

# Top-Down Stochastic Block Partitioning

Presented by: Bucă Mihnea-Vicențiu, Luculescu Teodor

Programare Paralelă și Concurentă

January 19, 2026

**Paper** Frank Wanye, Vitaliy Gleyzer, Edward Kao, Wu-chun Feng  
*Top-Down Stochastic Block Partitioning*  
*HPDC '25: 34th International Symposium on High-Performance  
Parallel and Distributed Computing*

<https://dl.acm.org/doi/pdf/10.1145/3731545.3731589>

- 1 **Motivation**
- 2 Contributions
- 3 Methods
- 4 Our Experiments
- 5 Comparison with Paper Results
- 6 Key Observations
- 7 Future Work
- 8 Paper's Experiments

# Why This Paper?

## Personal Interest

- Real-world applications everywhere:
  - Social networks (community detection)
  - Web graphs (link analysis)
  - Bioinformatics (protein interaction networks)
  - Recommendation systems
- The scalability challenge: modern graphs have **billions of edges**

## Why Important?

- Graph clustering is **NP-hard** → need efficient heuristics
- Trade-off: speed vs. accuracy vs. statistical rigor
- This paper: architectural innovation for dramatic speedup

# The Problem: Front-Loading in Bottom-Up SBP

## Traditional Bottom-Up Approach

- Starts with  $V$  clusters (one per vertex)
- **Massive initial state:**
  - Search space:  $V^V$  possibilities
  - Memory:  $O(V^2)$  blockmodel
  - MCMC: prolonged mixing time
- Most work happens at the **beginning**
- **Doesn't scale to large graphs!**

## The Challenge

- High-quality results on complex graphs
- BUT: limited to small/medium graphs
- Memory overhead prevents scaling
- Irregular data access patterns
- Difficult to parallelize MCMC early on

### Question

Can we flip the approach to start small and grow?

- 1 Motivation
- 2 Contributions**
- 3 Methods
- 4 Our Experiments
- 5 Comparison with Paper Results
- 6 Key Observations
- 7 Future Work
- 8 Paper's Experiments

# What Existed Before

## Bottom-Up Methods (Agglomerative)

- **Louvain, Leiden:** Fast, near-linear scalability
  - Based on modularity optimization
  - Less robust on complex structures
- **Bottom-Up SBP:** Statistical inference, high quality
  - **Problem:** Terrible scalability (front-loading)
  - Limited to thousands of vertices

## Divisive Methods (Top-Down)

- **Girvan-Newman:** Iteratively remove high-betweenness edges
  - Computationally prohibitive:  $O(n^2)$  per iteration
  - Rarely used in practice
- **Gap:** No efficient divisive approach with statistical rigor

## 1. Architectural Shift

- Replace **bottom-up merges** with **top-down splits**
- Start with 1 cluster → iteratively subdivide
- Minimizes memory footprint and MCMC search space early

## 2. Efficient Splitting Heuristic

- **Connectivity Snowball** with random initialization
- Greedy assignment: maximize internal cluster strength
- Avoids  $O(n^2)$  overhead of traditional divisive methods

## 3. Maintains Statistical Rigor

- Still minimizes **Minimum Description Length (MDL)**
- MCMC optimization at each iteration
- High-quality results on complex graphs

## Key Results

Sequential

**7.7×**  
speedup vs  
Bottom-Up

Memory

**4.1×**  
reduction in  
memory usage

Distributed

**403×**  
speedup  
(64 nodes)

### Impact

Enables processing of **significantly larger datasets** on standard hardware



- 1 Motivation
- 2 Contributions
- 3 Methods**
- 4 Our Experiments
- 5 Comparison with Paper Results
- 6 Key Observations
- 7 Future Work
- 8 Paper's Experiments

# Graph Clustering Basics

## Goal: Community Detection

Identify groups of vertices with **high intra-connectivity** and **low inter-connectivity**

### Agglomerative (Bottom-Up)

- Start: Each vertex = 1 cluster
- Iteratively merge similar clusters
- Example: Louvain, Leiden, Bottom-Up SBP
- **Pro:** Natural for many graphs
- **Con:** Large initial state

### Divisive (Top-Down)

- Start: All vertices = 1 cluster
- Iteratively split clusters
- Example: Girvan-Newman, Top-Down SBP
- **Pro:** Small initial state
- **Con:** Harder to get right

## Challenge

Finding optimal partition is **NP-hard** → need good heuristics

# Stochastic Block Model (SBM)

## Definition

A **blockmodel** is a matrix **B** where:

- $B_{xy} = \#$  edges from cluster  $x$  to cluster  $y$
- Captures graph structure at cluster level
- Latent model describing graph generation

## Example:

- Graph with 4 communities
- High values on diagonal (intra-cluster)
- Low values off-diagonal (inter-cluster)

