GSAP: A GPU-Accelerated Stochastic Graph Partitioner

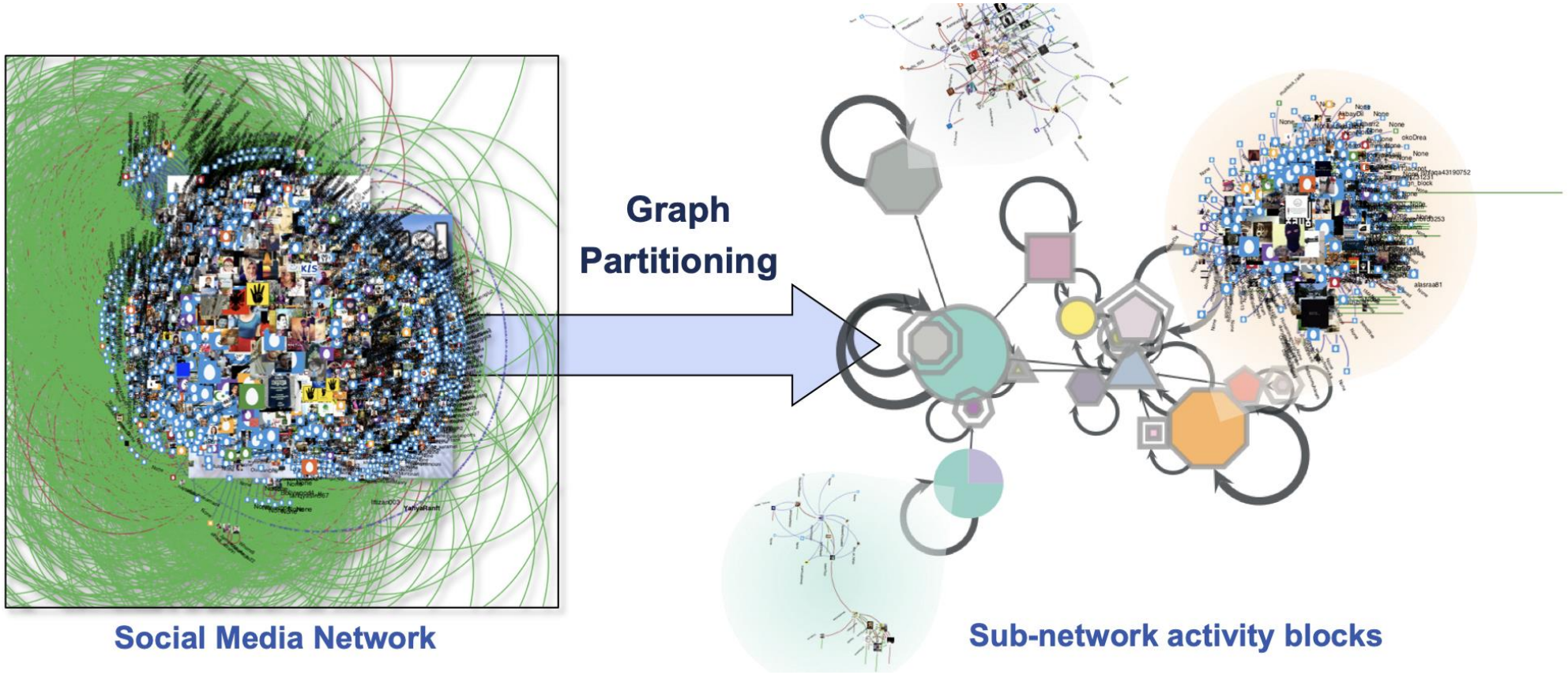
Chih-Chun Chang, Boyang Zhang, Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Wisconsin at Madison, Madison, WI
https://tsung-wei-huang.github.io/

Outline

- Introduce Stochastic Block Partitioning
- Introduce Limitations of Existing Solutions
- Introduce GPU-Accelerated Stochastic Block Partitioner
- Present Experimental Results

Graph Partitioning

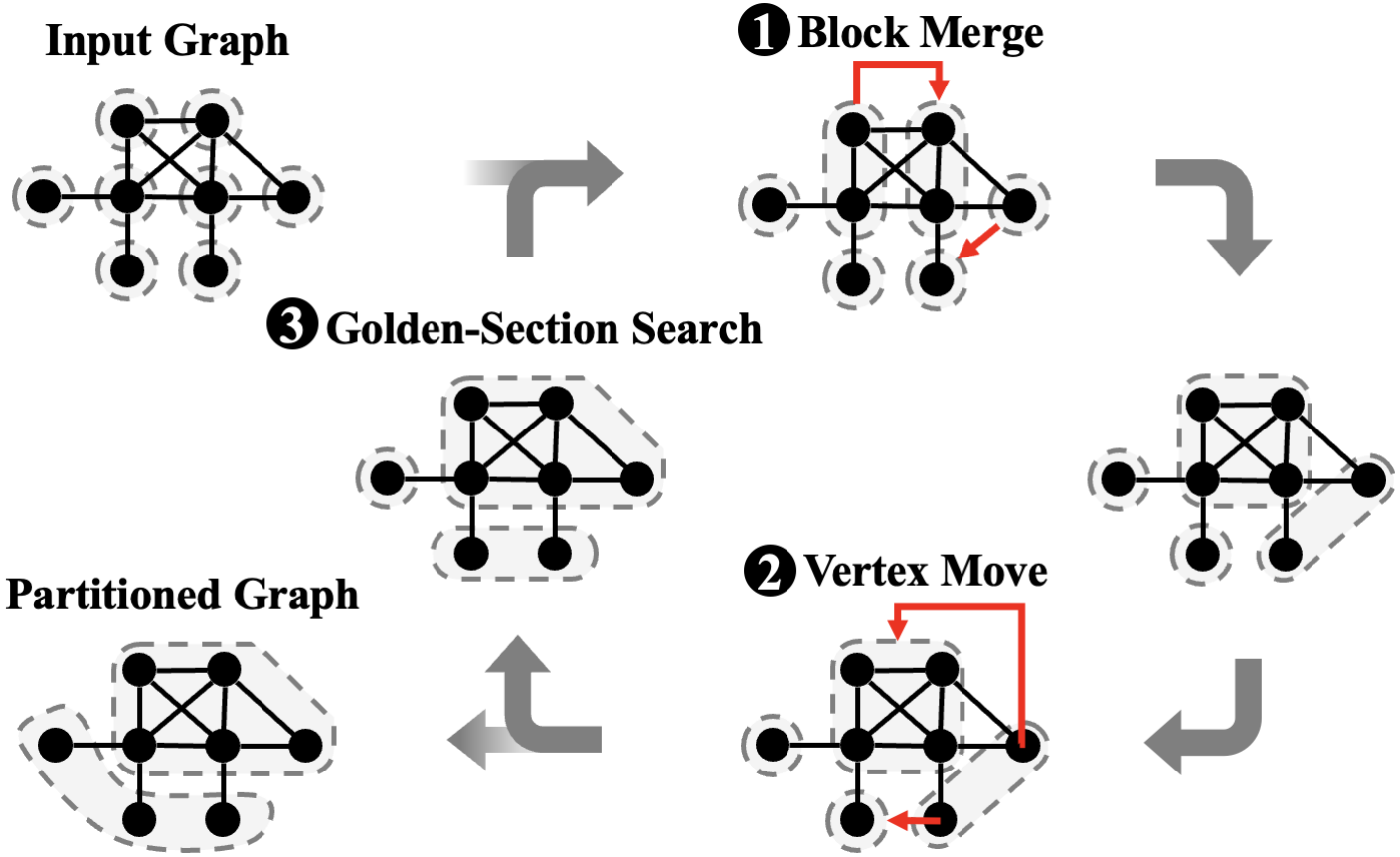
- Graph partitioning divides a graph into several blocks
  - Helps us understand complex graph datasets such as social networks[1]



[1] https://graphchallenge.mit.edu/sites/default/files/documents/GraphPartitionChallenge-2017-03-27.pdf

