ICPP 2022

51st International Conference on Parallel Processing

August 29 - September 1, 2022.

TileSpMSpV: A Tiled Algorithm for Sparse Matrix-Sparse Vector Multiplication on GPUs

Haonan Ji[1], Huimin Song[1], Shibo Lu[2], Zhou Jin[1], Guangming Tan[3] and Weifeng Liu[1]

1. Super Scientific Software Laboratory, China University of Petroleum-Beijing
2. Northeastern University
3. State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences

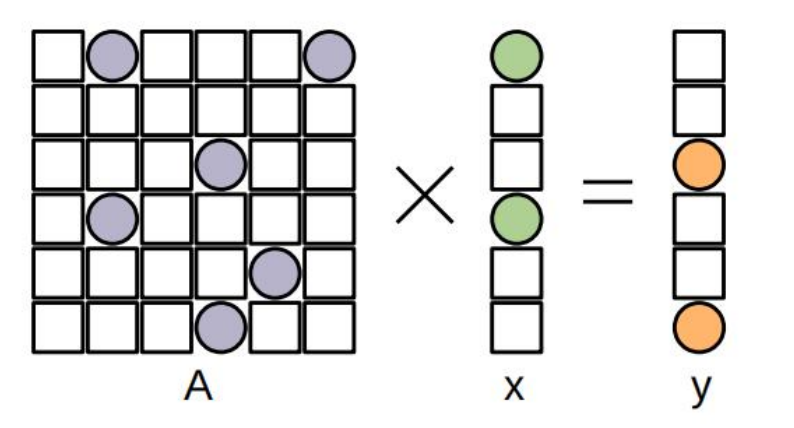
OUTLINE

Part I

Introduction

Sparse Matrix-Sparse Vector Multiplication (SpMSpV)

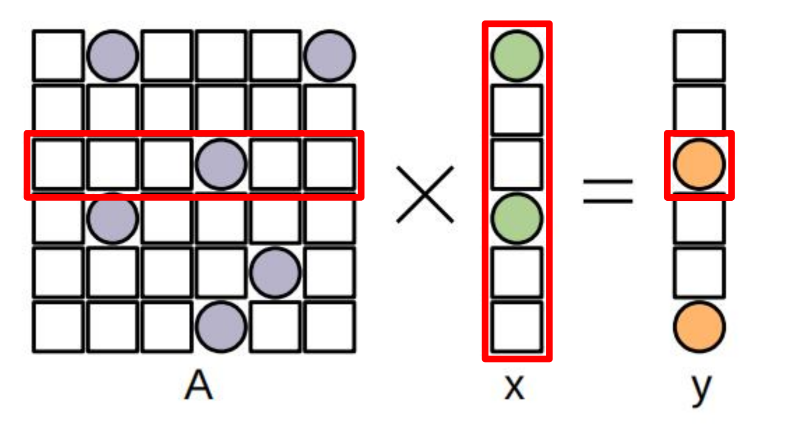
Sparse Matrix-Sparse Vector Multiplication (SpMSpV) operation multiplies a sparse matrix *A* with a sparse vector *x* and obtains a resulting sparse vector *y*.



SpMSpV

Row-wise SpMSpV

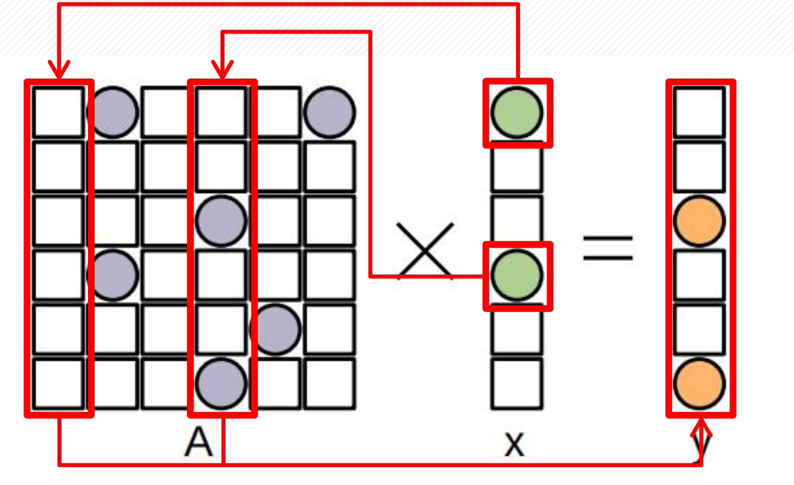
Each element of the resulting vector *y* is obtained by computing the dot product of the corresponding row of matrix *A* with the vector *x*.



Row-wise SpMSpV

Column-wise SpMSpV

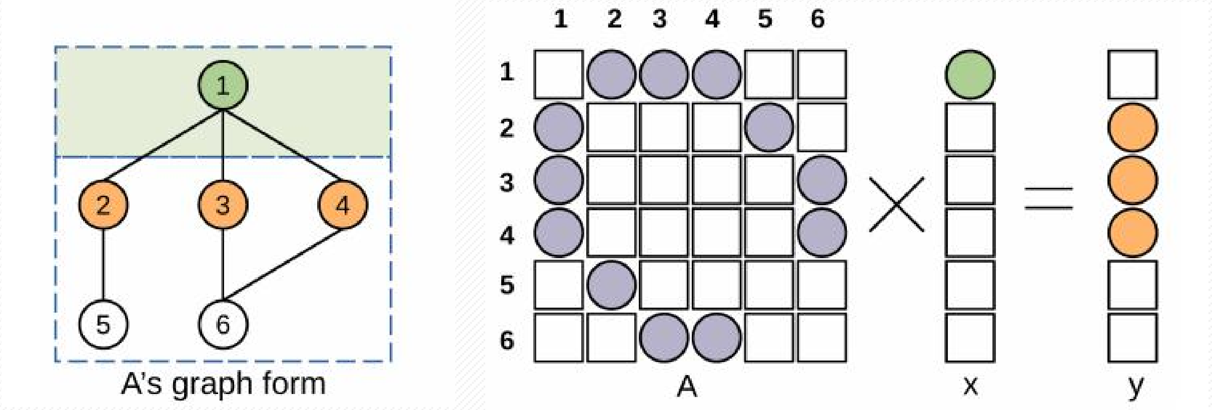
Each nonzero in *x* finds the corresponding column in the matrix, scales the nonzeros in the column, and merges the results into *y*.



Column-wise SpMSpV

BFS

BFS is one of the most basic traversal algorithms in graph computations. The algorithm starts from a source vertex in the graph and accesses all reachable vertices through multi-layer traversal.



An example of running the first iteration of BFS on the graph (left) by using SpMSpV (right).

Motivation 1

- Existing work ignored exploiting local sparsity in the input sparse matrix and vector largely.

Carl Yang, Yangzihao Wang and John Owens. “Fast Sparse Matrix and Sparse Vector Multiplication Algorithm on the GPU”. In IPDPSW '15, 2015, pp. 841-847.

Ariful Azad and Aydin Buluç. “A Work-Efficient Parallel Sparse Matrix Sparse Vector Multiplication Algorithm”. In IPDPS ’17, 2017, pp. 688-697.

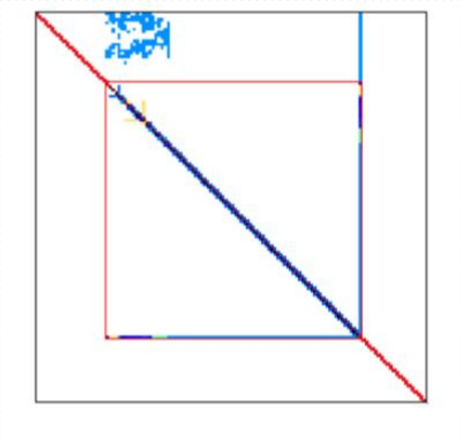
Leonid Yavits and Ran Ginosar. “Accelerator for Sparse Machine Learning”. In IEEE Computer Architecture Letters, 2018, pp. 21-24.

