Exploring the Design Space of Distributed Parallel SpMM

Hua Huang and Edmond Chow

Georgia Tech

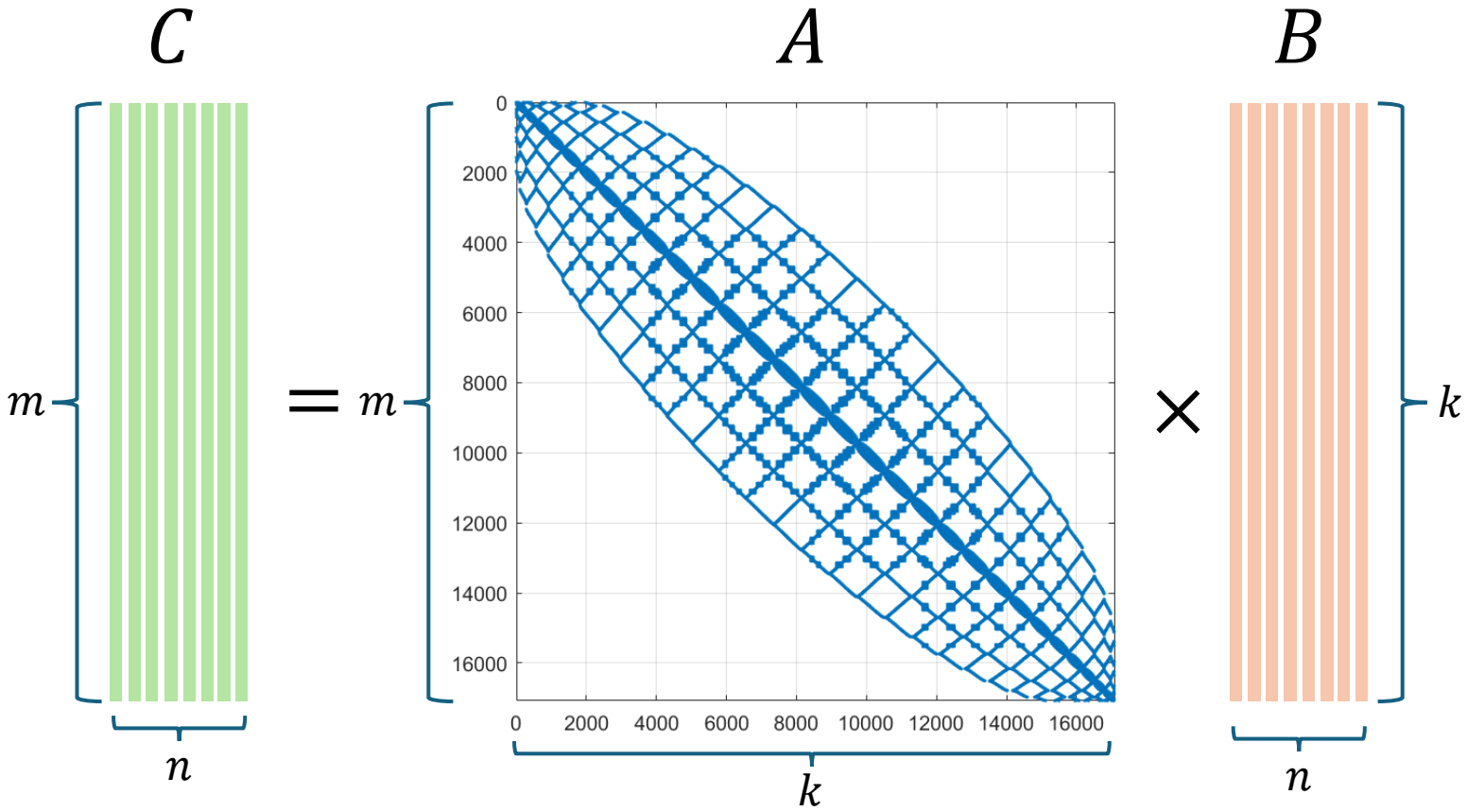
SIAM PP24, 03/07/2024

Overview

- Background
- SpMM parallelization design space
- Data transfers in parallel SpMM
- CRP-SpMM
- Numerical results

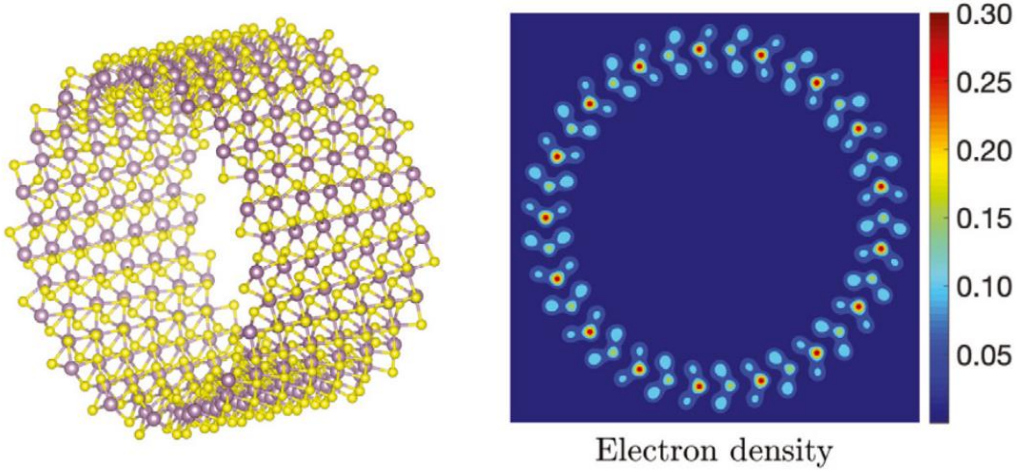
SpMM

- Sparse matrix – dense matrix (multiple vectors) multiplication
- $n = 1$ → SpMV, building block of many algorithms

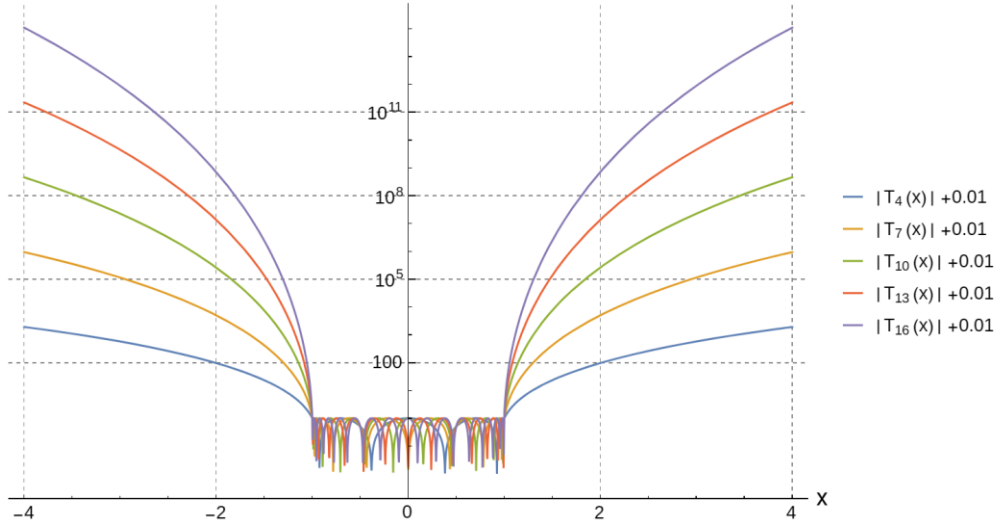


SpMM applications

- Blocked iterative solvers: CheFSI, LOBPCG, etc.