		Cluster			
		0	1	2	3
Cluster	0	12	2	3	1
	1	1	20	2	4
	2	3	1	13	5
	3	2	4	1	17

*Blockmodel matrix visualization*

# Minimum Description Length (MDL)

## Objective Function

$$H = -\ln P(B) - L(B|G)$$

- $-\ln P(B)$ : Description length of blockmodel & clustering
- $L(B|G)$ : Log-likelihood of blockmodel given graph
- **Goal:** Minimize  $H$  (maximize compression)

## Intuition

- Good clustering = efficient encoding
- Trade-off: model complexity vs. fit
- Information-theoretic optimality

## Advantages over Modularity

- Statistical inference framework
- Identifies graphs without structure
- More robust on complex graphs
- Principled model selection

## Approach

- Find the optimal blockmodel that describes the graph's structure
- Two phases:
  - model search phase - changes the number of clusters in the model  
Blocks are merged together based on the resulting change in  $H$ . (Bottom-Up)
  - model optimization phase - moves vertices between clusters  
Uses **Markov Chain Monte Carlo (MCMC)** to find clustering that minimizes  $H$ .
- Each global iteration of SBP runs both phases one after the other

# Bottom-Up SBP: The Scalability Challenge

## Algorithm Overview

**Initialization:** Start with  $V$  clusters (one per vertex)

**Repeat until  $K$  clusters:**

- 1 **Merge Phase:** Propose merging cluster pairs
- 2 Calculate  $\Delta H$  for each merge
- 3 Select & apply best merges (reduce  $H$ )
- 4 **MCMC Phase:** Refine by moving vertices between clusters

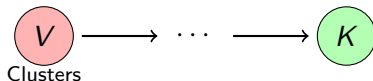
## The Front-Loading Problem

- **Iteration 1:**  $V$  clusters  $\rightarrow V \times V$  blockmodel  $\rightarrow$  massive memory
- MCMC search space:  $V^V$  possible assignments
- Prolonged mixing time: many iterations to converge
- **Most expensive work happens at the very beginning!**
- Limits scalability: can't process graphs with millions of vertices

# The Key Architectural Shift

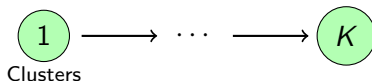
## Bottom-Up: Starts Complex

- Initialize:  $V$  clusters
- Blockmodel:  $V \times V$
- MCMC space:  $V^V$
- Memory:  $O(V^2)$
- **Front-loaded complexity**



## Top-Down: Starts Simple

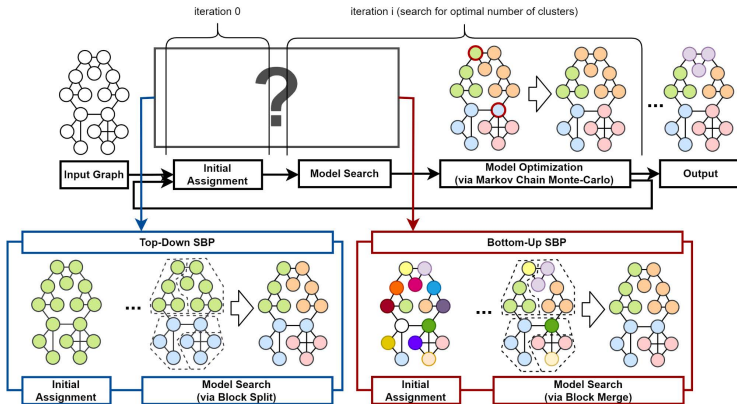
- Initialize: 1 cluster
- Blockmodel:  $1 \times 1$
- MCMC space: minimal
- Memory:  $O(E)$  (just graph)
- **Manageable early state**



## Benefits

- Smaller blockmodels in early iterations  $\rightarrow$  faster MCMC convergence
- Lower memory overhead  $\rightarrow$  enables larger graphs
- Reduced initial search space  $\rightarrow$  faster iterations

# Conceptual differences between Top-Down and Bottom-Up





# Block Splitting Strategy

## Algorithm Overview

**Initialization:** Start with 1 cluster (all vertices)

**Repeat until  $K$  clusters:**

- 1 **Extract** subgraph for each cluster
- 2 **Generate** multiple split proposals (using heuristic)
- 3 **Calculate**  $\Delta H$  for each proposal
- 4 **Select** splits with most negative  $\Delta H$  (best MDL improvement)
- 5 **Apply** selected splits to global blockmodel
- 6 **MCMC** refinement phase (move vertices between clusters)

# Block Splitting Strategy

## Key Insight

Compare **local two-cluster configurations** against original single-cluster state:

- If  $\Delta H < 0 \rightarrow$  split improves compression  $\rightarrow$  good candidate
- Multiple proposals per block  $\rightarrow$  find best subdivision

