HBMax: Optimizing Memory Efficiency for Parallel Influence Maximization on Multicore Architectures

Xinyu Chen (Washington State University)
Marco Minutoli (Pacific Northwest National Laboratory)
Jiannan Tian (Indiana University Bloomington)
Mahantesh Halappanavar (Pacific Northwest National Laboratory)
Ananth Kalyanaraman (Washington State University)
Dingwen Tao (Indiana University Bloomington)

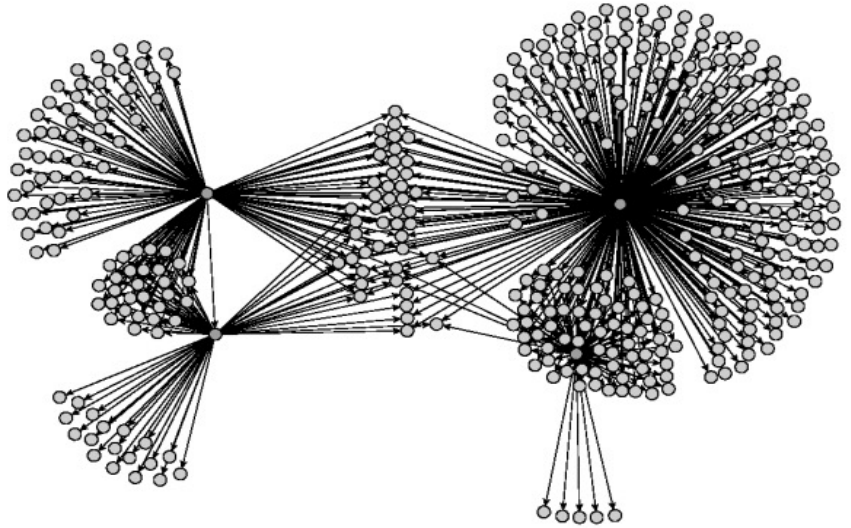
Background

➢ Influence Maximization (IM) Problem

- Given a graph G=(V,E), find k vertices that can activate maximal number of vertices in G
- Wide applications in viral marketing, politics, public health, sensor networks and bioinformatics

➢ Approximate Solutions

- A NP-hard optimization problem (Kempe *et al.*)
- Use Monte Carlo (MC) simulations to approximate (Borgs *et al.*)
  - **Intensive Computation** cost (Sampling $\theta \approx 10^5$ diffusions)
  - **High Memory** usage (store intermediate results)
  - **Easy Analysis** (Counting and "Pruning")



Japanese graph novel*

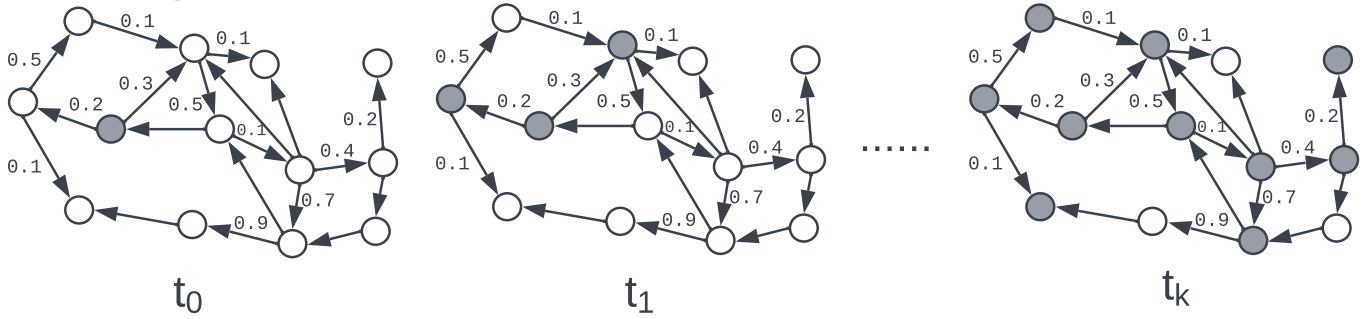
* The dynamics of viral marketing, Jure *et al.* 2007

Background

➢ Influence Maximization via Martingales Algorithm (IMM)

- Diffusion Models
  - Independent Cascade (IC)
- Random Reverse Reachable (RRR) Set
  - Intermediate results of visited vertices
- Sampling
  - Repeat many ($\theta \approx 10^5$) times
- Selection
  - Count the frequency of each vertex
  - Select the most frequent vertex (u^*)
  - "Pruning" RRRs that contain u^*
  - Reconstruct $\hat{h}$ and repeat k times

Computation and Memory challenging!



IC diffusion: Each activated vertex has one chance (at time step t_i) to activate its neighbors with some probabilities (edge weights)

sampling

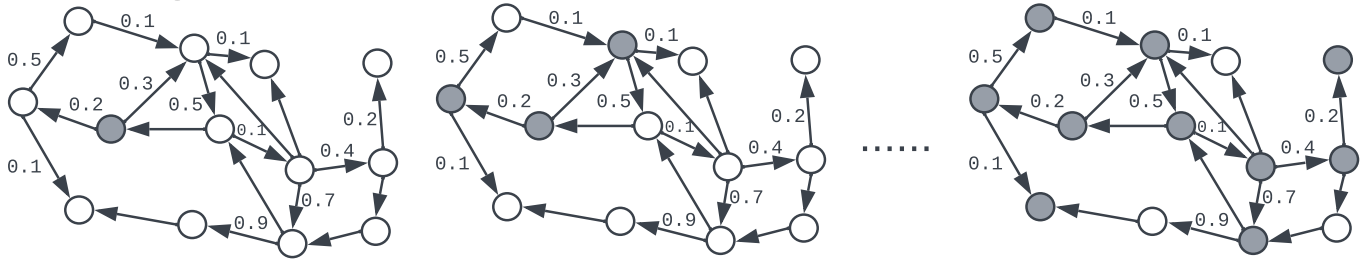
IC Samples				
R_1	v_1	v_3		
R_2	v_4	v_5		
R_3	v_2	v_3	v_4	
R_4	v_3			
R_5	v_1			
R_6	v_1	v_2	v_3	v_5
R_7	v_3	v_4		

Background

➢ Influence Maximization via Martingales Algorithm (IMM)

- Diffusion Models
  - Independent Cascade (IC)
- Random Reverse Reachable (RRR) Set
  - Intermediate results of visited vertices
- Sampling
  - Repeat many ($\theta \approx 10^5$) times
- Selection
  - Count the frequency of each vertex
  - Select the most frequent vertex (u^*)
  - "Pruning" RRRs that contain u^*
  - Reconstruct $\hat{h}$ and repeat k times

Computation and Memory challenging!



IC diffusion: Each activated vertex has one chance (at time step t_i) to activate its neighbors with some probabilities (edge weights)

sampling

IC Samples				
R_1	v_1	v_3		
R_2	v_4	v_5		
R_3	v_2	v_3	v_4	
R_4	v_3			
R_5	v_1			
R_6	v_1	v_2	v_3	v_5
R_7	v_3	v_4		

counting

$\hat{h}$	
v_1	3
v_2	2
v_3	5
v_4	3
v_5	2

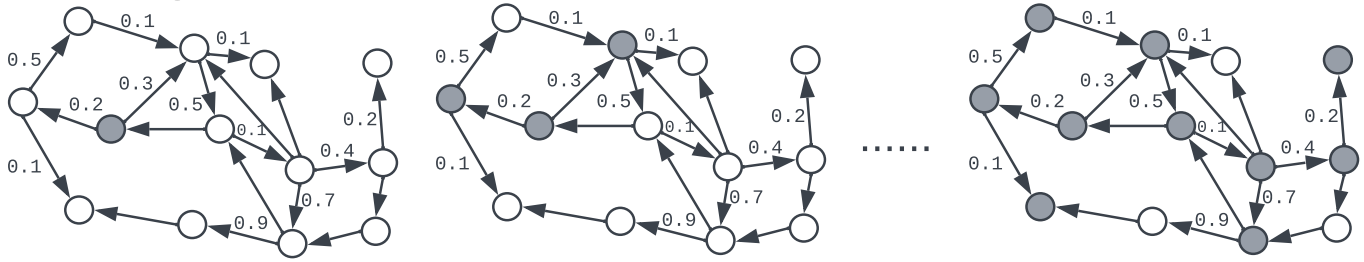
$u^* = v_3$

Background

➢ Influence Maximization via Martingales Algorithm (IMM)

- Diffusion Models
  - Independent Cascade (IC)
- Random Reverse Reachable (RRR) Set
  - Intermediate results of visited vertices
- Sampling
  - Repeat many ($\theta \approx 10^5$) times
- Selection
  - Count the frequency of each vertex
  - Select the most frequent vertex (u^*)
  - "Pruning" RRRs that contain u^*
  - Reconstruct $\hat{h}$ and repeat k times

Computation and Memory challenging!



IC diffusion: Each activated vertex has one chance (at time step t_i) to activate its neighbors with some probabilities (edge weights)

sampling

IC Samples				
R_1	v_1	v_3		
R_2	v_4	v_5		
R_3	v_2	v_3	v_4	
R_4	v_3			
R_5	v_1			
R_6	v_1	v_2	v_3	v_5
R_7	v_3	v_4		

counting

$\hat{h}$	
v_1	3
v_2	2
v_3	5
v_4	3
v_5	2

$u^* = v_3$

pruning

IC Samples				
R_1	v_1	v_3		
R_2	v_4	v_5		
R_3	v_2	v_3	v_4	
R_4	v_3			
R_5	v_1			
R_6	v_1	v_2	v_3	v_5
R_7	v_3	v_4		

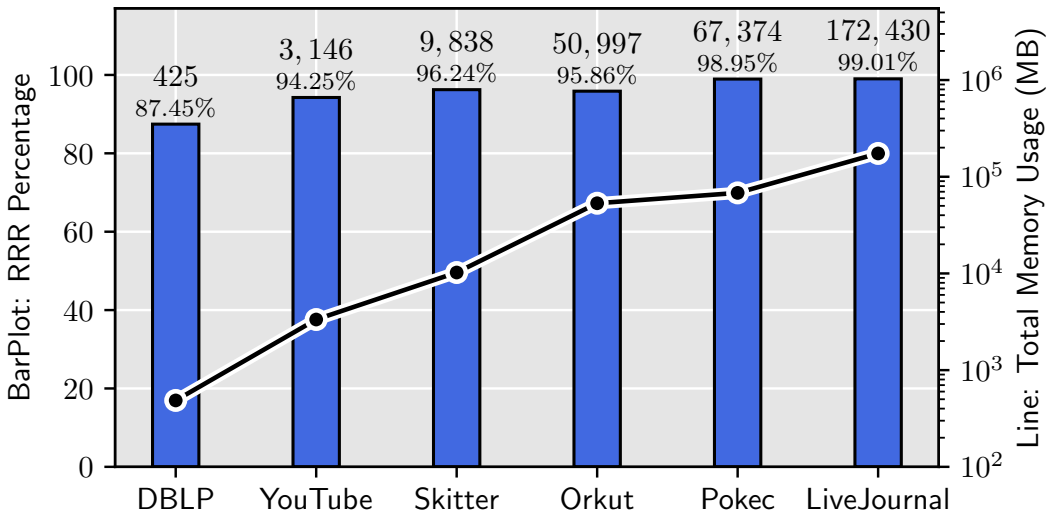
Motivation

➢ Address Computation Challenges

- Parallelization on shared and distributed-memory systems
  - Ripples (Minutoli *et al.*)
- Accelerated by GPUs
  - cuRipples (Minutoli *et al.*),
  - gIM (Shahrouz *et al.*)

➢ Memory Challenges Unaddressed

- Huge memory inflation (30x~165x) during computation
- Same for many stochastic graph applications



Memory usage of Ripples to store intermediate RRR sets

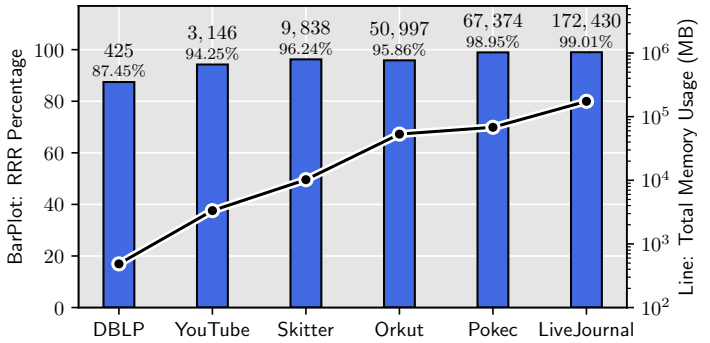
➢ Research Questions

- Can we leverage compression techniques?
- Can we compute with compressed data to preserve memory saving?
- If yes, which compression algorithms should we use?
- What are the benefits to reduce memory usage?

HBMax: Select Suitable Compression

➢ Profiling Memory Usage

- Huge memory inflation (30x~165x) during computation



Memory Inflation on six large social network graphs

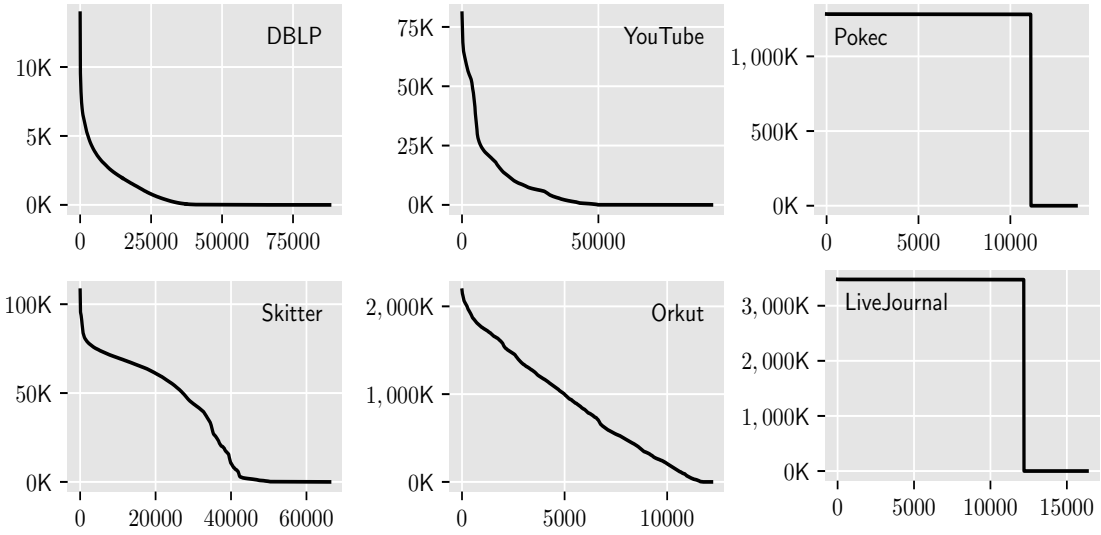
➢ Characterize Intermediate RRR Sets

- RRR sets have various distributions
  - Highly-skewed, Linear-decay, Flat-headed
- RRR sets have various densities
  - Sparse, Dense

Graph	Skewness	Density
DBLP	11.46	0.26%
YouTube	9.01	0.63%
Skitter	5.38	2.03%
Orkut	0.75	27.7%
Pokec	-1.43	66.0%
LiveJournal	-0.99	53.3%

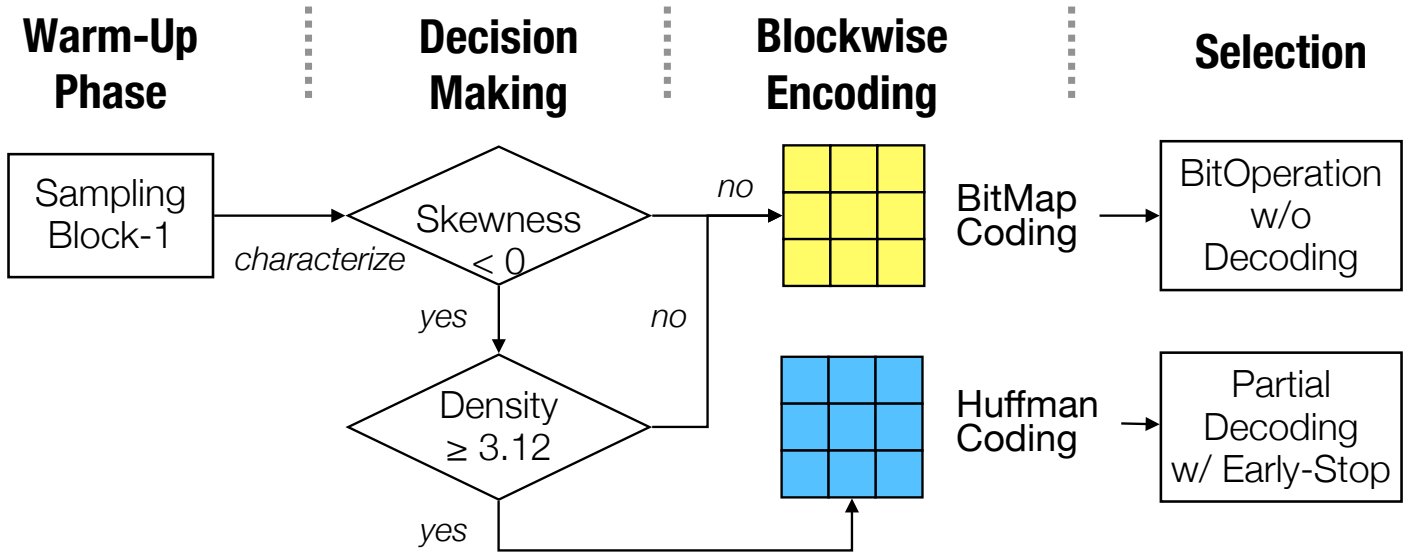
Huffman Coding (H)

bitmap Coding (B)



Characterization of intermediate RRR sets distributions

HBMAX: Overview of Workflow



Blockwise Sampling

Release memory after compressed
Avoid peak memory usage

➢ Warm-up

- Characterize skewness and density
- Huffman Coding (H)
- bitmap Coding (B)

➢ Sampling-and-Encoding

- Parallelize by OpenMP
- Consider NUMA effects
- Swap by data localities (H)

➢ Selection

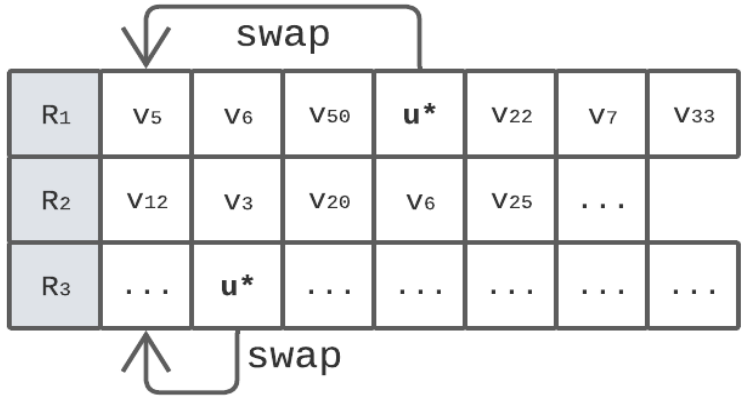
- Partially-decode and Early-Stop (H)
- Bit-Operations w/o Decoding (B)

HBMAX: Selection w/ Huffman Encoded Data (H)

➢ Partial-Decoding w/ Early-Stop

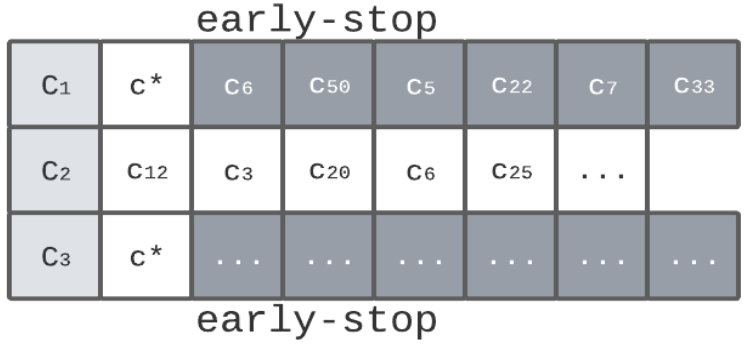
- Leverage the data locality of skewed distributions
- Swap during encoding to enable Early-Stop in partially-decoding

Encode RRRs



Swap u^* to the front and encoding

Query RRRs



Early-stop after decompress c^* and u^* is found

HBMAX: Selection w/ bitmap Encoded Data (B)

➢ Query the encoded RRRs

- Directly compute with encoded data
- Use POPCOUNT to construct frequency table
- Use v_i AND (v_i XOR u^*) to remove RRRs

POPCOUNT

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	$\hat{h}$
v_1	1	0	0	0	1	1	0	3
v_2	0	0	1	0	0	1	0	2
u^*	1	0	1	1	0	1	1	5
v_4	0	1	1	0	0	0	1	3
v_5	0	1	0	0	0	1	0	2

AND, XOR

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	$\hat{h}$
v_1	1	0	0	0	1	1	0	3
v_2	0	0	1	0	0	1	0	2
u^*	1	0	1	1	0	1	1	5
v_4	0	1	1	0	0	0	1	3
v_5	0	1	0	0	0	1	0	2

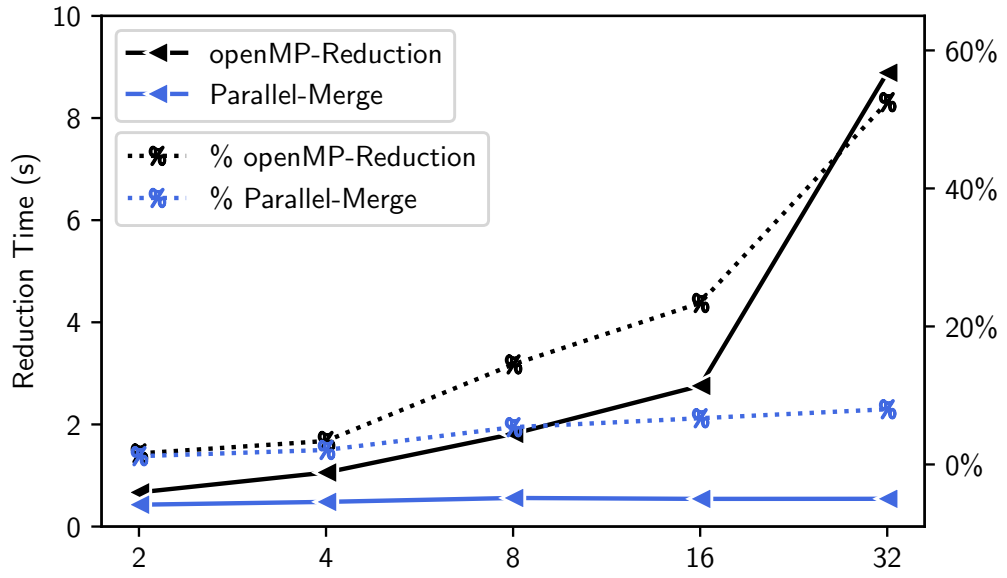
	R_1	R_2	R_3	R_4	R_5	R_6	R_7	$\hat{h}$
v_1	**0**	0	0	0	1	**0**	0	3
v_2	0	0	**0**	0	0	**0**	0	2
u^*	1	0	1	1	0	1	1	5
v_4	0	1	**0**	0	0	0	**0**	3
v_5	0	1	0	0	0	**0**	0	2

Selection with bitmap data. Columns represent RRRs

HBMAX: Selection u* w/ Parallel Merge

➢ Parallel Merge

- Select global maximum from local maxima
- Avoid parallel reduction to compute global frequency table
- Enhance scalability



Scalability of Parallel Merge and OpenMP reduction

For Skitter graph (n=1.6M, k=100)
Directly use OpenMP Reduce needs to reduce $1.6M \times 100 \times 4 \approx 650$ MB data;

Our parallel Merge on 32 threads needs to reduce $32 \times 100 \times 4 \approx 12.5$ KB data.

Evaluation: Setup

➢ **Platform**

- One Regular Memory (RM) node from Bridges-2
  - 2 AMD EPYC 7742 CPUs
  - 256 GB RAM
- Compiled by GCC 8.3.1

➢ **Datasets**

Network	#Vertices	#Edges	Avg Deg	Max Deg
DBLP	317,080	1,049,866	3.31	306
YouTube	1,134,890	2,987,624	2.63	28,576
Skitter	1,696,415	11,095,298	6.54	35,387
Orkut	3,072,441	117,185,083	76.28	33,313
Pokec	1,632,803	30,622,564	37.51	20,518
LiveJournal	4,847,571	68,993,773	28.47	22,889
arabic-2005	22,744,080	639,999,458	28.14	575,618
twitter7	41,652,230	1,468,365,182	35.25	770,155

Evaluation: Reduced Memory Usage

- Reduction of memory footprint
  - Up to 82.1% (LiveJournal) w/ bitmap coding
  - Ripples cannot process the 2 largest graphs (OOM)

Memory footprint (in MB) and reduction ratio (in parenthesis)

Graph	DBLP	YouTube	Skitter	Orkut	Pokec	Journal	Arabic	Twitter7
Ripples	424 (1.00)	3,143 (1.00)	9,838 (1.00)	46,506 (1.00)	55,682 (1.00)	163,745 (1.00)	348,606(1.00)	1,193,006(1.00)
Huffmax	316 (1.34)	1,722 (1.83)	5,293 (1.86)	30,130 (1.54)	-	-	-	-
Bitmax	-	-	-	-	10,661 (5.22)	29,329 (5.58)	81,504(4.28)	200,250(5.96)

Time-to-solution on tested graphs. Average time shortened is 14.5% on skew-distributed graphs.

Scalability of Parallel Merge and OpenMP reduction

Evaluation: Faster Time-to-solution

- Reduction of Time-to-Solution
  - Average 6.3% speedup
  - Reduced page fault

Reduced page fault in sampling steps

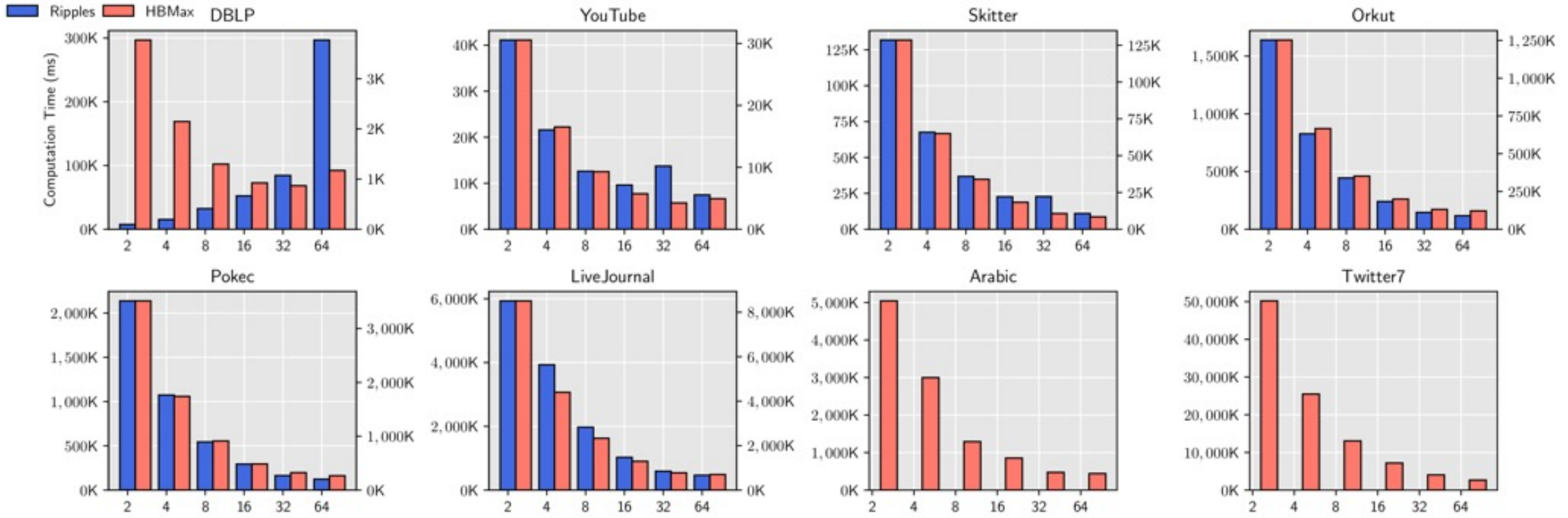
Graph	HBMax	Ripples	Reduced Page Faults
DBLP	0.45	0.50	29.5%
YouTube	4.30	5.78	24.3%
Skitter	13.43	17.56	10.5%
Orkut	196.02	245.25	25.2%
Pokec	196.79	257.96	31.0%
LiveJournal	612.60	761.59	42.8%
arabic-2005	1297.60	NA	NA
twitter7	11219.10	NA	NA

Time-to-solution (in seconds) and overhead ratio (in parenthesis)

Graph	DBLP	YouTube	Skitter	Orkut	Pokec	Journal	Arabic	Twitter7
Ripples	0.95 (1.0)	6.95 (1.0)	20.46 (1.0)	249.35 (1.0)	262.66 (1.0)	775.58 (1.0)	NA	NA
Huffmax	1.10 (1.16)	6.31 (0.91)	17.93 (0.88)	235.14 (0.94)	-	-	-	-
Bitmax	-	-	-	-	222.63 (0.85)	692.70 (0.89)	1,608.48	12,098.30

Evaluation: Scalability

- Strong scalability
  - 12.98x speedup on 64 cores
  - HBMax scales better on highly-skewed graphs (DBLP, YouTube)



Conclusion and Future Work

- A Compress-to-Compute approach
  - Huffman or bitmap Coding
  - Query by partially-decoding or no-decoding

- Evaluation on real-world large graphs
  - Reduce memory usage up to 82.1%
  - Average speedup is 6.3%
  - Strong scalability with 12.98x speedup on 64 cores

- Future Work
  - Extend to distributed-memory platforms
  - Leverage GPU accelerators
  - Explore compression techniques on broader graph algorithms

Thank You!

Any questions are welcome!

Contact Dingwen Tao: (ditao@iu.edu)
Xinyu Chen: (Xinyu.chen1@wsu.edu)