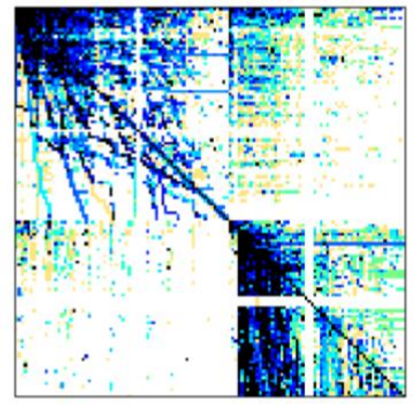
Min Li, Yulong Ao, and Chao Yang. “Adaptive SpMV/SpMSpV on GPUs for Input Vectors of Varied Sparsity”. In IEEE Transactions on Parallel and Distributed Systems, 2021, pp. 1842–1853.

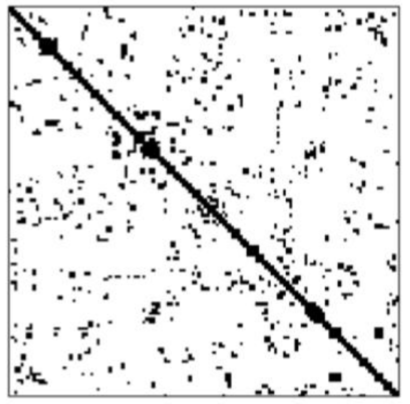
Paul Burkhardt. “Optimal Algebraic Breadth-First Search for Sparse Graphs”. In ACM Transactions on Knowledge Discovery from Data, 2021, pp. 1-19.

Motivation 2

- No one matrix storage formulation works for any sparsity structure.





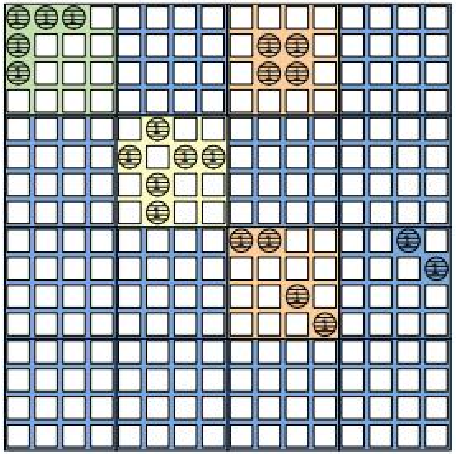


Part II

TileSpMSpV

Storage structure of sparse matrix

TileSpMSpV divides the input sparse matrix into several sparse matrix tiles of size *nt*-by-*nt*, where *nt* can be 16, 32 or 64.

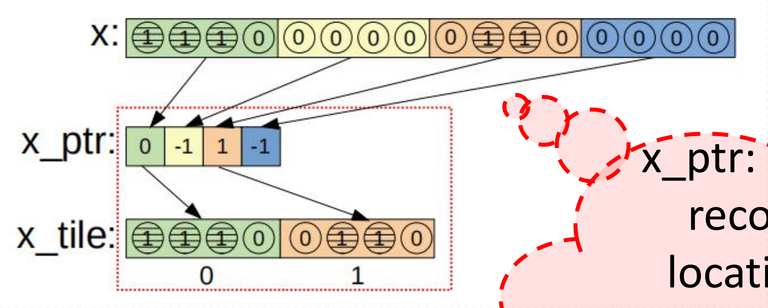


Using CSR, COO, ELL, HYB, dense, dense row and dense column to store non-empty tiles.

Yuyao Niu, Zhengyang Lu, Meichen Dong, Zhou Jin, Weifeng Liu, and Guangming Tan. "TileSpMV: A Tiled Algorithm for Sparse Matrix-Vector Multiplication on GPUs". IPDPS '21, 2021.

Storage structure of sparse vector

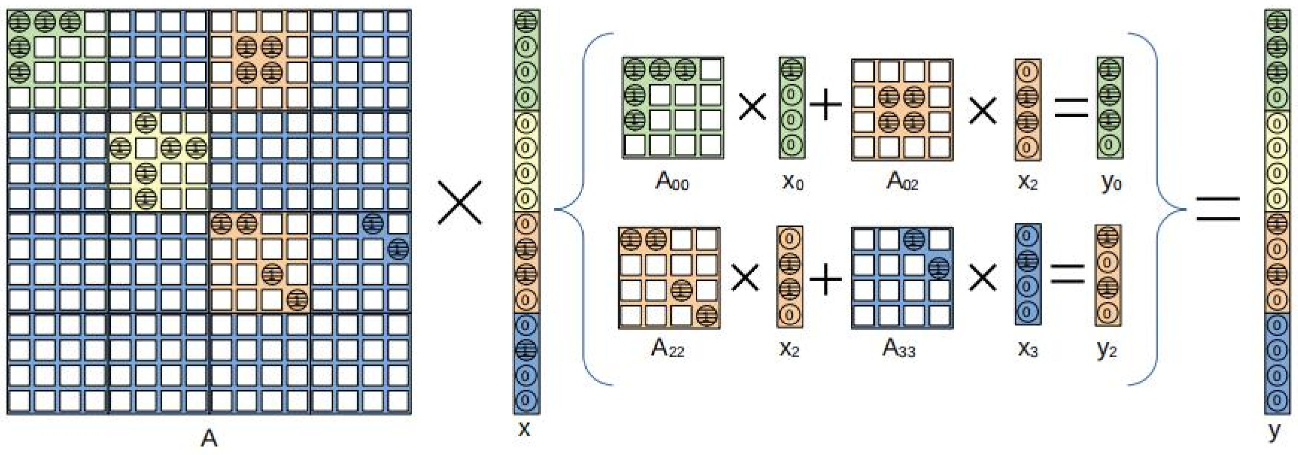
TileSpMSpV divides the input sparse vector into several sparse matrix tiles of size *nt*-by-1, then uses index and value arrays to mark the non-empty tiles information, and realizes the access of O(1) time complexity.



x_ptr: An index array that records the type and location of vector tiles.
x_tile: A value array that stores the elements of non-empty vector tiles.

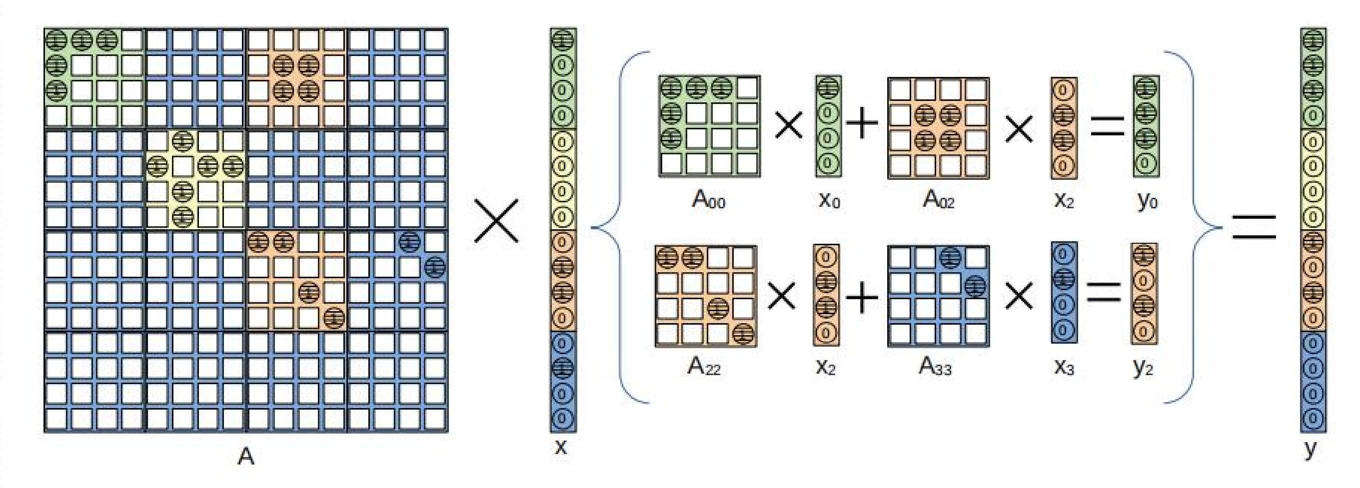
TileSpMSpV algorithm

- Load the corresponding matrix tile into the GPU shared memory.