$$H\Psi = E\Psi$$



$$\Phi = p_{cheb}(H, \Phi; m)$$

SpMM applications

- Non-negative matrix factorization

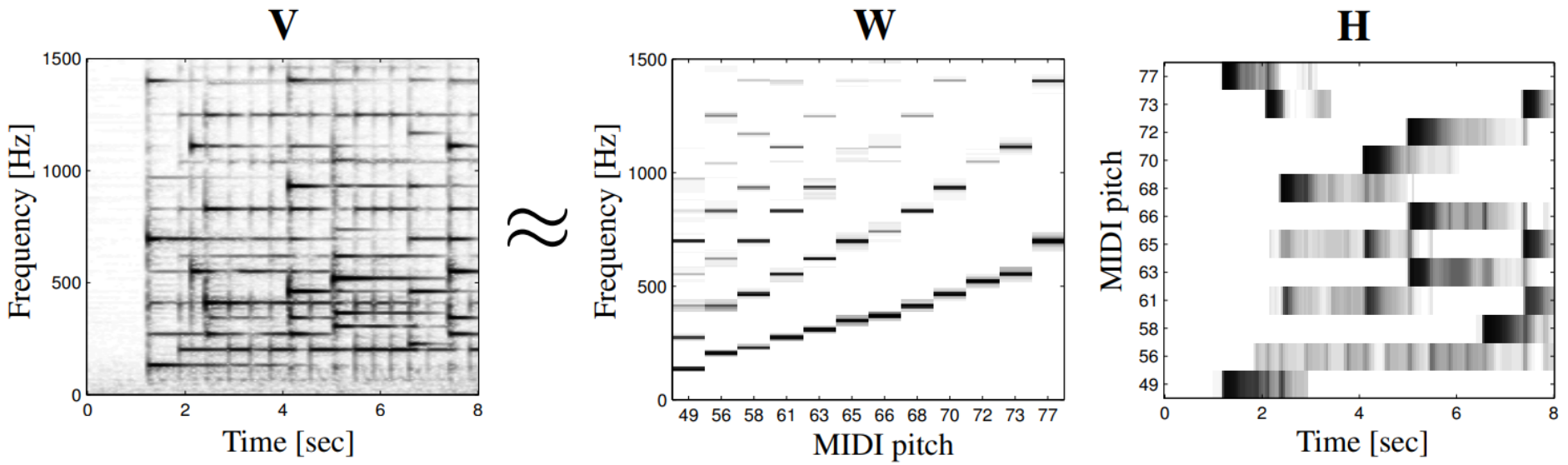
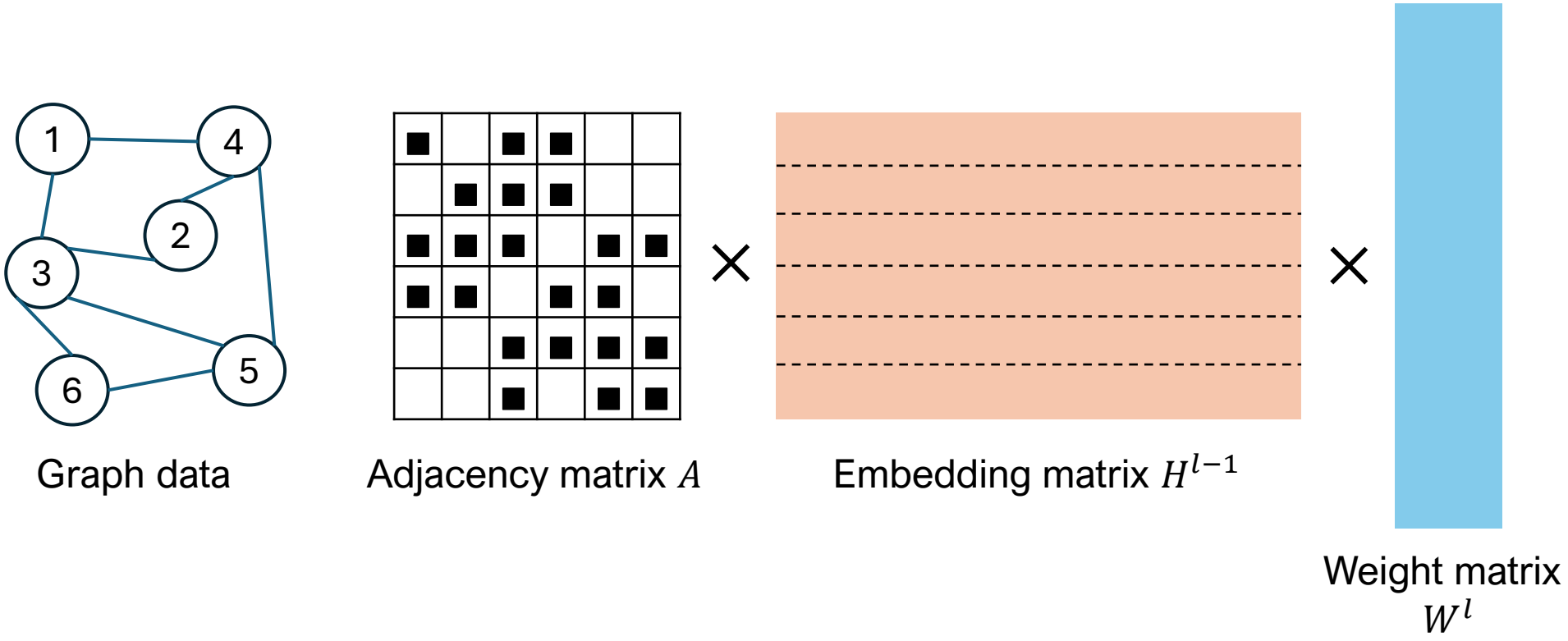