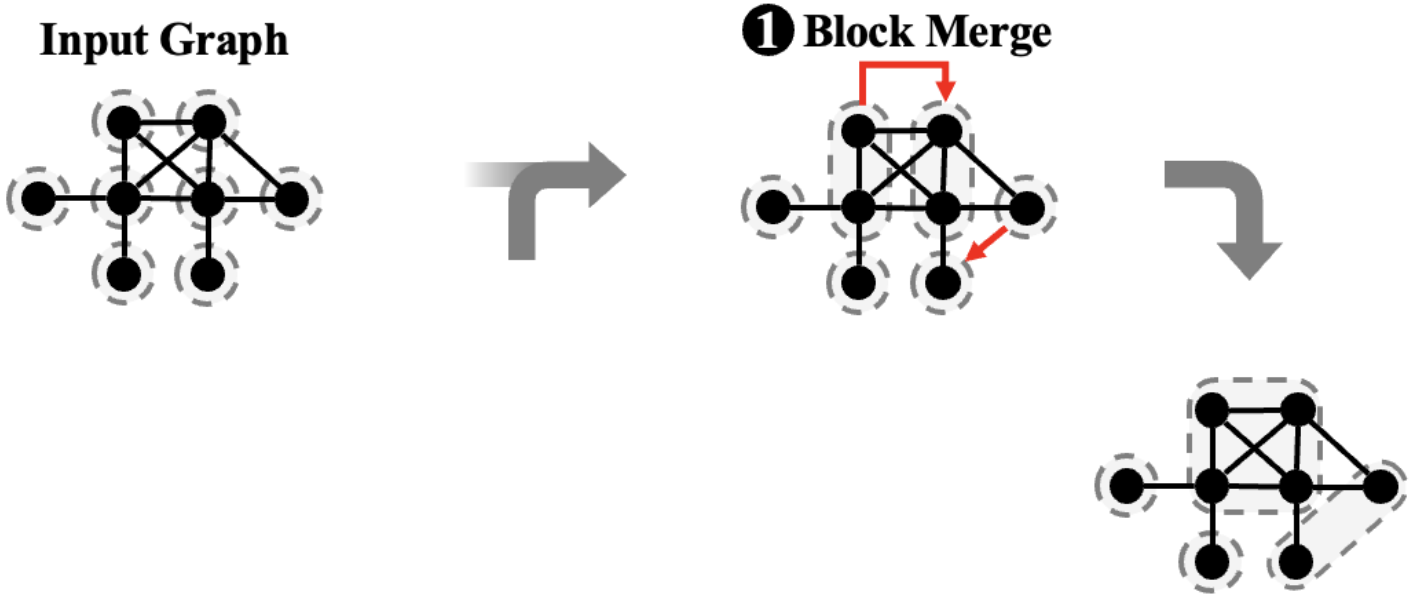
Stochastic Block Partitioning (SBP)

- SBP is one of the graph partitioning algorithms
  - Performs well on complex graphs with different block sizes
- SBP contains three phases
  - ❶ **Block merge**
  - ❷ **Vertex move**
  - ❸ **Golden-section search**



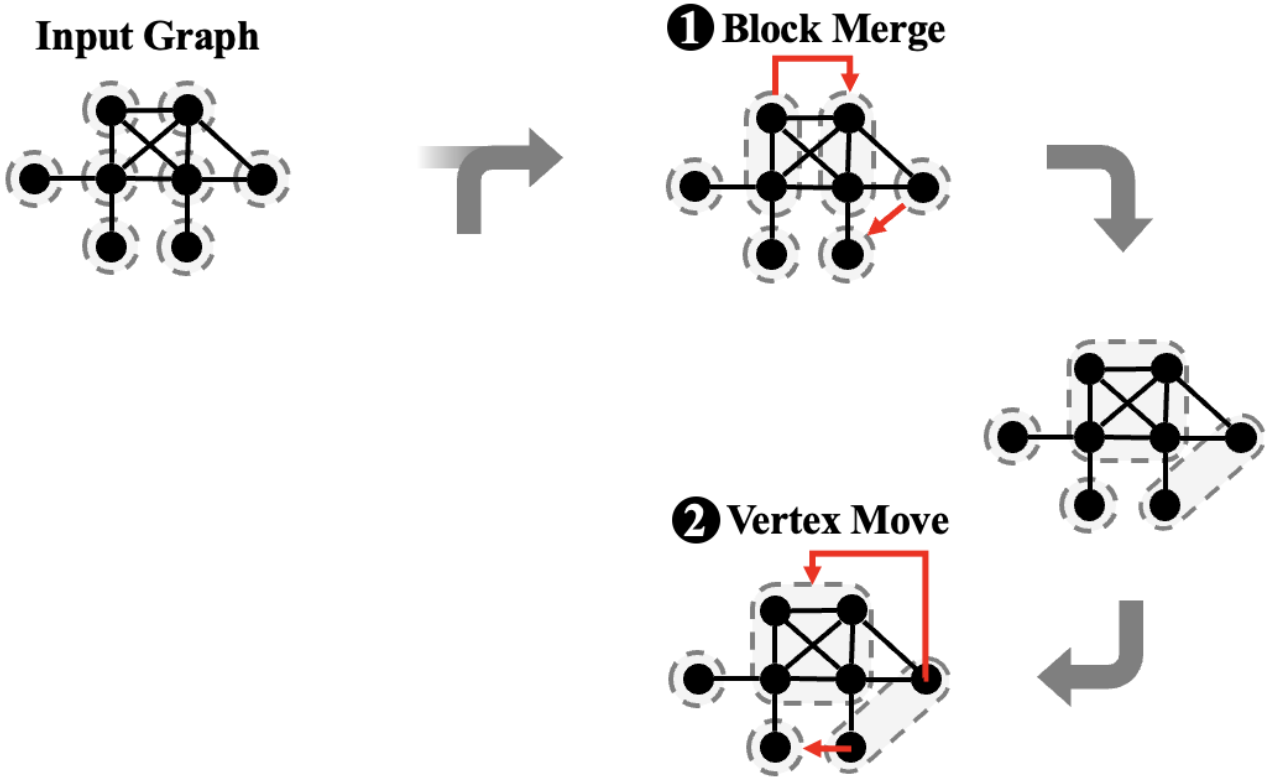
Stochastic Block Partitioning (SBP)

- ❶ **Block merge**
  - Proposes another block to merge stochastically
  - Computes the minimum description length (MDL)
    - MDL is a model selection principle to determine the best model
  - Merges those blocks whose proposals have the smallest MDL changes



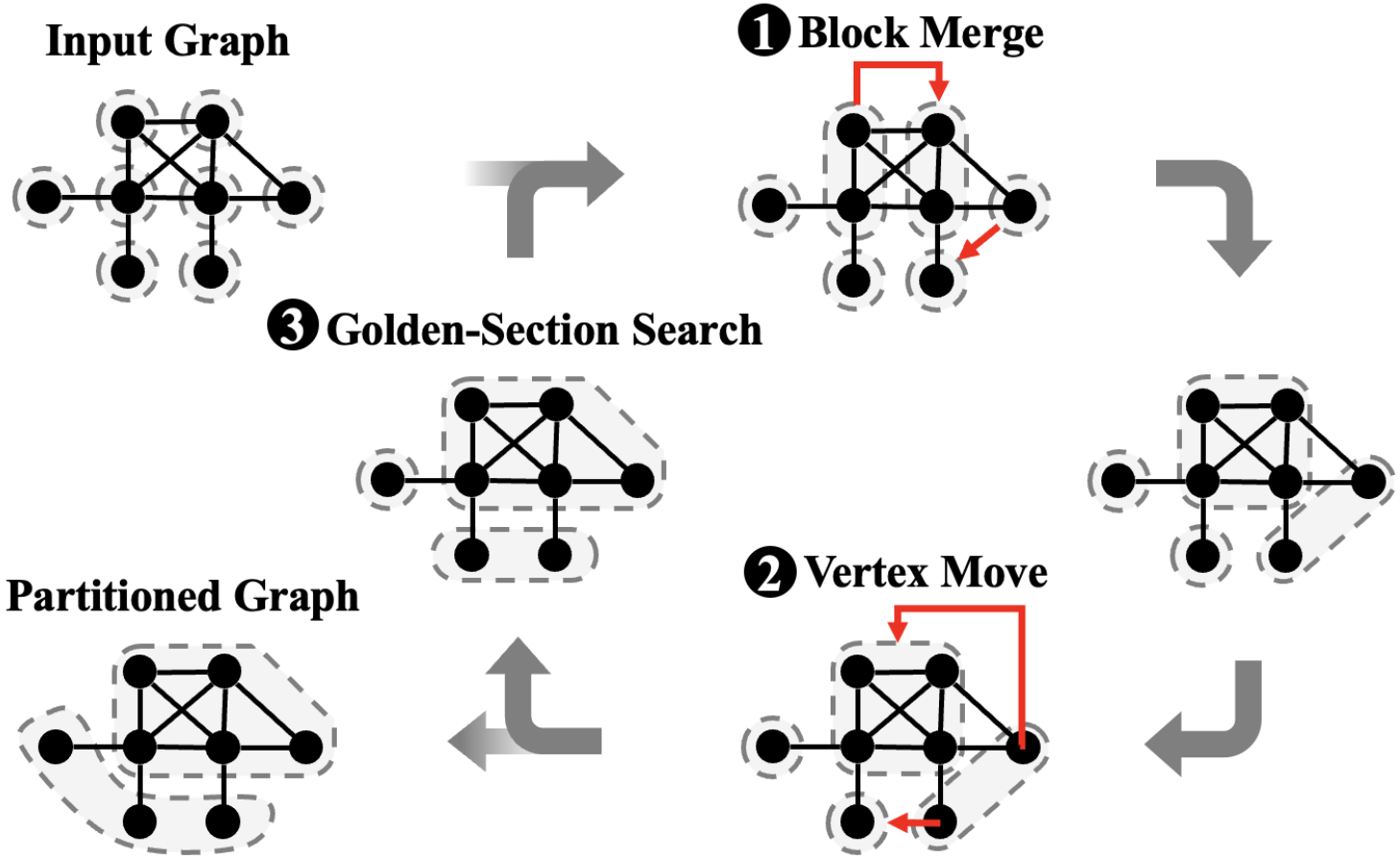
Stochastic Block Partitioning (SBP)