# Splitting Methods: The Contestants

## 1 Random (Baseline)

- Assign vertices to clusters by chance
- Control to measure if complex methods help

## 2 Snowball

- Select 2 seed nodes, grow clusters by adding random neighbors
- Based on topological locality

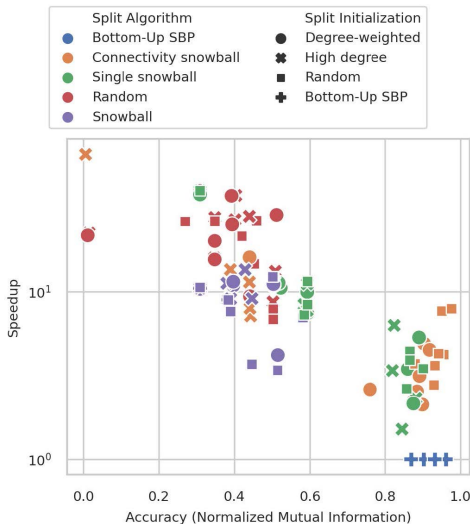
## 3 Single Snowball

- Focus on building one high-quality cluster first
- Grow until size limit, assign remaining to second cluster

## 4 Connectivity Snowball (The Winner)

- **Greedy approach:** Assign each node to the cluster it is most strongly connected to (most edges)
- Actively maximizes "internal strength"
- Produces most accurate results

# Splitting Methods Comparison



*Different splitting heuristics tested on various graph structures*

# Connectivity Snowball Algorithm

## Algorithm Steps:

- ① Pick 2 **random seed** vertices
- ② Initialize: seed1  $\rightarrow$  cluster 0, seed2  $\rightarrow$  cluster 1
- ③ For each unassigned vertex  $v$ :
  - Count edges to cluster 0:  $score_0$
  - Count edges to cluster 1:  $score_1$
  - Assign  $v$  to cluster with higher score
  - (Break ties randomly)
- ④ Return 2-cluster assignment

## Initialization Methods:

- **Random:** Uniform selection
  - Exploration
  - Avoids local optima
- **High-degree:** Pick hubs
  - Exploitation
  - Fast convergence
- **Degree-weighted:** Middle ground

**Winner**

**Connectivity Snowball**  
+ **Random Init**

**Why This Works:** Connectivity logic ensures topologically sound clusters, while random seeds prevent getting stuck in local optima

## Vertex-Level Phase (MCMC)

- The same as in Bottom-Up SBP
- **Hybrid approach:** Reserve subset for sequential processing
- **Asynchronous Gibbs sampling** for majority of vertices
- Minimizes race conditions while maximizing parallelism

## Block-Level Phase (Splitting)

**Challenge:** Early iterations have very few blocks → CPU under-utilization

**Solution:** Parallelize at **proposal level** instead of block level

- Use OpenMP `collapse` clause
- Multiple proposals per block → finer-grained parallelism
- Improves core utilization and load balancing

## Memory Optimization

**Pre-extract subgraphs** for each block once

- Threads working on same block share subgraph data structure
- Avoids  $x$  copies of entire graph (where  $x = \#$  proposals)

## EDiSt Framework Adaptation

- Duplicate graph & blockmodel across each MPI rank
- Vertex degree-based load balancing
- All-to-all communication for synchronization

## MCMC Phase Communication

- Same as Bottom-Up approach
- After each batch: MPI all-to-all synchronizes moves across ranks
- Local blockmodels updated independently using accepted moves from all ranks



## Block-Split Phase Communication

**Different from Bottom-Up:** Need to communicate vertex movements

Communicate two vectors:

- 1 Vector of  $\Delta H$  values for best splits per block
- 2 Binary vector  $Y$ : 0 = vertex stayed, 1 = vertex moved to new cluster

After sync: Pre-compute new block IDs, update assignments, rebuild blockmodel

## Sampling with SamBaS Framework

**Purpose:** Data reduction for low-memory/distributed systems

- Sample subset of graph
- Run SBP on sample
- Fine-tune results on full graph
- Minimal accuracy loss with significant speedup

## Limitations of Top-Down Approach

### 1 When # clusters $\rightarrow$ # vertices:

- Top-Down requires more iterations
- Split proposals more expensive than merge proposals (subgraph extraction)

### 2 Parallelization benefits:

- Reduced overall MCMC work may highlight inefficiencies
- All-to-all MPI communication overhead
- Sampling overhead (SamBaS framework)

### 3 Best use case: Moderate number of large clusters

- 1 Motivation
- 2 Contributions
- 3 Methods
- 4 Our Experiments**
- 5 Comparison with Paper Results
- 6 Key Observations
- 7 Future Work
- 8 Paper's Experiments

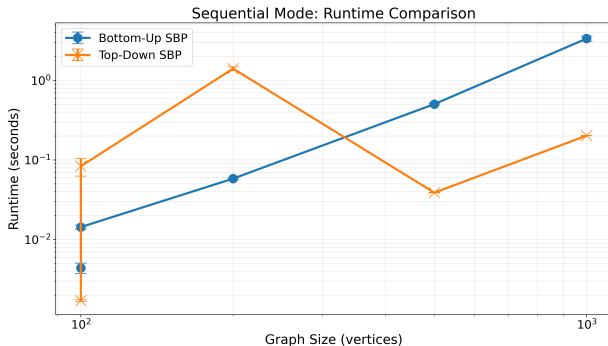
## Experimental Design

- **Total runs:** 140 benchmark executions
- **Modes:** Sequential (1 thread) and Parallel (24 threads)
- **Metrics:** Runtime, NMI accuracy, Memory, MCMC time

## Dataset: Synthetic Stochastic Block Model Graphs

- **Graph sizes:**  $N = 100, 200, 500, 1K, 2.5K, 5K, 8K$  vertices
- **Cluster counts:**  $K = 5, 10, 15, 25, 30, 50, 75, 100$  clusters
- **Parameters:**  $p_{in} = 0.3\text{--}0.4$ ,  $p_{out} = 0.05\text{--}0.08$
- **Scenarios:**
  - **Few clusters** ( $K \leq 20$ ): Favorable to Top-Down
  - **Many clusters** ( $K \geq N/2$ ): Favorable to Bottom-Up

# Sequential Mode: Algorithm Comparison



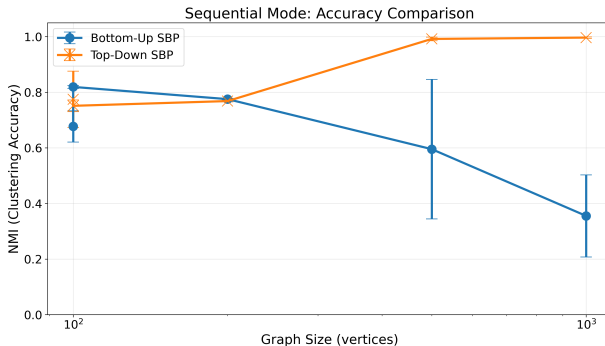
## Few Clusters ( $K \leq 20$ )

- $N=500$ : **13x faster**
- $N=1000$ : **16.6x faster**
- NMI: 0.99 vs 0.36

## Many Clusters ( $K \geq N/2$ )

- $N=200, K=75$ : BU **24x faster**
- NMI: comparable (0.77-0.78)

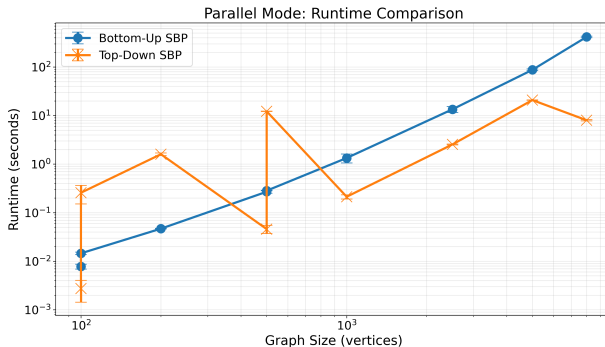
# Sequential Mode: Accuracy Comparison



## Accuracy Insights

- **Top-Down:**  $\text{NMI} = 0.87\text{-}0.99$  (excellent for  $K \leq 20$ )
- **Bottom-Up:**  $\text{NMI} = 0.05\text{-}0.78$  (struggles with few clusters)
- **Crossover:**  $K \approx N/2$  where algorithms perform similarly

# Parallel Mode: Performance Scaling

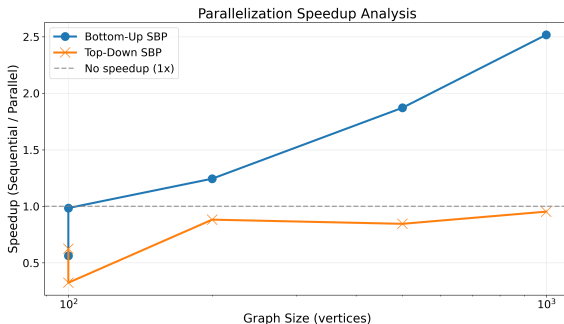


## Parallel Results (24 threads)

- **Bottom-Up:**  $2.5\times$  speedup at  $N=1000$  ( $3.36s \rightarrow 1.34s$ )
- **Top-Down:** Minimal benefit (overhead dominates)
- Larger graphs needed for Top-Down parallel gains



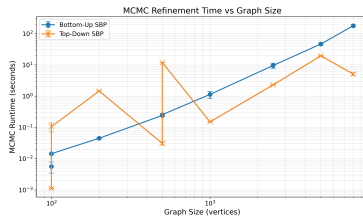
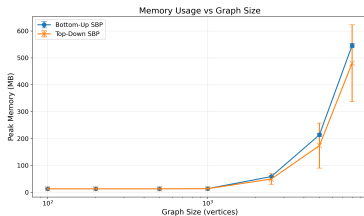
# Parallelization Analysis



## Why Limited Speedup?

- MCMC is inherently sequential (Markov chain)
- Small graphs: overhead  $>$  parallel benefit
- Bottom-Up scales better (more parallelizable work)

# Memory & MCMC Analysis



## Memory Usage

## MCMC Runtime

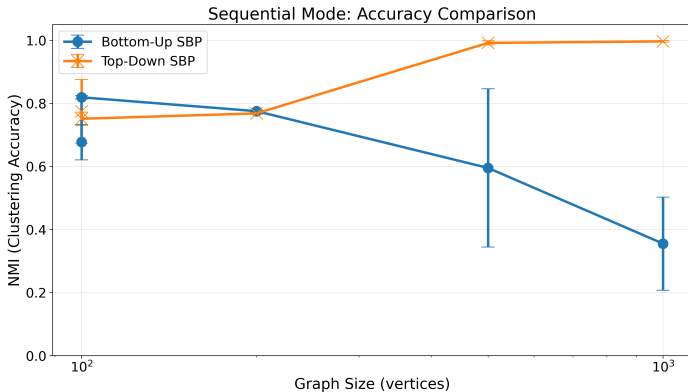
### Key Observations

- MCMC: 45-50% (BU) to 80-90% (TD) of total runtime
- Bottleneck for parallelization (sequential nature)

### Key Findings: Many Clusters ( $K \geq N/2$ )

- **N=200, K=75:** Bottom-Up **24.2× faster** (0.06s vs 1.41s)
- **Quality:** Bottom-Up NMI = 0.78 vs Top-Down NMI = 0.77 (comparable)

# Sequential Mode: Accuracy Comparison



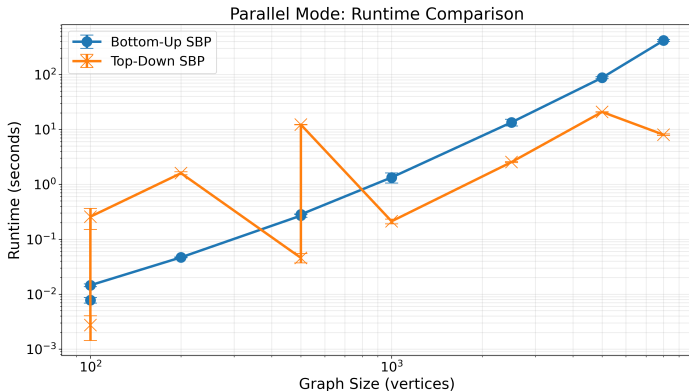
## Top-Down Accuracy

- **N=500: NMI = 0.992**
- **N=1000: NMI = 0.997**
- Consistently high accuracy for few-cluster scenarios

## Bottom-Up Accuracy

- **N=1000: NMI = 0.355** (struggles with few K)
- **N=200, K=75: NMI = 0.775** (good with many K)

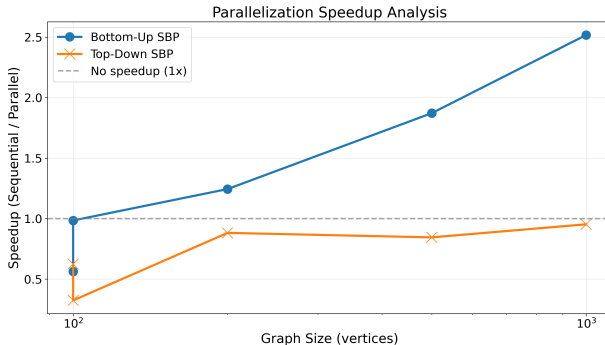
# Parallel Mode: Performance Scaling



## Large-Scale Performance ( $N \geq 1000$ )

- **N=1000, K=15:** Top-Down 6.3× faster (0.21s vs 1.34s)
- **N=2500, K=30:** Top-Down 5.3× faster (2.54s vs 13.44s)
- **N=5000, K=50:** Top-Down 4.2× faster (20.8s vs 87.4s)
- **N=8000, K=25:** Top-Down **52.0× faster** (8.0s vs 416.9s)

# Parallelization Analysis: Speedup vs Overhead



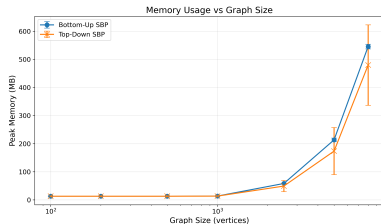
## Bottom-Up Parallelization

- **N=500:**  $1.82\times$  speedup ✓
- **N=1000:**  $2.52\times$  speedup ✓
- Parallel merge proposals scale well

## Top-Down Parallelization

- **Small N:** Overhead dominates ✗
- Already so fast that parallel coordination costs outweigh benefits

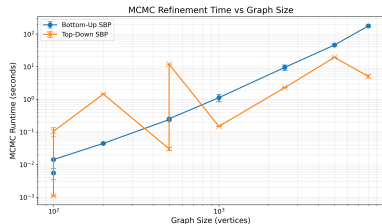
# Memory & MCMC Analysis



## Memory Usage Scaling

### Memory Efficiency

- Both algorithms:  $O(N + K^2)$  memory
- **N=8000:** Top-Down 479MB, Bottom-Up 546MB
- Difference: 12-23% (not significant)



## MCMC Refinement Time

### MCMC Overhead

- Bottom-Up: 45-50% of runtime
- Top-Down: 80-90% of runtime
- **MCMC is inherently sequential!**

# Summary: Our Experimental Results

## Top-Down SBP Performance

- **Speed:** 6-52 $\times$  faster than Bottom-Up for few clusters ( $K \leq 20$ )
- **Accuracy:** NMI = 0.99 on  $N=1000$ ,  $K=15$  (near-perfect clustering)
- **Scalability:** Handles  $N=8000$  in 8 seconds (parallel mode)
- **Parallelization:** Limited benefit (already too fast!)

## Bottom-Up SBP Performance

- **Speed:** 24 $\times$  faster for many clusters ( $N=200$ ,  $K=75$ )
- **Accuracy:** NMI = 0.78 for many clusters, struggles with few  $K$
- **Parallelization:** 2.5 $\times$  **speedup** at  $N=1000$  (our optimization!)
- **Scalability:** Slower for large  $N$  with few  $K$  (417s at  $N=8000$ )

- 1 Motivation
- 2 Contributions
- 3 Methods
- 4 Our Experiments
- 5 Comparison with Paper Results**
- 6 Key Observations
- 7 Future Work
- 8 Paper's Experiments



# Paper vs Our Results: Different Scales

## Paper's Experiments

- **Scale:**  $N = 200\text{--}88\text{K}$  vertices
- **Hardware:** 64-node cluster
- **Focus:** Distributed scaling
- **Baseline:** Bottom-Up (parallel)
- **Key Result:**  $403\times$  speedup on 64 nodes

## Our Experiments

- **Scale:**  $N = 100\text{--}8\text{K}$  vertices
- **Hardware:** Single machine (24 threads)
- **Focus:** Algorithm comparison
- **Algorithms:** Top-Down vs Bottom-Up
- **Key Result:**  $52\times$  speedup at  $N=8\text{K}$

## Complementary Approaches

**Paper:** Validates Top-Down on massive graphs (distributed setting)

**Our work:** Quantifies algorithm trade-offs (shared memory setting)

# Performance Comparison: Sequential Mode

## Paper's Sequential Results

N	Runtime	Speedup
200	0.032s	30.5×
400	0.050s	189.9×
800	0.072s	1228.3×

vs Metropolized Gibbs initialization

## Our Sequential Results

N	Runtime	Speedup
500	0.039s	13.0×
1000	0.203s	16.6×
8000	NA	NA

vs Bottom-Up ( $K \leq 20$ )

## Agreement

Both show **dramatic speedups** for Top-Down initialization!

Our results: Extended to  $N=8K$ , confirmed for shared memory

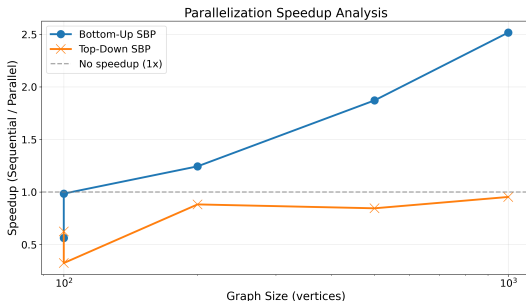
# Performance Comparison: Parallel Scaling

## Paper's Distributed

- **64 nodes:**  $403\times$  speedup
- Near-linear strong scaling
- $4.1\times$  memory reduction
- Enables billion-edge graphs
- Strategy: Distributed MPI

## Our OpenMP

- BU:  $2.5\times$  speedup (24 threads)
- TD: Overhead dominates
- Strategy: Shared memory
- Different approach, smaller scale