Figure from: Ewert *et al.*, Score-Informed Source Separation for Music Signals

SpMM applications

- Graph neural network

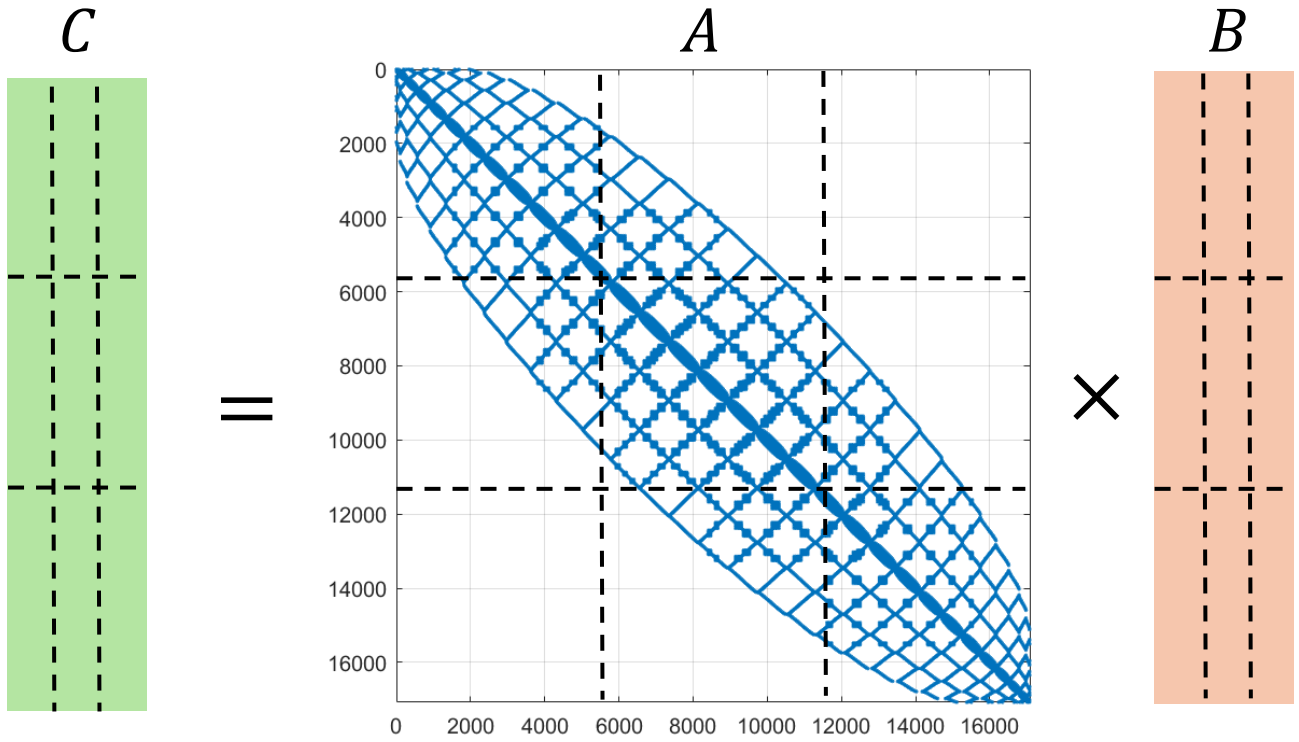


Distributed-memory parallel SpMM

- Communication dominates the overall cost
- Irregular calculation and communication patterns
- Much larger design space than dense matmul

GEMM-based SpMM

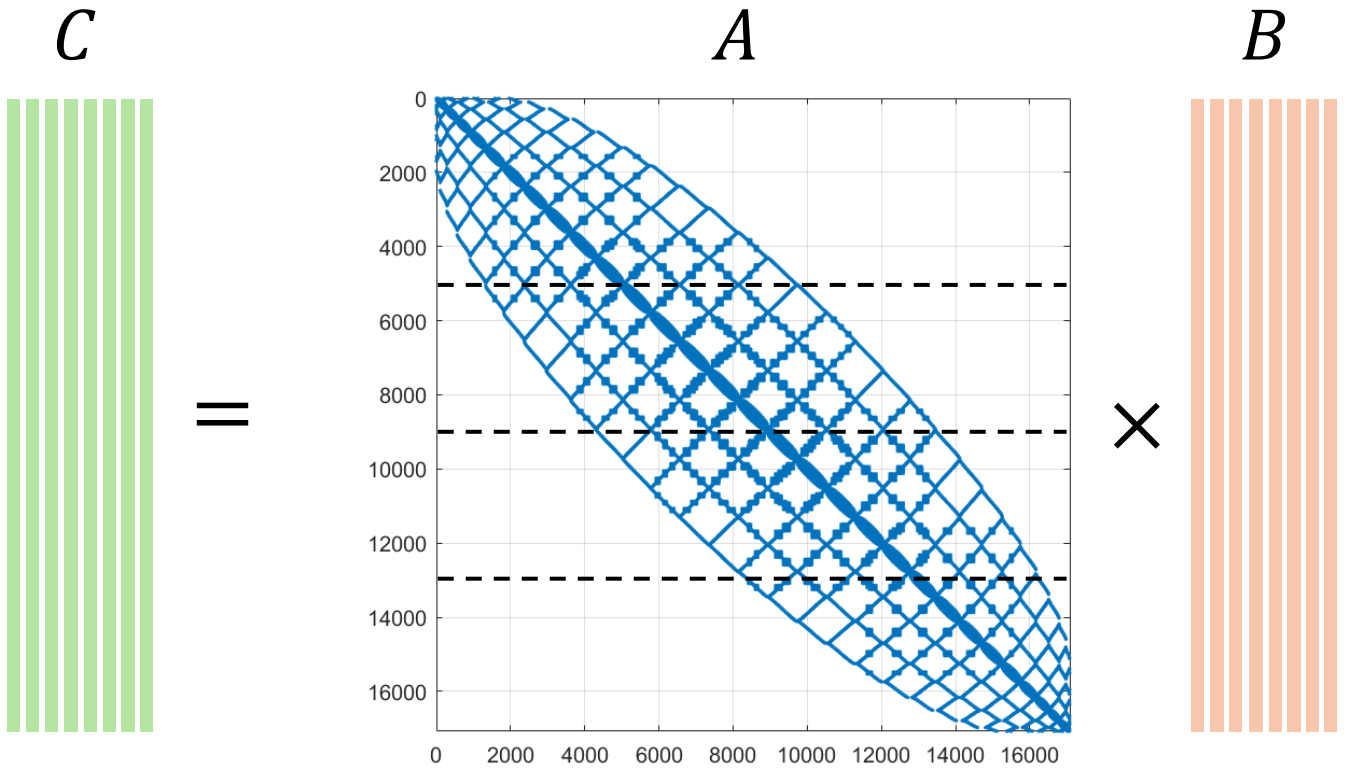
- Use parallel GEMM (dense matmul) algorithms for SpMM