- ❷ **Vertex move**
  - Proposes another block to move stochastically
  - Computes the MDL
  - Samples vertices to move via Monte Carlo Markov Chain (MCMC)



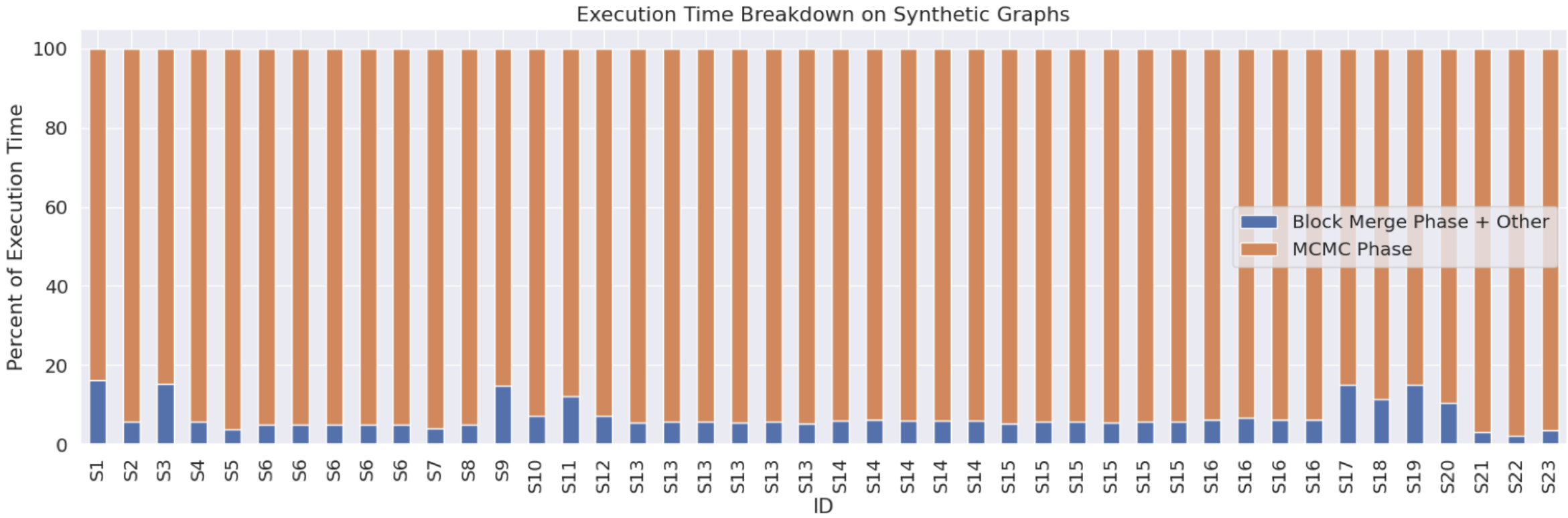
Stochastic Block Partitioning (SBP)

- ❸ **Golden-section search**
  - Finds the optimal partition by minimizing the MDL of the graph



Challenge of SBP

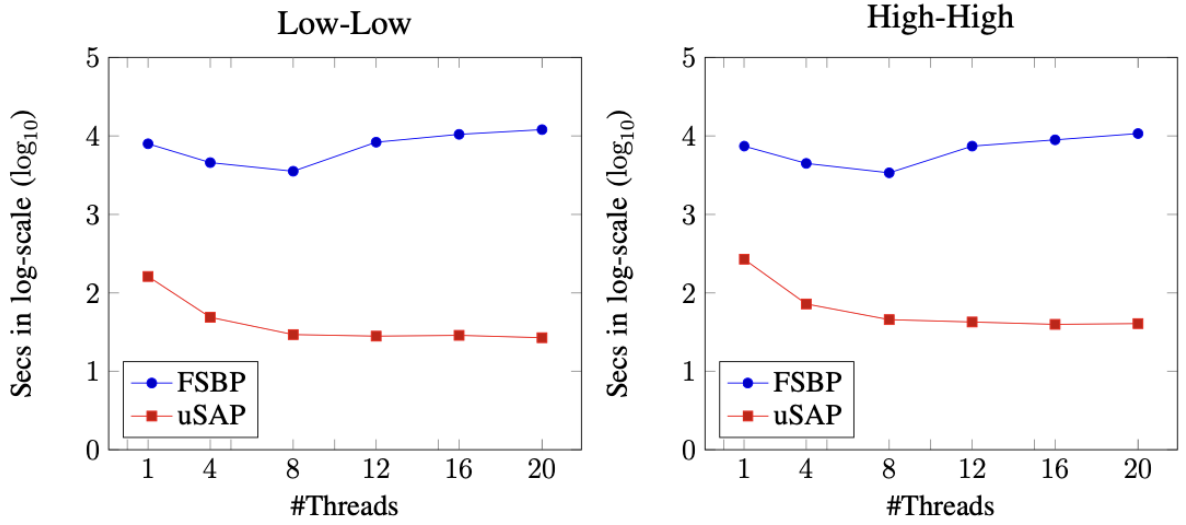
- The MCMC iterations are very time-consuming
  - The vertex move phase takes up to 98% of the overall SBP execution time[2]



[2] Wanye, Frank, et al. "On the parallelization of mcmc for community detection." Proceedings of the 51st International Conference on Parallel Processing. 2022.

Existing Solutions

- IEEE High Performance Extreme Computing Conference (HPEC) introduced the GraphChallenge[3]
- Limitations of existing solutions
  - Handle only small to medium-sized graphs[4] (FSBP)
  - Constrained by CPU parallelism[5] (uSAP)



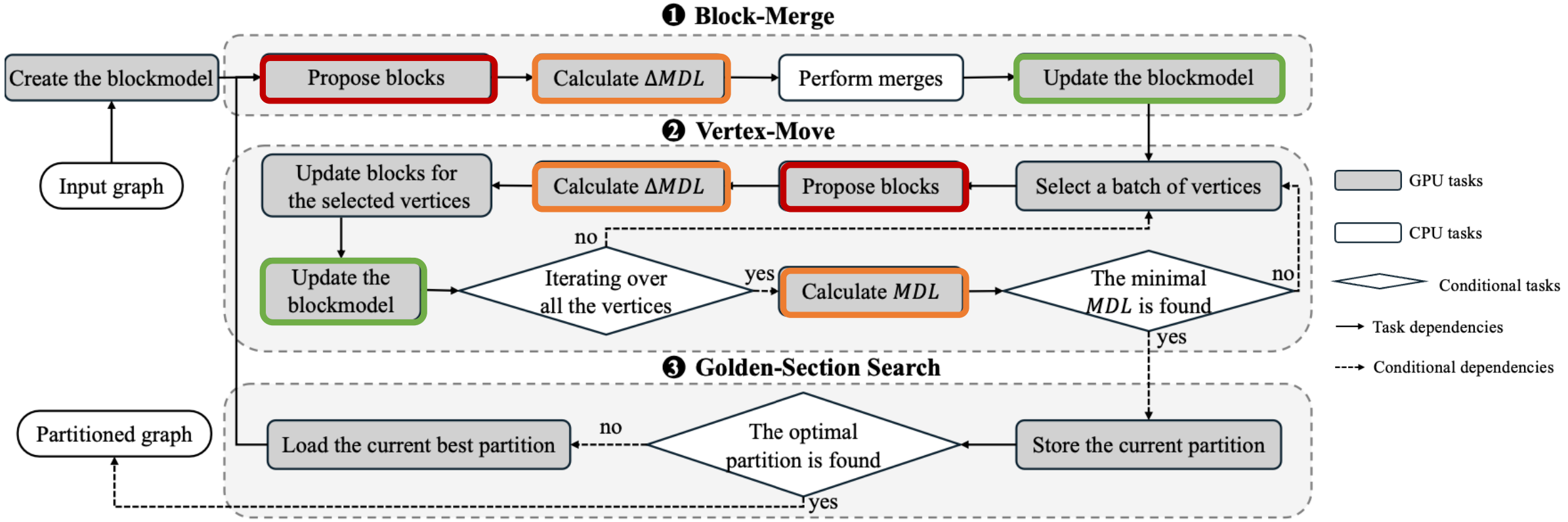
[3] https://graphchallenge.mit.edu
[4] Ahsen J. Uppal, Jaeseok Choi, Thomas B. Rolinger, and H. Howie Huang. 2021. Faster Stochastic Block Partition Using Aggressive Initial Merging, Compressed Representation, and Parallelism Control. In 2021 IEEE High Performance Extreme Computing Conference (HPEC).
[5] Chih-Chun Chang and Tsung-Wei Huang. 2023. uSAP: An Ultra-Fast Stochastic Graph Partitioner. In IEEE High-performance and Extreme Computing Conference (HPEC).

Limitations and Challenges

- Limitations
  - All existing works leverage multi-core CPU parallelism to accelerate SBP
  - The speedup of CPU-based parallelism diminishes as graph sizes increase
- Challenges
  - GPU implementation requires an efficient data structure to represent graphs
  - GPU implementation requires parallel algorithms for the irregular data pattern

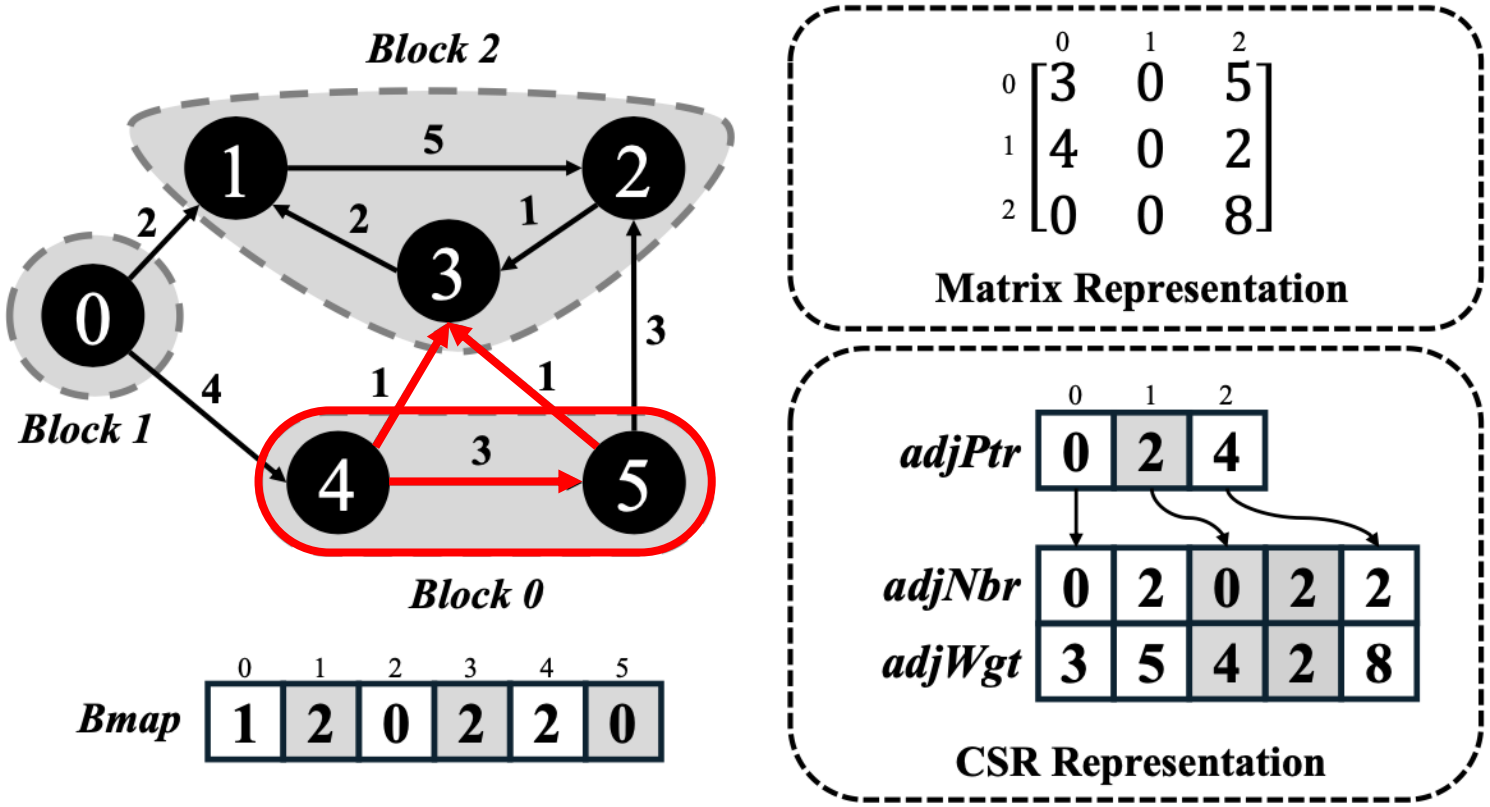
GPU-Accelerated Stochastic Graph Partitioner

- A GPU parallel algorithm for generating stochastic proposals
- A GPU parallel algorithm for computing MDL on irregular data
- A GPU parallel algorithm for updating the blockmodel matrix



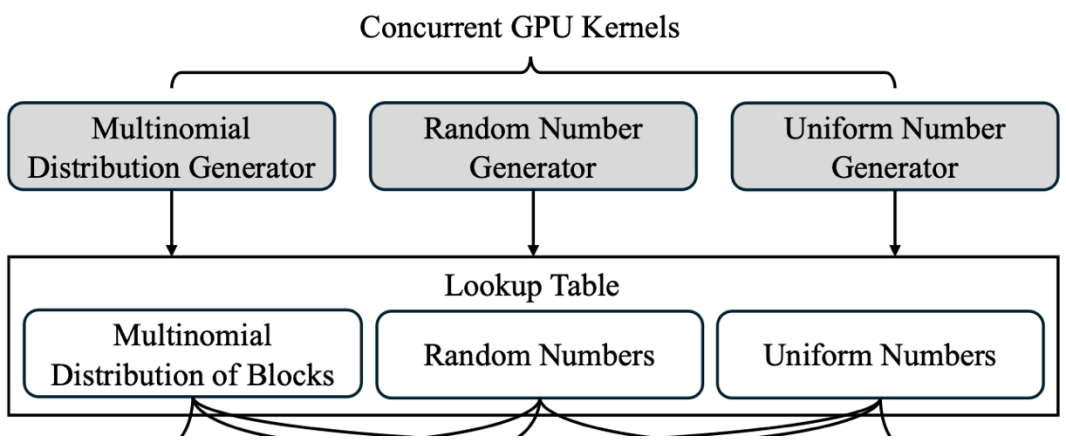
Graph Representation on GPU

- The blockmodel matrix is used to represent the structure of a graph
- The blockmodel matrix appears as a sparse matrix
- We use Compressed Sparse Row (CSR) format to save GPU memory

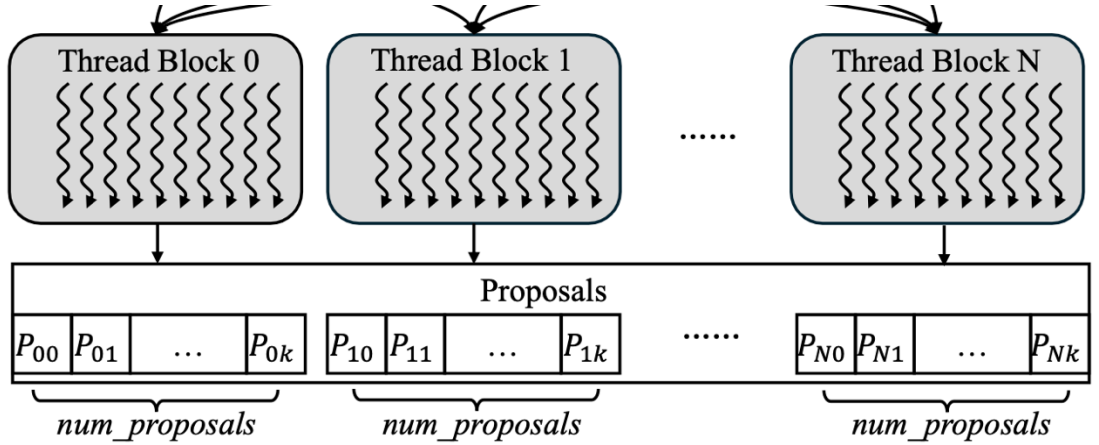


Generating Stochastic Proposals

- Generating stochastic proposal is time-consuming
- We launch sufficient GPU threads to generate random numbers simultaneously, rather than iteratively
- Each GPU thread uses these numbers to make proposals in parallel