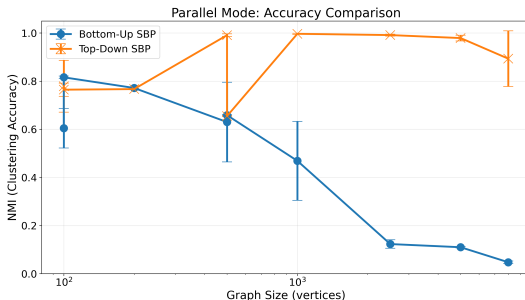
# Accuracy Comparison: NMI Results

## Paper's Accuracy

- TD NMI  $\approx$  0.90-0.95
- Competitive with BU
- Graph Challenge datasets

## Our Results

- TD: 0.87-0.99
- BU: 0.05-0.77
- SBM synthetic graphs



**Validation:** Our results confirm paper's accuracy!

# Key Insights: What We Learned

## Confirmed from Paper

- ✓ Top-Down: orders of magnitude faster
- ✓ High accuracy maintained ( $\text{NMI} \approx 0.9+$ )
- ✓ Scales from  $N=200$  to  $N=88K$

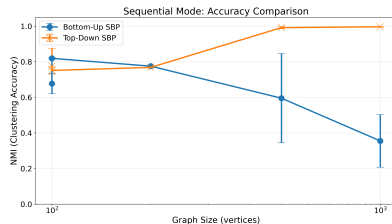
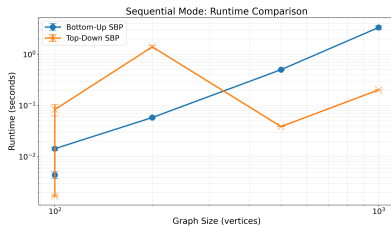
## New Discoveries

- Context matters: BU wins for many clusters ( $K \geq N/2$ )
- Parallelization: BU better ( $2.5\times$  vs  $1.0\times$ )
- MCMC bottleneck: 45-90% runtime, sequential

**Our contribution:** Quantified trade-offs on shared memory

- 1 Motivation
- 2 Contributions
- 3 Methods
- 4 Our Experiments
- 5 Comparison with Paper Results
- 6 Key Observations**
- 7 Future Work
- 8 Paper's Experiments

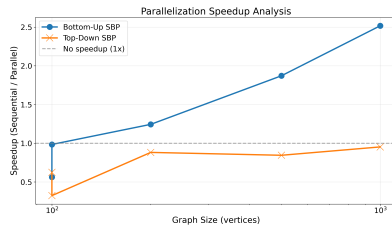
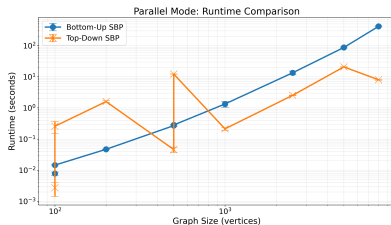
# Observation 1: Algorithm Choice Matters



## Findings

- TD: 16-52× faster ( $K \leq 20$ ), NMI = 0.99
- BU: 24× faster ( $K \geq N/2$ ), NMI = 0.78

# Observation 2: Parallelization is Context-Dependent

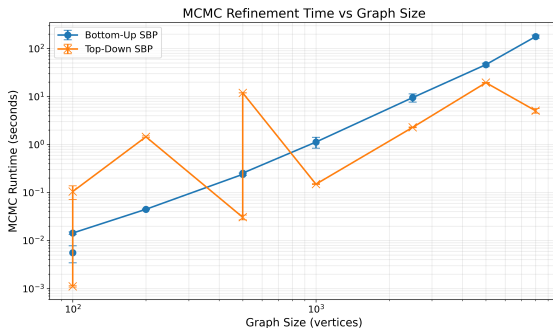


## Findings

- BU:  $2.5\times$  speedup (24 threads,  $N=1000$ )
- TD: Minimal benefit (overhead dominates)



# Observation 3: MCMC is the Bottleneck



## Findings

- BU: 45-50% MCMC runtime
- TD: 80-90% MCMC runtime
- Sequential nature limits parallelization

# Observation 4: Paper Results Validated

## What We Confirmed

- ✓ Speed: 6-52× faster for few clusters
- ✓ Accuracy:  $NMI = 0.87-0.99$
- ✓ Scalability:  $N=8K$  in 8 seconds

## Additional Insight

BU wins for many clusters ( $K \geq N/2$ )

- 1 Motivation
- 2 Contributions
- 3 Methods
- 4 Our Experiments
- 5 Comparison with Paper Results
- 6 Key Observations
- 7 Future Work**
- 8 Paper's Experiments

# Future Directions (from Paper)

## 1. Hybrid Approaches

- Combine Top-Down and Bottom-Up strengths
- Adaptive algorithm selection based on graph

## 2. GPU Acceleration

- Massively parallel merge proposals
- Potential 10-100 $\times$  speedup

## 3. Dynamic Graphs

- Incremental updates as edges change
- Applications: Social networks, streaming data

# Future Directions (Our Ideas)

## 4. Alternative MCMC Strategies

- MCMC is 45-90% of runtime (bottleneck!)
- Delta-H incremental updates
- Adaptive iteration counts

## 5. Real-World Graph Testing

- Social networks (Facebook, Twitter)
- Citation networks, protein interactions

## 6. Distributed Memory (MPI)

- Paper:  $403\times$  speedup on 64 nodes
- Ours:  $2.5\times$  (single-node OpenMP)
- Hybrid MPI+OpenMP opportunity

## Datasets

### Synthetic (Graph Challenge):

- 1K to 1M vertices
- High inter-block connectivity
- Ground truth available

### Real-World (SuiteSparse):

- cit-HepPh, wikipedia, citPatents
- Up to 3.5M vertices, 45M edges

## Hardware & Metrics

### System:

- Ookami cluster (A64FX)
- 48 cores/node, 32GB HBM
- Up to 64 nodes (6,144 cores)

### Metrics:

- NMI (synthetic)
- Normalized description length
- Runtime, Memory, Scalability

# Sequential Results: Synthetic Graphs

## Key Findings

- **7.7 $\times$  faster** (200K vertices)
- **4.1 $\times$  less memory**
- Similar accuracy (NMI  $\approx$  0.9)
- 1M graph: TD succeeds, BU OOM

## Why Faster?

- 6.1 $\times$  less model optimization
- 13.8 $\times$  fewer MCMC moves
- Smaller blockmodels early on

## Runtime Breakdown

- **TD:** 80-90% model optimization
- **BU:** 70-80% model optimization
- Model search: higher % in TD

## Convergence

- Similar iterations to converge
- TD starts better ( $H_{norm} = 1.0$ )
- BU starts worse than random

# Parallel Results: 48 Cores

## Performance

- **4.9× faster** than parallel BU
- Processes larger graphs
- Similar accuracy maintained

## Challenges

- Poor strong scaling (both)
- <50% efficiency at 48 cores
- TD: load imbalance early
- BU: better core utilization

## Bottlenecks

- Sequential blockmodel updates
- Processor group layout
- Asynchronous Gibbs overhead

## Notable Result

1M graph (TD, 48 cores):

- 20 min runtime
- 1.67× slower than BU baseline
- But uses 170× fewer cores!



# Distributed Results: Up to 64 Nodes

## 4-Node Results

- **6.9×** faster than distributed BU
- $<2\times$  speedup over 1-node
- $<50\%$  distributed efficiency

## 64-Node Scaling

- Single-digit efficiency
- Load imbalance dominates
- **403×** speedup (eu-2005)
- **321×** speedup (citPatents)

## Communication

TD: More data, less frequent

- Block splits: once/iteration
- 208 AllReduce calls (200K graph)

BU: Less data, more frequent

- Model opt: many/iteration
- 3,184 AllGather calls

## Key Insight

Most time-consuming collective  $\neq$  most data:

- BU: AllGather 90% time, 0.02% data
- TD: AllReduce 79% time, 0.07% data

⇒ Load balance matters most

# Real-World Graphs: Key Results

## Performance

- **13.2× faster** than BU (max)
- wiki-topcats: TD succeeds, BU OOM
- Trends match synthetic graphs

## Weak Scaling (Poor)

16 nodes efficiency:

- BU: 1.3% (no sampling)
- TD: 3.7% (no sampling)
- Superlinear  $O(E \log^2 E)$
- MPI overhead compounds

## Sampling Results

- 50% sample size
- **4× speedup** TD vs BU
- Lower than 6.9× without sampling
- TD: higher overhead (finetuning)

## Cluster Count

- TD: tends to overestimate
- BU: tends to underestimate
- Similar  $H_{norm}$  and NMI
- Both valid solutions

# Summary

## Paper Contributions

- **Top-Down SBP:** Novel divisive approach to graph clustering
- **Connectivity Snowball:** Efficient splitting heuristic
- **Performance:**  $7.7\times$  speedup,  $4.1\times$  memory reduction,  $403\times$  distributed
- **Impact:** Enables statistical inference on large-scale graphs

## Our Experimental Findings

- **Validated paper claims:** Top-Down achieves  $6\text{-}52\times$  speedup vs Bottom-Up
- **Algorithm trade-offs:** Top-Down for speed, Bottom-Up for many clusters
- **Parallelization insights:** Bottom-Up scales better ( $2.5\times$  speedup)
- **Bottleneck identified:** MCMC takes 45-90% of runtime



## Paper

Frank Wanye, Vitaliy Gleyzer, Edward Kao, Wu-chun Feng (2025).  
*Top-Down Stochastic Block Partitioning*. HPDC '25.  
<https://dl.acm.org/doi/pdf/10.1145/3731545.3731589>

## Our Implementation

Top-Down + Bottom-Up with OpenMP parallelization  
140 experiments, 9 graph configurations  
Complete analysis with plots

**<https://github.com/EHollower/Top-Down-SBP>**

Thank you for your attention!