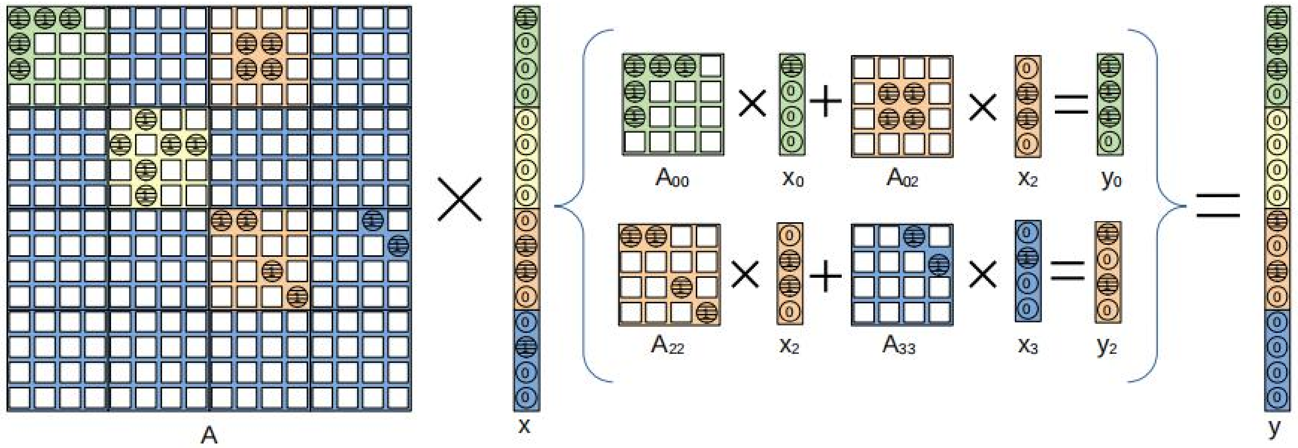
TileSpMSpV algorithm

- Load the corresponding matrix tile into the GPU shared memory.
- Find the actual storage position of the corresponding vector tile called *x_tile_id*. If *x_tile_id*=-1, skip the calculation, otherwise, obtain the vector tile information.



TileSpMSpV algorithm

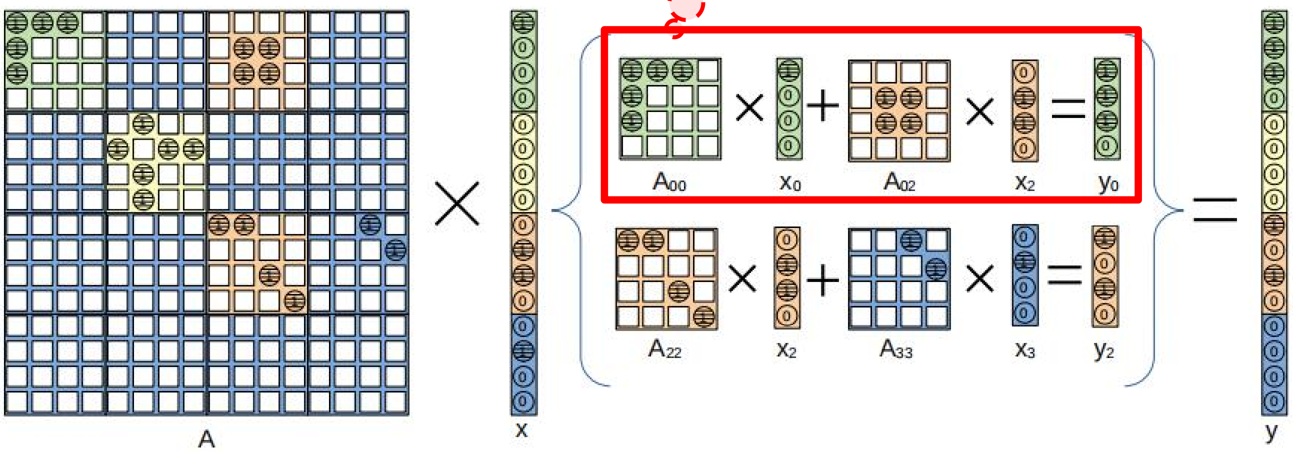
- Load the corresponding matrix tile into the GPU shared memory.
- Find the actual storage position of the corresponding vector tile called *x_tile_id*. If *x_tile_id*=-1, skip the calculation, otherwise, obtain the vector tile information.
- Different kernel calculations are selected according to different formats of matrix tiles.



TileSpMSpV: vector tile is not empty

- Load the corresponding matrix tile into the GPU shared memory.
- Find the actual storage position of the corresponding vector tile called x_tile_id. If x_tile_id=-1, skip the calculation, otherwise, obtain the vector tile information.
- Different kernel calculations are selected according to different formats of matrix tiles.

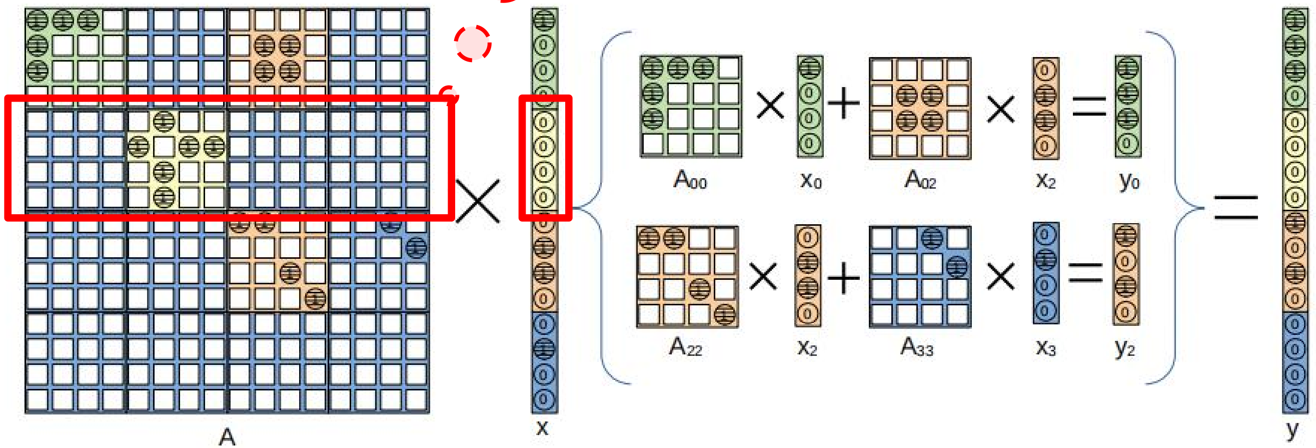
Multiply the two non-empty tiles A_{00} and A_{02} of the first row tile by the vector tiles x_0 and x_2, and add the resulting vectors to obtain y_0, that is, the first tile of y.



TileSpMSpV: vector tile is empty

- Load the corresponding matrix tile into the GPU shared memory.
- Find the actual storage position of the corresponding vector tile called x_tile_id. If x_tile_id==-1, skip the calculation, otherwise, obtain the vector tile information.
- Different kernel calculations are selected according to different formats of matrix tiles.