(1) Generate lookup tables



(2) Generate stochastic proposals

Computing MDL in Parallel

- The MDL computation consists of heavy floating-point operations
- We decompose the formulation to compute the change of MDL
  - The out-edge part before block merging or vertex moving
  - The in-edge part before block merging or vertex moving
  - The out-edge part after block merging or vertex moving
  - The in-edge part after block merging or vertex moving

$$\Delta MDL = \left(C_{b'}^{out} + C_{b'}^{in}\right) - \left(C_{b}^{out} + C_{b}^{in}\right)$$

$$\text{where} \quad b' = b \cup s$$

- Each part can be computed in parallel

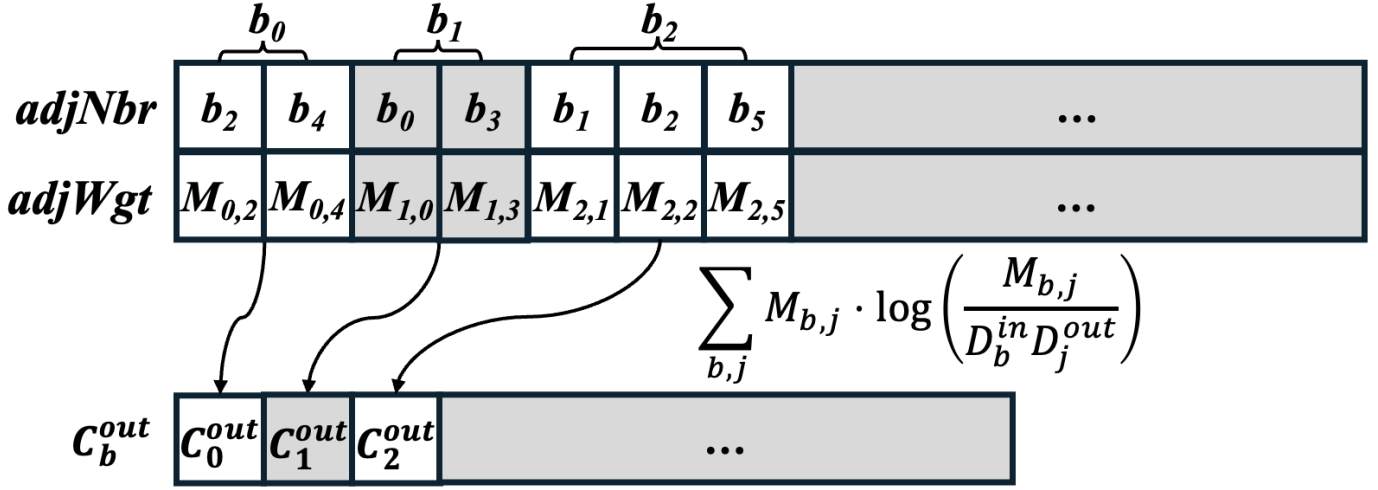
Computing MDL in Parallel

- The MDL is obtained through edge weights, in-degree and out-degree

$$C_b^{out} = \sum_{b,j} M_{b,j} \cdot \log\left(\frac{M_{b,j}}{D_b^{in} D_j^{out}}\right)$$

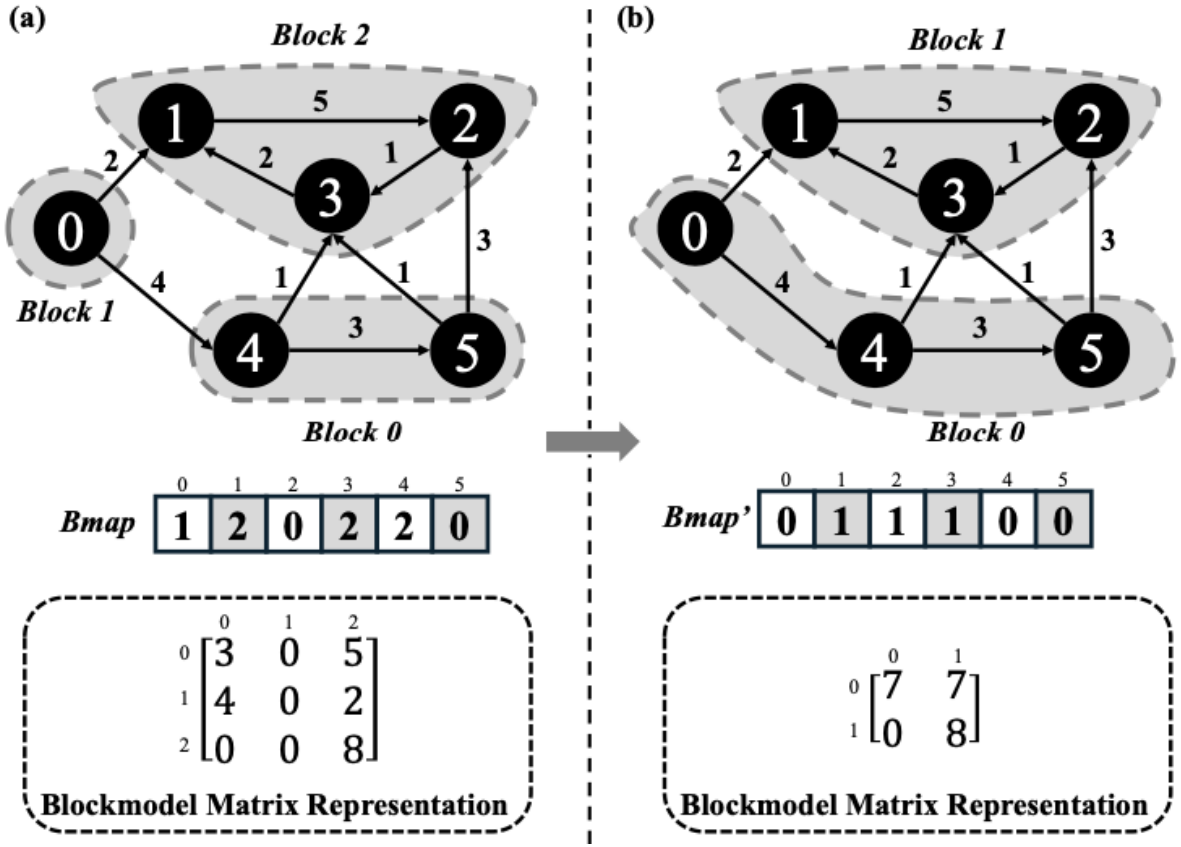
- We perform transformation for each edge to get $M_{b,j} \cdot \log\left(\frac{M_{b,j}}{D_b^{in} D_j^{out}}\right)$

- We perform segmented reduction for each block to get C_b^{out}



Updating the Blockmodel Matrix

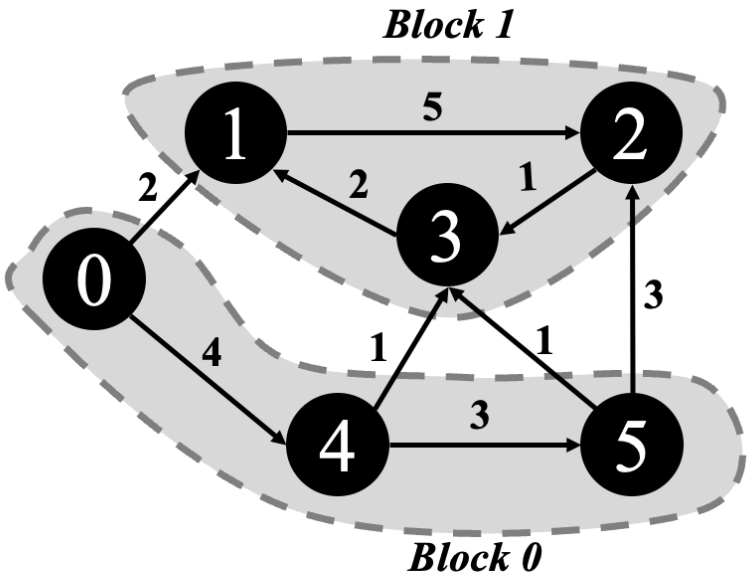
- The blockmodel matrix is frequently updated in SBP.
- It's hard to update a blockmodel matrix in CSR format.

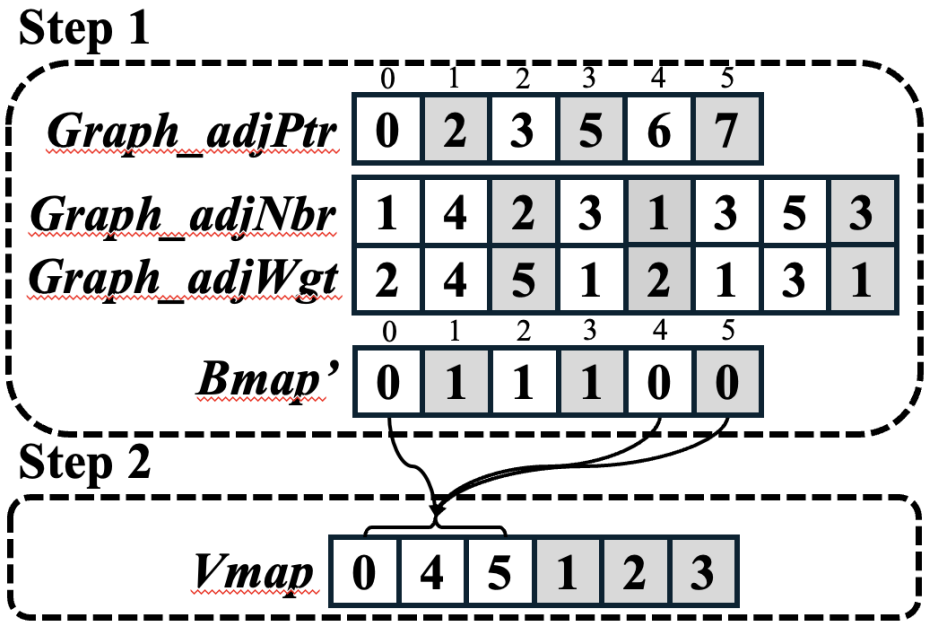


Block 1 merges with block 0

Updating the Blockmodel Matrix

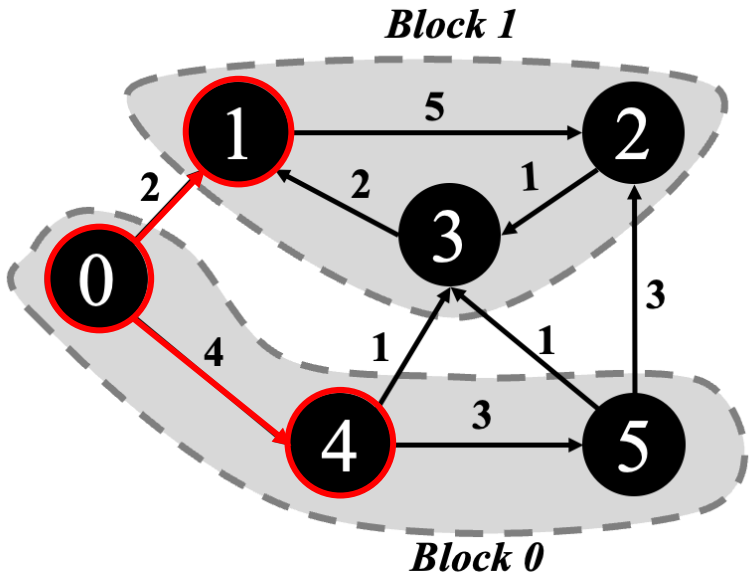
- The goal is to obtain the new CSR of the current partition
- Step 1: prepare the graph's CSR and block ID of each vertex (**`Bmap'`**)
- Step 2: sort the **`Bmap'`** and record the vertex ID

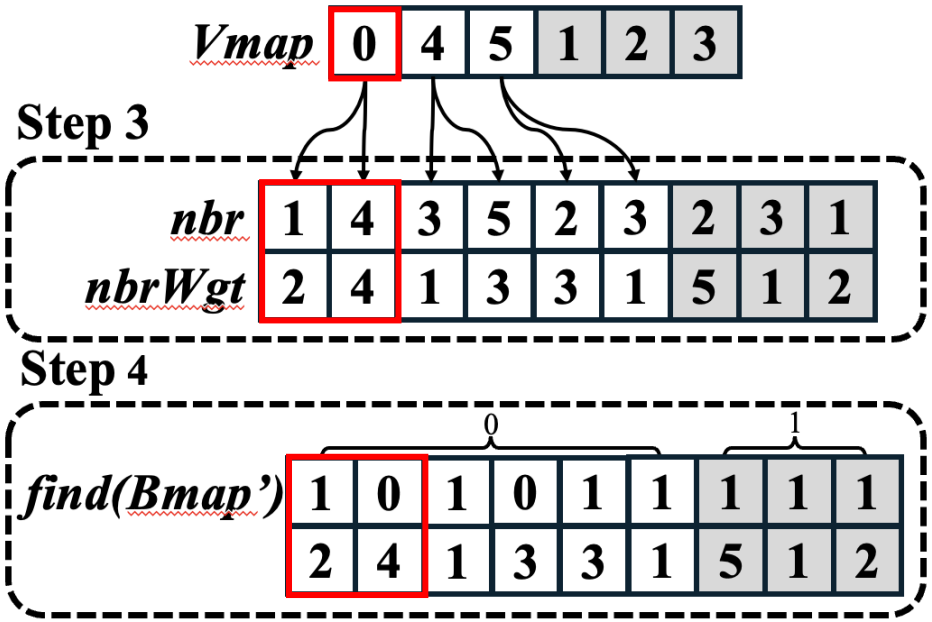




Updating the Blockmodel Matrix

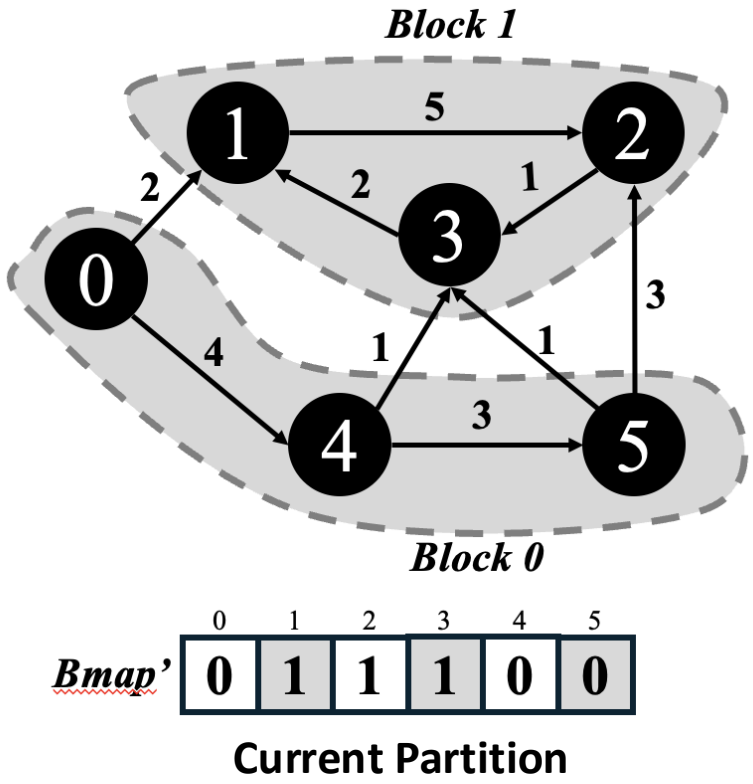
- Step 3: identify each vertex's neighbors and the edges
- Step 4: find the block ID for each neighbor with **`Bmap'`**

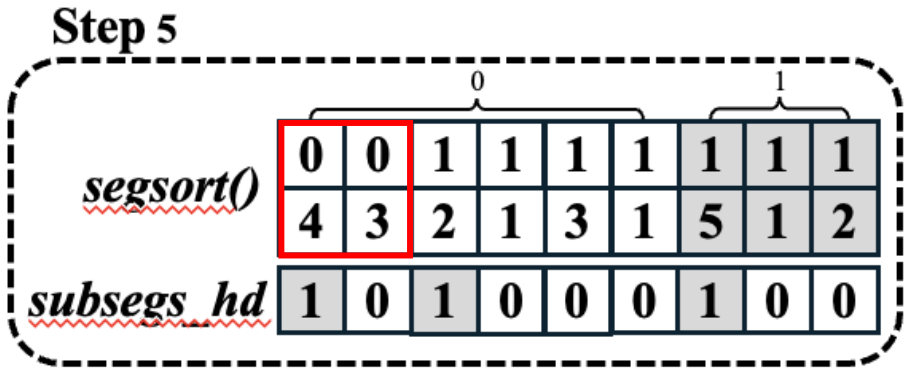




Updating the Blockmodel Matrix

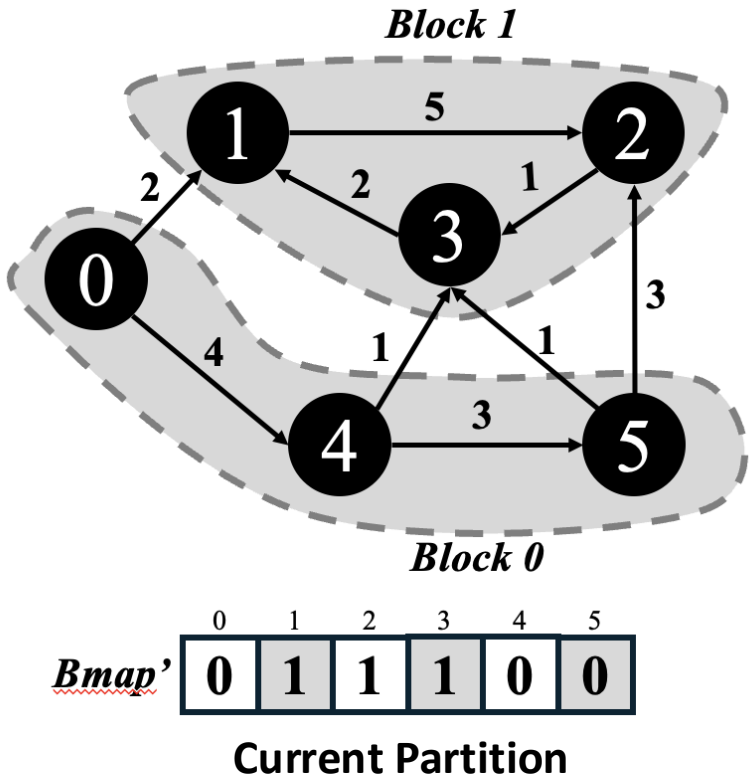
- Step 5.1: perform segmented sorting on the neighboring block ID
- Step 5.2: find the starting position of each subsegment

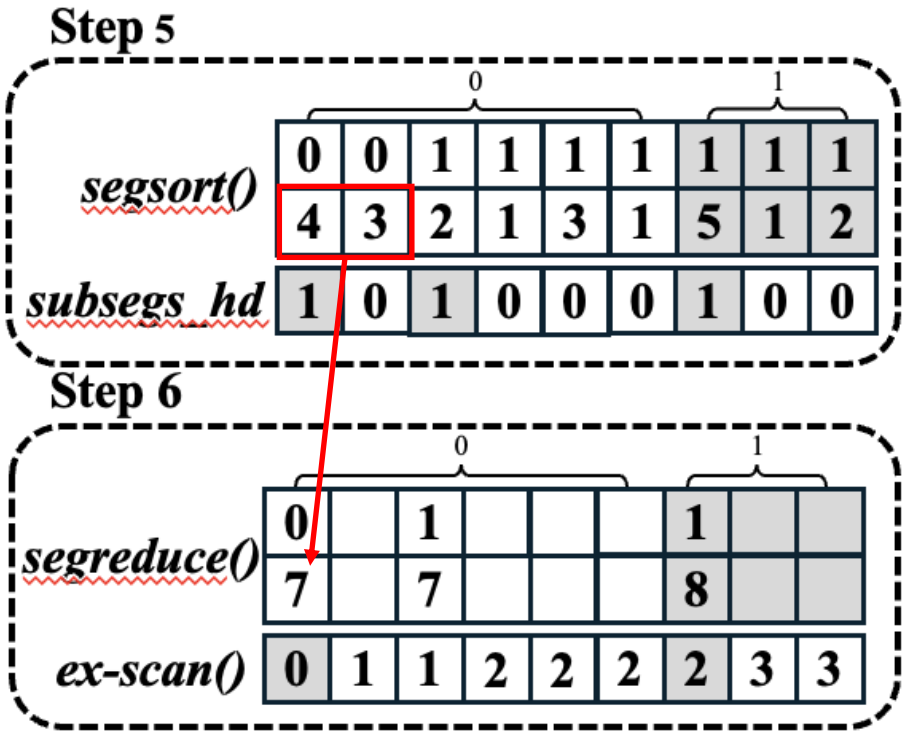




Updating the Blockmodel Matrix

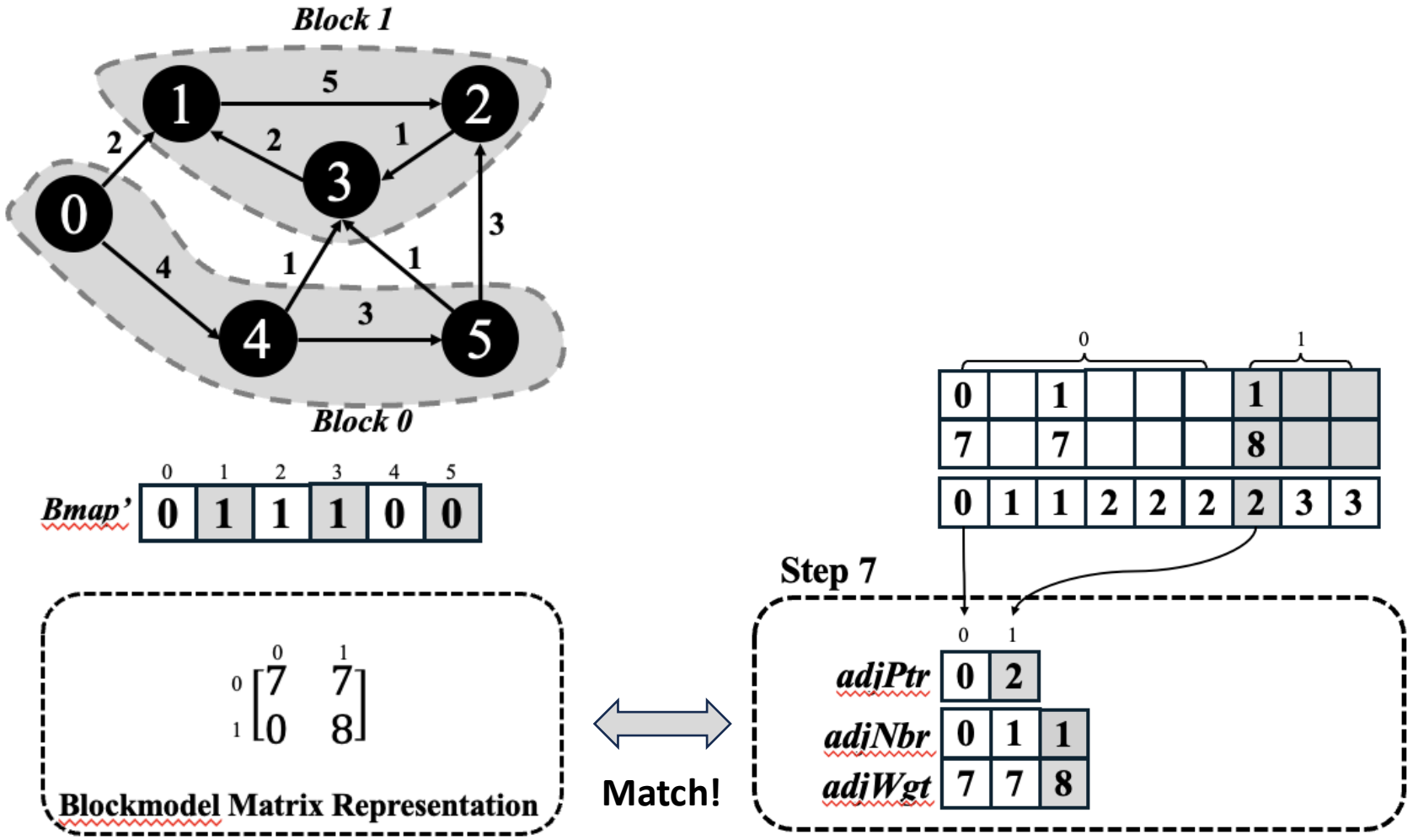
- Step 6.1: perform segmented reduction to remove duplicate block IDs and sum up the edge weights
- Step 6.2: perform prefix-sum on the subsegment head array





Updating the Blockmodel Matrix

- Step 7: obtain the updated blockmodel matrix in CSR format.



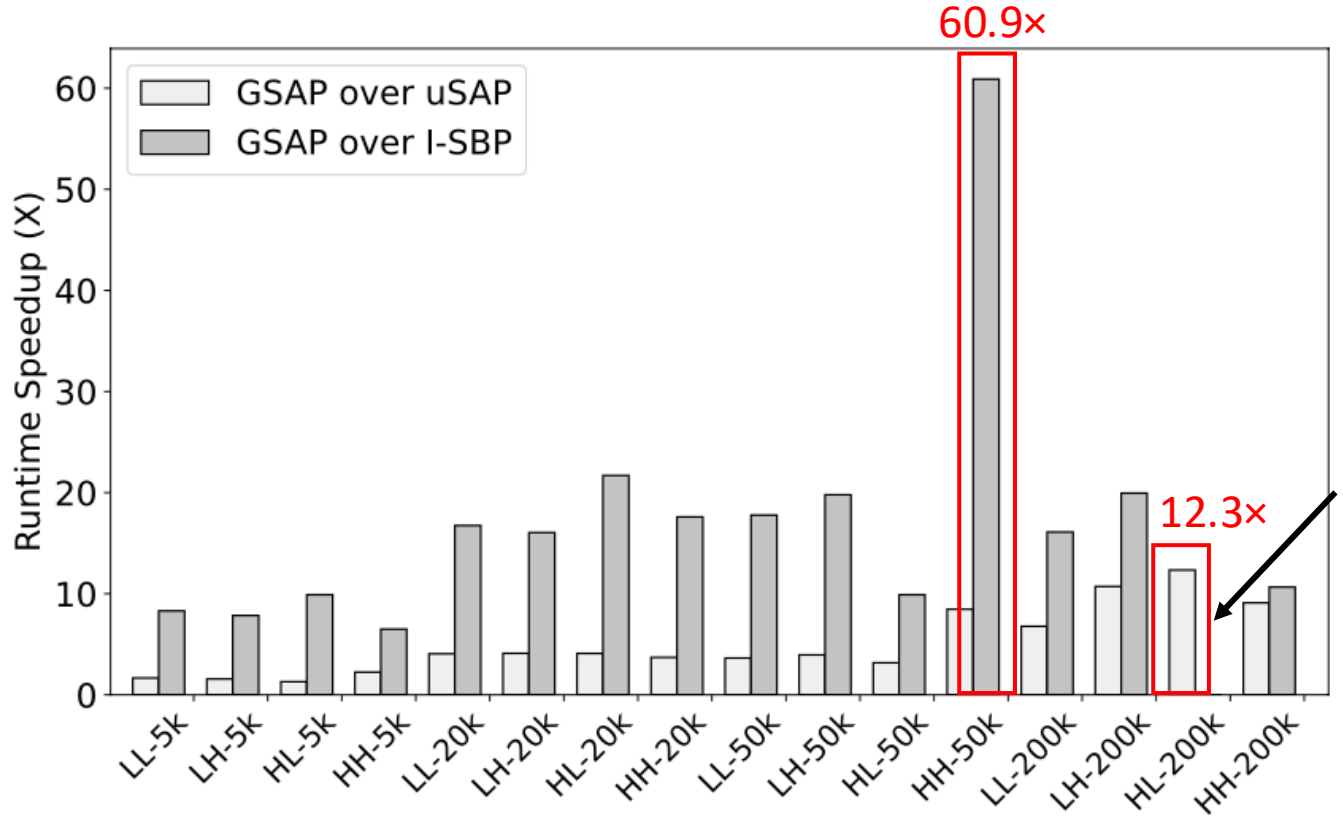
Experiments Setup

- Hardware
  - A 20-thread CPU and a NVIDIA RTX A4000 16 GB GPU
- GraphChallenge Dataset
  - The number of vertices varies from 1K to 1M with different difficulties
- GraphChallenge Baseline
  - The champion, I-SBP, is implemented in OpenMP
  - The innovation award, uSAP, is implemented in Taskflow[6]

[6] Tsung-Wei Huang, Dian-Lun Lin, Chun-Xun Lin, and Yibo Lin. 2022. Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System. IEEE Transactions on Parallel and Distributed Systems (TPDS) (2022).

Runtime Speedup

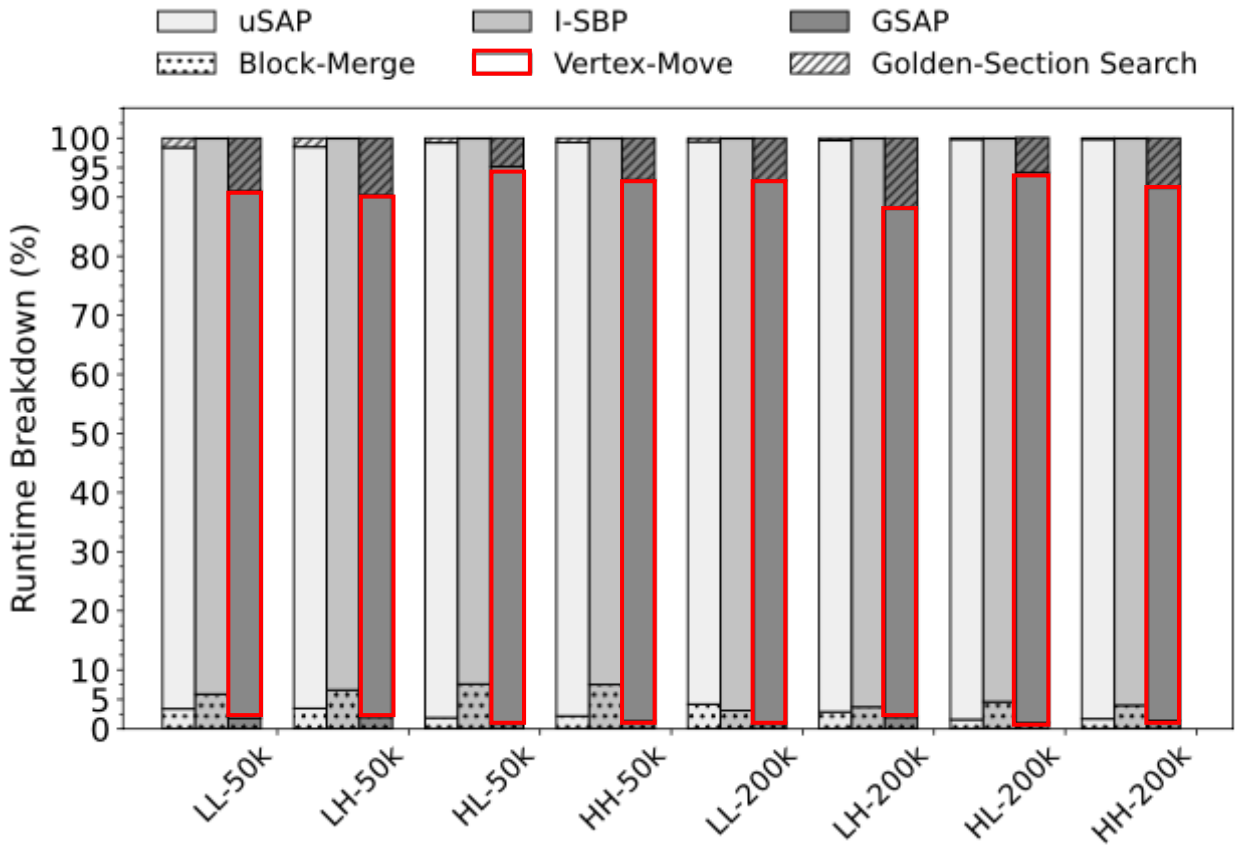
- GSAP is 12.3× and 60.9× faster than uSAP and I-SBP under the 200K-vertex and the 50K-vertex graph
- GSAP is 4.5× and 14.2× faster than uSAP and I-SBP on average



I-SBP failed to find the optimal partition.

Runtime Breakdown

- The vertex move phase in GSAP takes up to 86% of the runtime compared to at least 95% for uSAP and 92.3% for I-SBP



Partition Quality

- We use normalized mutual information to evaluate the partition quality
- GSAP generally outperforms uSAP and I-SBP across all graph

	Graph Categories											
	Low - Low			Low - High			High - Low			High - High		
#Vertices	uSAP	I-SBP	GSAP	uSAP	I-SBP	GSAP	uSAP	I-SBP	GSAP	uSAP	I-SBP	GSAP
1k	0.94	0.94	**0.99**	0.87	0.84	**0.92**	0.78	failed	**0.88**	0.76	0.60	**0.80**
5k	0.99	0.99	**1.00**	0.96	0.93	**0.98**	0.92	0.88	**0.95**	0.69	0.69	**0.80**
20k	1.00	0.98	1.00	0.95	**0.97**	0.96	0.98	0.92	**0.99**	0.88	0.85	**0.89**
50k	0.99	0.99	0.99	0.94	0.93	**0.95**	**0.95**	0.72	0.76	**0.82**	0.33	0.78
200k	0.90	0.90	**0.92**	0.89	0.89	0.79	**0.79**	failed	0.76	**0.85**	0.77	0.71
1m	-	-	**0.84**	-	-	**0.91**	-	-	**0.69**	-	-	**0.73**

Summary

- Introduced SBP
- Introduced limitations of existing solutions
- Introduced GPU-accelerated stochastic block partitioner
- Present experimental results