Distributed K-core decomposition using MPI

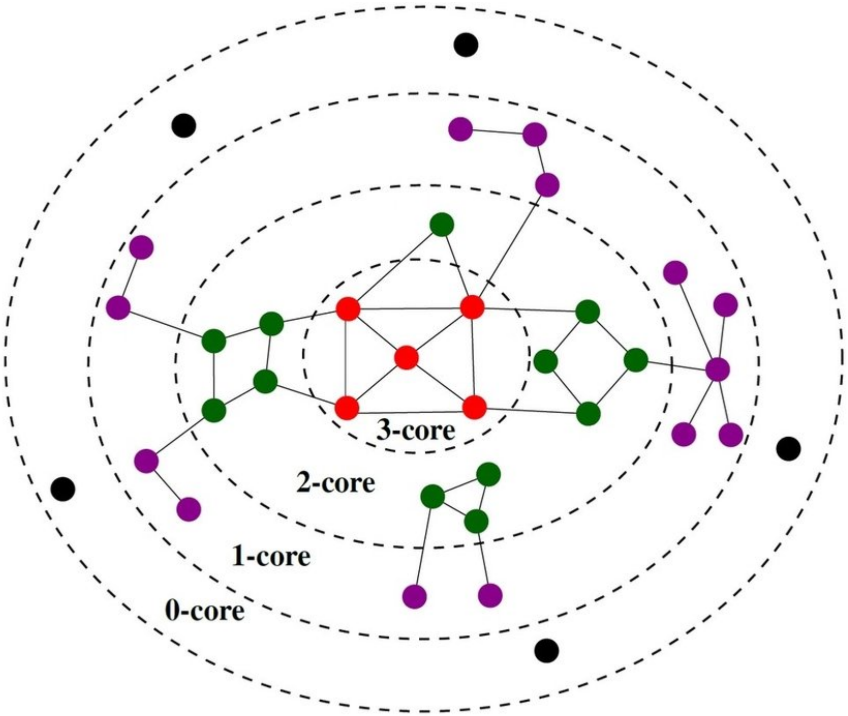
Penghang Liu

CSE633 presentation

University at Buffalo The State University of New York

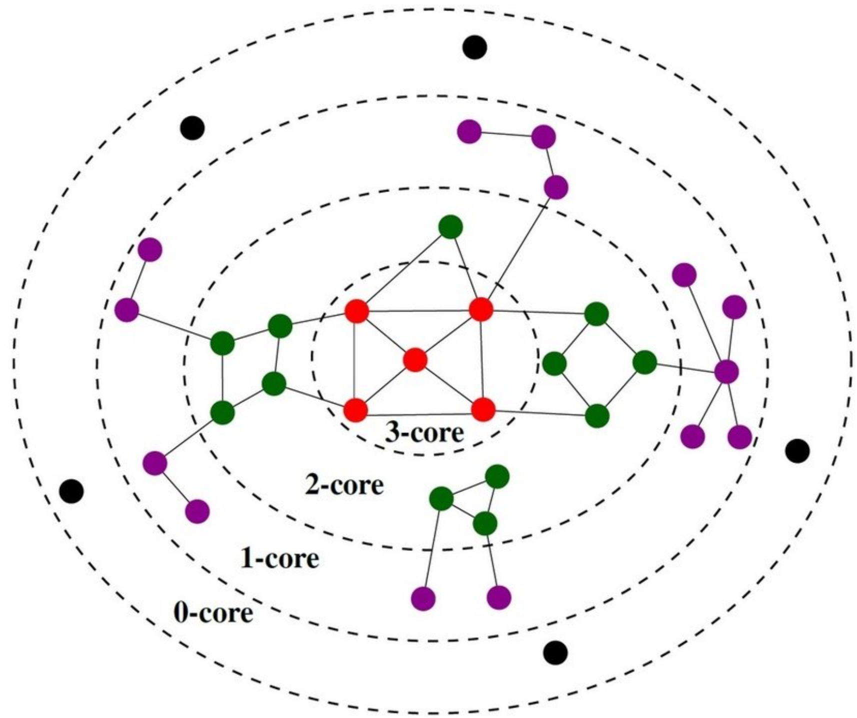
K-Core

- k core G(V, E) is the maximal subgraph where each vertex $v \in V$ is connected to at least k other vertices.
- k core is a more reliable approach in dense subgraph discovery than vertex degree.



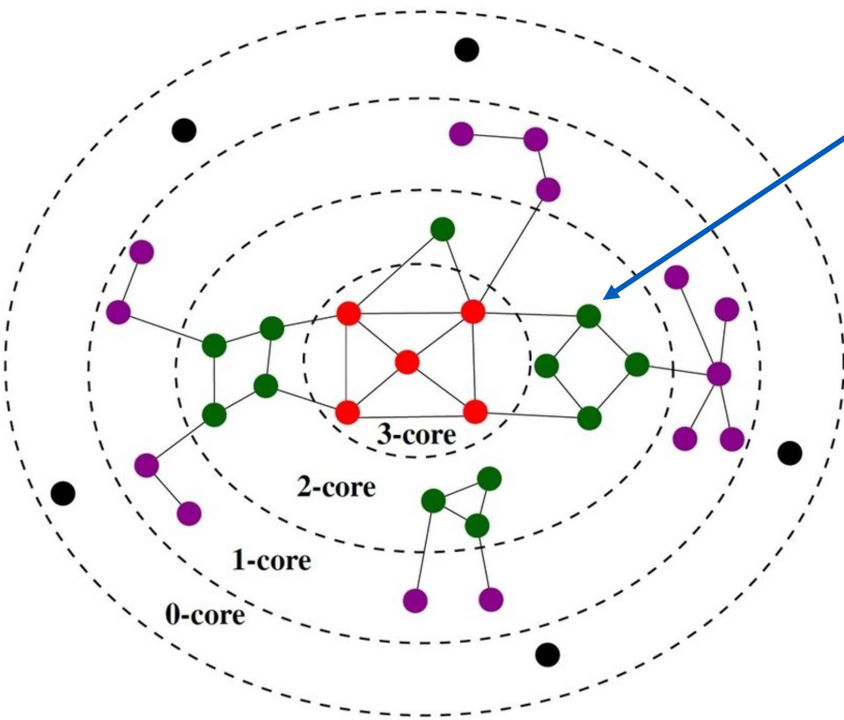
K-Core decomposition

- Given a undirected unweighted graph G(V, E), find the core value k_{max} for every vertex $v \in$.
- The core value k_{max} for a vertex v , is the maximum k core that v belongs to.



- k_{max} =0
- k_{max} =1
- k_{max} =2
- k_{max} =3

Calculating K value



Real K value of its neighbors

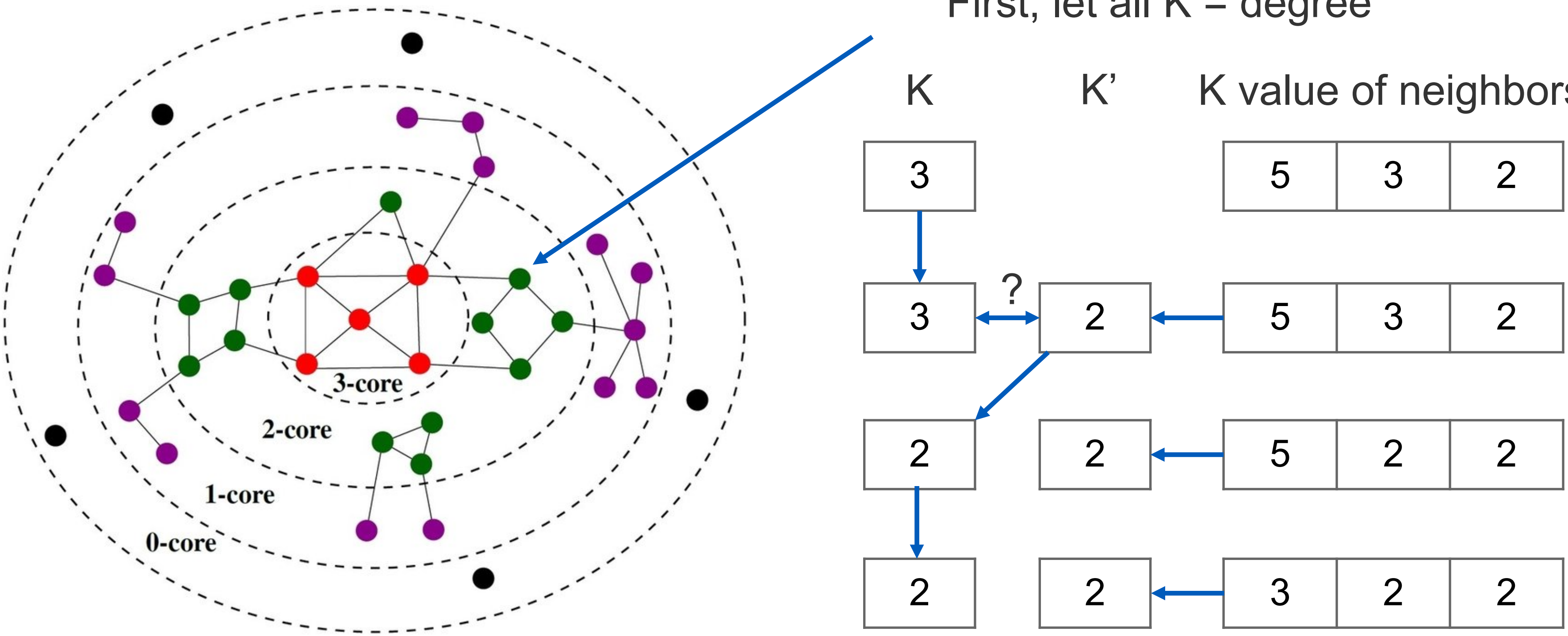
Vertex v_i :			
	3	2	2

$$Count(k \geq 3) = 1 < 3$$
$$Count(k \geq 2) = 3 \geq 2$$

So K = 2

Calculating K value from degree

First, let all K = degree



Solution (single processer)

Input: Adjacent list

Vertex:

v_1	v_2	v_3	v_4	$\ldots$	v_i	$\ldots$	v_j	$\ldots$	v_n

$deg_1 = 2$

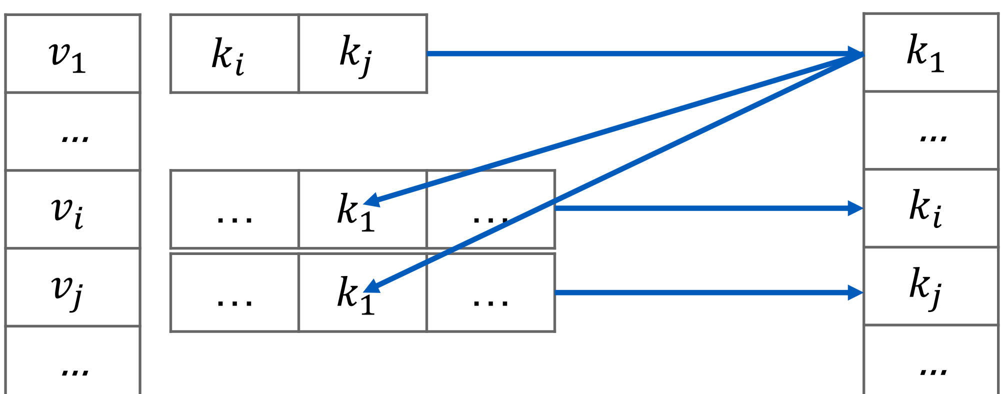
v_i
v_j

Initialize $k_i = deg_i$:

K:

k_1	k_2	k_3	k_4	$\ldots$	k_5	$\ldots$	k_6	$\ldots$	k_n

Update K until convergence:



Distribute n vertices to m processors:

	P_1			P_2		$\dots$	P_m		
Vertex:									
v_1	v_2	v_3	v_4	$\dots$	v_5	$\dots$	v_6	$\dots$	v_n

Vertex v_i is assigned to processor $(i \ mod \ m)$

Initialize each processor:

Partial adjacent list

...	v_i	$\dots$	v_j	$\dots$	v_n
	v_j				
	v_k				

Initialize: $k_i = deg_i$ if $v_i \in P$
$k_i = \infty$ if $v_i \notin P$

...	k_i	$\dots$	k_j	$\dots$	k_n

Sending messages:

Send local k value

...	k_i	…	k_j	…	k_n

Send k_i to v_j
Send k_i to v_k

Send k_i to v_j → Local vertex v_j

Send k_i to v_k → Send to v_k in processor $(k \ mod \ m)$

Receiving messages:

On initialization:

k_i		$k_j = deg_j$	$k_k = \infty$

k'_i	←	k_j	k_k	←

Receive new k_k from processor $(k \ mod \ m)$

If $k_i < k'_i$: Then $k_i \leftarrow k'_i$, send message of k_i

Pseudocode for the processor

on *initialization* **do**
  changed ← **false**;
  core ← $d(u)$;
  foreach $v \in neighbor_V(u)$ **do** $est[v] \leftarrow \infty$;
  send $\langle u, core \rangle$ **to** $neighbor_V(u)$;

Initialization

on *receive* $\langle v, k \rangle$ **do**
  if $k < est[v]$ **then**
    $est[v] \leftarrow k$;
    $t \leftarrow$ computeIndex$(est, u, core)$;
    if $t < core$ **then**
      core ← t;
      changed ← **true**;

Receive message

repeat
  if *changed* **then**
    send $\langle u, core \rangle$ **to** $neighbor_V(u)$;
    changed ← **false**;

Update and send new message

Pseudocode for updating K value

Algorithm 2: **int** computeIndex(**int**[ ] *est*, **int** u, k)

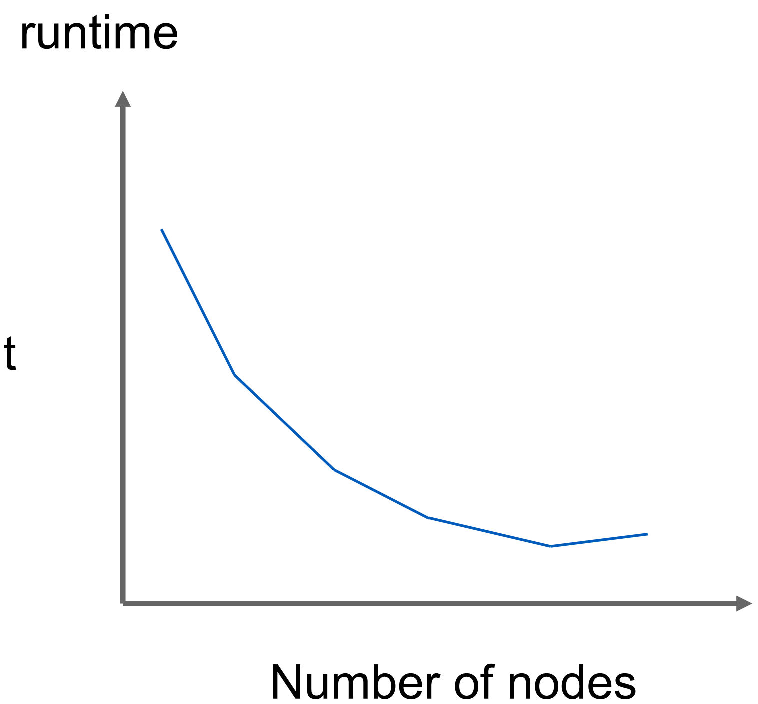
for $i = 1$ **to** k **do** $count[i] \leftarrow 0$;
foreach $v \in neighbor_V(u)$ **do**
  $j \leftarrow \min(k, est[v])$;
  $count[j] = count[j] + 1$;
for $i = k$ **downto** 2 **do**
  $count[i-1] \leftarrow count[i-1] + count[i]$;
$i \leftarrow k$;
while $i > 1$ **and** $count[i] < i$ **do**
  $i \leftarrow i - 1$;
return i;

Update k_i based on the current K value of the neighbors of v_i .

Experiment I: same input size, increase number of processors

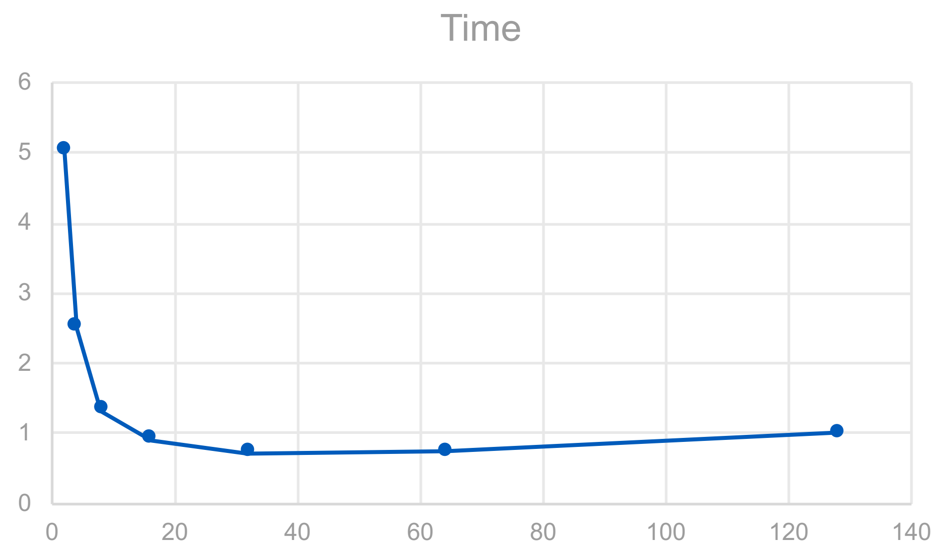
Nodes	2	4	8	16	32	64	128
Input size	1,200,000						

Expected result



Experiment I: same input size, increase number of processors

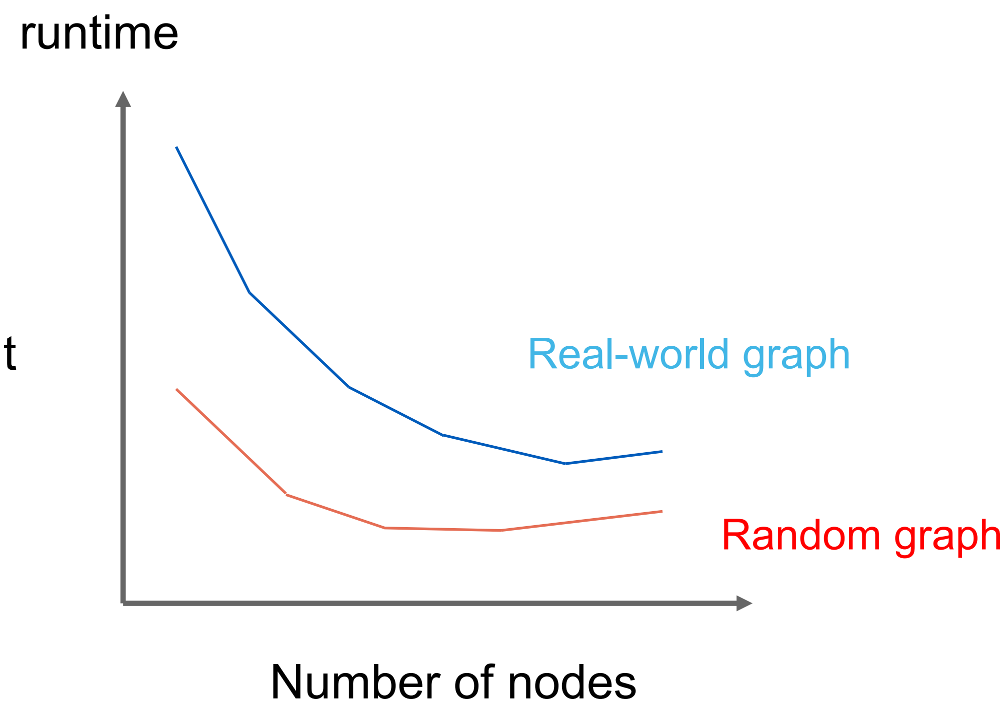
Processor	Time
2	5.037009
4	2.515878
8	1.323840
16	0.916787
32	0.718203
64	0.746985
128	1.018293



Experiment II: Real-world graph vs random graph

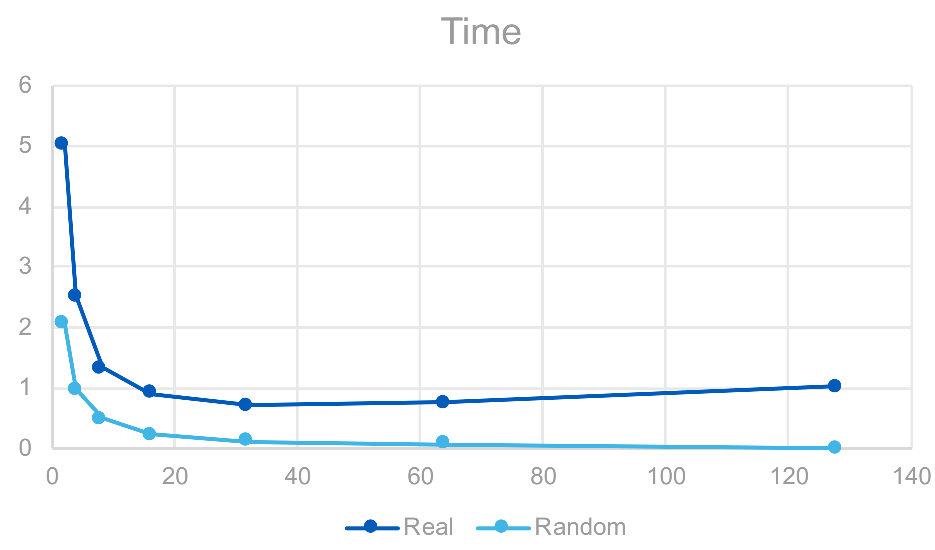
Nodes	2	4	8	16	32	64	128
Input 1	YouTube friendships (1,200,000 nodes)						
Input 2	ER model random graph (1,200,000 nodes)						

Expected result



Experiment II: Real-world graph vs random graph

Processor	Time	Random
2	5.037009	2.056716
4	2.515878	0.970089
8	1.323840	0.499263
16	0.916787	0.227319
32	0.718203	0.107626
64	0.746985	0.077976
128	1.018293	0.003473



Accuracy Validation

```
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 2.txt youtube_true.txt | aw]
k '{print $1-$11}'|sort -n|uniq -c
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 4.txt youtube_true.txt | aw]
k '{print $1-$11}'|sort -n|uniq -c
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 8.txt youtube_true.txt | aw]
k '{print $1-$11}'|sort -n|uniq -c
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 16.txt youtube_true.txt | a]
wk '{print $1-$11}'|sort -n|uniq -c
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 32.txt youtube_true.txt | a]
wk '{print $1-$11}'|sort -n|uniq -c
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 64.txt youtube_true.txt | a]
wk '{print $1-$11}'|sort -n|uniq -c
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 128.txt youtube_true.txt | ]
awk '{print $1-$11}'|sort -n|uniq -c
1134890 0
```

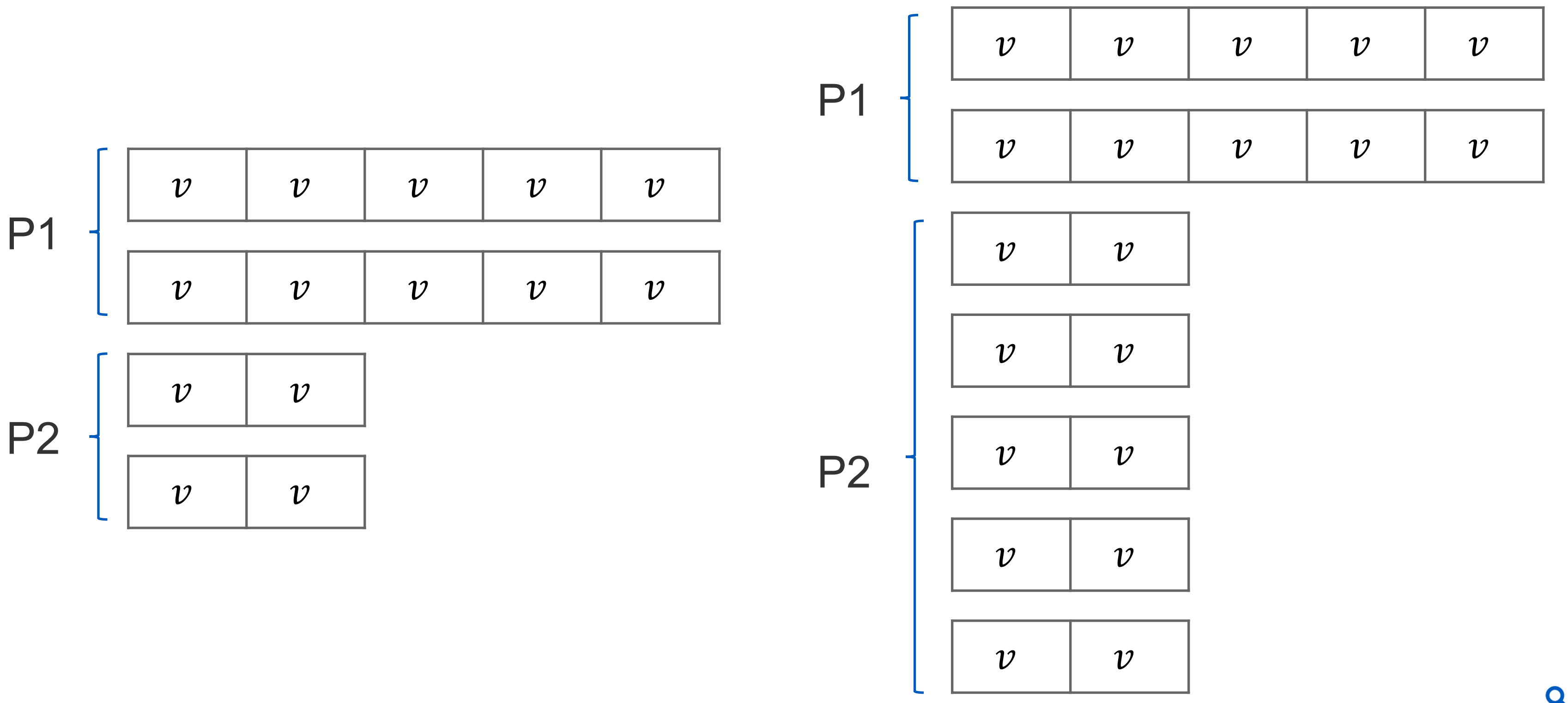
University at Buffalo The State University of New York

Challenges

- Proposed experiment: Double the input size as well as the number of processors.
- Real-world data: Different real-world data doesn't work the same. Can not control input size
- Random graphs: Randomly generated data is so uniform that the communication is finished in a few rounds.
- Data are not exactly distributed equally.

Future Work

Distribute the data by edges instead of nodes?



Reference

Montresor, A., De Pellegrini, F., & Miorandi, D. (2013). Distributed k-core decomposition. *IEEE Transactions on parallel and distributed systems*, *24*(2), 288-300.

Malliaros, F. D., Papadopoulos, A. N., & Vazirgiannis, M. (2016). Core Decomposition in Graphs: Concepts, Algorithms and Applications. In *EDBT* (pp. 720-721).

Thanks!