- Not utilizing sparsity → overhead and load imbalance

SpMV-based SpMM

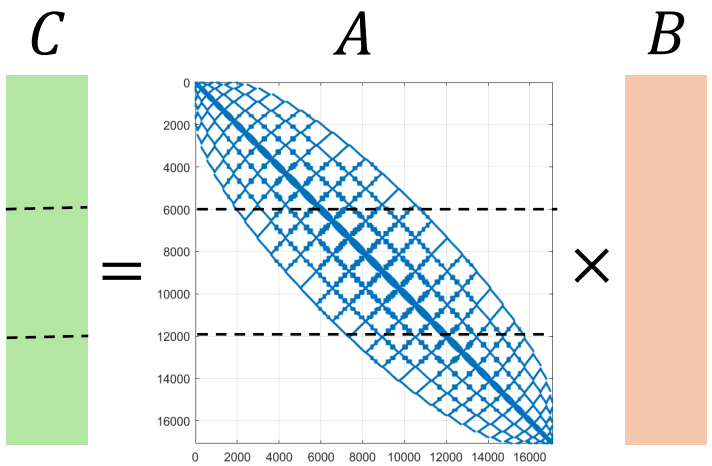
- Use parallel SpMV algorithms for multiple vectors



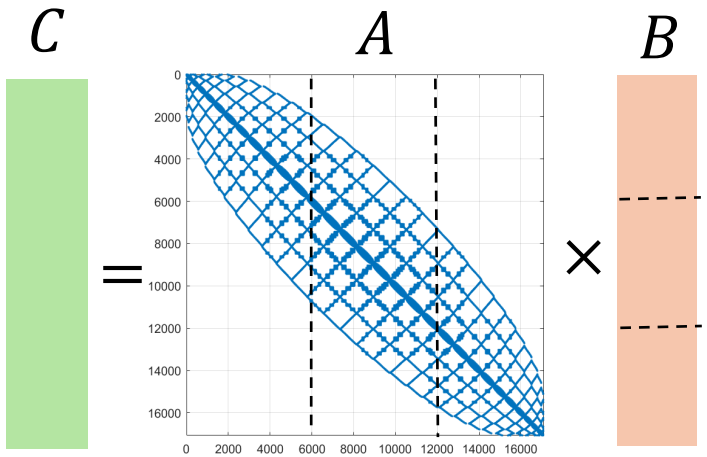
- Not utilizing n-dimension parallelism (columns of *B*)

SpMM parallelization design space

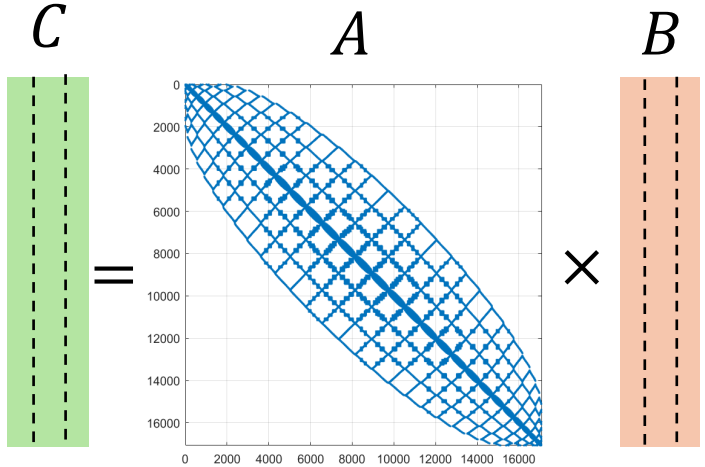
1D parallelization of SpMM



1D m-parallelization



1D k-parallelization



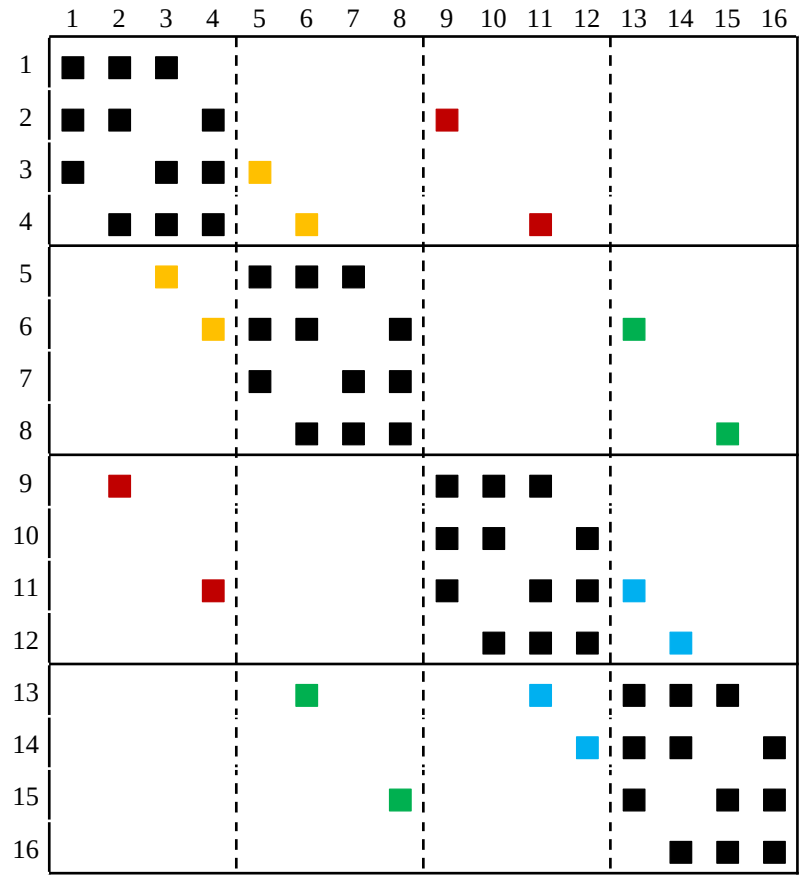
1D n-parallelization

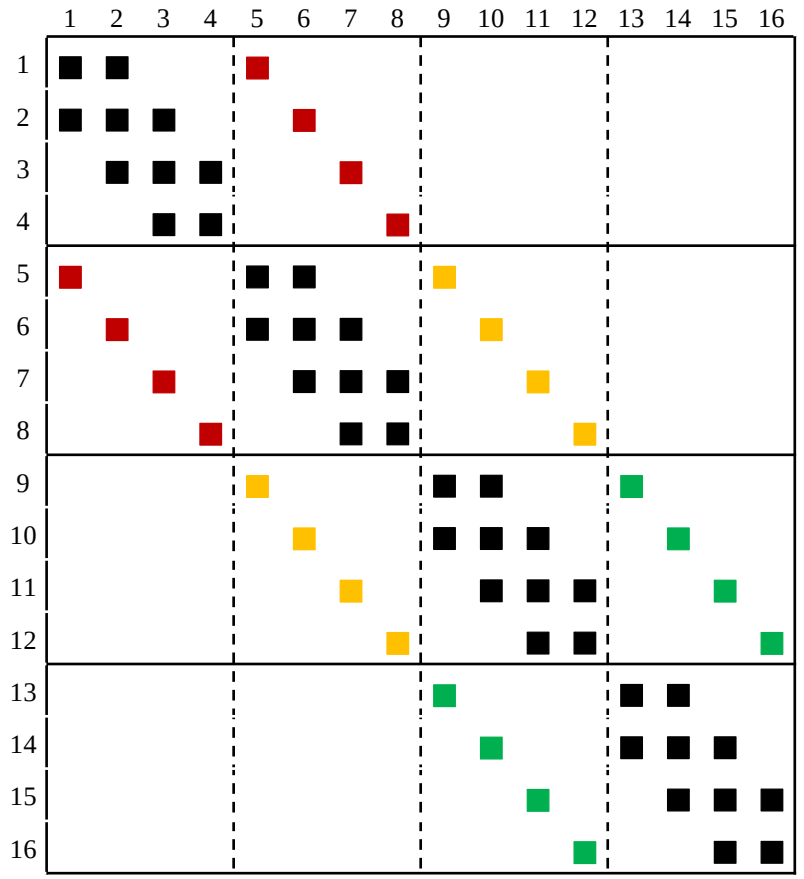
2D and 3D parallelization of SpMM

- 2D and 3D schemes: combining 1D schemes
  - mk-, mn-, kn-, and mkn-parallelization
- Spanned by two perpendicular subspaces
  - Sparse matrix partitioning
  - Process grid size

Design subspace 1: matrix partitioning

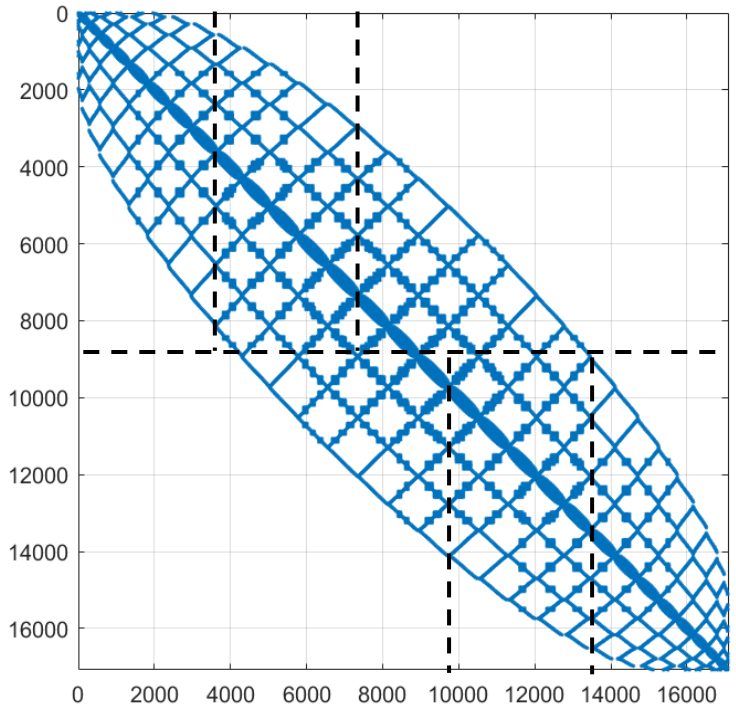
- Row/column reordering (RCM, AMD, …)

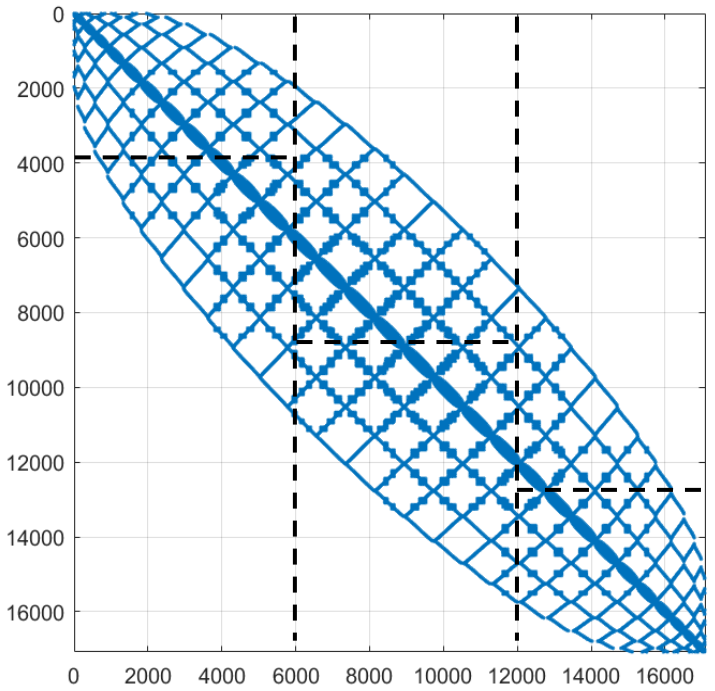




Design subspace 1: matrix partitioning

- 2D partitioning methods (jagged, fine-grain, …)

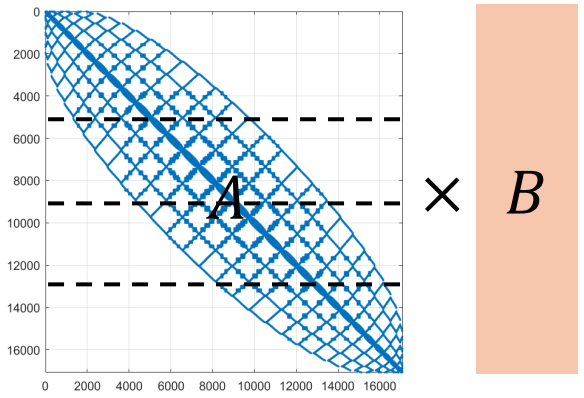




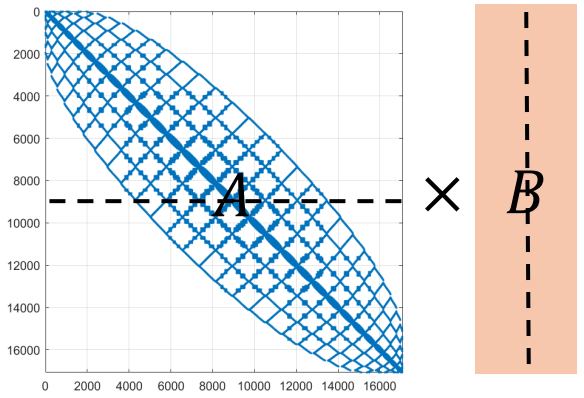
Design subspace 2: process grid size

$$p = p_m \times p_k \times p_n$$

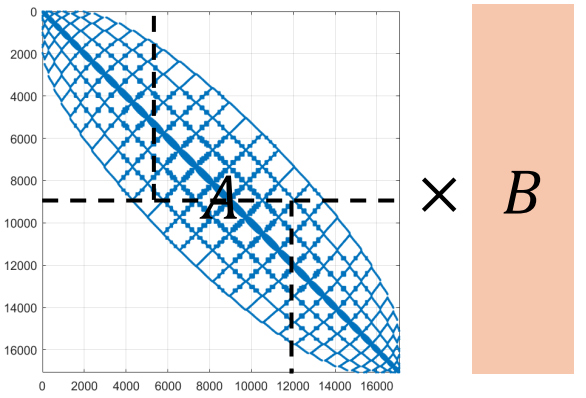
Large p with many prime factors?



$p_m = 4, p_n = p_k = 1$



$p_m = p_n = 2, p_k = 1$



$p_m = p_k = 2, p_n = 1$

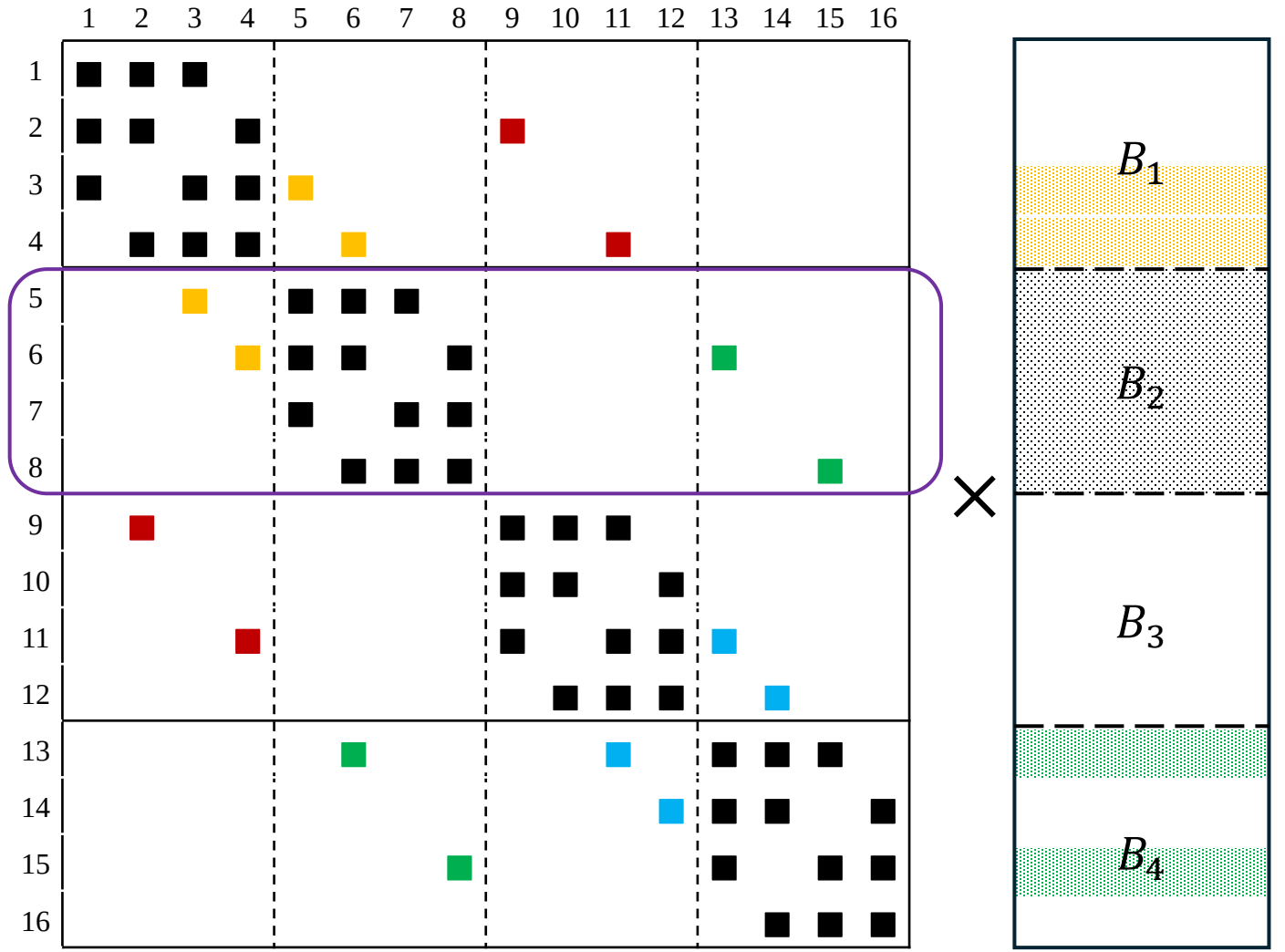
Data transfer in parallel SpMM

Notations

- $nnz(\cdot)$: number of non-zeros in a matrix block
- $|\cdot|$: size of a set
- I_i, J_i: row and column index sets
- $\boldsymbol{nec}(\cdot)$: set of non-empty column indices in a matrix block
- $\boldsymbol{ner}(\cdot)$: set of non-empty row indices in a matrix block

Data transfer in 1D parallelization

- 1D m-parallelization



Proc. 2 owns $\begin{cases} A_2 = A(5:8,:) \\ B_2 = B(5:8,:) \end{cases}$

$nec(A_2) = \{3, 4, 5, 6, 7, 8, 13, 15\}$

$nec(A_2) = ner(A(:, 5:8))$

Proc. 2 receives

$B(\{3, 4, 13, 15\}, :)$

for $C(5:8,:) = A_2 \times B$

Data transfer in 1D parallelization

- 1D m-para. total # matrix elem. to be transferred

$$S_m(p) = \left(\sum_{i=1}^{p} \left| nec\left(A(I_i, :)\right) \right| - k \right) n$$

$$nec(A_2) = \{3, 4, 5, 6, 7, 8, 13, 15\}$$

Proc. 2 receives $B(\{3, 4, 13, 15\}, :)$

Data transfer in 1D parallelization

- 1D k-para. total # matrix elem. to be transferred

$$S_k(p) = \left(\sum_{i=1}^{p} \left| ner\left(A(:, J_i)\right) \right| - m \right) n$$

- 1D n-para. total # matrix elem. to be transferred

$$S_n(p) = (p-1) \cdot nnz(A)$$

Data transfer in 2D parallelization

- 2D mn-parallelization

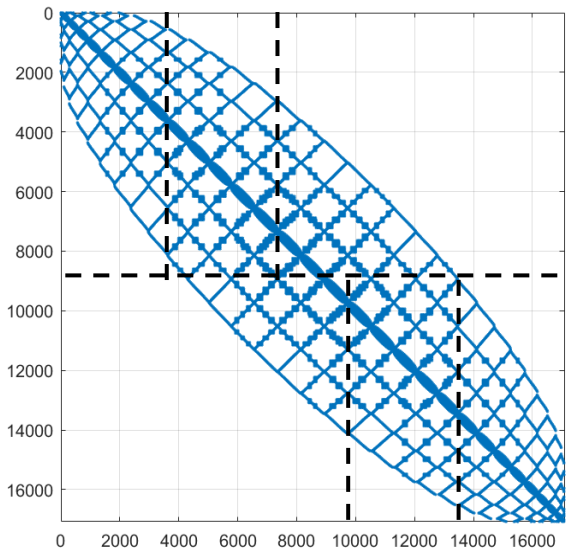
$$S_{mn}(p_m, p_n) = \left(\sum_{i=1}^{p_m} \left| nec\left(A(I_i, :)\right) \right| - k \right) n \; + \; (p_n - 1) \cdot nnz(A)$$

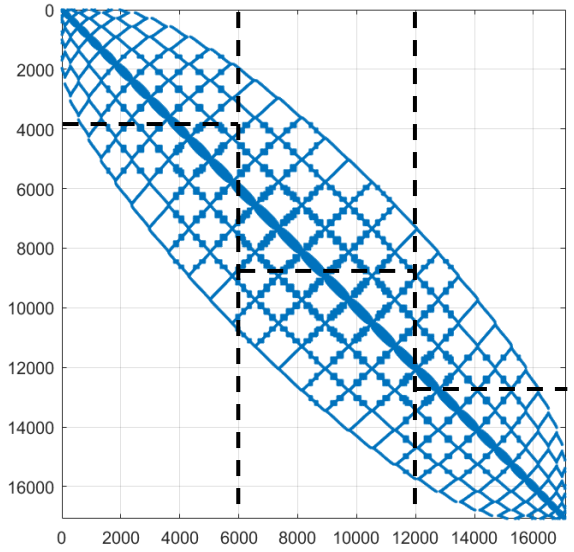
- 2D kn-parallelization

$$S_{kn}(p_k, p_n) = \left(\sum_{i=1}^{p_k} \left| ner\left(A(:, J_i)\right) \right| - m \right) n \; + \; (p_n - 1) \cdot nnz(A)$$

Data transfer in 2D/3D parallelization

- 2D mk-para. and 3D para.
  - Depends on the 2D partitioning of A
  - No simple formula for total communication size





CRP-SpMM

CRP-SpMM

- Communication Reduced Parallel SpMM
- Uses 2D mn-parallelization
  - The solution space of $p_m \times p_n = p$ is smaller
  - Finding balanced 1D partitioning is cheaper
  - 1D m-parallelization is commonly used

Communication cost model

$$S(p_m, p_n) = r(A) \cdot \left(\sum_{i=1}^{p_m} \left| nec\left(A(I_i, :)\right) \right| - k \right) n$$
$$+ f(A) \cdot (p_n - 1) \cdot nnz(A)$$

- Differs from $S_{mn}(p_m, p_n)$
  - $r(A)$: number of times A will be reused (e.g., in iterative solver)
  - $f(A)$: non-zero storage cost overhead (value / (value + indices))

2D process grid search

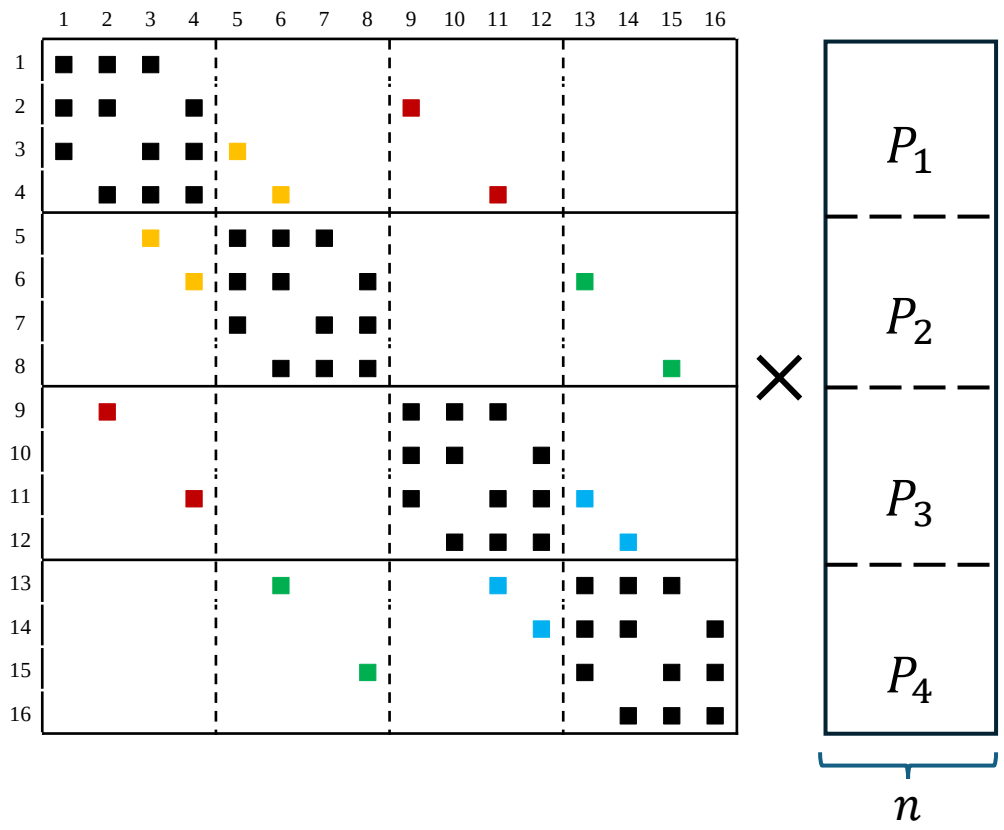
- Minimize the cost function:

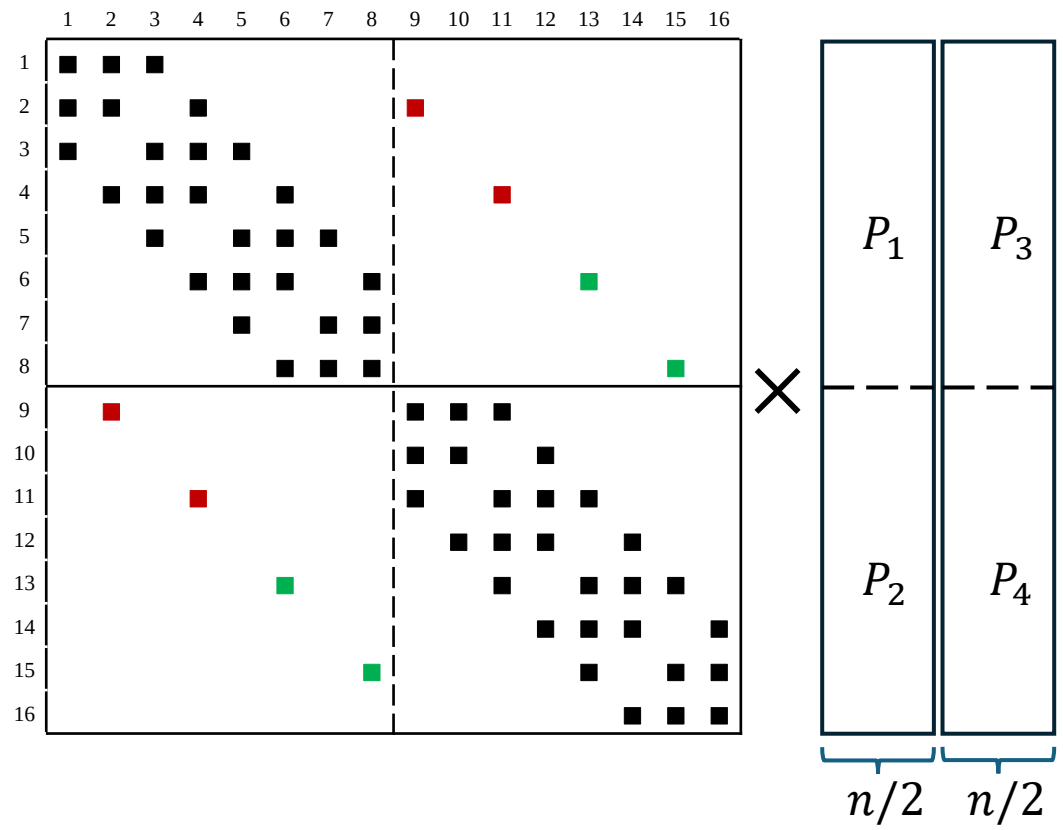
$$S(p_m, p_n) = r(A) \cdot \left(\sum_{i=1}^{p_m} \left| nec\left(A(I_i, :)\right) \right| - k \right) n + f(A) \cdot (p_n - 1) \cdot nnz(A)$$

Assumption: a smaller p_m → a smaller number B rows to copy

2D process grid search

Assumption: a smaller p_m → a smaller number B rows to copy





2D process grid search algorithm

1. Baseline: 1D p-way row partitioning ($p_m = p, p_n = 1$)
2. Fast 2D grid search: multiple p_n with prime factors of p
3. Calculate a new p_m-way row partitioning using the baseline
4. If the new parallelization is better, save it

CRP-SpMM Computation

1. Replicate A once between p_n groups of processes
2. Each p_m-process group performs 1D m-parallel SpMM
   (a) Pack B matrix rows for exchange
   (b) Send and receive B matrix rows
   (c) Unpack received B matrix rows
   (d) Local SpMM

Numerical results

Experiment settings

- Georgia Tech PACE-Phoenix cluster
  - 2 * Intel Xeon Gold 6226 12-core, 192 GB DDR4, 100 Gbps IB
  - Intel C/C++ compiler v19.1, Intel MKL v19.1
- $r(A) = 1$
- $f(A) = 1.5$ (CSR format)

Communication sizes

- Comm. size of 1D m-para. and CRP-SpMM w/ METIS

Matrix Name	METIS Time (s)	Alg. 1 Time (s)	m-para. Comm. Size ($\times 10^6$)	mn-para. $p_m \times p_n$	mn-para. Comm. Size ($\times 10^6$)
PFlow_742	8.08	0.01	359	128×2	315
Serena	5.78	0.01	666	128×2	625
Geo_1438	5.08	0.02	674	128×2	630
StocF-1465	4.14	0.01	442	64×4	361
Long_Coup_dt6	7.10	0.02	836	128×2	802
Hook_1498	5.13	0.01	620	128×2	557
Flan_1565	6.90	0.02	604	128×2	603
Bump_2911	13.22	0.04	1160	128×2	1136

256 processes ($p = 256$), B has 1024 columns ($n = 1024$)

Communication sizes

- Comm. size of 1D m-para. and CRP-SpMM

Matrix Name	m-para. Size ($\times 10^6$)	mn-para. Comm. $p_m \times p_n$	mn-para. Comm Size ($\times 10^6$)
com-Orkut	26938	16×16	12059
Amazon	30204	16×16	16514
reddit	8583	32×8	2823
cage15	7281	32×8	5170
kmer_V2a	28765	32×8	28646
delaunay_n23	2518	256×1	2518
wb-edu	184	256×1	184
nlpkkt160	5530	64×4	3999
Hardesty3	2201	16×16	1919
ss	1647	64×4	883
circuit5M	5599	256×1	5599

256 processes ($p = 256$), B has 256 columns ($n = 256$)

Parallel runtime

- 128 nodes, 2 MPI ranks per node, 12 cores per MPI rank
- CB: CombBLAS (GEMM-based alg.), always use $p_m = p_n = 16$

Matrix Name	CB-best Runtime (s)	1D m-para. Runtime (s)	CRP-SpMM $p_m \times p_n$	CRP-SpMM Runtime (s)
PFlow_742	0.79	0.01	256×1	0.01
Serena	1.07	0.01		0.01
Geo_1438	1.22	0.01		0.01
StocF-1465	0.55	0.01		0.01
Long_Coup_dt6	1.59	0.02		0.02
Hook_1498	1.19	0.01		0.01
Flan_1565	2.20	0.01		0.01
Bump_2911	2.46	0.01		0.01
com-Orkut	2.31	0.75	16×16	0.31
Amazon	2.85	0.80	16×16	0.43
reddit	0.59	0.23	32×8	0.07
cage15	2.26	0.10	32×8	0.08
kmer_V2a	5.99	0.87	32×8	0.92
delaunay_n23	1.97	0.20	256×1	0.20
wb-edu	2.25	0.03	256×1	0.03
nlpkkt160	4.65	0.25	64×4	0.16
Hardesty3	1.40	0.03	16×16	0.03
ss	0.60	0.06	64×4	0.04
circuit5M	4.43	(N/A)	256×1	(N/A)

Future work

- Better algorithm for fast row partitioning in 2D grid search
- Extending for generalized 3D parallelization
- Performance improvement with low-level optimizations

Take-aways

- Parallelizing SpMM has a vast design space
- CRP-SpMM
  - Fast search of 2D mn-para. scheme for a lower comm. cost
  - Reuse existing algorithms and codes from SpMV
- Source code: https://github.com/scalable-matrix/CRP-SpMM

Acknowledgement:

- U.S. Department of Energy, grant DE-SC0019410
- Partnership for an Advanced Computing Environment (PACE) @ Georgia Tech

Thank you for your attention.
Questions?