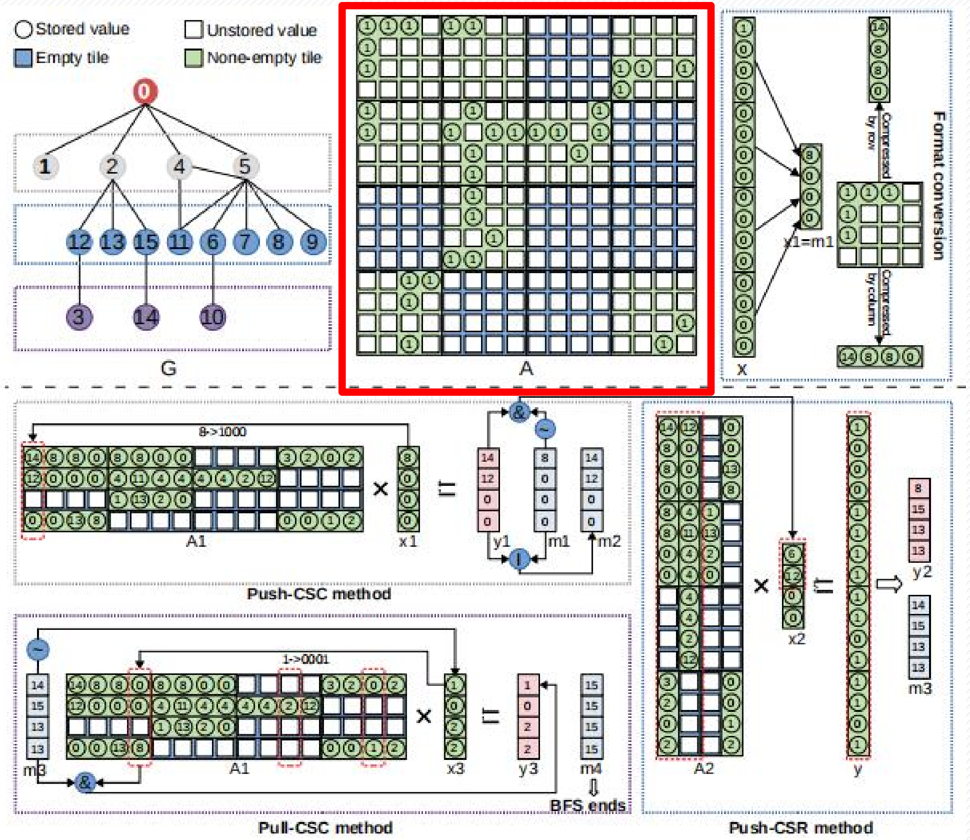
The non-empty tile A_{11} of the second row tile is matched to the second vector tile x_1. According to its index, it is judged that x_1 is a zero tile, so there is no need to calculate.



Part III

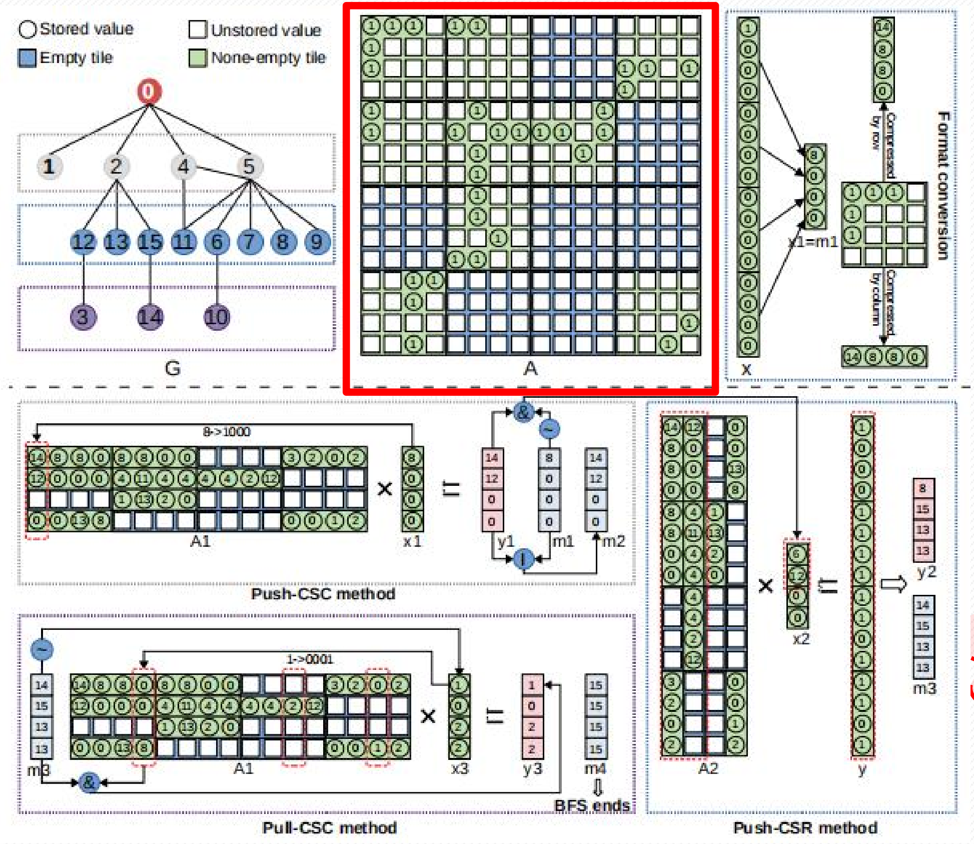
TileBFS

Auxiliary data structure for TileBFS



The non-empty tiles use a binary bitmask to record whether the elements in a tile are zero.

Auxiliary data structure for TileBFS

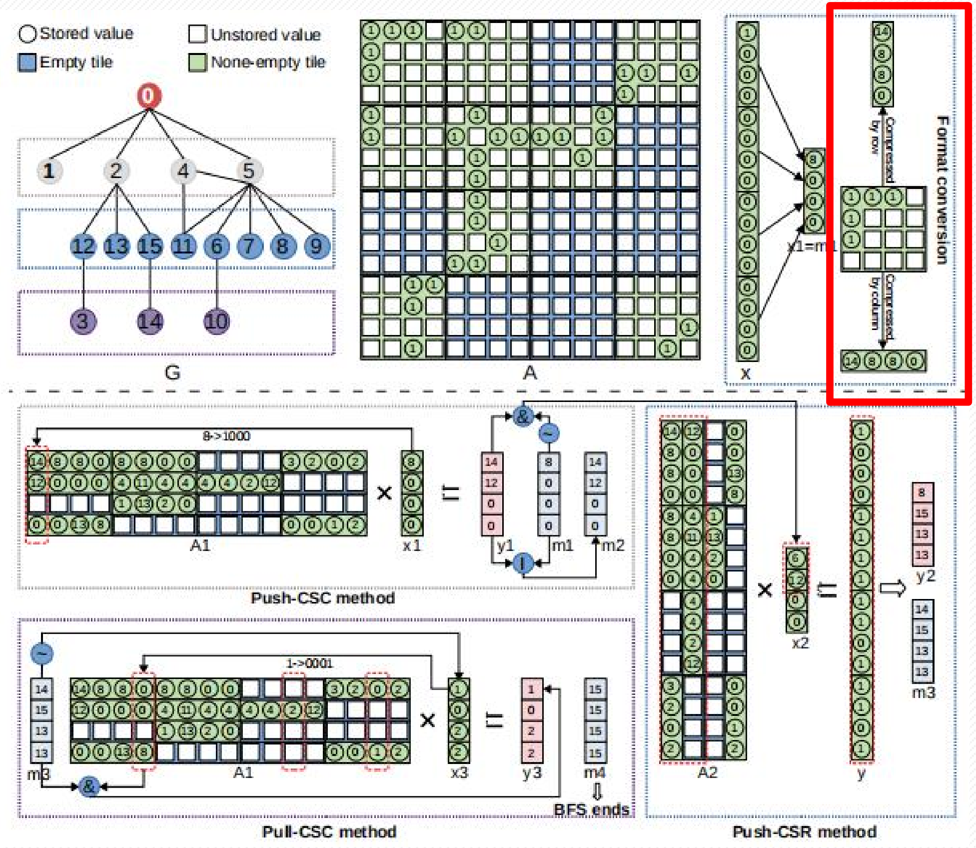


The non-empty tiles use a binary bitmask to record whether the elements in a tile are zero.

Adaptive selection of tile size (*nt* = 16, 32, or 64).

16: unsigned char
32: unsigned int
64: unsigned long long

Auxiliary data structure for TileBFS



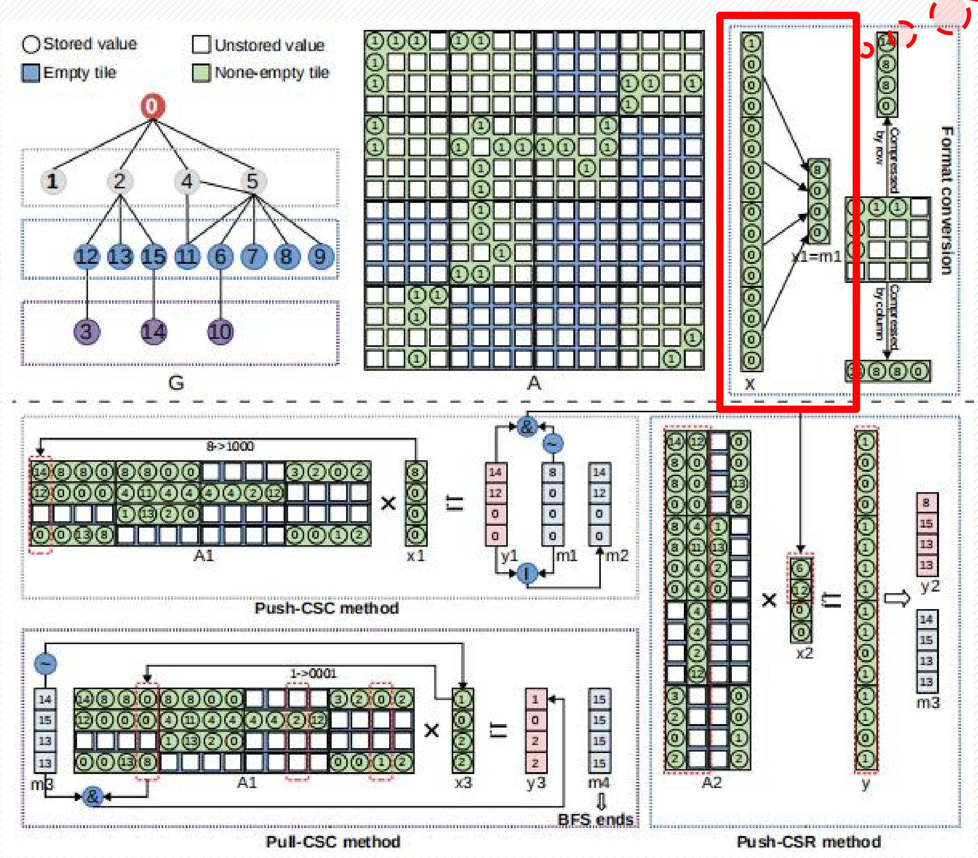
The non-empty tiles use a binary bitmask to record whether the elements in a tile are zero.

Adaptive selection of tile size (*nt* = 16, 32, or 64).

In tile format of row-wise and column-wise SpMSpV.

Auxiliary data structure for TileBFS

The vector *m* marks whether the vertex has been visited.



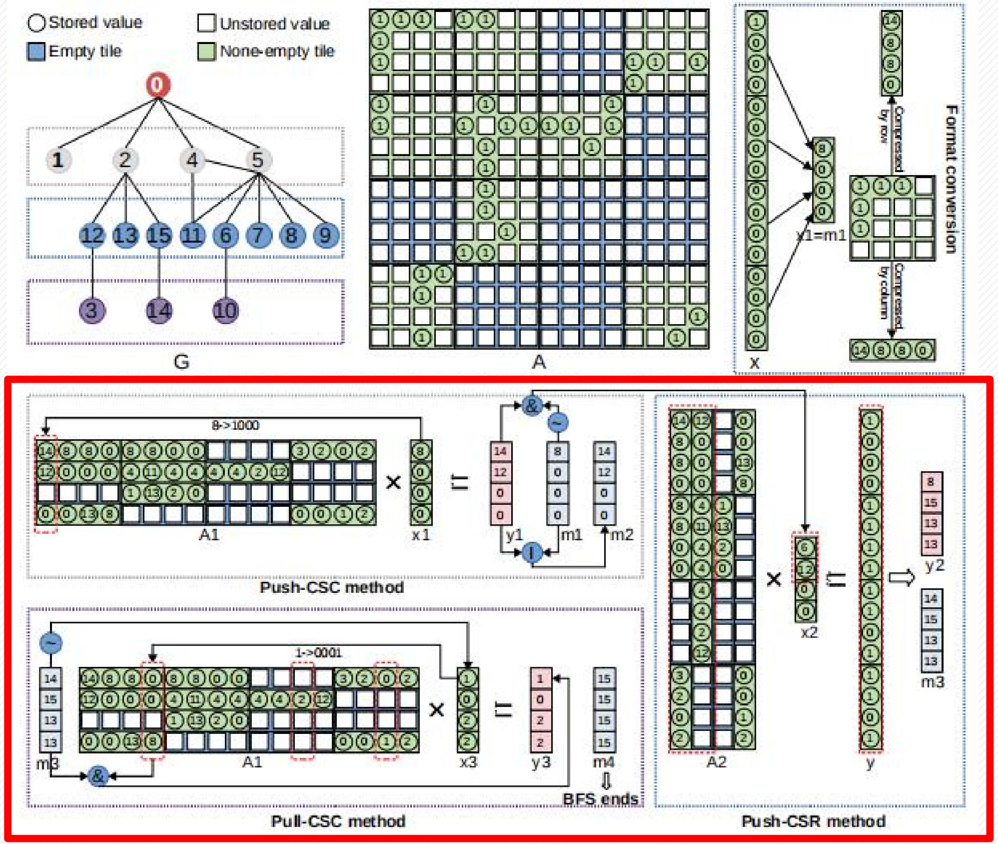
The non-empty tiles use a binary bitmask to record whether the elements in a tile are zero.

Adaptive selection of tile size (*nt* = 16, 32, or 64).

In tile format of row-wise and column-wise SpMSpV.

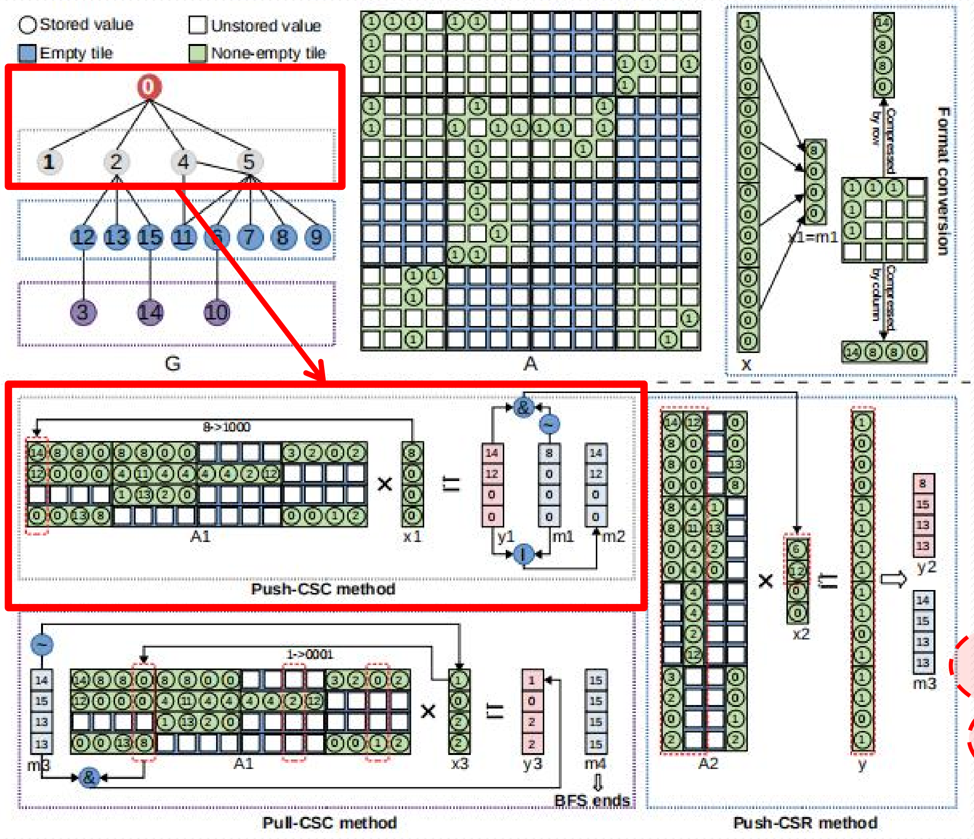
Vectors are stored as dense tiled vectors.

Direction optimization of BFS



- A single operation method is difficult to maintain high performance when dealing with vectors with different sparsity.
- TileBFS designs three types of SpMSpV methods: Push-CSC, Push-CSR and Pull-CSC.

Direction optimization of BFS——Push-CSC (First layer)

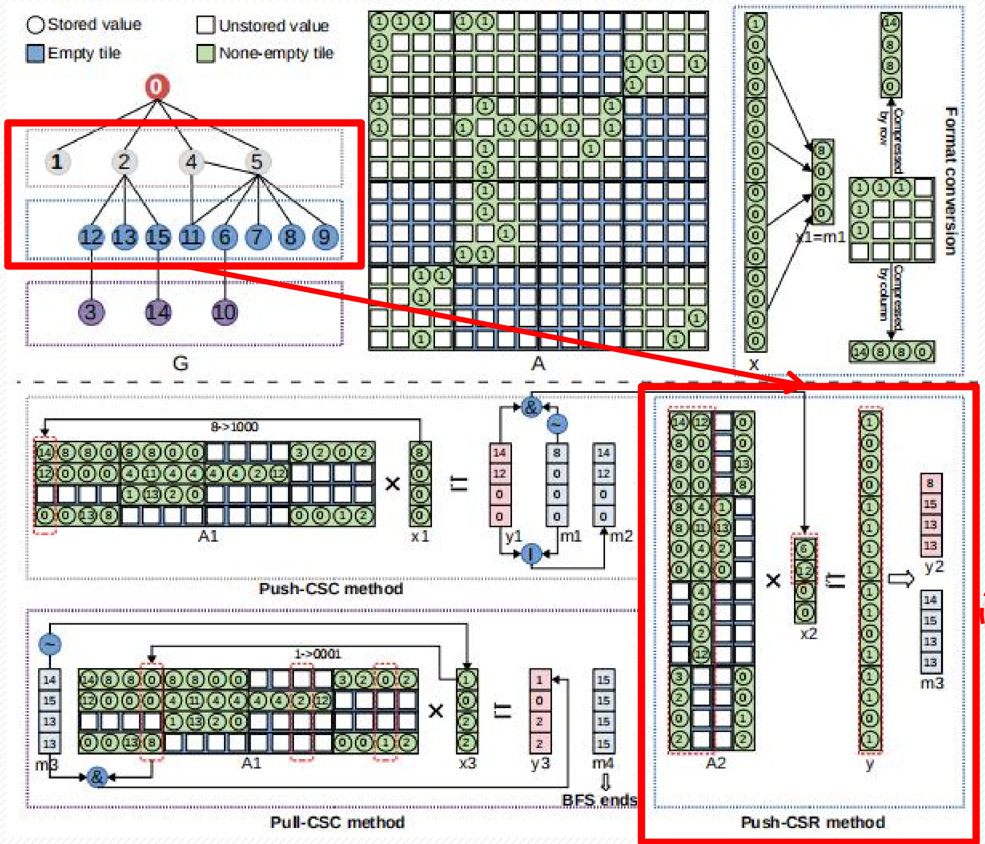


According to the non-empty element position of the input vector, several corresponding matrix columns are found, and then the corresponding matrix columns are merged into the result vector.

When the sparsity of the *x* is less than 0.01 and the number of unvisited vertices is large, we will use Push-CSC.

TileBFS

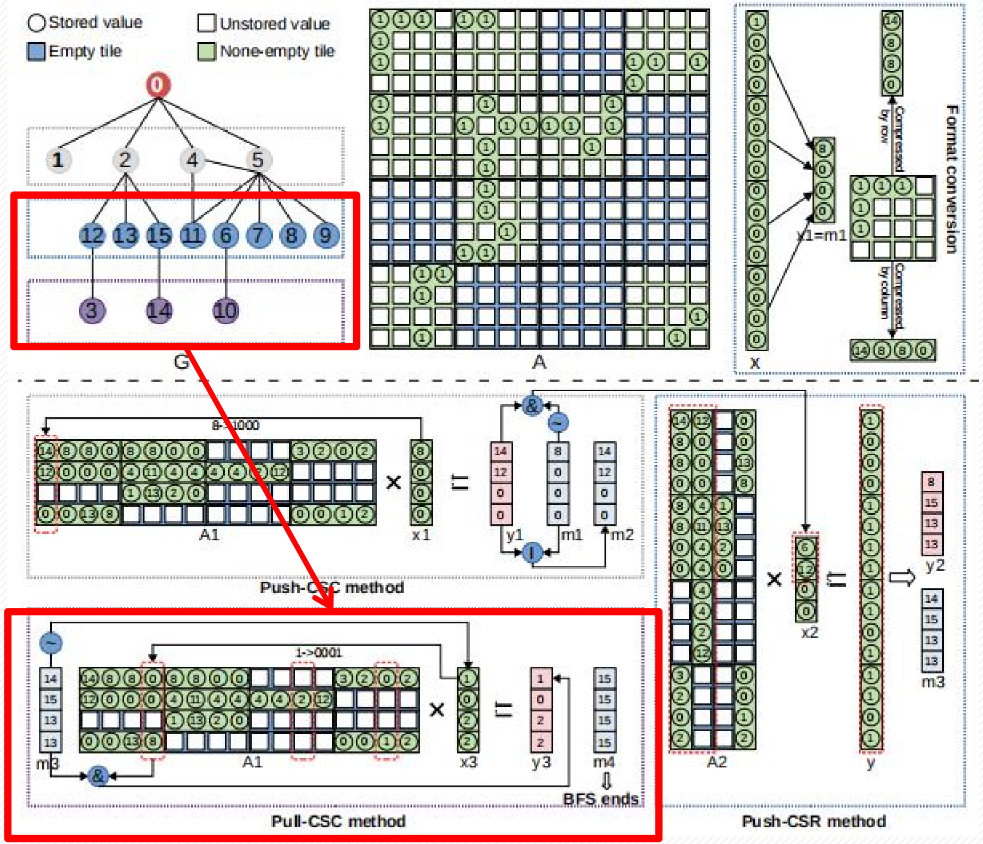
Direction optimization of BFS——Push-CSR (Second layer)



By multiplying each row tile of the matrix by the corresponding vector tile, the resulting vector tile is obtained.

When the sparsity of the input vector x is greater than or equal to 0.01 and the number of unvisited vertices is large, we will use Push-CSR.

Direction optimization of BFS——Pull-CSC (Third layer)



- Calculate the input vector.
- Find the corresponding matrix columns.
- AND the selected matrix columns with the mask vector.
- Update mask vector according to AND result.

When the number of unvisited vertices is small, we will use Pull-CSC.

Part IV

Performance Evaluation

Performance Evaluation

Experimental setup

	Algorithm	Machine specification
SpMSpV	(1) TileSpMV [40]	(1) NVIDIA Geforce RTX 3060 (Ampere), 3,584 CUDA cores @ 1.78 GHz, 12 GB GDDR6, B/W 360.0 GB/s, (2) NVIDIA Geforce RTX 3090 (Ampere), 10,496 CUDA cores @ 1.70 GHz, 24 GB GDDR6X, B/W 936.2 GB/s.
	(2) cuSPARSE v11.4 BSR	
	(3) CombBLAS [3]	
	(4) TileSpMSpV (this work)	
BFS	(1) Gunrock [48]	
	(2) GSwitch [37]	
	(3) TileBFS (this work)	

Yuyao Niu, Zhengyang Lu, Meichen Dong, Zhou Jin, Weifeng Liu, and Guangming Tan. “TileSpMV: A Tiled Algorithm for Sparse Matrix-Vector Multiplication on GPUs”. IPDPS ’21, 2021.

Ariful Azad and Aydin Buluç. “A Work-Efficient Parallel Sparse Matrix Sparse Vector Multiplication Algorithm”. In IPDPS ’17, 2017.

Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D. Owens. “Gunrock: A High-Performance Graph Processing Library on the GPU”. In PPoPP ’16, 2016.

Ke Meng, Jiajia Li, Guangming Tan, and Ninghui Sun. “A Pattern Based Algorithmic Autotuner for Graph Processing on GPUs.” In PPoPP ’19, 2019.

