

6,144 cores on Edison. 790 inner iterations. 0.095%nz (avg). 0.013%nz (final)



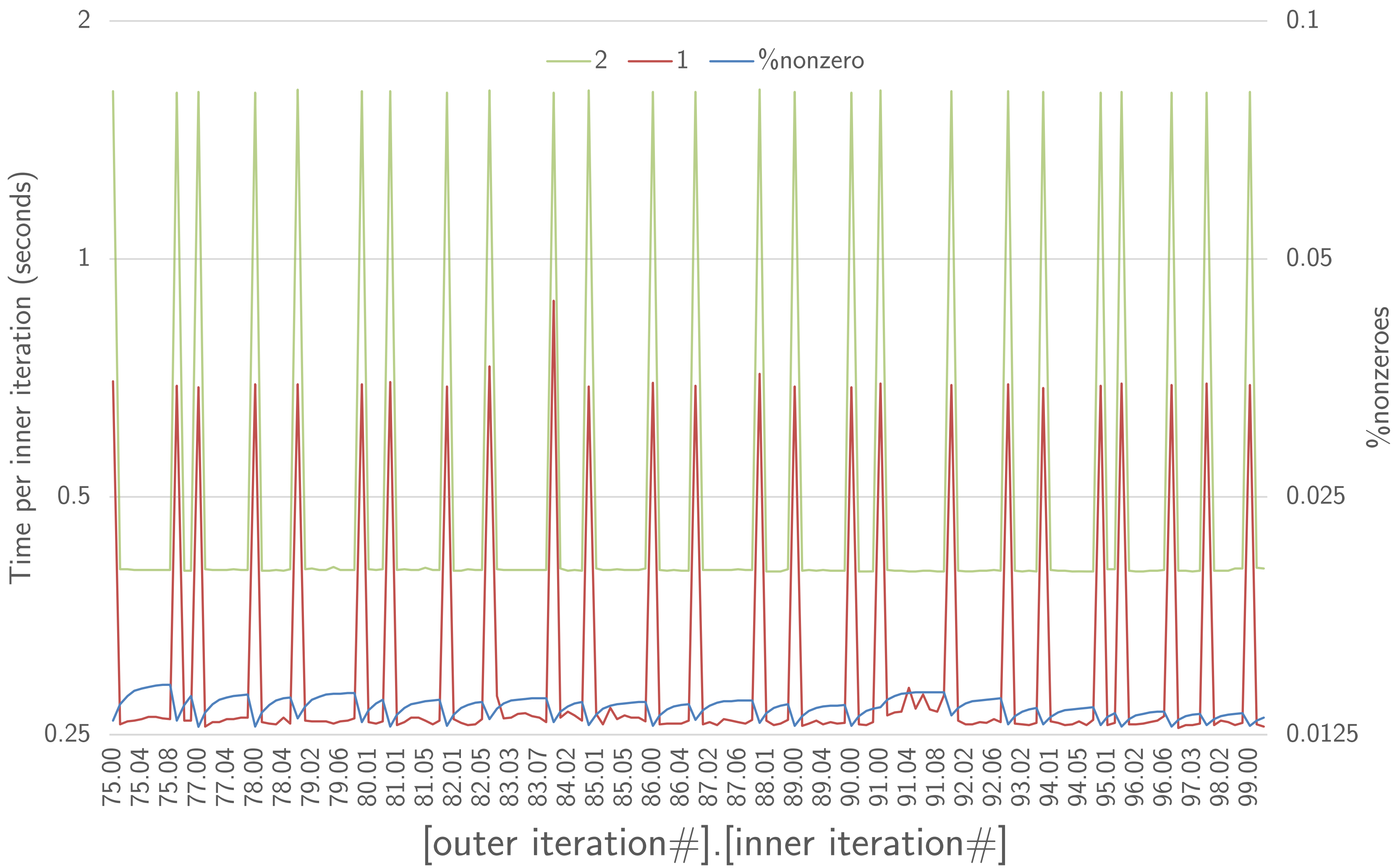
1.2k x 110k: Iterations 50–74

6,144 cores on Edison. 790 inner iterations. 0.095%nz (avg). 0.013%nz (final)



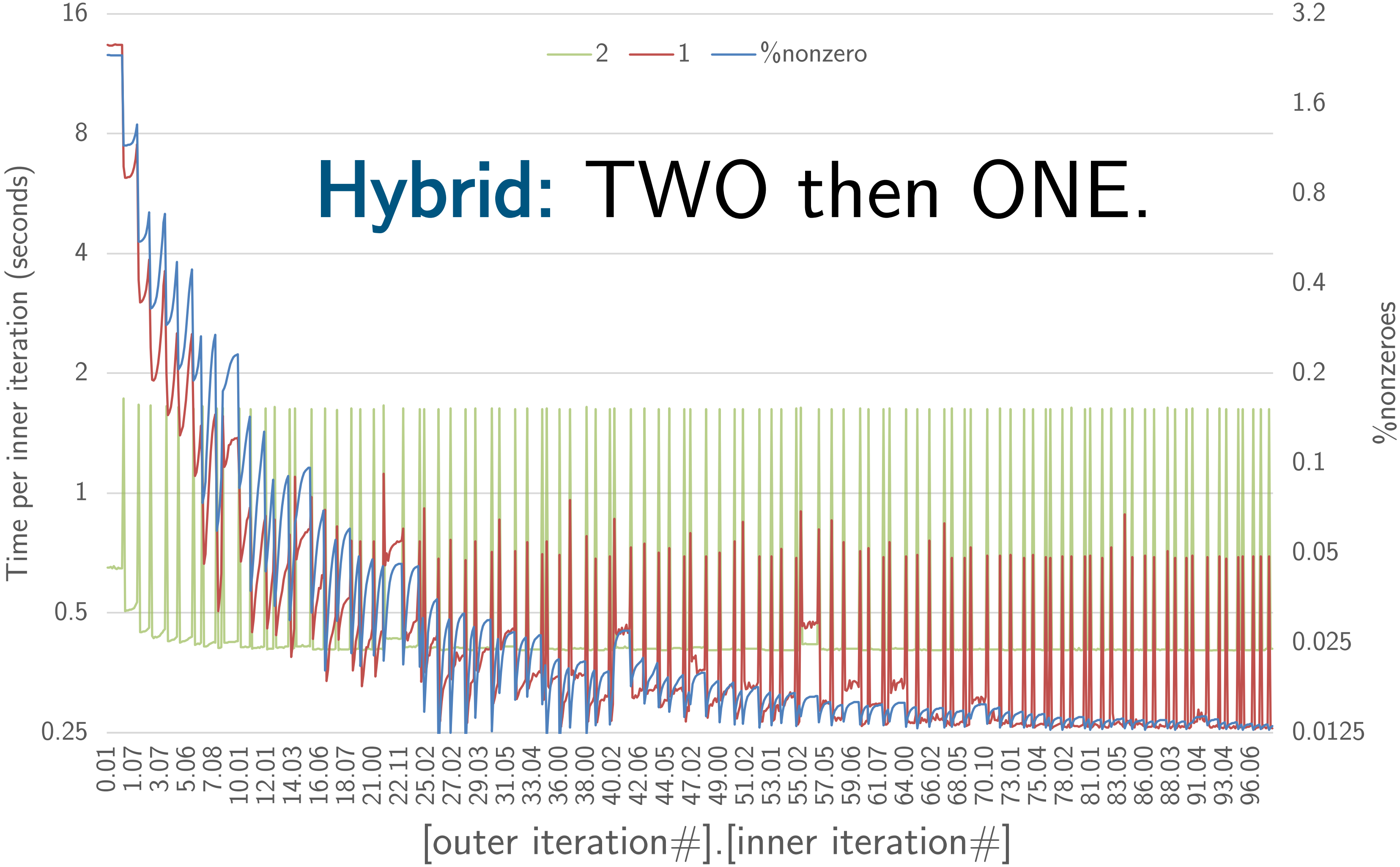
1.2k x 110k: Iterations 75-99

6,144 cores on Edison. 790 inner iterations. 0.095%nz (avg). 0.013%nz (final)



1.2k x 110k: Iteration costs

6,144 cores on Edison. 790 inner iterations. 0.095%nz (avg). 0.013%nz (final)



Conclusion

- **Highly scalable.** Enables processing high-dimensional datasets with arbitrary underlying structure.
- **2 approaches:** ONE wins when %nonzero $\cdot$ $\#$avg inner iter $\ll n/p$
- **Communication-avoiding:** up to 10.18× faster than non-CA.
- Plenty of room for improvement
  - **Hybrid:** switch from TWO to ONE midway.
  - Replication factors can change halfway too.
  - Communication overlapping
  - Account for different flop rates.
  - Approximate step size and use s-step trick.
- Software releases:
  - https://people.eecs.berkeley.edu/~penpornk/spdm3
  - https://people.eecs.berkeley.edu/~penpornk/hp-concord
  - Python package released soon.
- I'm looking for more applications!

References

- **CONCORD**
  S. Oh, O. Dalal, K. Khare, and B. Rajaratnam,
  Optimization methods for sparse pseudo-likelihood graphical model selection.
  In NIPS 27, 2014, pp. 667–675.
- **BigQUIC**
  Cho-Jui Hsieh, Mátyás A Sustik, Inderjit S Dhillon, Pradeep K Ravikumar, and Russell Poldrack.
  Big & quic: Sparse inverse covariance estimation for a million variables.
  In NIPS, pages 3165–3173, 2013.
- **SpDM^3**
  P. Koanantakool, A. Azad, A. Buluç, D. Morozov, S. Oh, L. Oliker, K. Yelick.
  Communication-avoiding parallel sparse-dense matrix-matrix multiplication.
  In IPDPS 2016.

Thank you!
Questions?