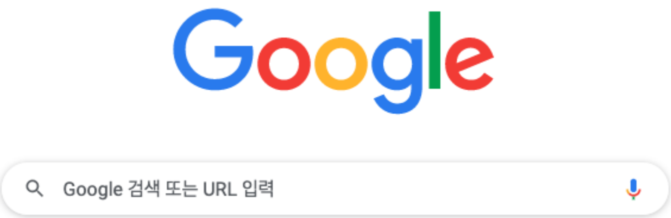
CR²: Community-aware Compressed Regular Representation for Graph Processing on a GPU

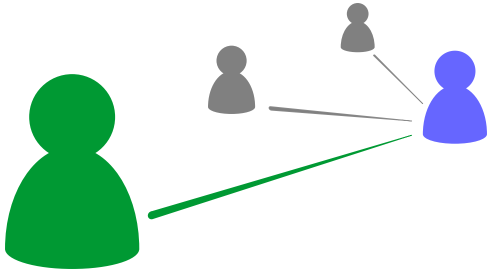
Shinnung Jeong[1], Sungjun Cho[2], Yongwoo Lee[1], Hyunjun Park[1], Seonyeong Heo[3], Gwangsun Kim[2], Yongsok Kim[1], and Hanjun Kim[1]

Yonsei University[1], Postech[2], Kyung Hee University[3]

Graphs are One of the Important Data Structures to Abstract and Analyze Real-World Data

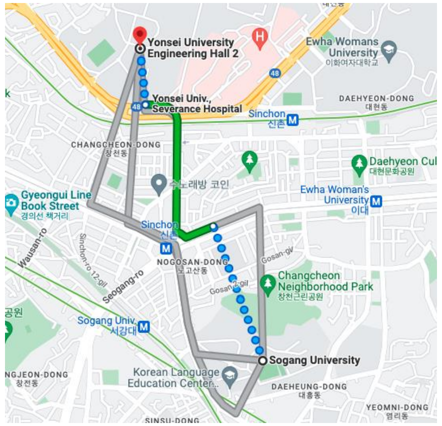


Web Search



Social Network

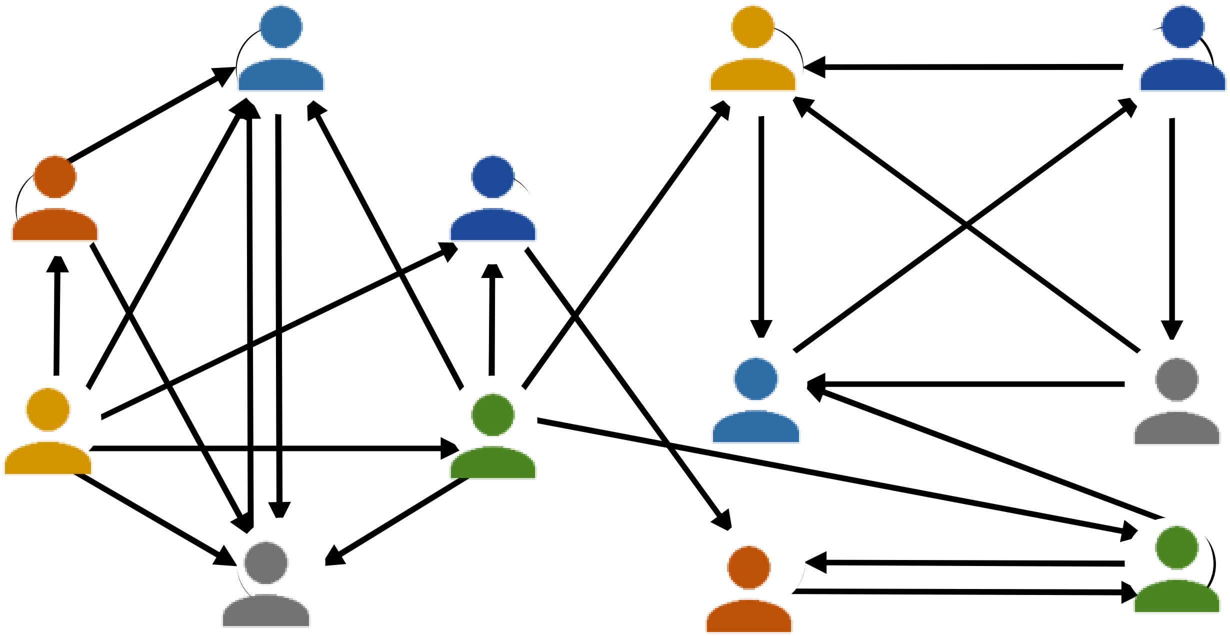
Neuroscience



Path Finding

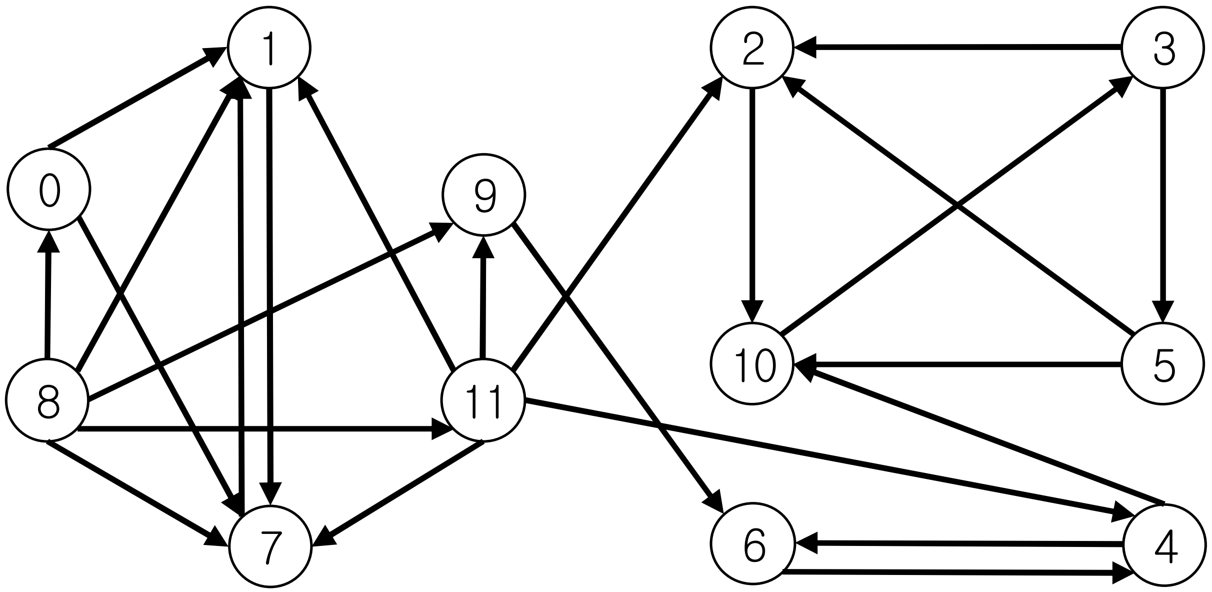
What is Graph?

- An abstract data structure with vertices and their pairs(edges)
- Graph = (Vertex, Edge)



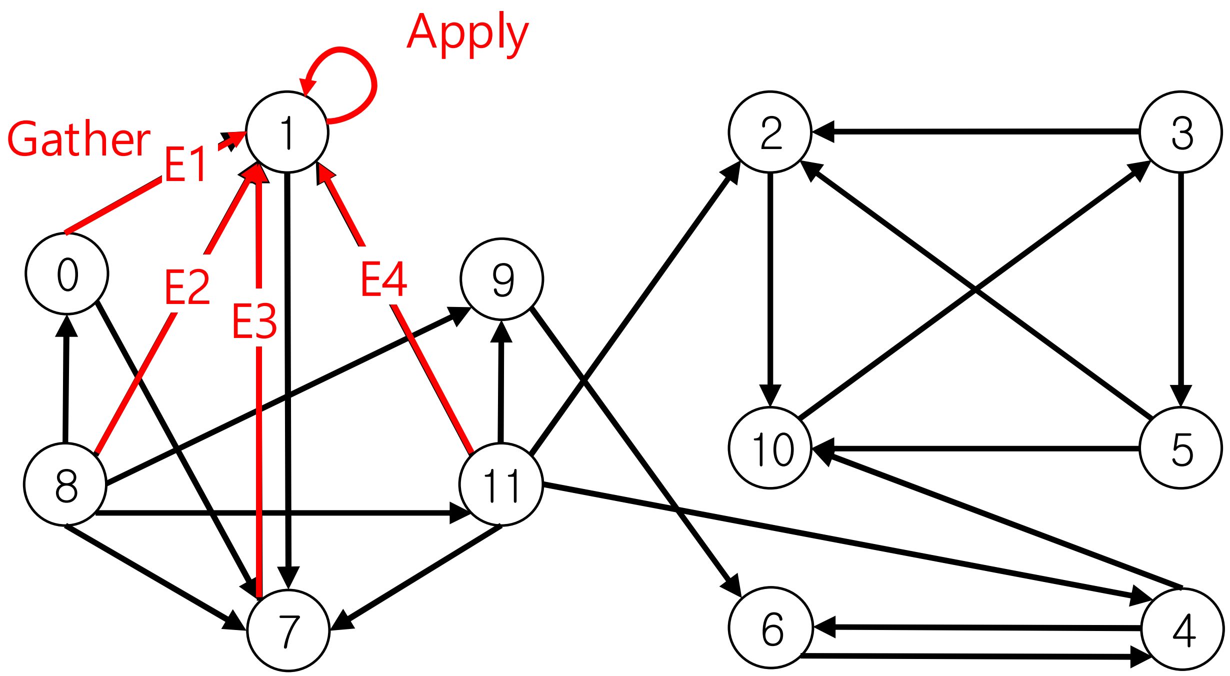
What is Graph?

- An abstract data structure with vertices and their pairs(edges)
- Graph = (Vertex, Edge)



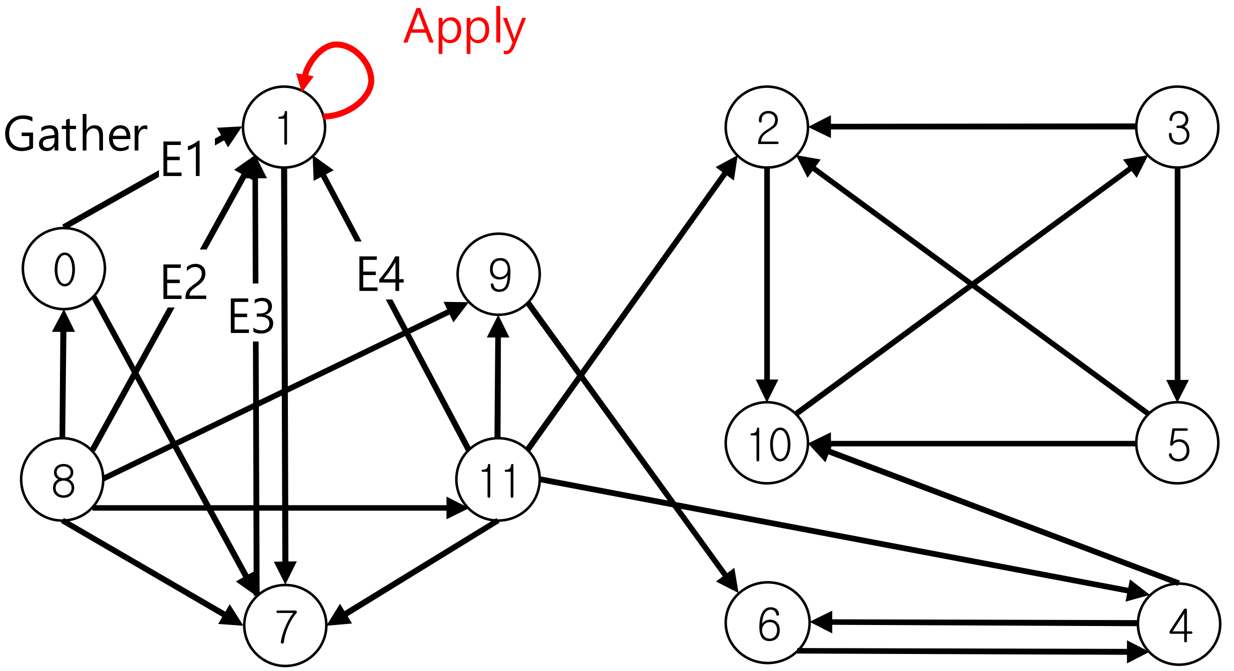
What is the Graph Processing?

- Update vertex data by **gather**ing neighbor edge data
- **Apply** computation on vertex data after finishing gathering
- Perform the same algorithm to each vertex and to each edge



GPU is a Promising Platform for Graph Processing

- Real-world graphs become larger and apply the same algorithms to each vertex and edges
- GPU is designed for Single Instruction Multiple Data(SIMD) processing



Map to each thread

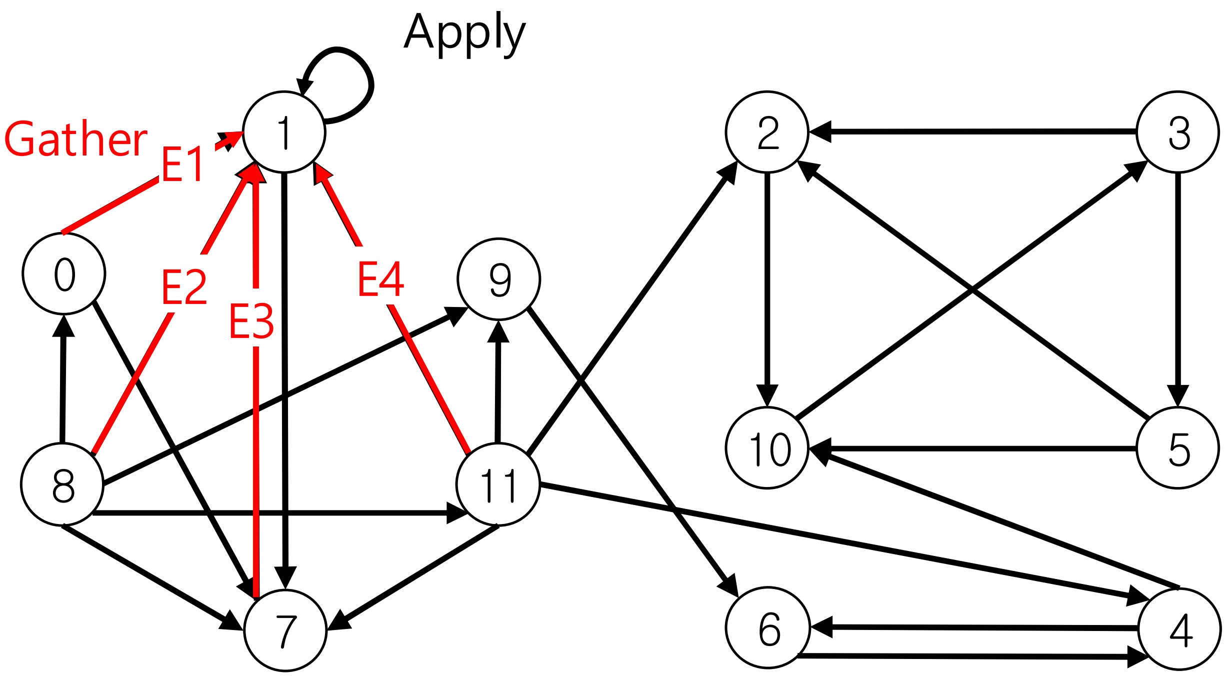
Apply Processing on GPU

T0	T1	T2	T3	T4	T5	T6	T7
V0	V1	V2	V3	V4	V5	V6	V7
V8	V9	V10	V11				

Time

However, Graph Processing on GPU is Still Challenging!

- The characteristics of real-world graph are not fit for GPU, such as skewness and sparsity
- Cause irregular distribution inducing **workload imbalance** and reducing resource utilization



Gather Processing on GPU

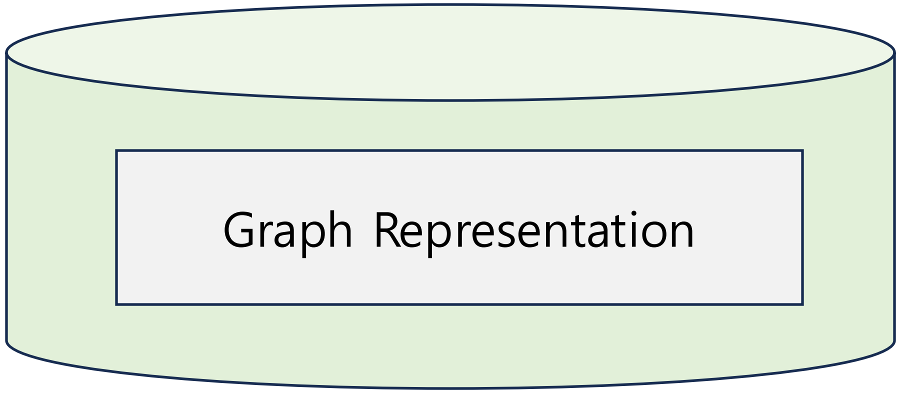
T0	T1	T2	T3	T4	T5	T6	T7
V0	V1	V2	V3	V4	V5	V6	V7
E0	E1	E5	E8	E9	E11	E12	E14
	E2	E6		E10		E13	E15
	E3	E7					E16
	E4						

Time

Map each vertex and its neighboring edges to each thread

Changing Graph Representation for GPU can be a Solution

- The graph representation defines how to store a given graph in GPU memory
- Since the graph is large, the graph representation affect to memory usage and perf



Read graph data from memory

Gather Processing on GPU

T0	T1	T2	T3	T4	T5	T6	T7
V0	V1	V2	V3	V4	V5	V6	V7
E0	E1	E5	E8	E9	E11	E12	E14
	E2	E6		E10		E13	E15
	E3	E7					E16
	E4						

Time

Changing Graph Representation for GPU can be a Solution

- CSR can reduce memory usage and enable neighbor list friendly access

dest id	0	1	2	3	4	5	6	7	8	9	10	11	end
offset	0	1	5	8	9	11	12	14	18	18	20	23	24

edge list	8	0	7	8	11	3	5	11	10	6	11	3	4	9	0	1	8	11	8	11	2	4	5	8

Compressed Sparse Row (CSR) [1]

Neighbor list access friendly representation

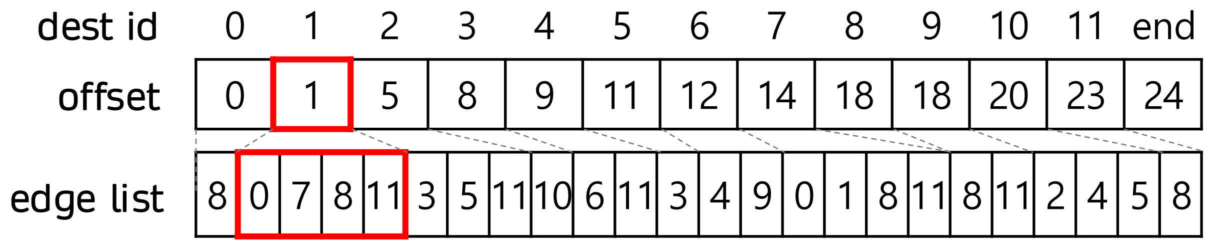
Gather Processing on GPU

T0	T1	T2	T3	T4	T5	T6	T7
V0	V1	V2	V3	V4	V5	V6	V7
E0	E1	E5	E8	E9	E11	E12	E14
	E2	E6		E10		E13	E15
	E3	E7					E16
	E4						

Time

Changing Graph Representation for GPU can be a Solution

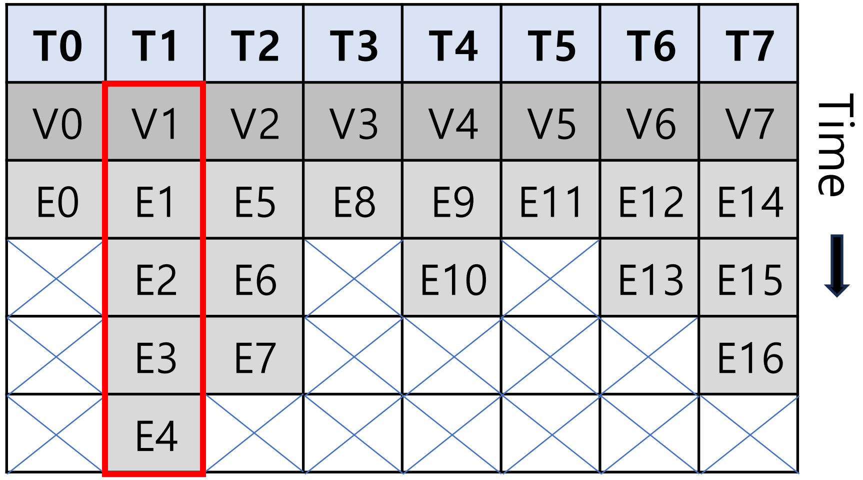
- CSR can reduce memory usage and enable neighbor list friendly access
- Graph representation affects **workload imbalance, locality, and memory access pattern**



Compressed Sparse Row (CSR) [1]

Neighbor list access friendly representation

Gather Processing on GPU



Existing Graph Representation for GPU: G-shards

Shard 0

src id	0	3	3	0	1	2
dest id	1	2	4	7	7	10
src value	v_1	v_2	v_4	v_7	v_7	v_{10}

Shard 1

src id	7	4	6	5	4	5
dest id	1	2	5	6	10	10
src value	v_1	v_2	v_5	v_6	v_{10}	v_{10}

Shard 3

src id	8	8	11	11	10	11	9	8	11	8	11	8
dest id	0	1	1	2	3	5	6	7	7	9	9	11
src value	v_0	v_1	v_1	v_2	v_3	v_5	v_6	v_7	v_7	v_9	v_9	v_{11}

G-Shards[2]

- Achieve balanced execution
- Improve vertex value access locality
- **Cause more memory usage** than CSR

Rep	**Edge Rep**	**Value**	
		Vertex	Edge
CSR	|E| + |V| + 1	|V|	|E|
G-Shards	2|E|	|V| + |E|	|E|

[2] R. Gupta F. Khorasani et al. 2014. CuSha: Vertex-centric Graph Processing on GPUs. HPCA’14

Existing Graph Representation for GPU: Tigr

dest id	0	1	1	2	2	3	4	5	6	7	7	8	9	10	10	11	end
offset	0	1	3	5	7	8	9	11	12	14	16	18	18	20	22	23	24

edge list	8	0	7	8	11	3	5	11	10	6	11	3	4	9	0	1	8	11	8	11	2	4	5	8

Tigr[3]

- Reduce imbalance by vertex split
- Still remain balancing opertunity
- Cause more memory usage
- Do not address vertex value access locality

Rep	**Edge Rep**	**Value**	
		Vertex	Edge
CSR	|E| + |V| + 1	|V|	|E|
G-Shards	2|E|	|V| + |E|	|E|
Tigr	|E| + (**2 + Ft**)|V|	|V|	|E|

[3] Q. Junqiao N. Sabet et al. 2018. Tigr: Transforming Irregular Graphs for GPU-Friendly Graph Processing. ASPLOS'18

Existing Graph Representation for GPU: Tigr

dest id	0	1	1	2	2	3	4	5	6	7	7	8	9	10	10	11	end
offset	0	1	3	5	7	8	9	11	12	14	16	18	18	20	22	23	24

Existing graph representation for GPUs fails to **balance workload** and to improve **locality** while **reducing memory size**

- Still remain workload imbalance
- Cause more memory usage
- Do not address vertex value access locality

Tigr	|E| + (**2 + Ft**)|V|	|V|	|E|

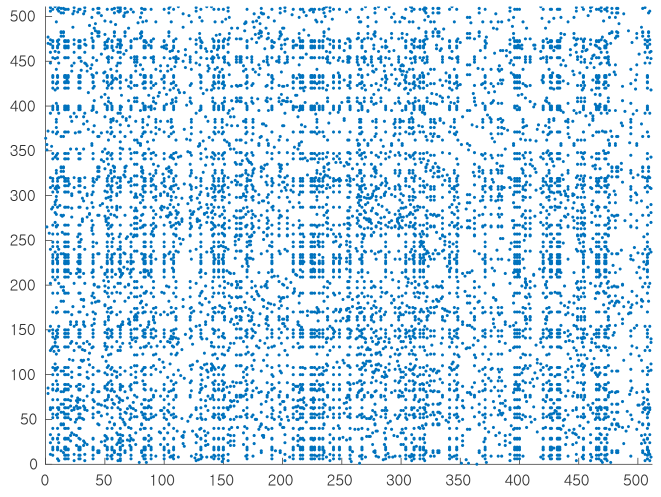
[3] Q. Junqiao N. Sabet et al. 2018. Tigr: Transforming Irregular Graphs for GPU-Friendly Graph Processing. ASPLOS'18

Need a New Graph Representation!

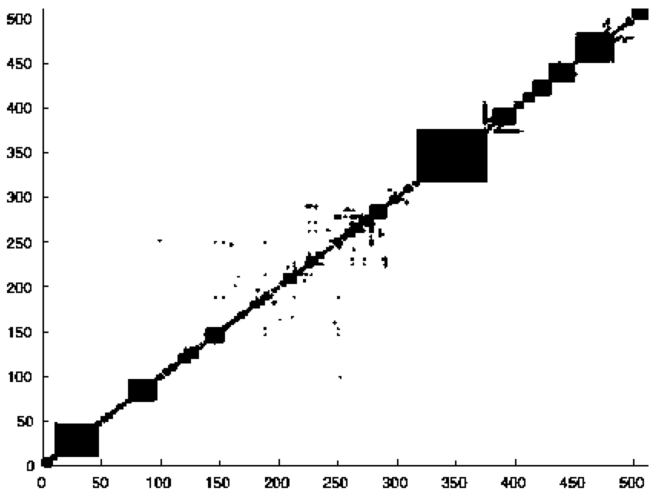
- Leverage the **high locality**
- **Minimize memory usage** to support larger graphs
- **Align with GPU architecture** to maximize performance

Observation 1: The Reordered Graph Has a Community on a Diagonal

- Most vertices are densely connected within certain groups
- With reordering algorithms, real-world graphs can be reordered in diagonal lines



Random Order

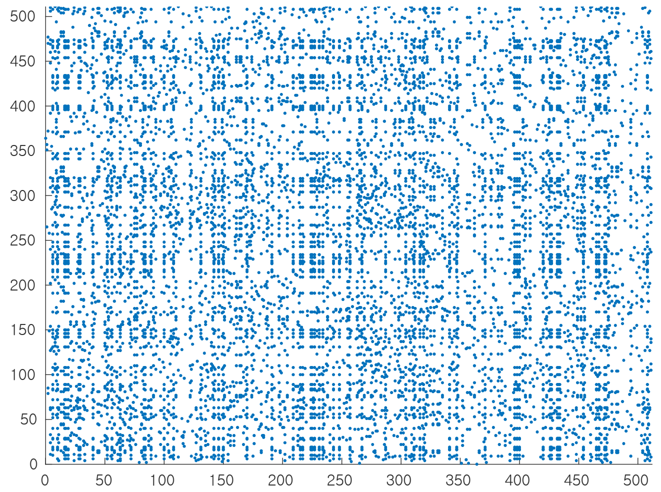


Community-based

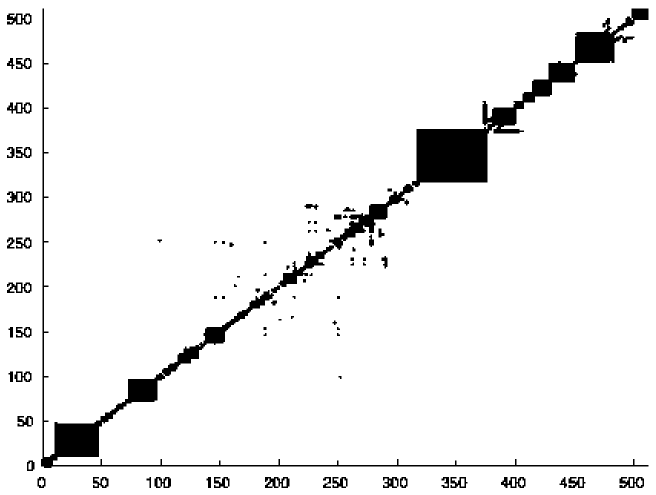
Solution 1: Community-aware Clustered Graph Representation

- Extract communities of real-world graph as clusters
- Execute edges in the same cluster on the same GPU core

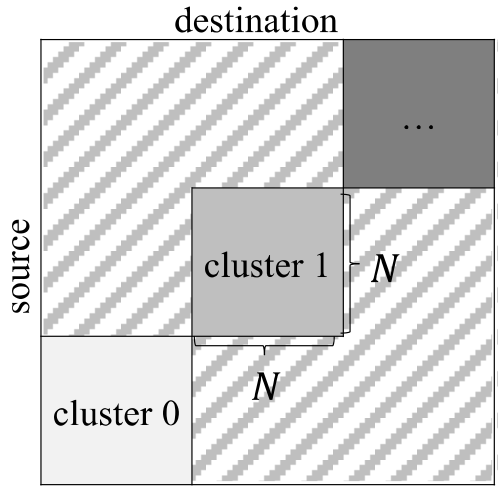
➔ CR^2 can **exploit the vertex locality**



Random Order



Community-based



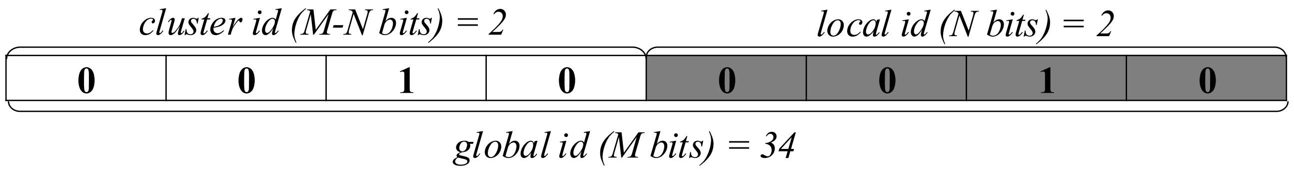
Clustering in CR^2

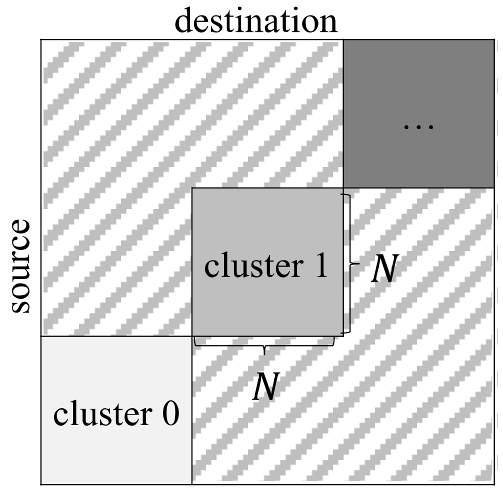
Solution 1: Community-aware Clustered Graph Representation

- Extract communities of real-world graph as clusters
- Execute edges in the same cluster on the same GPU core

➔ CR^2 can **exploit the vertex locality**

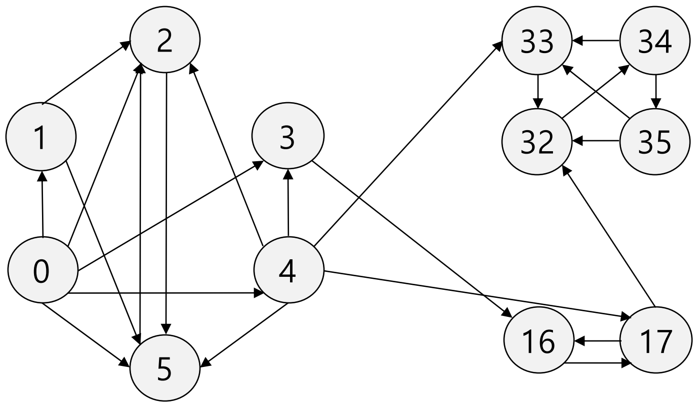
➔ **Reduce memory usage** by representing vertex id with local id





Clustering in CR^2

Solution 1: Community-aware Clustered Graph Representation

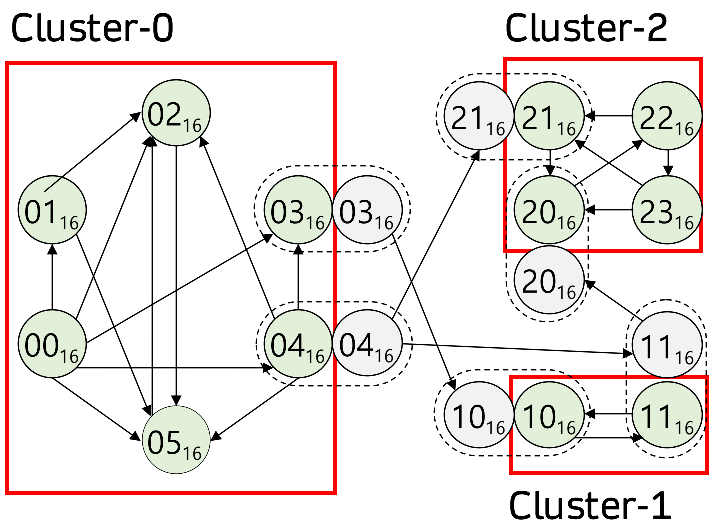


src id	0	1	2	3	4	5	6	...	16	17	18	...	32	33	34	35	36	...
offset$_{8bit}$	0	5	7	8	9	14	15	...	15	16	18	...	18	19	20	22	24	...

edge list$_{8bit}$: | 1 | 2 | 3 | 4 | 5 | 2 | 5 | 5 | 16 | 2 | 3 | 5 | 17 | 33 | 2 | 17 | 16 | 32 | 34 | 32 | 33 | 35 | 32 | 33 |

Make clusters to improve locality

Reduce memory usage by using local id



Cluster-0 graph

src id$_{4bit}$	0 1 2 4 5
offset$_{8bit}$	0 5 7 8 11 12
edge list$_{4bit}$	1 2 3 4 5 2 5 5 2 3 5 2

Cluster-1 graph

src id$_{4bit}$	0 1
offset$_{8bit}$	0 1 2
edge list$_{4bit}$	1 0

Cluster-2 graph

src id$_{4bit}$	0 1 2 3
offset$_{8bit}$	0 1 2 4 6
edge list$_{4bit}$	2 0 1 3 0 1

Inter-cluster graph

src id$_{8bit}$	3 4 17
offset$_{8bit}$	0 1 3 4
edge list$_{8bit}$	16 33 17 32

Seperated subgraph for inter cluster

Solution 1: Community-aware Clustered Graph Representation

Cluster-0 graph

src id$_{4bit}$	0	1	2	4	5							
offset$_{8bit}$	0	5	7	8	11	12						
edge list$_{4bit}$	1	2	3	4	5	2	5	5	2	3	5	2

Cluster-1 graph

src id$_{4bit}$	0	1	
offset$_{8bit}$	0	1	2
edge list$_{4bit}$	1	0	

Cluster-2 graph

src id$_{4bit}$	0	1	2	3		
offset$_{8bit}$	0	1	2	4	6	
edge list$_{4bit}$	2	0	1	3	0	1

Inter-cluster graph

src id$_{8bit}$	3	4	17	
offset$_{8bit}$	0	1	3	4
edge list$_{8bit}$	16	33	17	32

Gather Processing
Execute edges in the same cluster on the same GPU core

Cluster-0

T0	T1	T2	T3	T4	T5	T6	T7
0	1	2	4	5			
0	5	7	8	11			
5	7	8	11	12			
1	2	5	2	2			
2	5		3				
3			5				
4							
5							

Cluster-1

T0	T1	T2	T3	T4	T5	T6	T7
0	1						
0	1						
1	2						
1	0						

Cluster-2

T0	T1	T2	T3	T4	T5	T6	T7
0	1	2	3				
0	1	2	4				
1	2	4	6				
2	0	1	0				
		3	1				

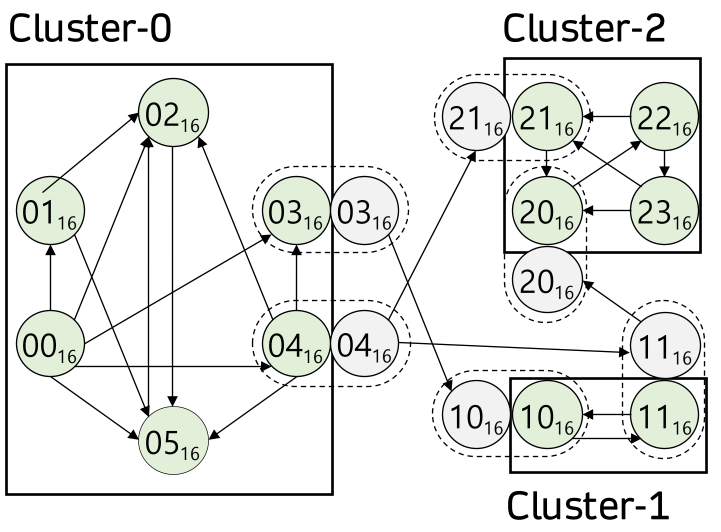
Inter-cluster

T0	T1	T2	T3	T4	T5	T6	T7
3	4	17					
0	1	3					
1	3	4					
16	33	32					
	17						

- Improve locality
- Reduce memory usage with local id
- But, **still suffered from workload imbalance**

Observation 2: Degree Regulation can Remove Offest Array

- Offset array indicates the number of neighboring edges
- If graph representation has a fixed number of edges per vertex,
  - Can remove offset array
  - Can balance workload



Cluster-0 graph

src id$_{4bit}$	0	1	2	4	5							
offset$_{8bit}$	0	5	7	8	11	12						
edge list$_{4bit}$	1	2	3	4	5	2	5	5	2	3	5	2

Cluster-1 graph

src id$_{4bit}$	0	1	
offset$_{8bit}$	0	1	2
edge list$_{4bit}$	1	0	

Cluster-2 graph

src id$_{4bit}$	0	1	2	3		
offset$_{8bit}$	0	1	2	4	6	
edge list$_{4bit}$	2	0	1	3	0	1

Inter-cluster graph

src id$_{8bit}$	3	4	17	
offset$_{8bit}$	0	1	3	4
edge list$_{8bit}$	16	33	17	32

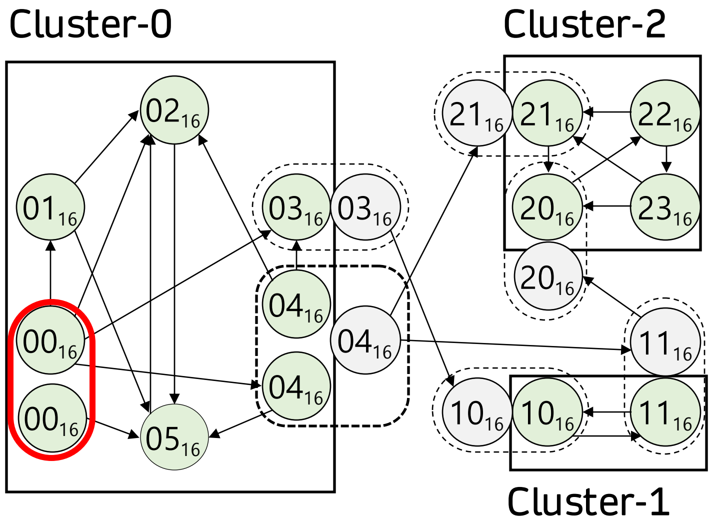
Solution 2: Degree-ordered Subgraph

- Partition the graph into multiple degree-ordered subgraphs in which all the vertices have the **Degree-n regularized number of edges**
- Enable **fine-grained workload balancing** across GPU warps with less memory usage

Degree-1	E0															
Degree-2	E1	E2														
Degree-4	E3	E4	E5	E6												
Degree-8	E7	E8	E9	E10	E11	E12	E13	E14								
Degree-16	E15	E16	E17	E18	E19	E20	E21	E22	E23	E24	E25	E26	E27	E28	E19	E30

⋮

Solution 2: Degree-ordered Subgraph



Cluster-0 graph

src id$_{4bit}$	0	0	1	2	4	4	5					
offset$_{8bit}$	0	4	5	7	8	10	11	12
edge list$_{4bit}$	1	2	3	4	5	2	5	5	2	3	5	2

Cluster-1 graph

src id$_{4bit}$	0	1	
offset$_{8bit}$	0	1	2
edge list$_{4bit}$	1	0	

Cluster-2 graph

src id$_{4bit}$	0	1	2	3		
offset$_{8bit}$	0	1	2	4	6	
edge list$_{4bit}$	2	0	1	3	0	1

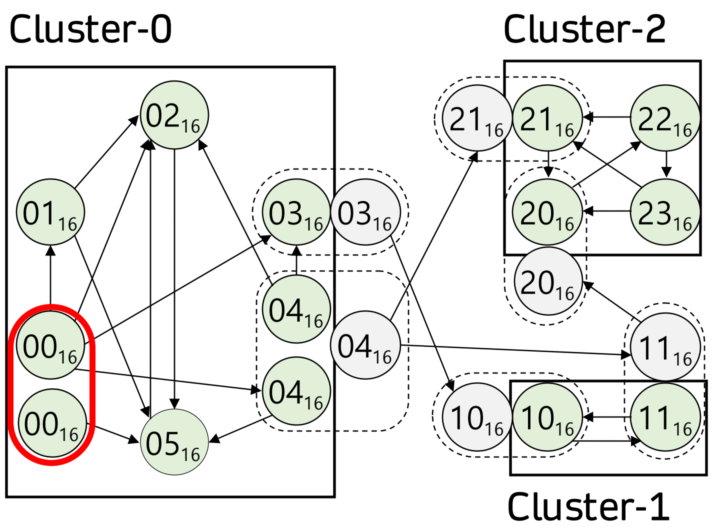
Inter-cluster graph

src id$_{8bit}$	3	4	17	
offset$_{8bit}$	0	1	3	4
edge list$_{8bit}$	16	33	17	32

Vertex-split graph & representation

Split vertex to make it a power-of-two

Solution: CR² Representation



Cluster-0 graph

src id$_{4bit}$	0	0	1	2	4	4	5					
offset$_{8bit}$	0	4	5	7	8	10	11	12				
edge list$_{4bit}$	1	2	3	4	5	2	5	5	2	3	5	2

Cluster-1 graph

src id$_{4bit}$	0	1	
offset$_{8bit}$	0	1	2
edge list$_{4bit}$	1	0	

Cluster-2 graph

src id$_{4bit}$	0	1	2	3		
offset$_{8bit}$	0	1	2	4	6	
edge list$_{4bit}$	2	0	1	3	0	1

Inter-cluster graph

src id$_{8bit}$	3	4	17	
offset$_{8bit}$	0	1	3	4
edge list$_{8bit}$	16	33	17	32

Remove offset array & Generate Degree-n Subgraph

Cluster-0 degree-4

src id$_{4bit}$	0			
edge list$_{4bit}$	1	2	3	4

Cluster-0 degree-2

src id$_{4bit}$	1	4		
edge list$_{4bit}$	2	5	2	3

Cluster-0 degree-1

src id$_{4bit}$	0	2	4	5
edge list$_{4bit}$	5	5	5	2

Cluster-1 degree-1

src id$_{4bit}$	0	1
edge list$_{4bit}$	1	0

Cluster-0 degree-2

src id$_{4bit}$	2	3		
edge list$_{4bit}$	1	3	0	1

Cluster-2 degree-1

src id$_{4bit}$	0	1
edge list$_{4bit}$	2	0

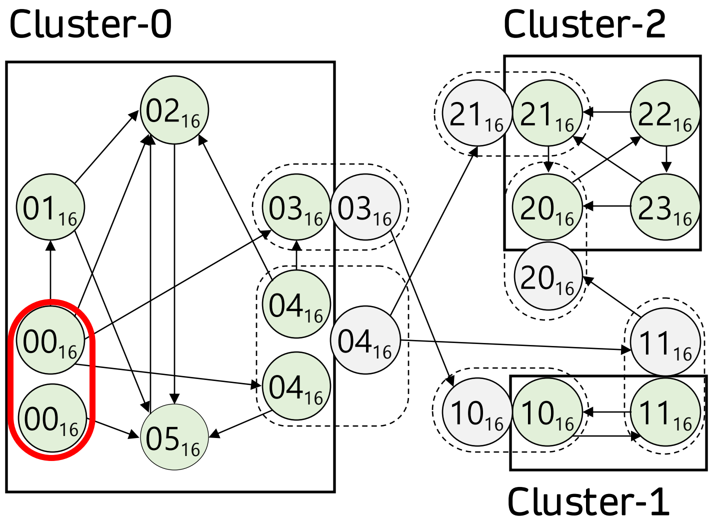
Inter-Cluster degree-1

src id$_{8bit}$	4	
edge list$_{8bit}$	33	17

Inter-Cluster degree-1

src id$_{8bit}$	3	17
edge list$_{8bit}$	16	32

Solution: CR² Representation



Cluster-0 graph

src id$_{4bit}$	0	0	1	2	4	4	5					
offset$_{8bit}$	0	4	5	7	8	10	11	12				
edge list$_{4bit}$	1	2	3	4	5	2	5	5	2	3	5	2

Cluster-1 graph

src id$_{4bit}$	0	1	
offset$_{8bit}$	0	1	2
edge list$_{4bit}$	1	0	

Cluster-2 graph

src id$_{4bit}$	0	1	2	3		
offset$_{8bit}$	0	1	2	4	6	
edge list$_{4bit}$	2	0	1	3	0	1

Inter-cluster graph

src id$_{8bit}$	3	4	17	
offset$_{8bit}$	0	1	3	4
edge list$_{8bit}$	16	33	17	32

Remove offset array & Generate Degree-n Subgraph

Cluster-0 degree-4

src id$_{4bit}$	0			
edge list$_{4bit}$	1	2	3	4

Cluster-0 degree-2

src id$_{4bit}$	1	4		
edge list$_{4bit}$	2	5	2	3

Cluster-0 degree-1

src id$_{4bit}$	0	2	4	5
edge list$_{4bit}$	5	5	5	2

Cluster-1 degree-1

src id$_{4bit}$	0	1
edge list$_{4bit}$	1	0

Cluster-0 degree-2

src id$_{4bit}$	2	3		
edge list$_{4bit}$	1	3	0	1

Cluster-2 degree-1

src id$_{4bit}$	0	1
edge list$_{4bit}$	2	0

Inter-Cluster degree-1

src id$_{8bit}$	4	
edge list$_{8bit}$	33	17

Inter-Cluster degree-1

src id$_{8bit}$	3	17
edge list$_{8bit}$	16	32

- High locality
- Minimize memory usage
- Align with GPU architecture

CORELAB
COMPILER RESEARCH LAB

Solution: CR^2 Representation

Cluster-0 degree-4
src id_{4bit}: 0
edge $list_{4bit}$: 1 | 2 | 3 | 4

Cluster-0 degree-2
src id_{4bit}: 1 | 4
edge $list_{4bit}$: 2 | 5 | 2 | 3

Cluster-0 degree-1
src id_{4bit}: 0 | 2 | 4 | 5
edge $list_{4bit}$: 5 | 5 | 5 | 2

Cluster-1 degree-1
src id_{4bit}: 0 | 1
edge $list_{4bit}$: 1 | 0

Cluster-0 degree-2
src id_{4bit}: 2 | 3
edge $list_{4bit}$: 1 | 3 | 0 | 1

Cluster-2 degree-1
src id_{4bit}: 0 | 1
edge $list_{4bit}$: 2 | 0

Inter-Cluster degree-1
src id_{8bit}: 4
edge $list_{8bit}$: 33 | 17

Inter-Cluster degree-1
src id_{8bit}: 3 | 17
edge $list_{8bit}$: 16 | 32

Gather Processing

Intra-Cluster Execution

Degree-4 Execution

T0	T1	T2	T3	T4	T5	T6	T7
0	0	0	0				
1	2	3	4				

Degree-2 Execution

T0	T1	T2	T3	T4	T5	T6	T7
1	1	4	4	2	2	3	3
2	5	2	3	1	3	0	1

Degree-1 Execution

T0	T1	T2	T3	T4	T5	T6	T7
0	2	4	5	0	1	0	1
5	5	5	2	1	0	2	0

Inter-Cluster Execution

Degree-4 Execution

T0	T1	T2	T3	T4	T5	T6	T7

Degree-2 Execution

T0	T1	T2	T3	T4	T5	T6	T7
4	4						
33	17						

Degree-1 Execution

T0	T1	T2	T3	T4	T5	T6	T7
3	17						
16	32						

Execute the degree-n subgraphs of each cluster simultaneously

Solution: CR^2 Representation

Cluster-0 degree-4
src id$_{4bit}$ | 0 |
edge list$_{4bit}$ | 1 | 2 | 3 | 4 |

Cluster-0 degree-2
src id$_{4bit}$ | 1 | 4 |
edge list$_{4bit}$ | 2 | 5 | 2 | 3 |

Cluster-0 degree-1
src id$_{4bit}$ | 0 | 2 | 4 | 5 |
edge list$_{4bit}$ | 5 | 5 | 5 | 2 |

Cluster-1 degree-1
src id$_{4bit}$ | 0 | 1 |
edge list$_{4bit}$ | 1 | 0 |

Cluster-0 degree-2
src id$_{4bit}$ | 2 | 3 |
edge list$_{4bit}$ | 1 | 3 | 0 | 1 |

Cluster-2 degree-1
src id$_{4bit}$ | 0 | 1 |
edge list$_{4bit}$ | 2 | 0 |

Inter-Cluster degree-1
src id$_{8bit}$ | 4 |
edge list$_{8bit}$ | 33 | 17 |

Inter-Cluster degree-1
src id$_{8bit}$ | 3 | 17 |
edge list$_{8bit}$ | 16 | 32 |

Gather Processing

Intra-Cluster Execution

Degree-4 Execution

T0	T1	T2	T3	T4	T5	T6	T7
0	0	0	0				
1	2	3	4				

Degree-2 Execution

T0	T1	T2	T3	T4	T5	T6	T7
1	1	4	4	2	2	3	3
2	5	2	3	1	3	0	1

Degree-1 Execution

T0	T1	T2	T3	T4	T5	T6	T7
0	2	4	5	0	1	0	1
5	5	5	2	1	0	2	0

Inter-Cluster Execution

Degree-4 Execution

T0	T1	T2	T3	T4	T5	T6	T7

Degree-2 Execution

T0	T1	T2	T3	T4	T5	T6	T7
4	4						
33	17						

Degree-1 Execution

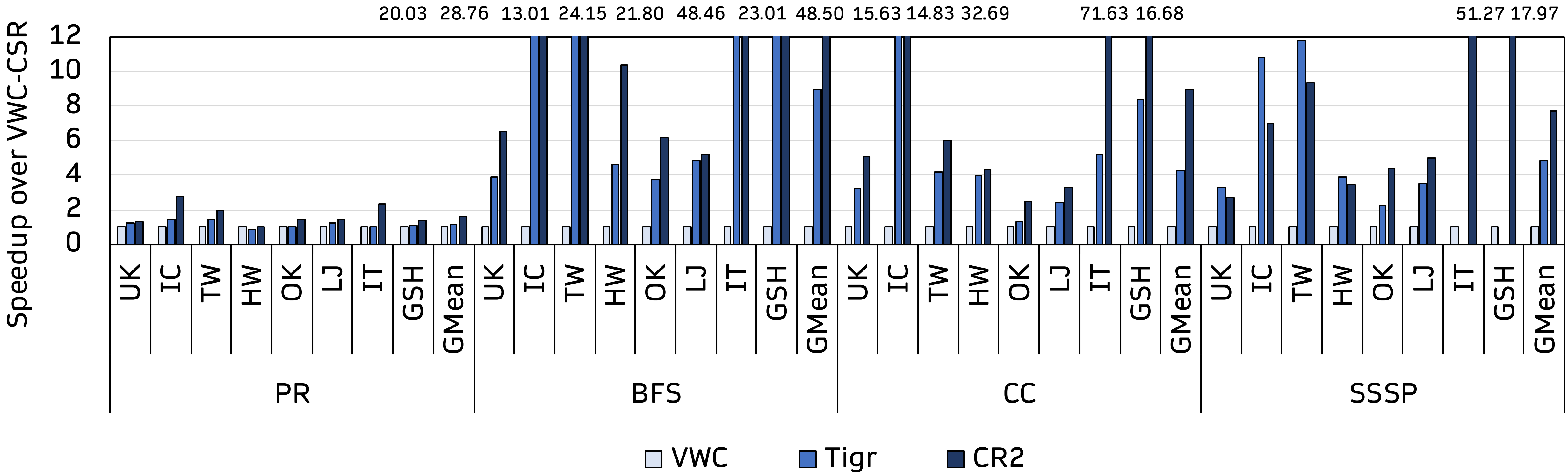
T0	T1	T2	T3	T4	T5	T6	T7
3	17						
16	32						

CSR Execution

T0	T1	T2	T3	T4	T5	T6	T7
0	5	7	8	9	14	15	15
5	7	8	9	14	15	15	15
1	2	5	5	2	2		
2	5			3			
3				5			
4				17			
5				33			
15	15	15	15	15	15	15	15
15	15	15	15	15	15	15	15
15	16	18	18	18	18	18	18
16	18	18	18	18	18	18	18
17	16						
	32						
18	18	18	18	18	18	18	18
18	18	18	18	18	18	18	18
18	19	20	22				
19	20	22	24				
34	32	33	32				
		35	33				

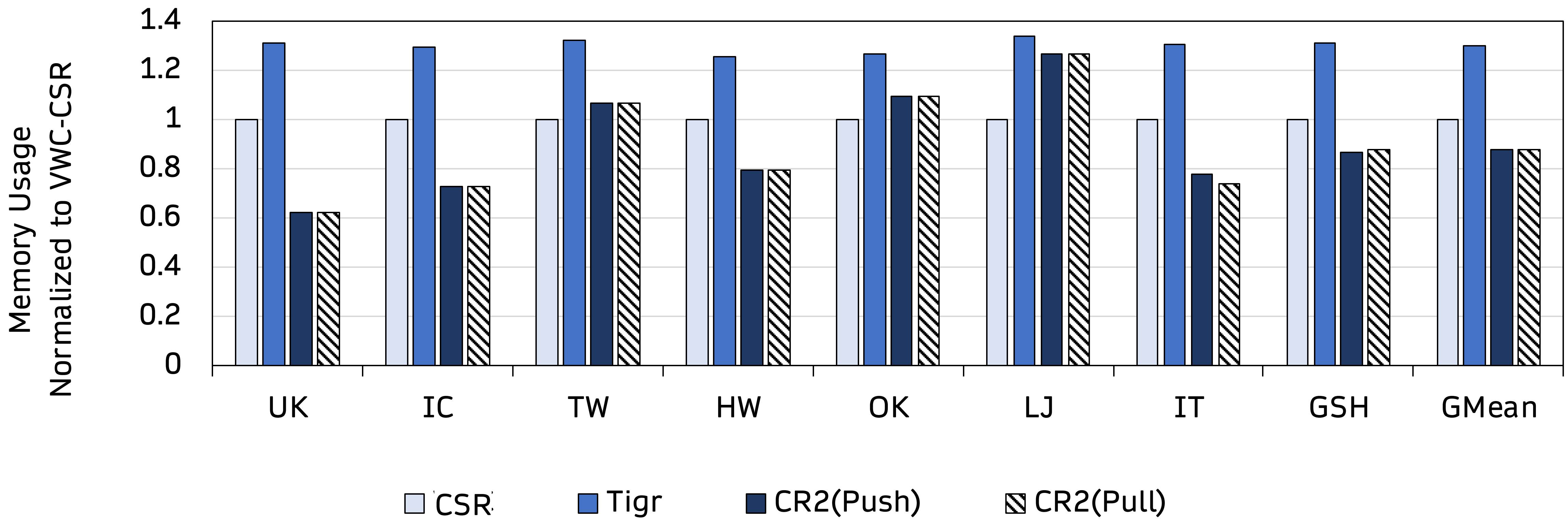
Overall Performance

- Achieve **6.47**x and **1.55**x performance geomean speedup compared to VWC-CSR and Tigr
- Use NVIDIA Geforce RTX 3090
- Use Four graph algorithms: PageRank(PR), Breadth-First Search(BFS), Connected Components(CC), and Single Source Shortest Path(SSSP)