Information of the representative matrices

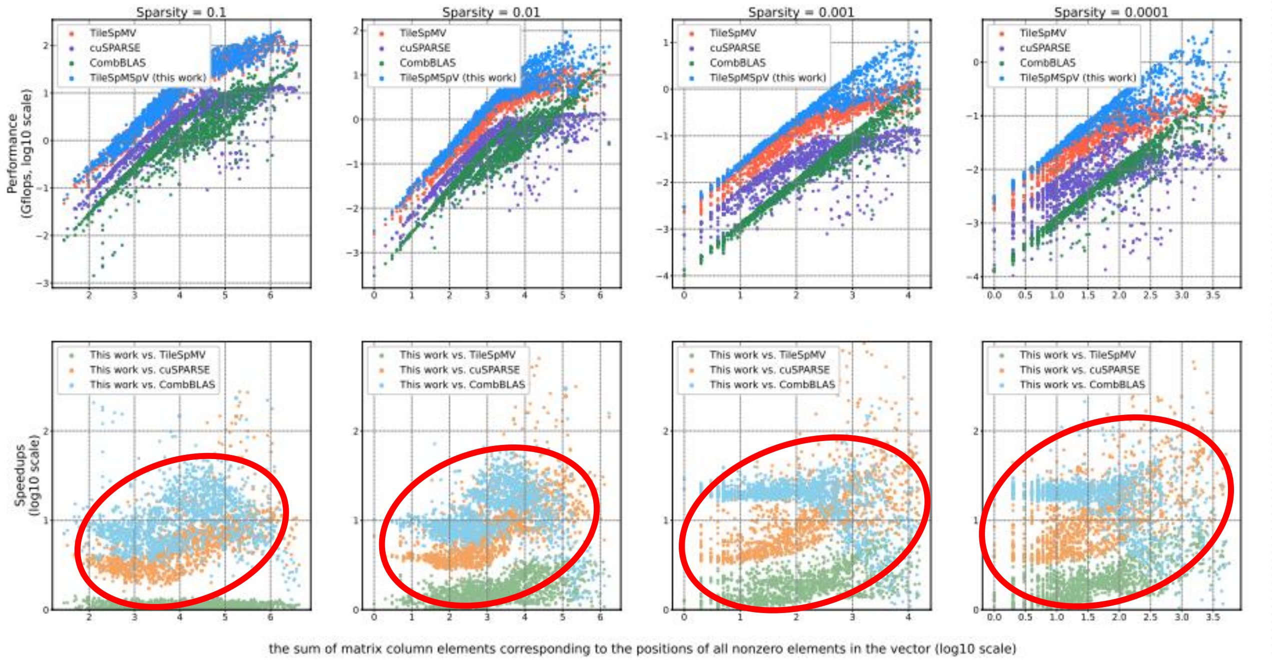
Matrix	Size	#nonzeros	#tiles (16*16)	#tiles (32*32)	#tiles (64*64)
af_5_k101	503K x 503K	17M	257K	110K	55K
cant	62K x 62K	4M	62K	20K	8K
cavity23	4K x 4K	144K	2K	1K	1K
pdb1HYS	36K x 36K	4M	50K	19K	8K
fullb	199K x 199K	11M	31K	112K	220K
ldoor	952K x 952K	46M	998K	574K	380K
in-2004	1M x 1M	27M	1M	641K	363K
msdoor	415K x 415K	20M	484K	288K	191K
roadNet-TX	1M x 1M	3M	1M	740K	464K
ML_Geer	1M x 1M	110M	1M	694K	332K
333SP	3M x 3M	22M	8M	7M	7M
dielFilterV2clx	607K x 607K	25M	2M	1M	481K

Dataset

The dataset of SpMSpV contains all 2757 sparse matrices from the SuiteSparse Matrix Collection. Inside the dataset, 2081 sparse matrices are square and are used for testing BFS.

Performance Evaluation

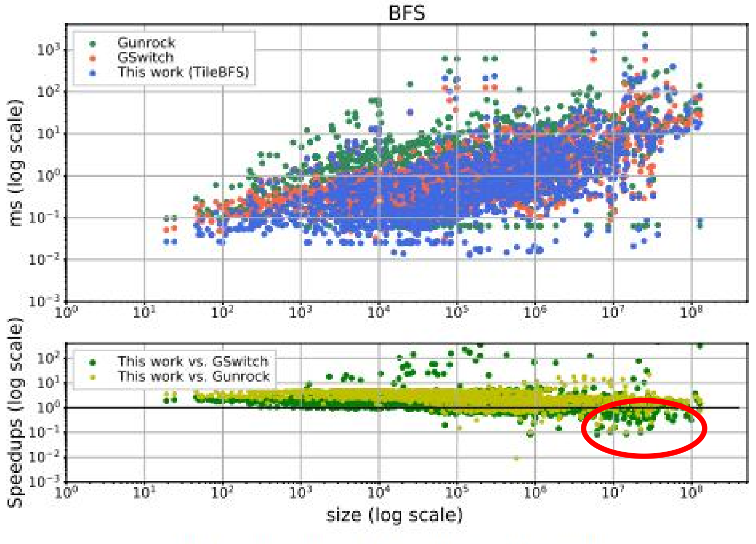
SpMSpV performance comparison of four methods with different sparsity



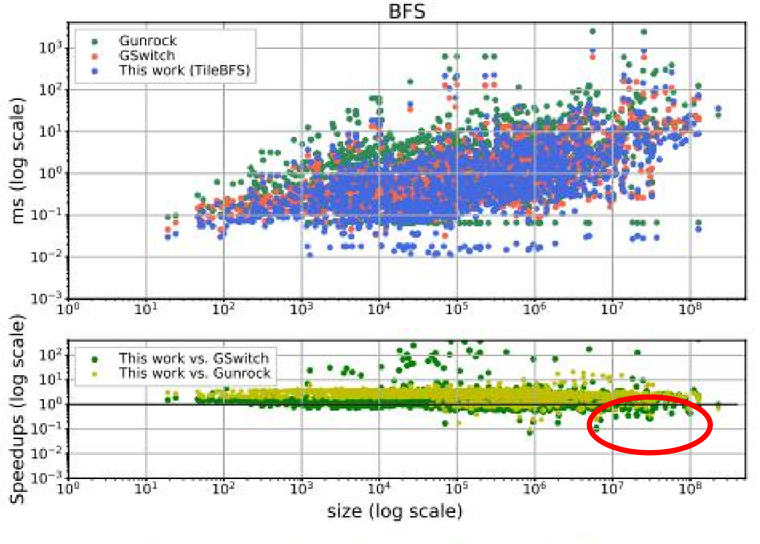
TileSpMSpV achieves speedups of on average 1.83x (up to 7.68x) over TileSpMV, 17.18 (up to 1050.02x) over cuSPARSE and 17.20x (up to 235.90x) over CombBLAS at vector sparsity of 0.1, 0.01, 0.001 and 0.0001 on RTX 3090.

Performance Evaluation

BFS performance comparison



(a) BFS performance and speedups on RTX 3060

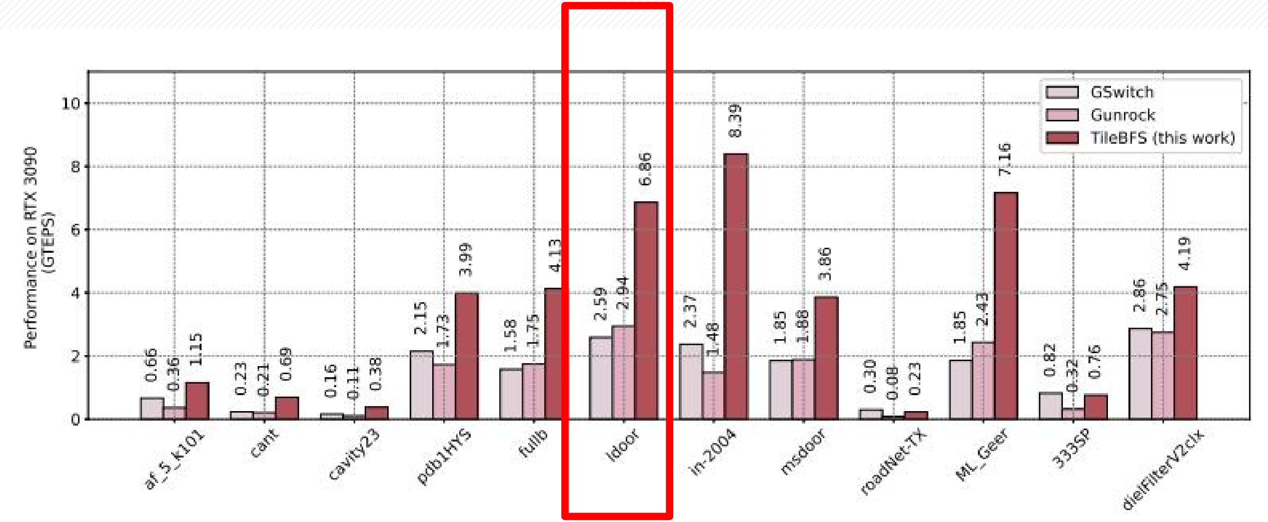


(b) BFS performance and speedups on RTX 3090

On RTX 3060, the average speedups over Gunrock and GSwitch are 3.03x and 4.35x, the best speedups are 21.70x and 837.36x, respectively.

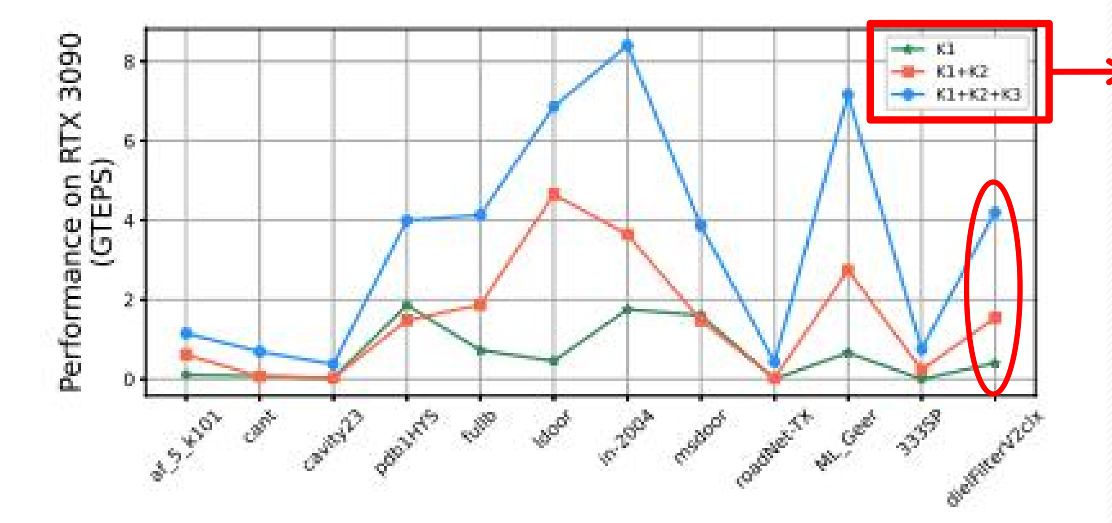
On RTX 3090, the average speedups over Gunrock and GSwitch are 2.74x and 20.01x, the best speedups are 4.69x and 1164.35x, respectively.

Comparison over Gunrock and GSwitch



Performance comparison of 12 representative matrices on RTX 3090 GPU.

Directional optimization analysis

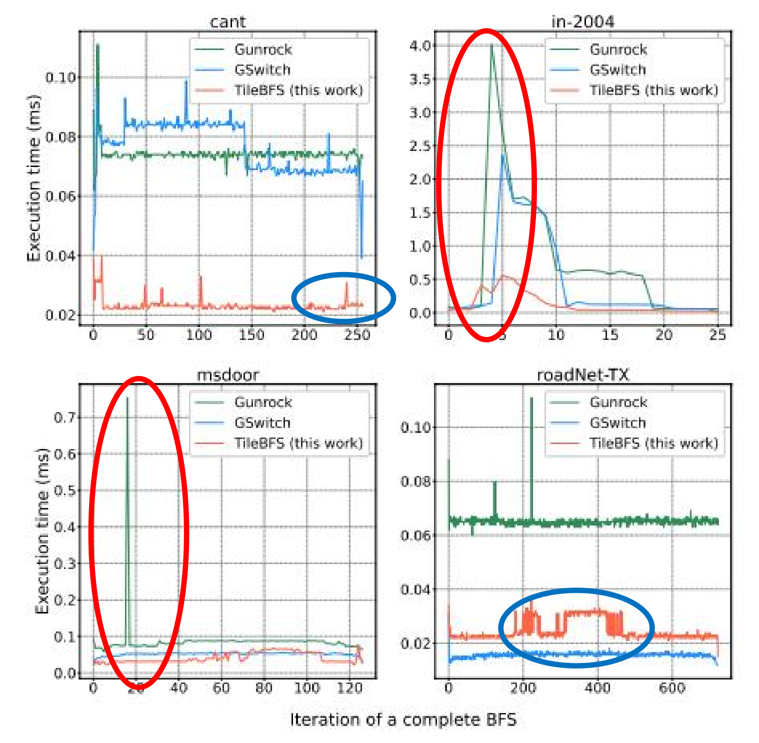


Push-CSC (K1)
Push-CSR (K2)
Pull-CSC (K3)

Comparison of BFS performance using three direction optimization step by step of the representative matrices. The performance improvement of kernel conversion is significant.

Performance Evaluation

Iteration time analysis

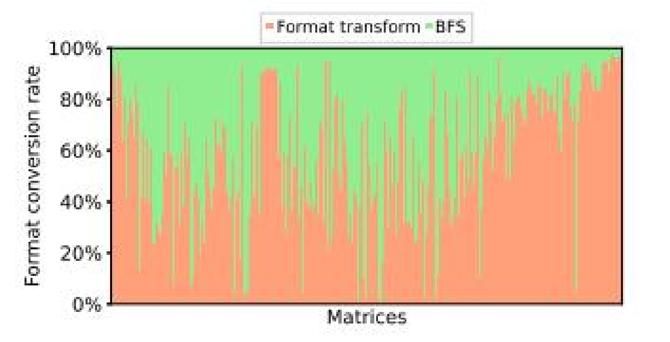


On RTX 3090, the iteration time comparison of Gunrock, GSwitch and TileBFS.

- Suppress peak execution time(see the matrices ‘in-2004’ and ‘msdoor’)
- Invalid kernel switch(see the matrices ‘msdoor’ and ‘cant’)

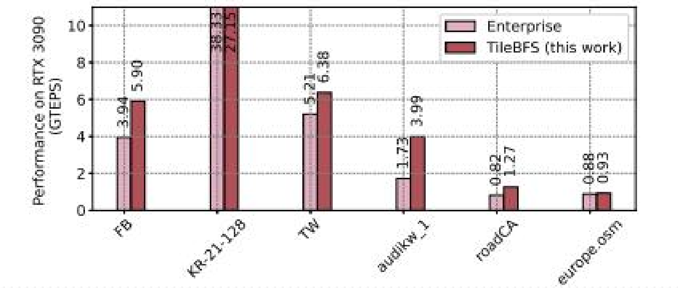
Performance Evaluation

Format conversion overhead



Comparison of preprocessing time and a BFS time of the all matrices on RTX 3090. The time for format conversion does not exceed a single BFS processing time in most cases.

Comparison over Enterprise



Performance comparison of 6 representative matrices on RTX 3090 GPU. The average speedup is 1.39x, and the maximum speedup is 2.31x.

Hang Liu and H. Howie Huang. “Enterprise: Breadth-First Graph Traversal on GPUs”. In SC ’15, 2015.

Part V

Conclusion

Conclusion

We develop tiled storage structures for the sparse matrix and vectors involved in SpMSpV.

Conclusion

We develop tiled storage structures for the sparse matrix and vectors involved in SpMSpV.

We design a tiled sparse algorithm called TileSpMSpV and a directional optimization BFS algorithm called TileBFS.

Conclusion

We develop tiled storage structures for the sparse matrix and vectors involved in SpMSpV.

We design a tiled sparse algorithm called TileSpMSpV and a directional optimization BFS algorithm called TileBFS.

We evaluate our algorithms on latest NVIDIA GPU and significantly outperform existing work.

ICPP 2022

Thanks for your time!

TileSpMSpV: A Tiled Algorithm for Sparse Matrix-Sparse Vector Multiplication on GPUs

Haonan Ji[1], Huimin Song[1], Shibo Lu[2], Zhou Jin[1], Guangming Tan[3] and Weifeng Liu[1]

1. Super Scientific Software Laboratory, China University of Petroleum-Beijing
2. Northeastern University
3. State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences