AN ORDER SAMPLING PROCESSING-IN-MEMORY ARCHITECTURE FOR APPROXIMATE GRAPH PATTERN MINING

Ziqian Wan (reporter)
Guohao Dai
Yun Joon Soh
Jishen Zhao
Yu Wang

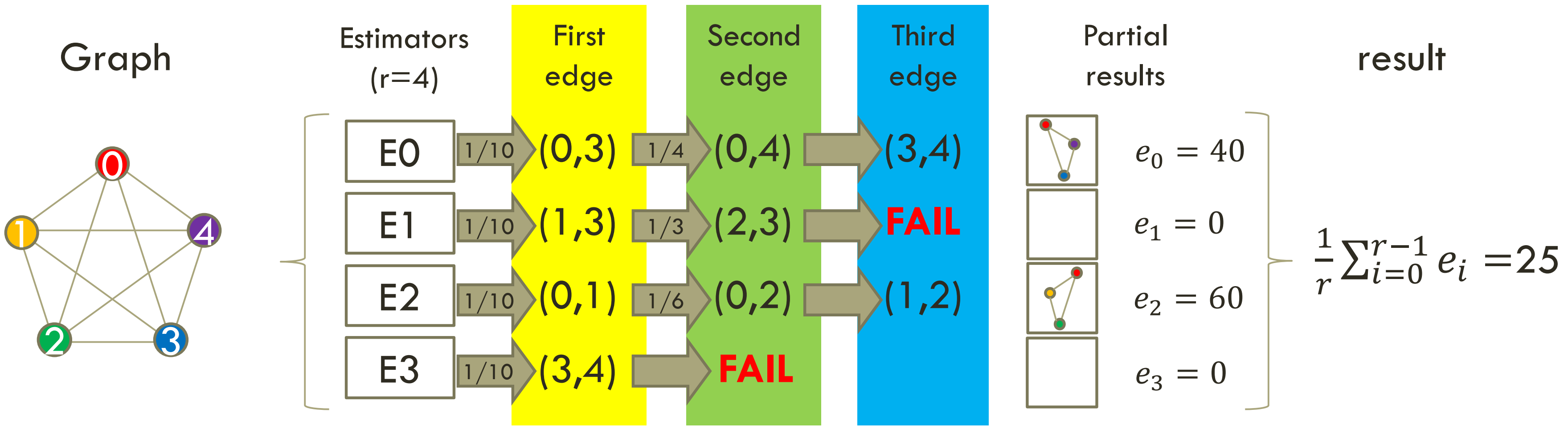
CONTENTS

BACKGROUND & RELATED WORK

GRAPH PATTERN MINING

- **Task:** discover structural patterns in a graph
- **Applications:** detecting similarity between graphlets in social networking; counting pattern frequencies to do credit card fraud detection
- **Challenges:**
  - Lack of scalability
  - Inefficient

CURRENT SOLUTION: ASAP ALGORITHM



edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

PROBLEMS AND SOLUTIONS

	ASAP system	Our Design
Algorithm	Expensive searching operation Inefficient sampling order	Sorting and Dictionaries Order-Aware Sampling
Architecture	Lack of parallelism	Processing-in-Memory

◆All these three designs, lead to 97× performance improvement compared with the ASAP system.

ALGORITHM OPTIMIZATIONS

SORTING & DICTIONARIES

SORTING AND DICTIONARIES

- In the original algorithm, the edge stream is in random order, thus the search operation must traverse the whole edges list to find a certain edge, which is very expensive.
- If we sort the edge stream before executing the algorithm, we can generate several dictionaries which indicates the neighboring and location information of edges in the edge stream. With the help of these dictionaries, we can greatly reduce the complexity of search operations.

DICTIONARIES

- dict_loc: store the location information of edges in the memory
- dict_node: store the neighboring information of nodes
- dict_edge: store edges that comes out of a node

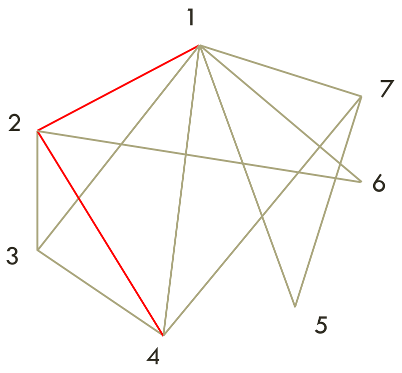
CONDITIONAL SAMPLE EDGE

L

original sample range

#edge	0	1	2	3	4	5	6	7	8	9	10	11
first-node	1	1	1	1	1	1	2	2	2	3	4	5
second-node	2	3	4	5	6	7	3	4	6	4	7	7

shortened sample range



edge_list: [[1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [2, 3], [2, 4], [2, 6], [3, 4], [4, 7], [5, 7]]

dict_edge: {1: [0, 1, 2, 3, 4, 5], 2: [0, 6, 7, 8], 3: [1, 6, 9], 4: [2, 7, 9, 10], 5: [3, 11], 6: [4, 8], 7: [5, 10, 11]}

filter / deduplicate → neighbor edges: [1 ,2 ,3 ,4 ,5 ,6 ,7 ,8]

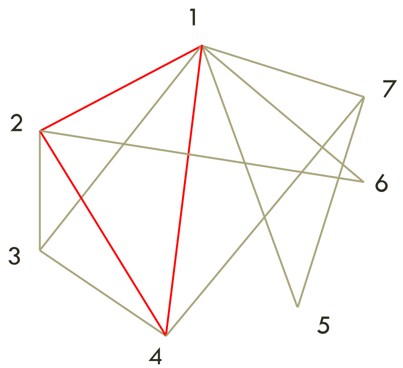
CONDITIONAL CLOSE

L

original search range

#edge	0	1	2	3	4	5	6	7	8	9	10	11
first-node	1	1	1	1	1	1	2	2	2	3	4	5
second-node	2	3	4	5	6	7	3	4	6	4	7	7

shortened search range



edge_list: [[1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [2, 3], [2, 4], [2, 6], [3, 4], [4, 7], [5, 7]]

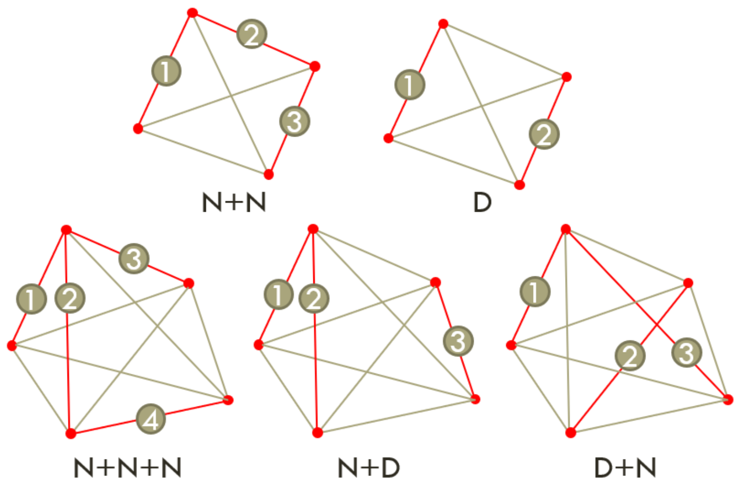
dict_node: {1: [2, 3, 4, 5, 6, 7], 2: [3, 4, 6], 3: [4], 4: [7], 5: [7]}

dict_loc: {1: [0, 5], 2: [6, 8], 3: [9, 9], 4: [10, 10], 5: [11, 11]}

ORDER-AWARE SAMPLING

DIFFERENT SAMPLING ORDERS

- In the ASAP system, only the **neighborhood sampling** is used.
- However, **disjoint sampling** has great potential to outperform neighborhood sampling.
- For k-clique, we can extend from (k-2)-clique by sampling a disjoint edge, or extend from (k-1)-clique by sampling a neighboring edge.



FLOW OF ORDER-AWARE SAMPLING

- Give out all possible sampling orders according to the pattern.
- Estimate the execution time of an estimator for each sampling order by running 10K estimators.
- Calculate the number of estimators needed for an (ϵ, δ)- approximation for each sampling order.
- For each sampling order, calculate the execution time by multiplying the time of each estimator with the number of estimators.
- Choose the sampling order that costs least time as our sampling order.

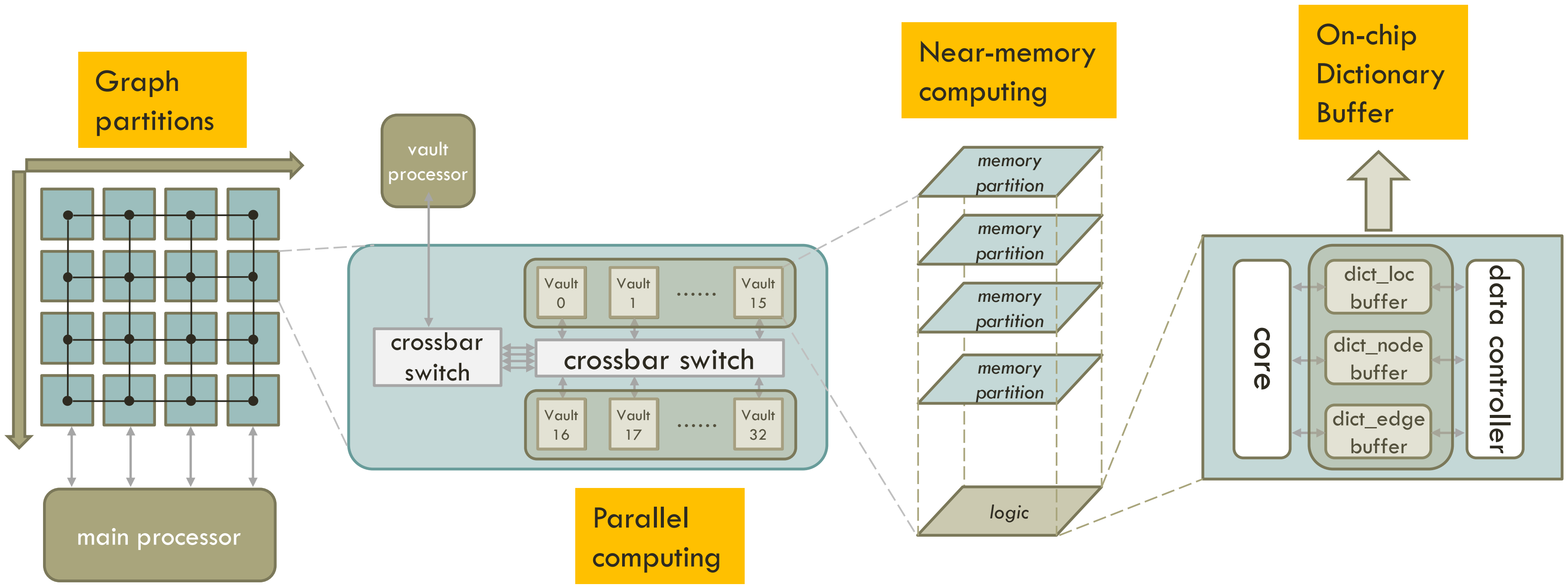
ARCHITECTURE DESIGN

PIM ARCHITECTURE

Processing in memory (PIM) architectures is suitable for ASAP implementation for 3 reasons:

- PIM provides *memory-capacity-proportional bandwidth*, thus tackle the bottleneck of limited memory bandwidth
- We can integrate an On-chip Dictionary Buffer on logic layer to further reduce data access
- The HMC array structure provides large parallelism, which is very useful for the ASAP algorithm which features parallel computing

ARCHITECTURE DESIGN



WORK FLOW

- Inputting. Input the graph.
- Partitioning. Partition the graph into 16 subgraphs.
- Preprocessing. Sort the stream and generate dictionaries.
- Selecting. Select the most efficient sampling order and compute the number of estimators.
- Sampling. Execute estimators and output partial results.
- Gathering. Gather partial results and compute the final result.
- Outputting. Output the final result.

EXPERIMENTAL EVALUATION

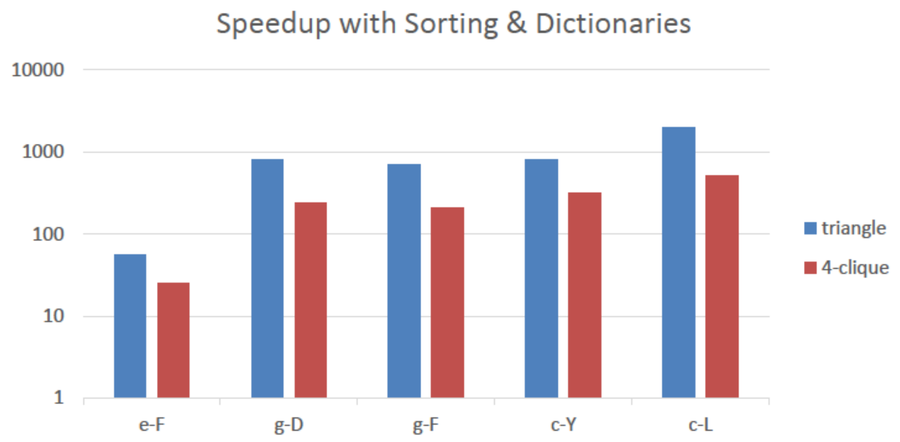
EXPERIMENTAL SETUP

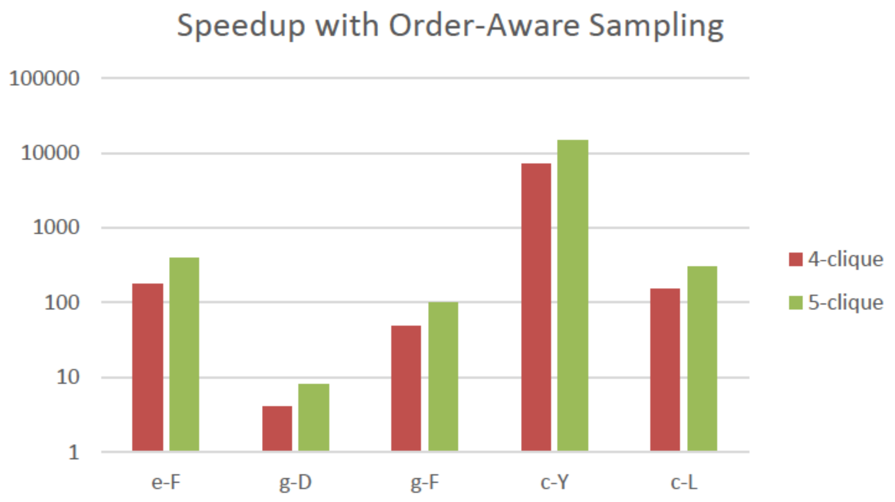
- Simulator: cycle accurate in-house simulator
- Dataset: Stanford Large Network Dataset
- Workloads: triangle counting, four & five-clique mining

Table 2: Properties of Graphs

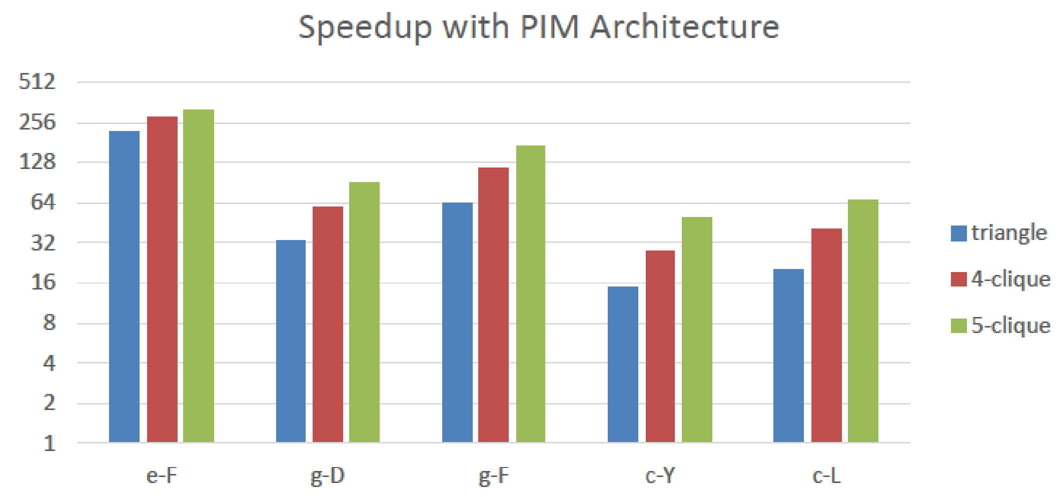
Graph	# Edges	# Vertices	Max Degree
ego-Facebook (e-F) [8]	88,234	4,039	1,045
gemsec-Deezer-HR (g-D) [11]	498,202	54,573	420
gemsec-Facebook-Artist (g-F) [11]	819,306	50,515	1,469
com-Youtube (c-Y) [18]	2,987,624	1,134,890	28,754
com-LiveJournal (c-L) [18]	34,681,189	3,997,962	14,815

BENEFITS OF ALGORITHM OPTIMIZATIONS

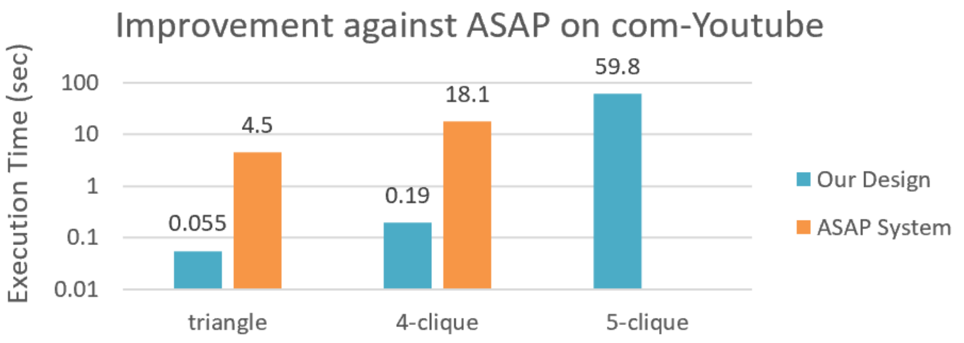


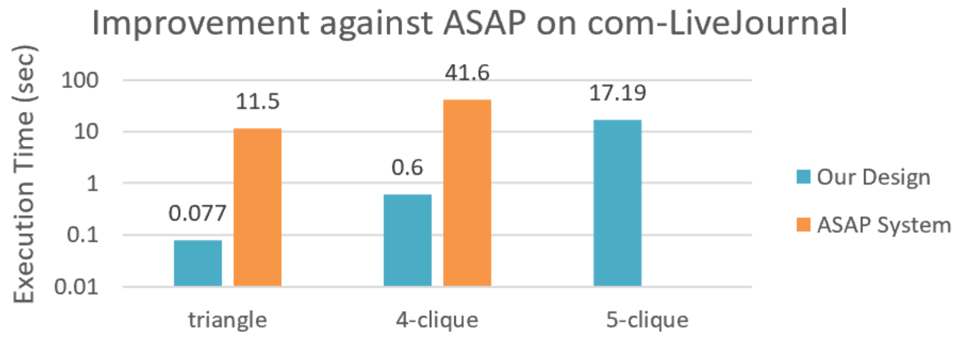


BENEFITS OF PIM ARCHITECTURE



OVERALL PERFORMANCE





THANKS