Distributed-memory parallel algorithms based on rank-structured matrices

Chao Chen, University of Texas at Austin

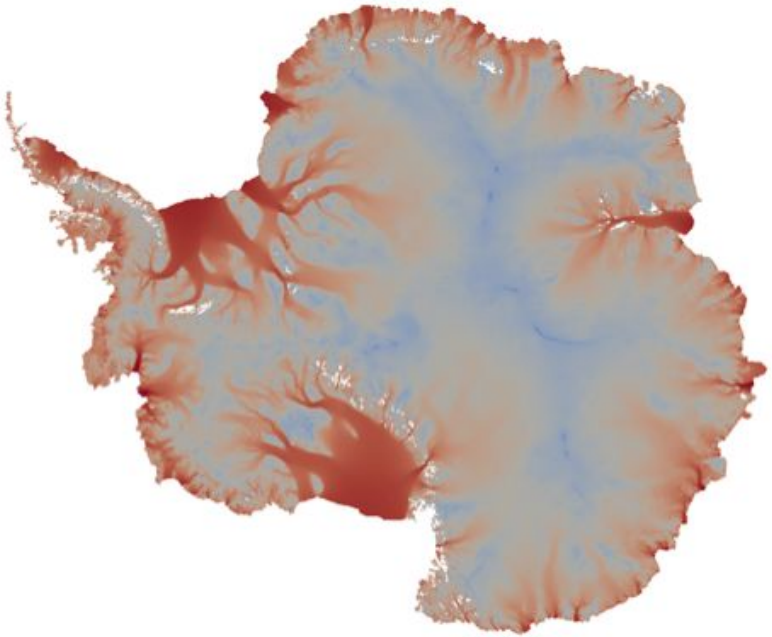
Conference on Fast Direct Solvers
October 23, 2021

Collaborators: G. Biros, E. Boman, L. Cambier, E. Darve, T. Liang, H. Pouransari, S. Rajamanickam, and R. Tuminaro

Sparse linear solver

- Sea level could rise up to ~58 meters*

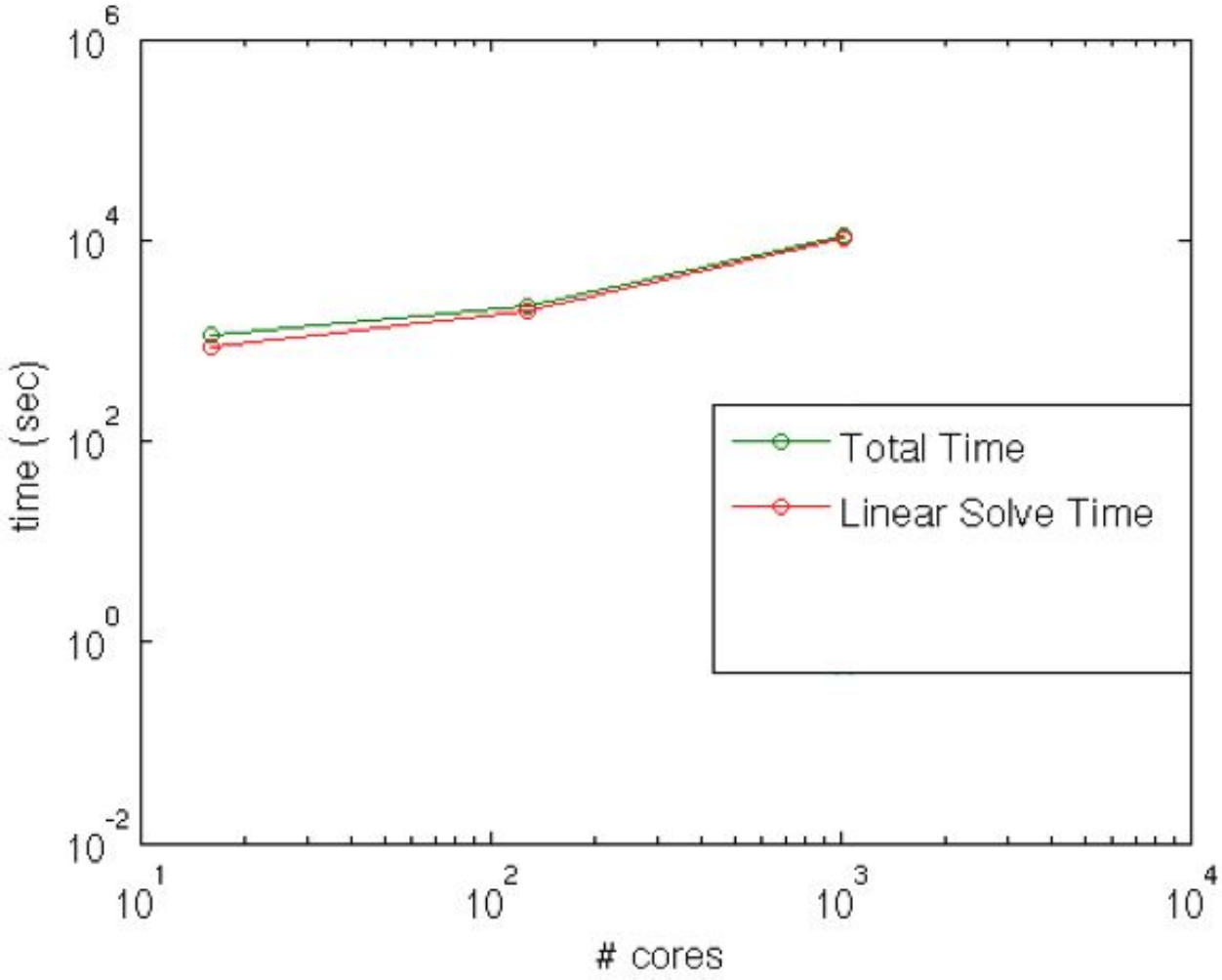
- Ice sheet modeling of Antarctica
  - numerous linear solves
  - up to 1 billion unknowns
  - thousands of cores



*Estimates given by Prof. Richard Alley of Penn State, who testified in 1999 about climate change to Al Gore.

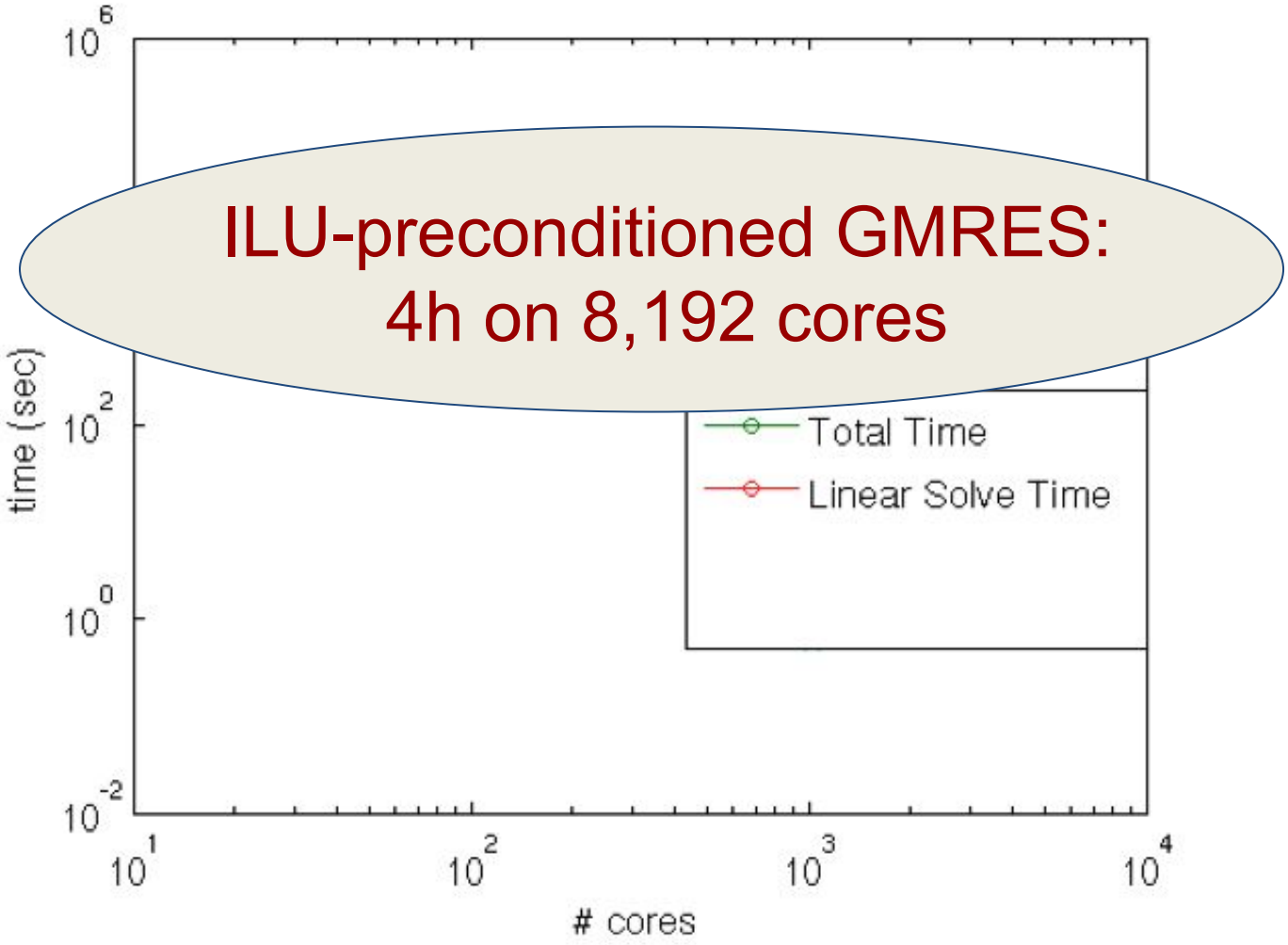
Tuminaro et al., 2016

Sparse linear solver



Tezaur et al., 2015

Sparse linear solver

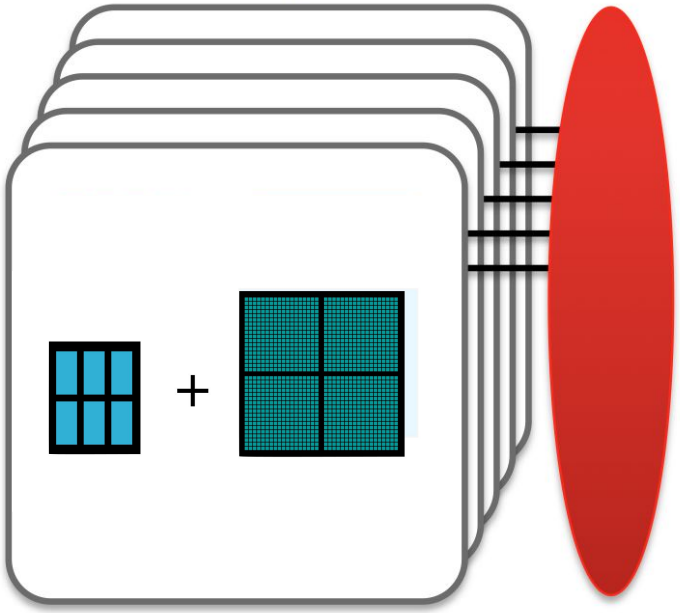


Tezaur et al., 2015

What's needed?

- Fast and robust algorithms
  - $O(N (\log N)^k)$ computation and memory
  - elliptic PDEs (not highly indefinite)

- Highly parallel softwares
  - distributed memory
  - CPUs + GPUs



Supercomputer with ~10,000 nodes

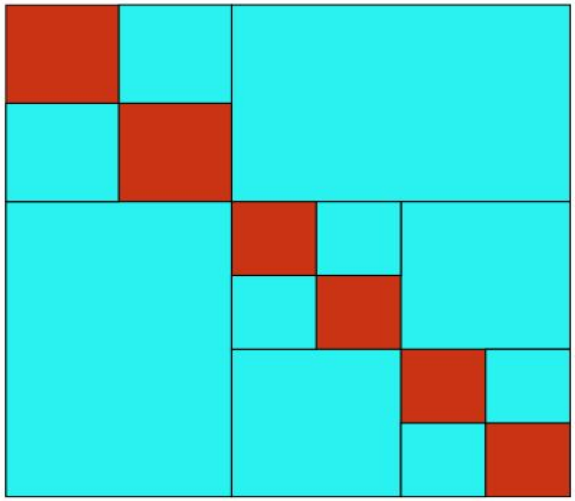
Existing approaches

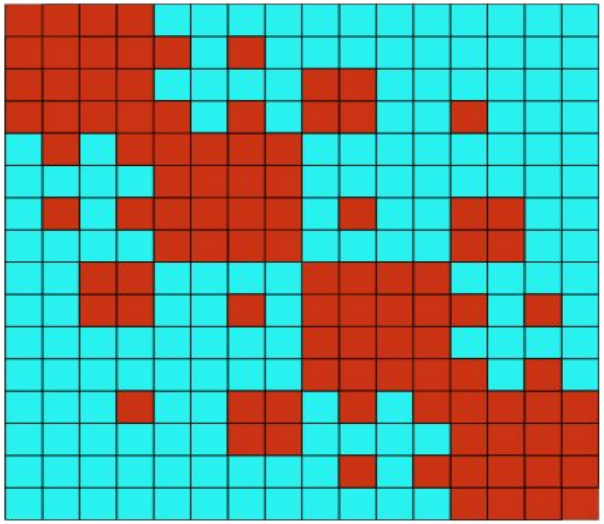
- Sparse direct solvers
  - use efficient elimination ordering to minimize fill-in
  - UMFPACK ('\' in Matlab), SuperLU, Pardiso ….
  - fill-in in 3D –> computation: $O(N^2)$, memory: $O(N^{4/3})$
- Iterative solvers
  - CG, multigrid, ...
  - Hypre, Trilinos, PETSc, …
  - computation: $O(N)$ x #iter, memory: $O(N)$
- Fast direct solvers
  - tunable accuracy
  - fast and robust
  - high arithmetic intensity (# flops/byte)
  - reduced communication and synchronization

Fast direct solvers

- Hierarchical matrices [Bebendorf, Borm, Darve, Gu, Hackbusch, Martinsson, Xia, etc.]

  - elliptic boundary value problems lead to low-rank off-diagonal blocks.





Aminfar et al., 2015

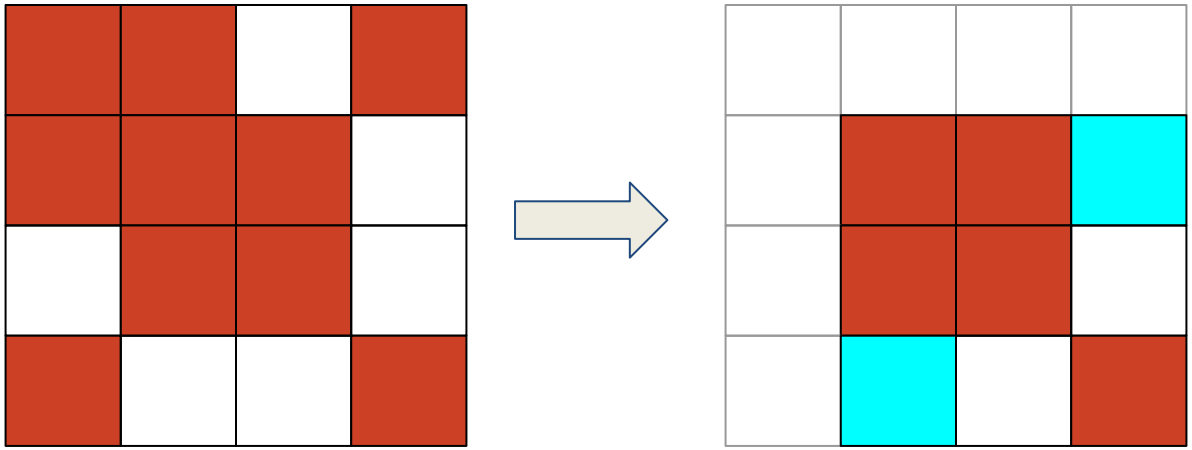
A paradigm of fast direct solvers

$$\mathcal{W}_{m-1} \dots \mathcal{W}_1 \mathcal{W}_0 \, A \, \mathcal{W}_0^T \mathcal{W}_1^T \dots \mathcal{W}_{m-1}^T \approx I$$

Form a block matrix with “near” and “far” matrix blocks
For every level {
    For every block rows/columns {
        compress “far” matrix blocks
        eliminate “fine” rows/columns
    }
    merge ‘‘coarse’’ rows/columns to form next level
}

Sparse solvers: Pouransari et al., 2017, Sushnikova et al., 2018
Dense solvers: Ambikasaran et al., arXiv, Coulier et al., 2017, Minden et al.,2017

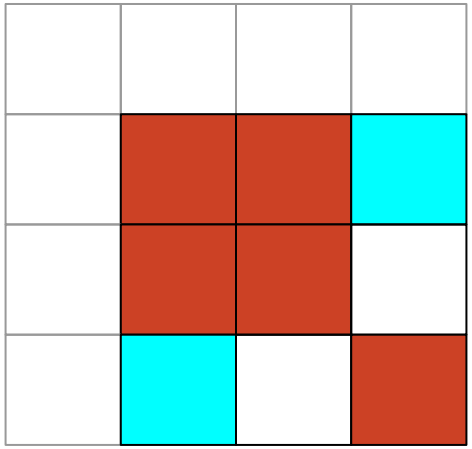
A block sparse matrix



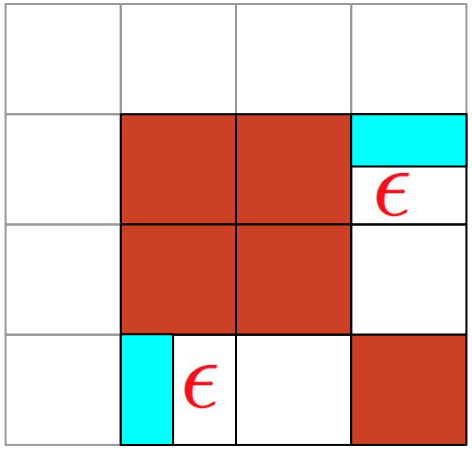
Sushnikova et al., 2018

“near” matrix blocks: original nonzero blocks.
“far” matrix blocks: other blocks (initially zero).

Compress “far”/fill-in matrix blocks



$$F = U \begin{pmatrix} \tilde{V}^T \\ \epsilon \end{pmatrix}$$



A_1

$\mathcal{E}_s A_1 \mathcal{E}_s^T$

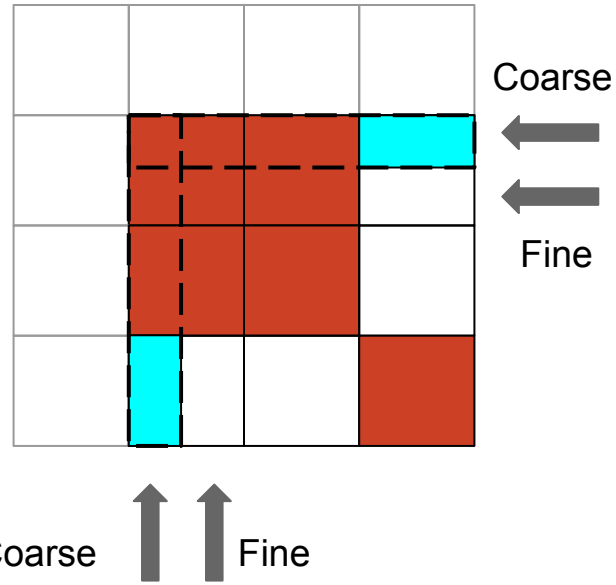
$$\mathcal{E}_s = \begin{pmatrix} U^T & & \\ & I & \\ & & I \end{pmatrix}$$

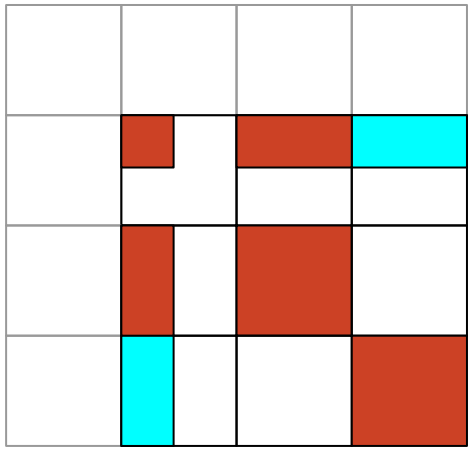
$$PF = \begin{pmatrix} \hat{F} \\ T\hat{F} + \epsilon \end{pmatrix}$$

Compress “far”/fill-in matrix blocks

- “coarse” rows/columns -> keep to next level
- “fine” rows/columns -> eliminate (existing fill-in not touched)

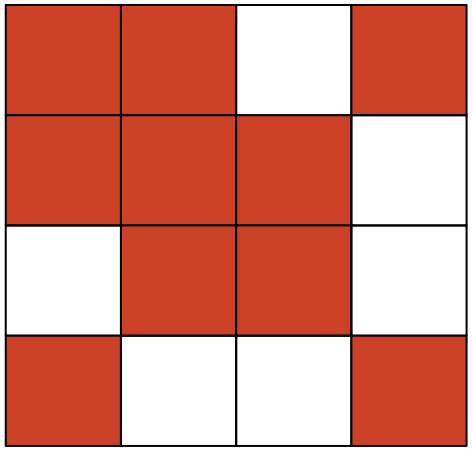
$\mathcal{E}_s A_1 \mathcal{E}_s^T \approx$

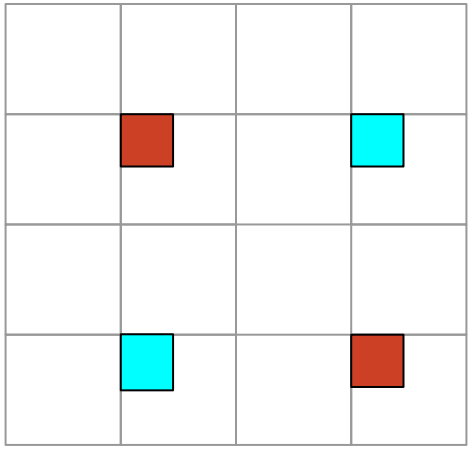




$\mathcal{G}_s \mathcal{E}_s A_1 \mathcal{E}_s^T \mathcal{G}_s^T$

After one level





$$\mathcal{W}_{m-1} \ldots \mathcal{W}_1 \mathcal{W}_0 \, A \, \mathcal{W}_0^T \mathcal{W}_1^T \ldots \mathcal{W}_{m-1}^T \approx \begin{pmatrix} I & \\ & A_c \end{pmatrix}$$

A paradigm of fast direct solvers

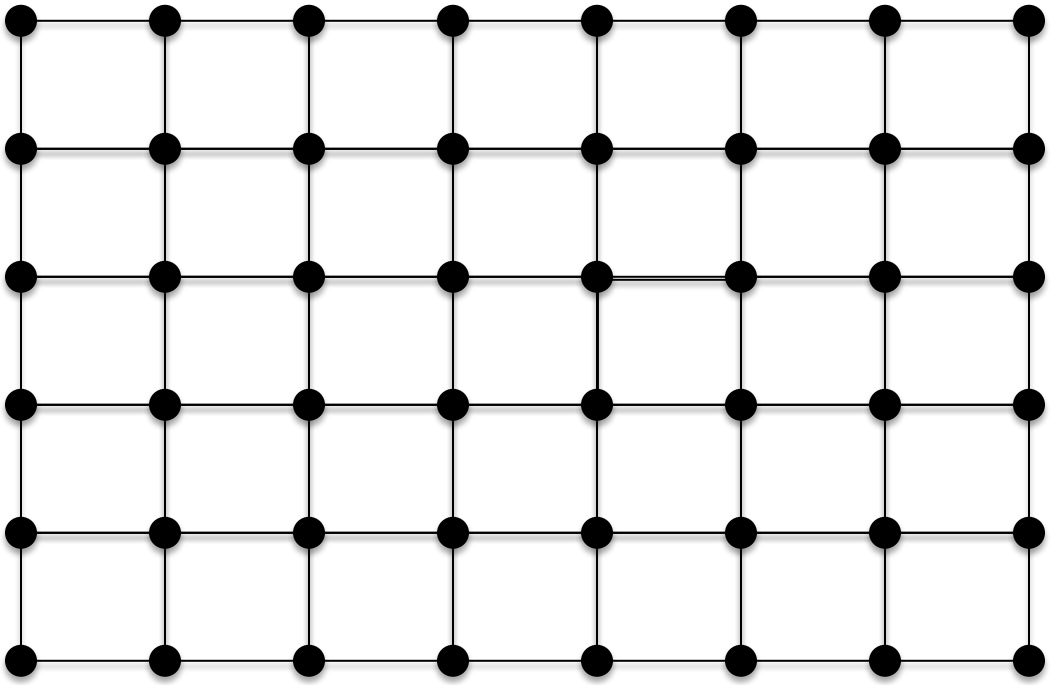
$$\mathcal{W}_{m-1} \ldots \mathcal{W}_1 \mathcal{W}_0 \, A \, \mathcal{W}_0^T \mathcal{W}_1^T \ldots \mathcal{W}_{m-1}^T \approx I$$

Form a block matrix with “near” and “far” matrix blocks
For every level {
    For every block rows/columns {
        compress “far” matrix blocks
        eliminate “fine” rows/columns
    }
    merge ‘‘coarse’’ rows/columns to form next level
}

Sparse solvers: Pouransari et al., 2017, Sushnikova et al., 2018
Dense solvers: Ambikasaran et al., arXive, Coulier et al., 2017, Minden et al.,2017

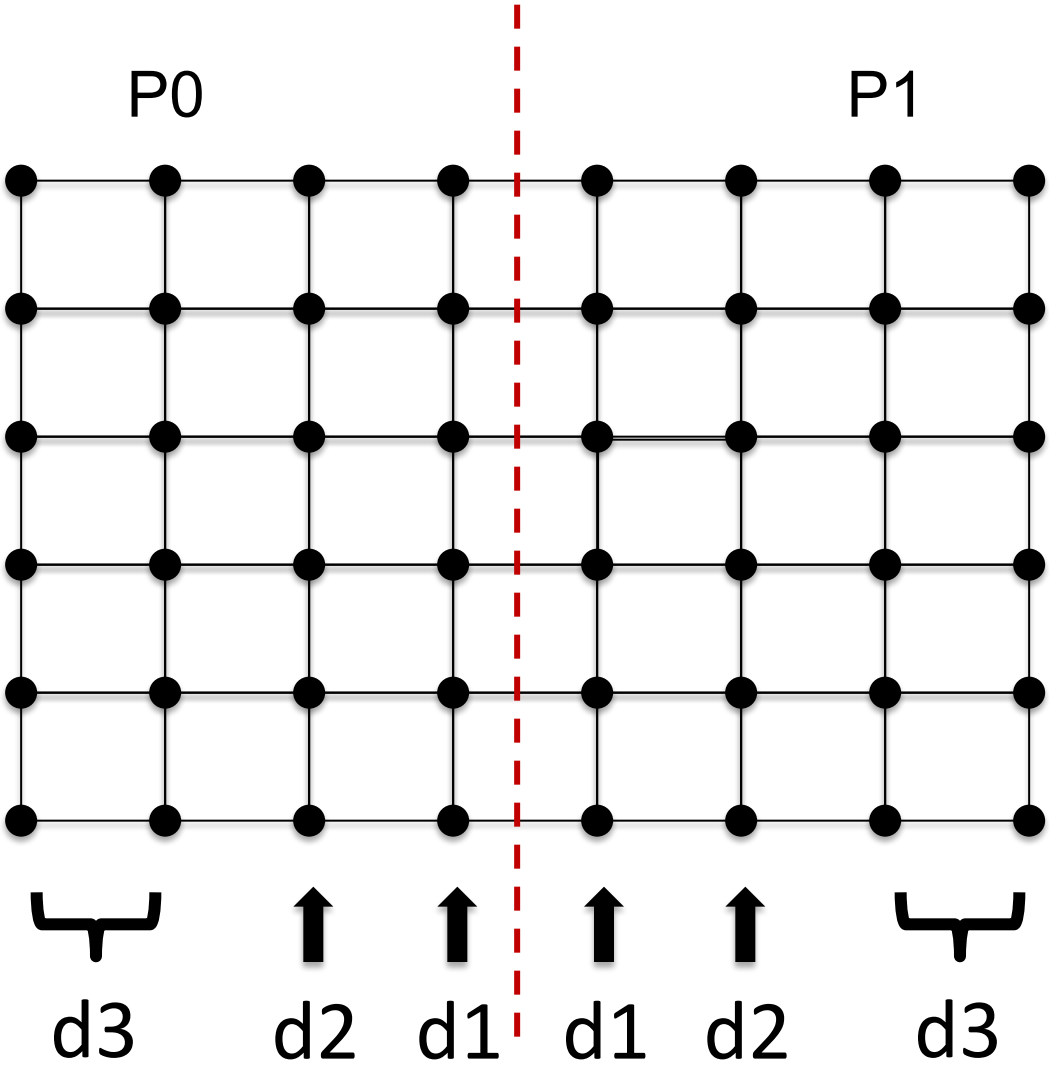
Parallelism

Theorem [Chen et al., 2018]:
fill-in exists only between two nodes of distance 2 (i.e., neighbor's neighbor) in the underlying graph.



Data decomposition

- d1
  - coupled among neighbors
- d2
  - processed in parallel
  - requires communication
- d3
  - processed in parallel
  - no communication



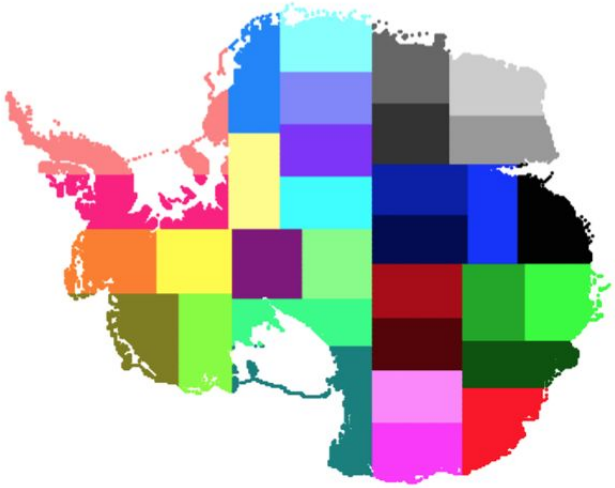
Communication & Computation in 3D

Matrix size: N, # processors: p, rank: r

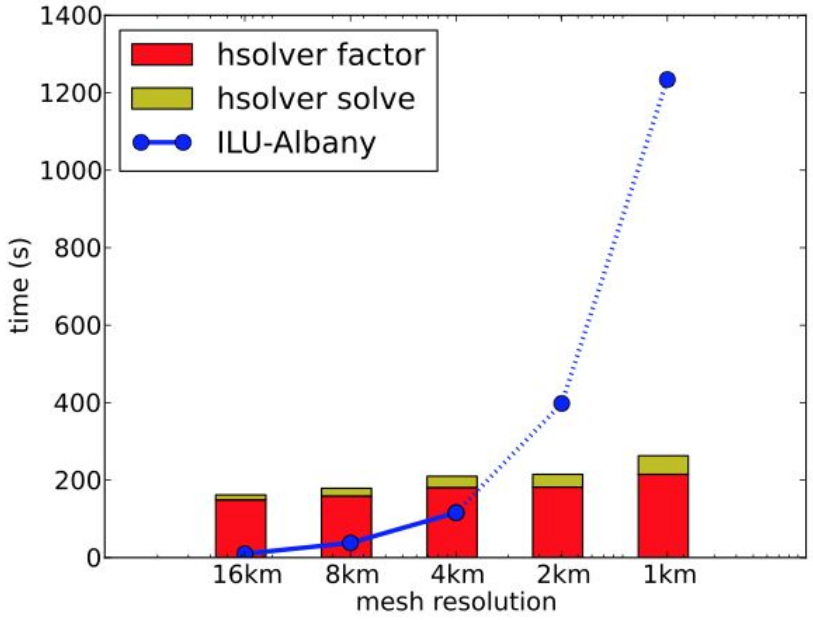
Define $M = (Nr^2/p)^{1/3}$

- Communication
  - local communication
  - exchange $O(M^2)$ boundary data
  - # messages: $O(\log(N/rp)+\log(p))$
- Computation
  - computation: $O(M^3) = O(Nr^2/p)$
  - local dense linear algebra
- Memory
  - memory: $O(Nr/p)$

Ice sheets simulation



- Parallel asynchronous implementation using MPI
- Deferred compression scheme [Gu 2010, Xia 2010&2017, Chen et al., 2019, Feliu-Fabà and Ying, 2020, Cambier et al., 2020]



weak scalability (12-digits' accuracy)

Ice sheets simulation

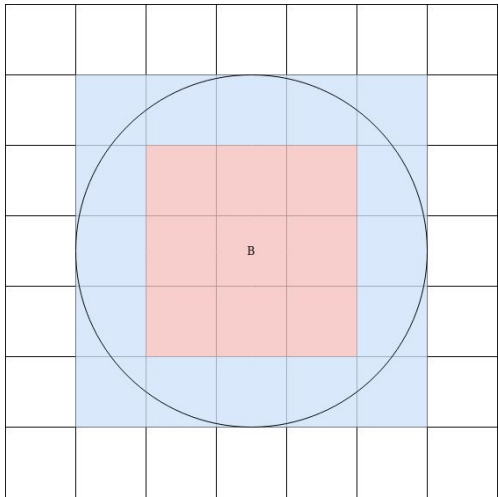
Table 8: 11 vertical mesh layers: hierarchical solver ($\epsilon = 10^{-2}$) vs. ILU.

			ILU		hierarchical solver		
h	N	P	iter #	total time	iter #	factor	solve
16km	1.1M	4	90	7	18	147	22
8km	4.6M	16	183	21	23	186	38
4km	18.5M	64	468	66	24	213	53
2km	74M	256	1000[a]	—	27	214	65
1km	296M	1024	1000[b]	—	27	243	71

time (s) for 12-digits' accuracy

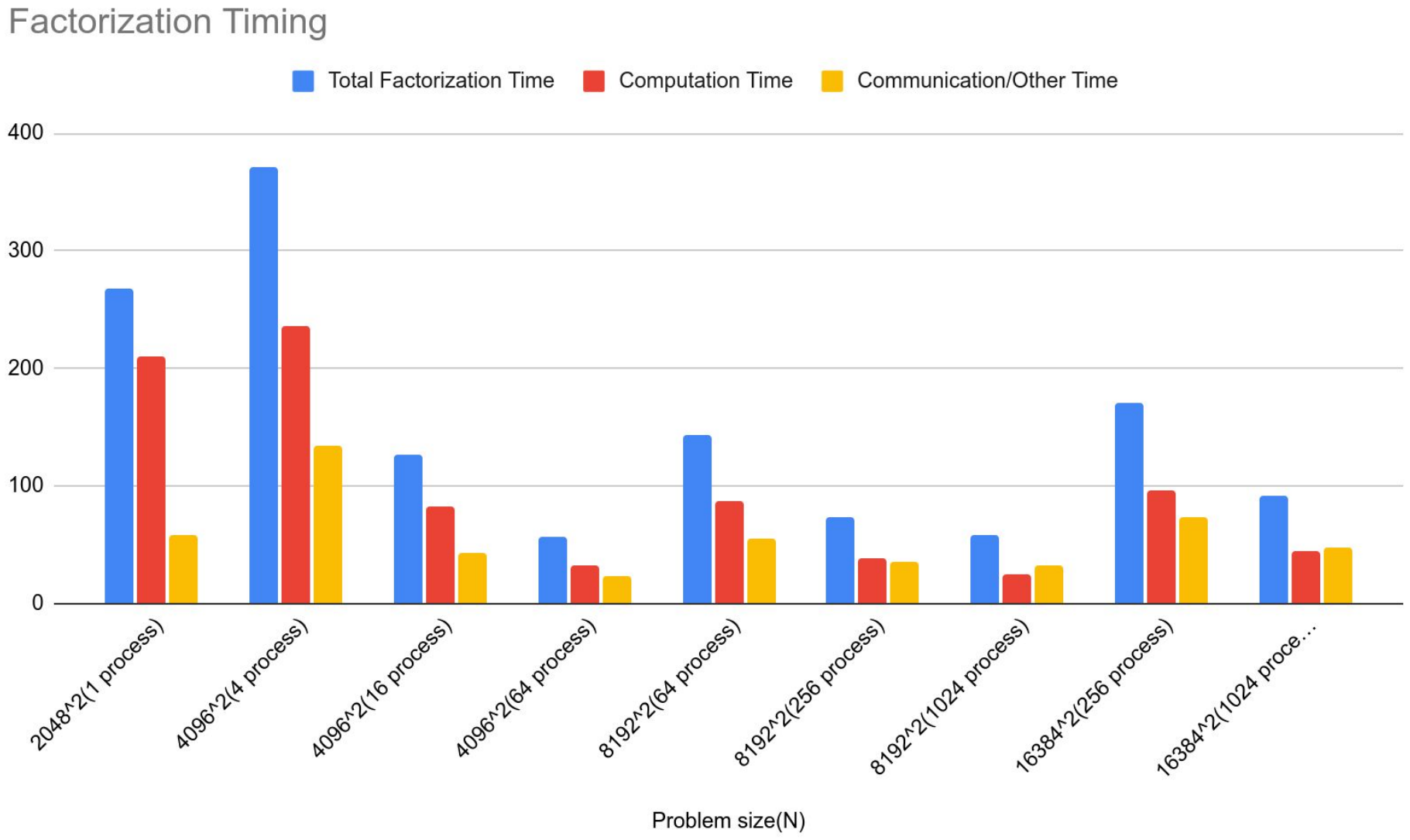
Same framework for FMM-matrices

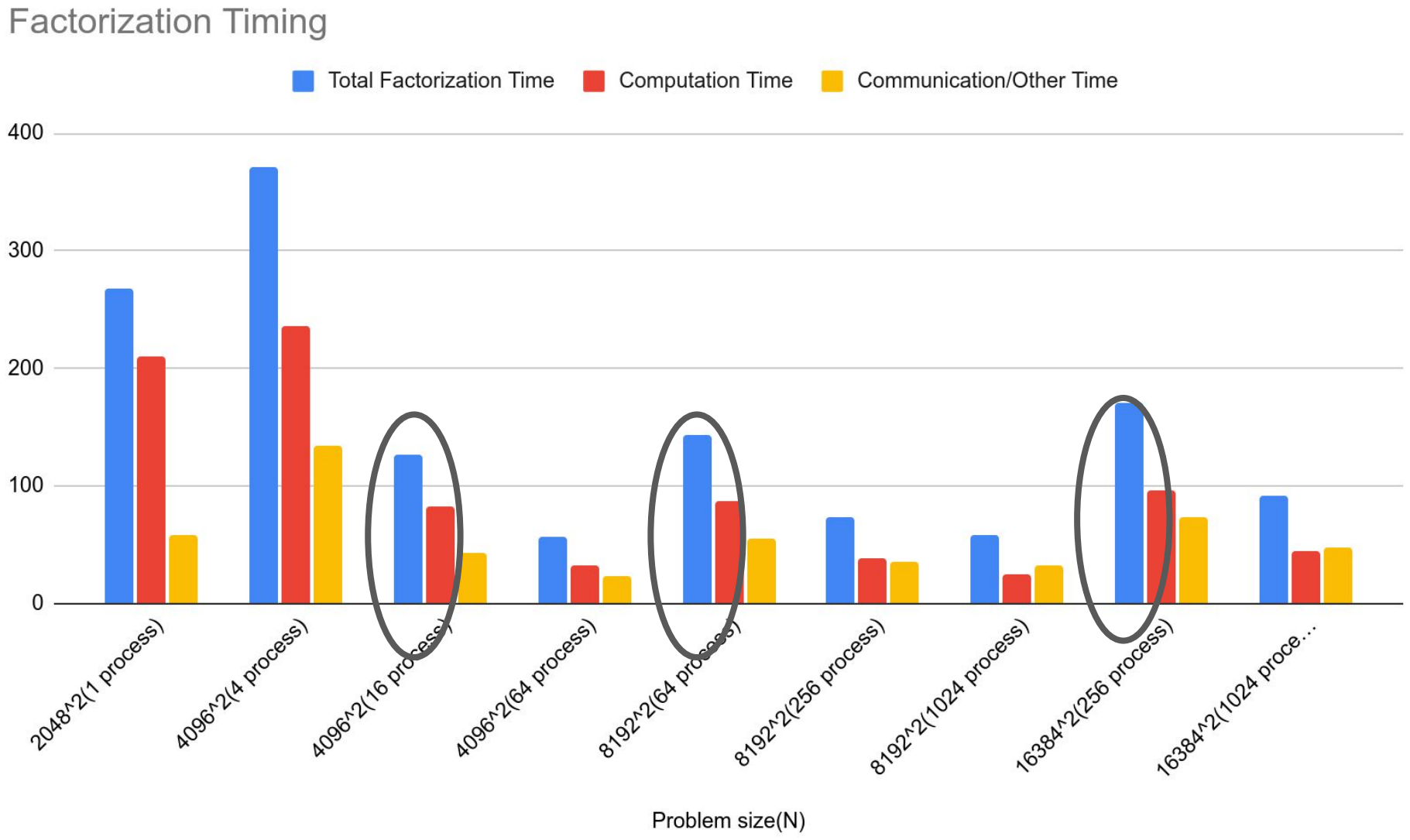
Form a block matrix with “near” and “far” matrix blocks
For every level {
    For every block rows/columns {
        compress “far” matrix blocks
        eliminate “fine” rows/columns
    }
    merge ‘‘coarse’’ rows/columns to form next level
}

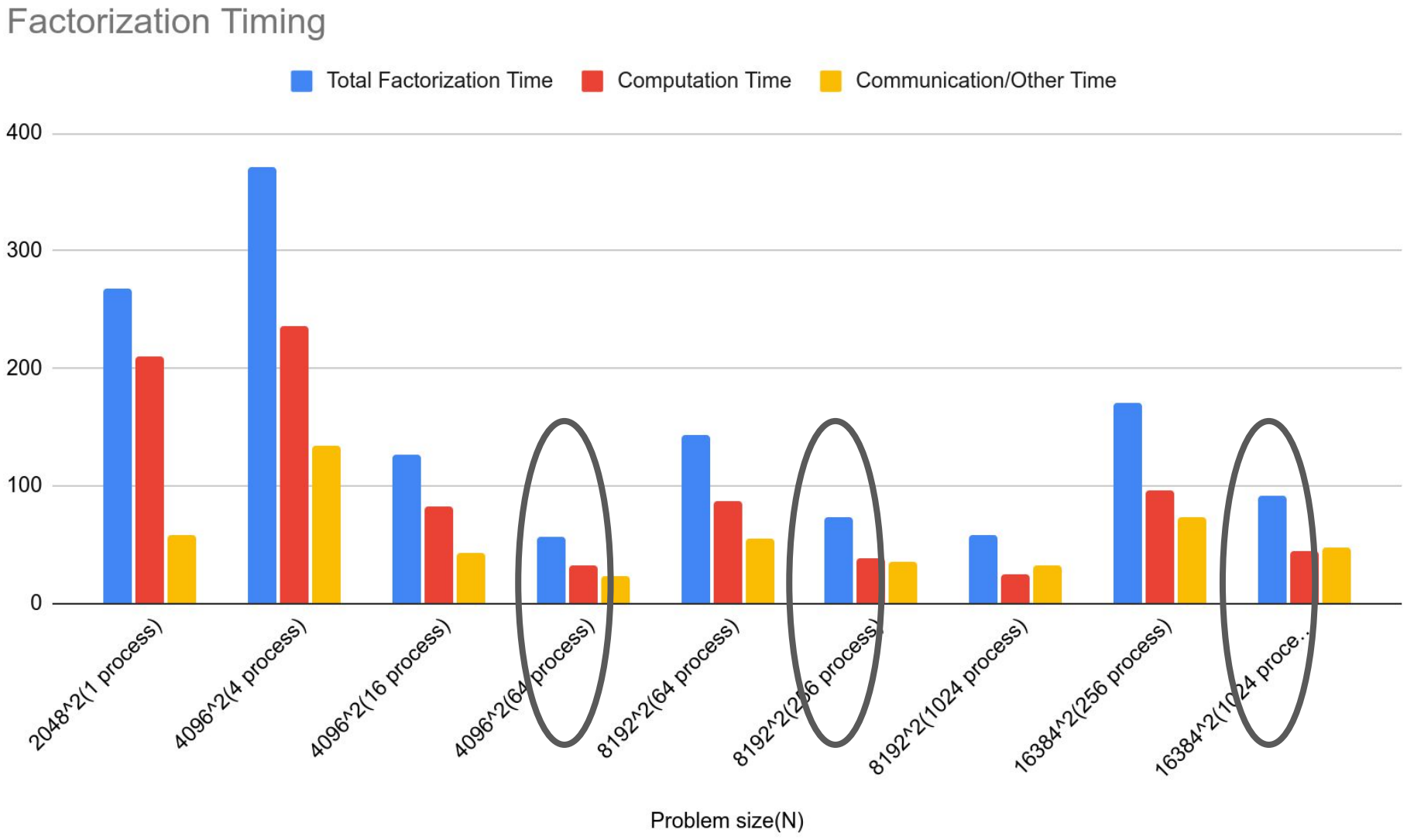


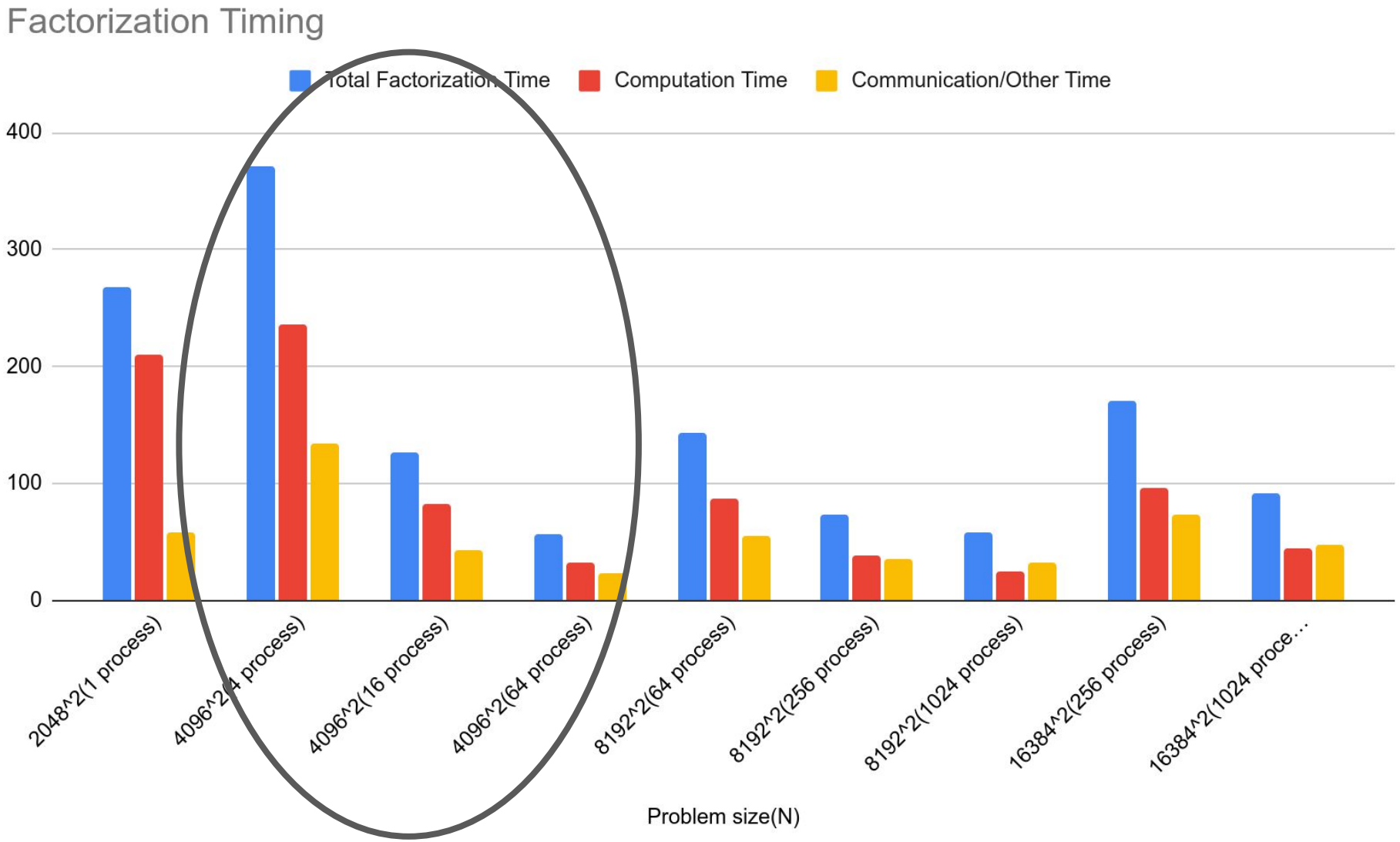
compression:

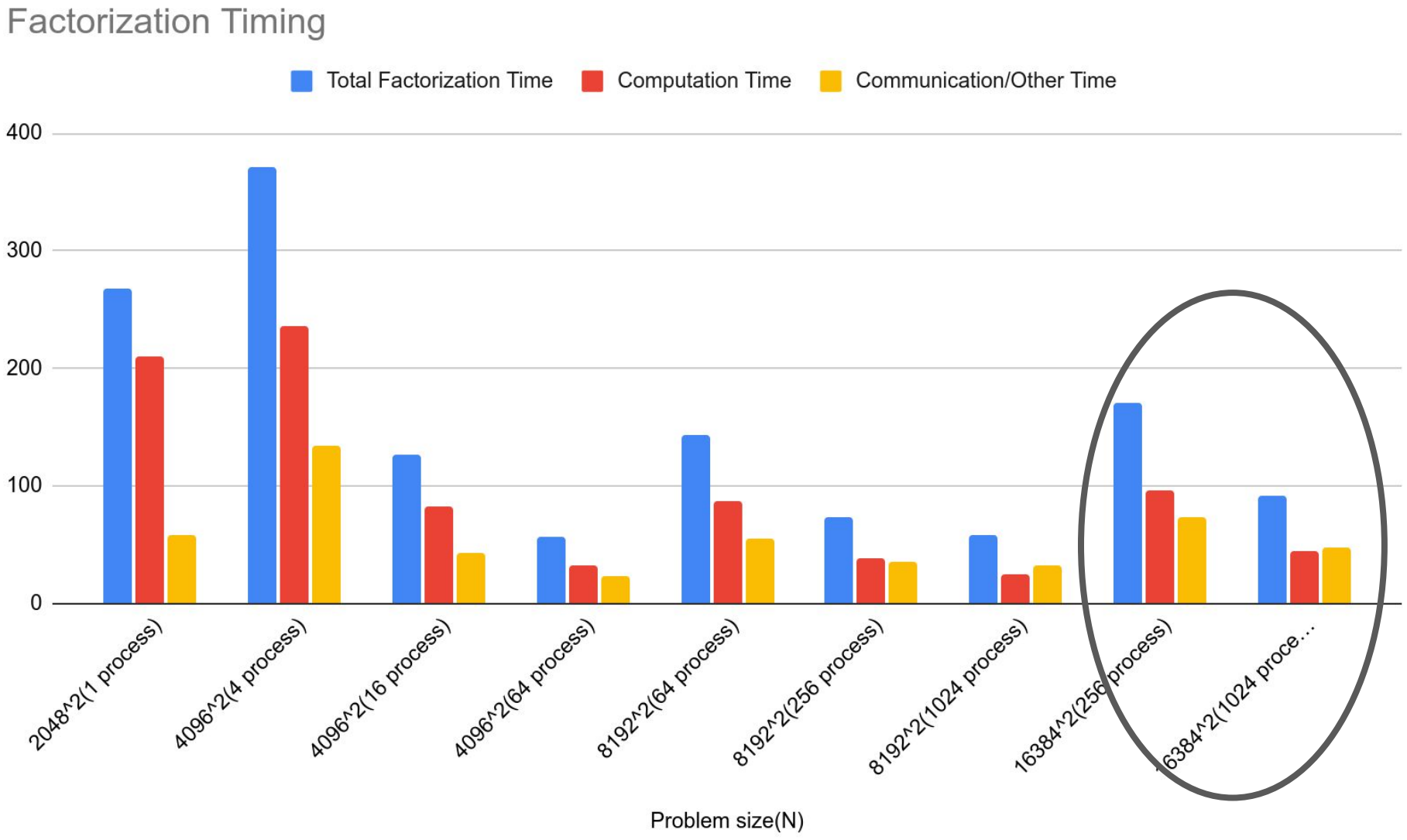
- ID + proxy surface [Minden et al., 2017]
- Chebyshev interpolation + SVD
  [Ambikasaran and Darve, arXiv, Coulier et al., 2017]


Factorization Timing
Total Factorization Time
Computation Time
Communication/Other Time
400
300
200
100
0
2048^2(1 process)
4096^2(4 process)
4096^2(16 process)
4096^2(64 process)
8192^2(64 process)
8192^2(256 process)
8192^2(1024 process)
16384^2(256 process)
16384^2(1024 proce…
Problem size(N)


Factorization Timing
Total Factorization Time
Computation Time
Communication/Other Time
400
300
200
100
0
2048^2(1 process)
4096^2(4 process)
4096^2(16 process)
4096^2(64 process)
8192^2(64 process)
8192^2(256 process)
8192^2(1024 process)
16384^2(256 process)
16384^2(1024 proce…
Problem size(N)


Factorization Timing
Total Factorization Time
Computation Time
Communication/Other Time
400
300
200
100
0
2048^2(1 process)
4096^2(4 process)
4096^2(16 process)
4096^2(64 process)
8192^2(64 process)
8192^2(256 process)
8192^2(1024 process)
16384^2(256 process)
16384^2(1024 proce...
Problem size(N)


Factorization Timing
Total Factorization Time
Computation Time
Communication/Other Time
400
300
200
100
0
2048^2(1 process)
4096^2(4 process)
4096^2(16 process)
4096^2(64 process)
8192^2(64 process)
8192^2(256 process)
8192^2(1024 process)
16384^2(256 process)
16384^2(1024 proce…
Problem size(N)


Factorization Timing
Total Factorization Time
Computation Time
Communication/Other Time
400
300
200
100
0
2048^2(1 process)
4096^2(4 process)
4096^2(16 process)
4096^2(64 process)
8192^2(64 process)
8192^2(256 process)
8192^2(1024 process)
16384^2(256 process)
16384^2(1024 proce…
Problem size(N)