Memory Usage

- Achieve **12.3**% and **32.2**% less memory on geomean average compared with CSR and Tigr



Conclusion

- This work proposes a new graph representation called **CR^2**
- Based on two key solution,
  - Community-aware clustered subgraph
  - Degree-ordered subgraphs with vertex degree regulation

- CR^2 enhance graph processing on GPU by,
  - Improving locality
  - Minimizing memory usage
  - Aligning with GPU architecture

- Achieve 1.53 times performance speedup while using 32.1% less memory on the geomean average compared to the state-of-the-art techniques

Thank you 😊

shin0403@yonsei.ac.kr

Specification of Existing Graph Representation

Rep	Edge Rep	Value	
		Vertex	Edge
CSR	$\|E\| + \|V\| + 1$	$\|V\|$	$\|E\|$
G-Shards	$2\|E\|$	$\|V\| + \|E\|$	$\|E\|$
Tigr	$\|E\| + (1 + Ft)\|V\|$	$\|V\|$	$\|E\|$
CR^2	$(1 - r)\|E\| + (1 + Fc)\|V\|$	$\|V\|$	$\|E\|$

CSR & CSR-based virtual wrap-centric model[1]

- Suffered from imbalance among threads within warps
- Statically fixed, leading to underutilized GPU lanes

G-Shards [2]

- Improve lane utilization and memory
- Cause more memory than CSR

Tigr [3]

- Reduce imbalance by splitting vertices and redistributing edges
- Still faces inter-warp workload imbalance
- Cause memory overheads

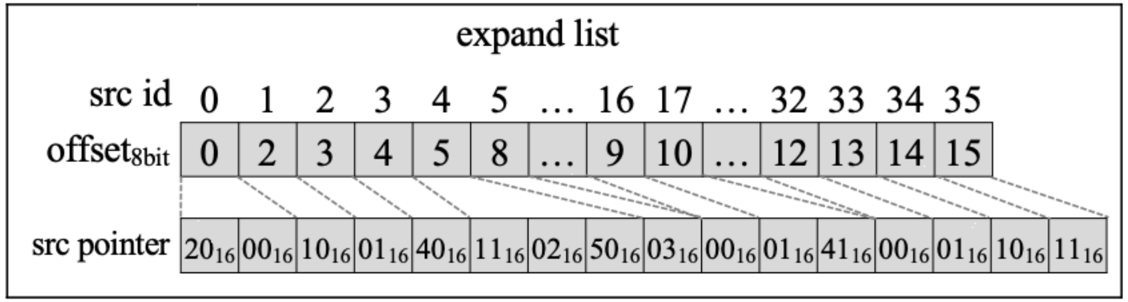
[1] S. S. Hong et al. 2011. Accelerating CUDA Graph Algorithms at Maximum Warp. PPoPP'11
[2] R. Gupta F. Khorasani et al. 2014. CuSha: Vertex-centric Graph Processing on GPUs. HPCA'14
[3] Q. Junqiao N. Sabet et al. 2018. Tigr: Transforming Irregular Graphs for GPU-Friendly Graph Processing. ASPLOS'18

CR2 Processing Kernel

Algorithm 4: Example of CR^2 processing kernel

```
Input: deg : Degree of this kernel
       baseIDList: List of base vertex id
       edgeList: List of remaining vertex id
       isPull: Indicator for direction
       workload: Number of virtual nodes
eid ← threadIdx.x + blockDim.x * blockIdx.x
vid = eid >> deg
if vid >= workload then
    return;
src = baseIDList[vid]
if isDense then
    globalID ← src & 0xffff0000
    dest ← globalID | (DENSE_TYPE *)edgeList[eid]
else
    dest ← (SPARSE_TYPE*)edgeList[eid]
if isPull then
    swap(src, dest)
...
// perform algorithm with source and destination ID.
```

Expand List for Active Vertex Supporting



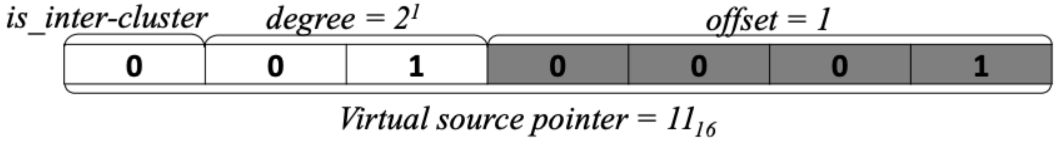


Figure 7: Source pointer representation in the expand list

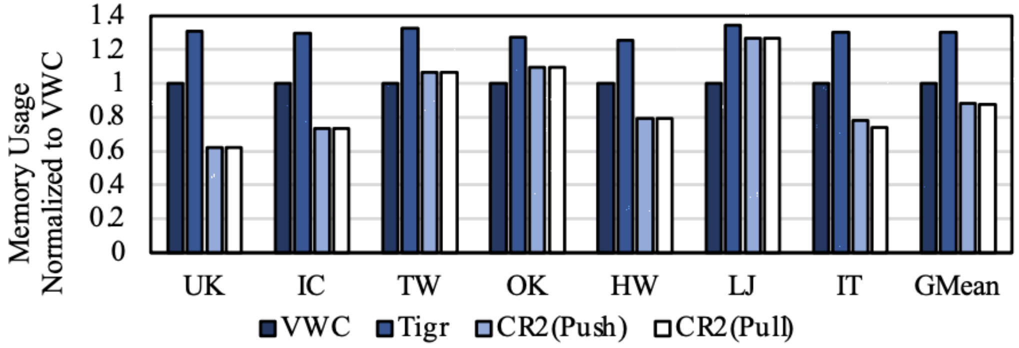


Figure 9: Memory usage of VWC-CSR, Tigr and CR^2. Each graph is normalized to the memory usage of VWC-CSR.

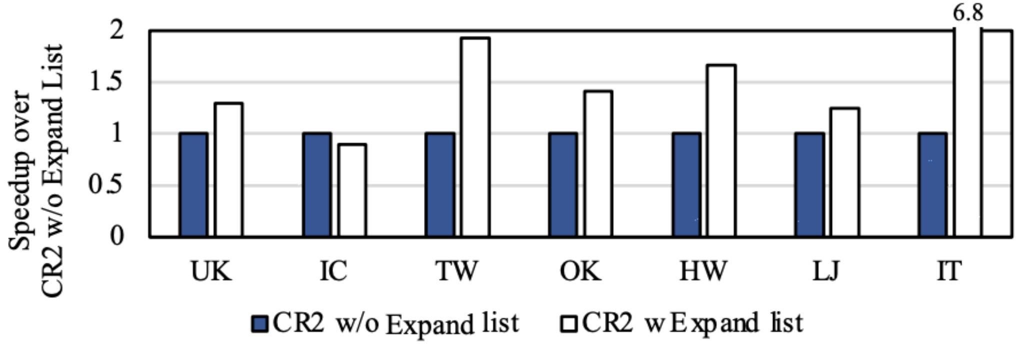
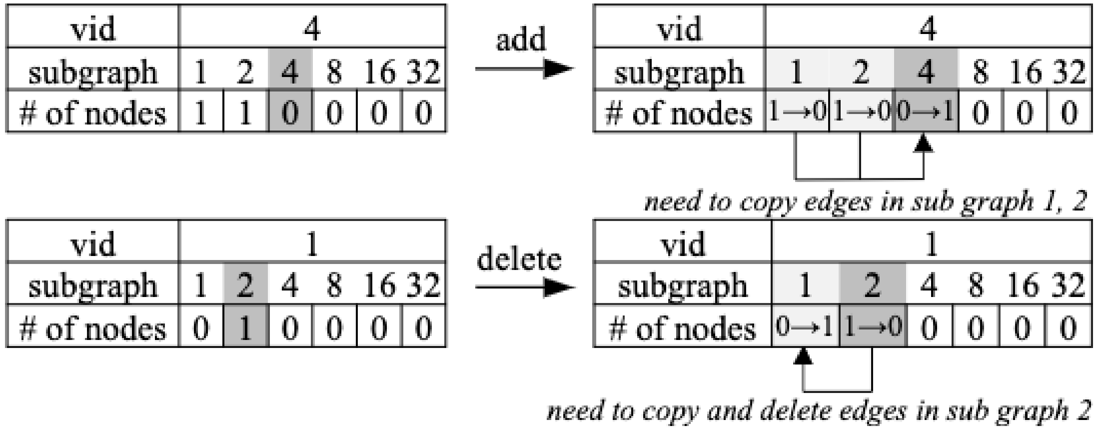
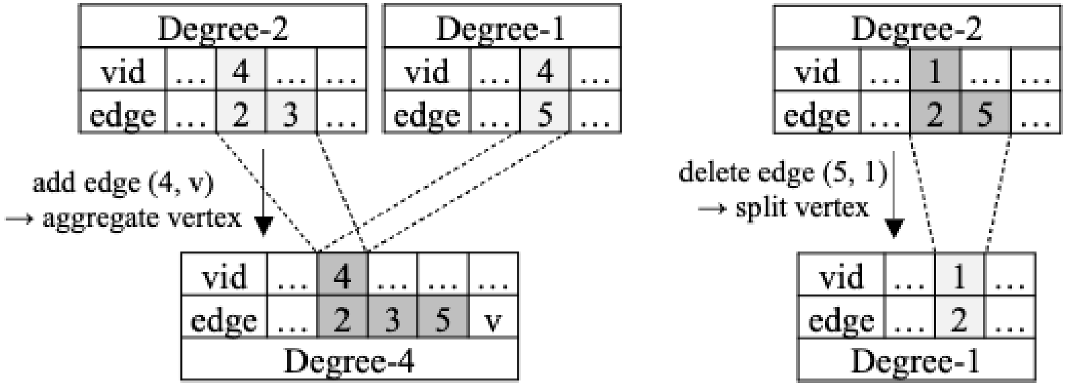


Figure 10: Performance comparison between CR^2 without the expand list and with the expand list for the BFS algorithm.

Add & Delete Cost



(a) Comparison of Degree-n subgraphs



(b) Add edge (4, v)

(c) Delete edge (1, 5)

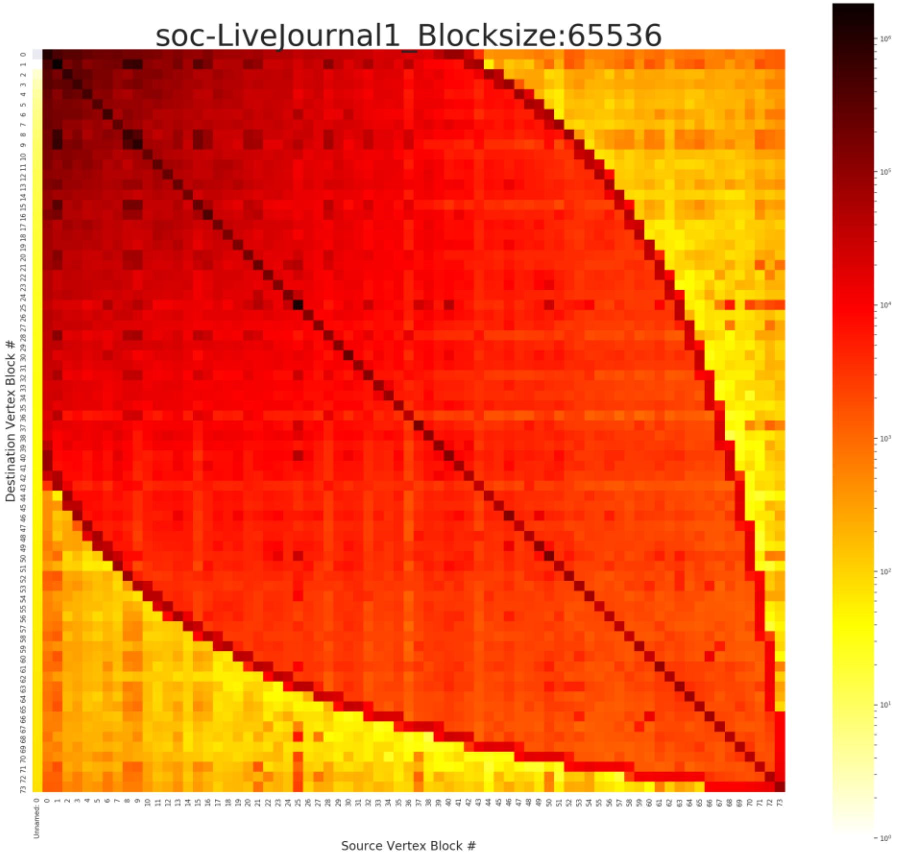
Operation	CSR	CR^2
add & delete	$O(\|V\| + \|E\|)$	$O(\|V\| + \|V_v\|)$
find neighbors	$O(deg(v))$	$O(deg(v))$

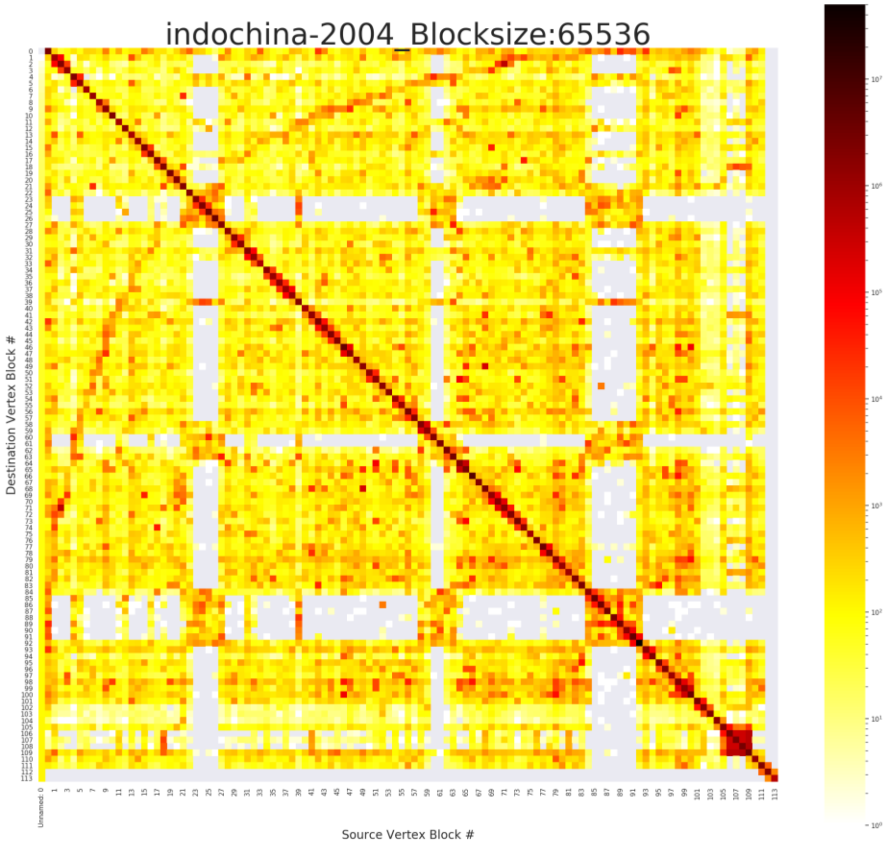
Dataset Specification

Table 3: Dataset specification [6]. $\mathrm{edges_d}$ denotes the ratio of the edges in the intra-cluster graph with cluster size $2^{16} \times 2^{16}$ over the entire edges.

Dataset	# nodes	# edges	Max deg	% $\mathrm{edges_d}$	sym
uk-2005(UK)	130K	23M	850	99 %	sym
indochina-2004(IC)	7,415K	302M	56,425	93 %	sym
hollywood-2009(HW)	1,140K	113M	11,467	70 %	sym
soc-twitter-2010(TW)	21,298K	530M	698,112	38%	sym
soc-LiveJournal(LJ)	4,848K	86M	20,333	19%	sym
soc-orkut(OK)	2,997K	213M	27,466	15%	sym
it-2004(IT)	41M	1,151M	1,326,745	92%	asym

Soc-LiveJournal vs IndoChina-2004 Graph Dataset





Build Time

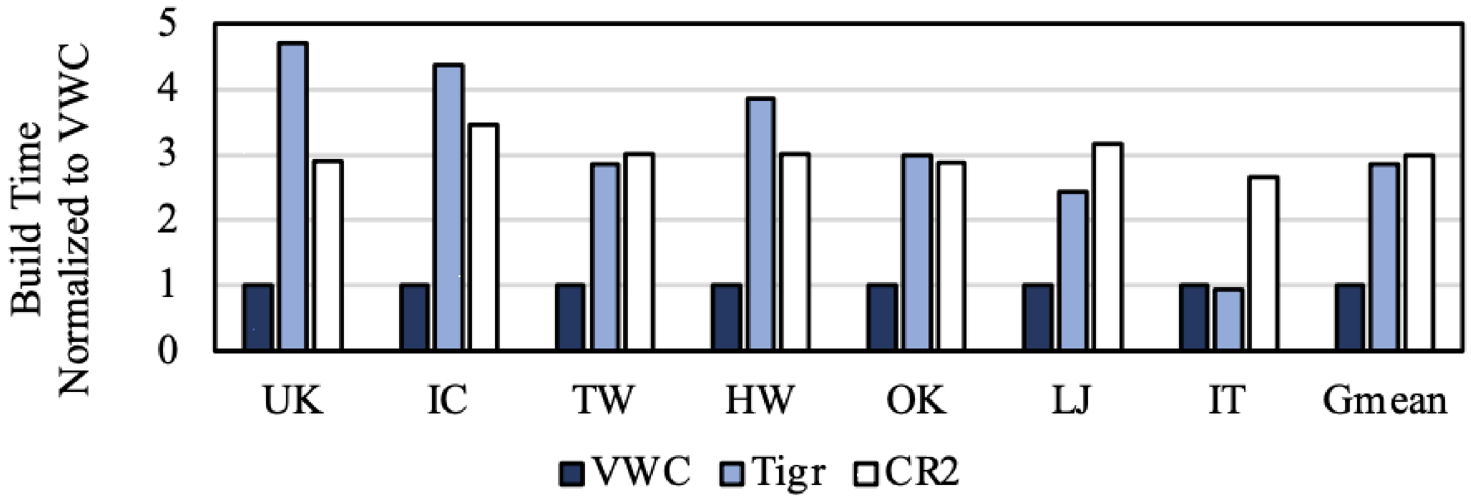


Figure 12: Build Time of VWC-CSR, Tigr and CR^2. Each graph is normalized to the build time of VWC-CSR and performs one Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz.