Parallel Clustered Low-Rank Approximation and Its Application to Link Prediction*

Joyce Jiyoung Whang,
Department of Computer Science & Engineering, SKKU

* Published in *International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, 2012.

Social Network Analysis

- Huge size of social network graphs poses great challenge on the analysis

facebook Over 900 million active users

twitter Over 500 million active users

LinkedIn Over 175 million registered users

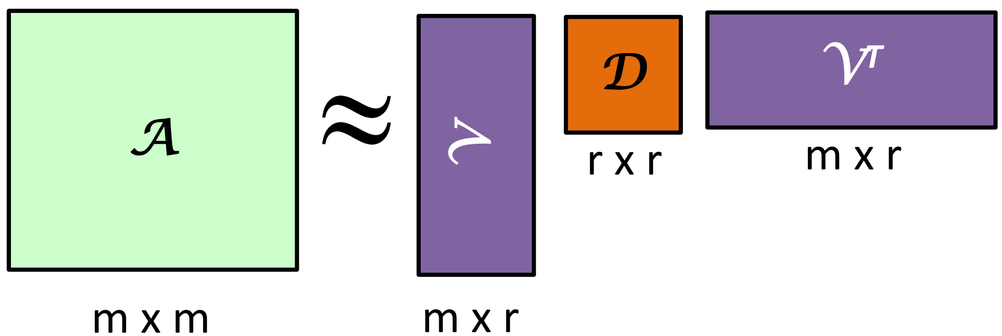
- Two important ways to solve the challenge
  - Parallelization
    - Large scale distributed parallelization
  - Approximation
    - In many cases, approximate answers are sufficient, e.g. friend recommendations

Need for Approximation

- Problem: compute the number of *length-k* paths between every two vertices in a graph
- Solution 1: graph traversals
  - Too expensive for large graphs
- Solution 2: linear algebra formulation
  - Represent a graph by its adjacency matrix A
  - $A^K(i,j)$ is number of *length-k* paths between vertices i and j
  - Still very expensive

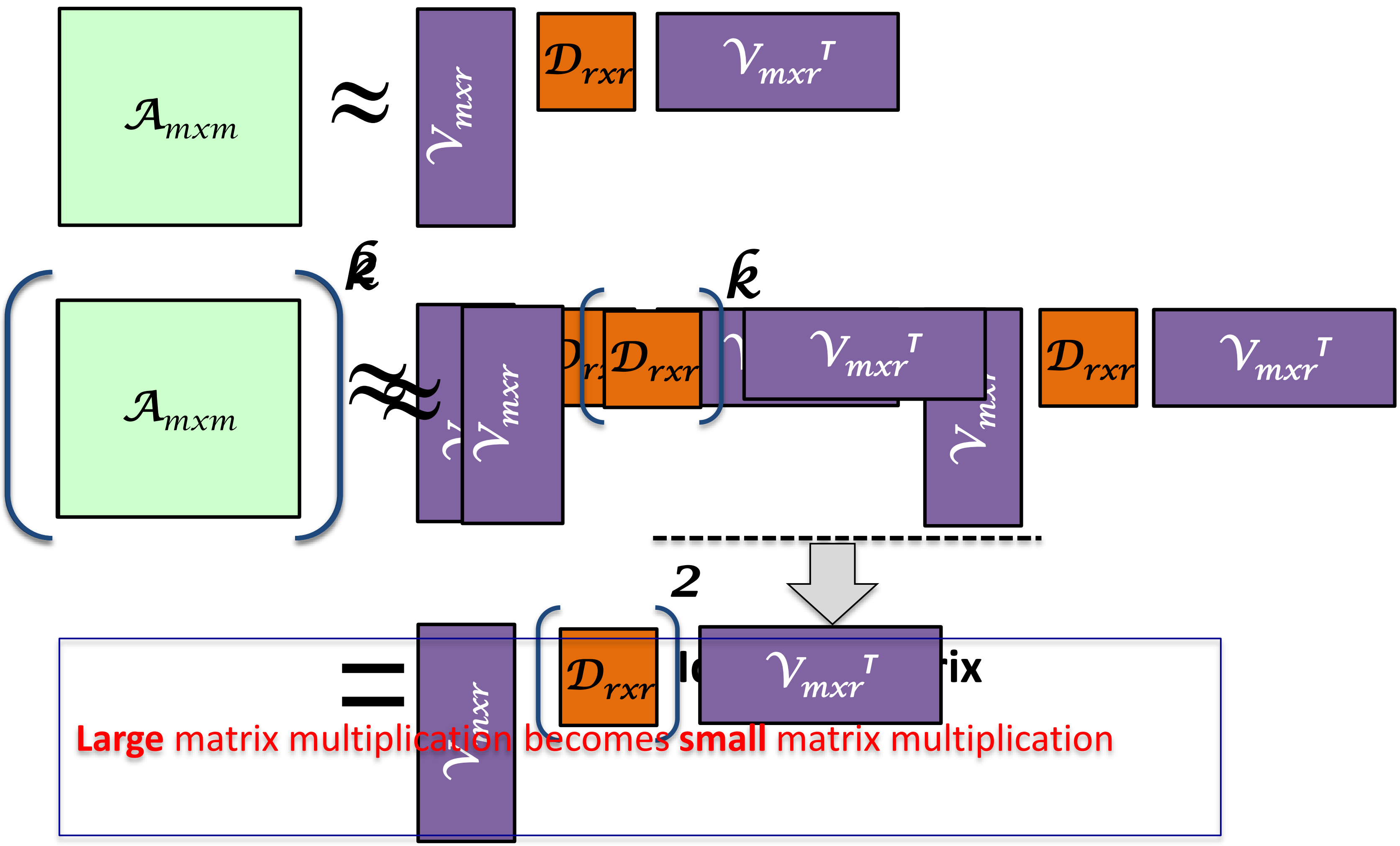
Low-rank Approximation

- The adjacency matrix of an undirected graph can be approximated by the product of three matrices $\mathcal{V}$, $\mathcal{D}$ and $\mathcal{V}^T$



  - $r << m$, called **rank,** is an input parameter
  - The larger the r, the smaller the approximation error
  - $\mathcal{V}^T\mathcal{V} = \mathcal{I}$ (Identity matrix)

Approximating A^k by low-rank approximation



Large matrix multiplication becomes **small** matrix multiplication

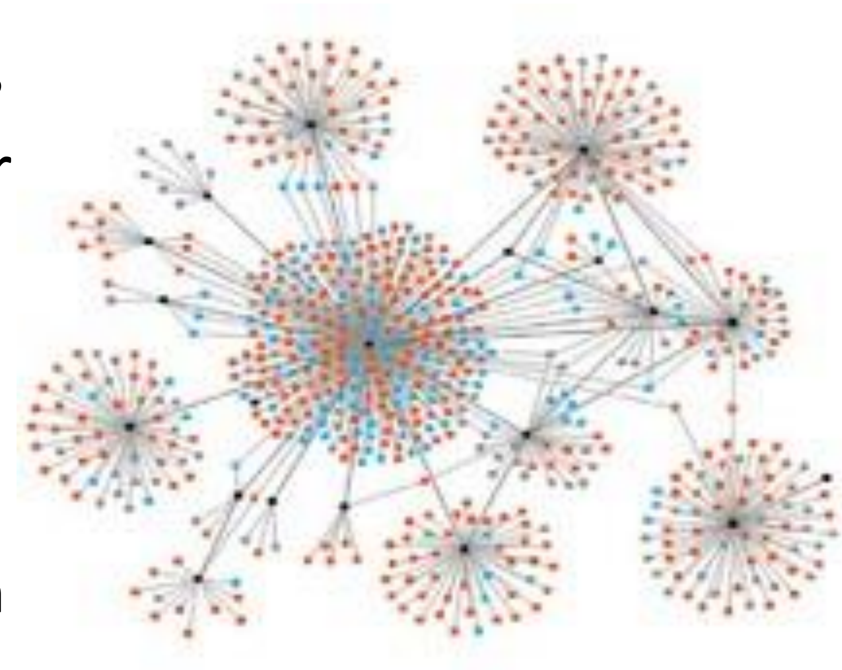
The Limitation of Low-rank Approximation

- Large rank *r* is needed for large graphs to make the approximation error acceptable
- The computation and memory costs are expensive for large graphs and rank

How to improve it

Structure In Social Networks

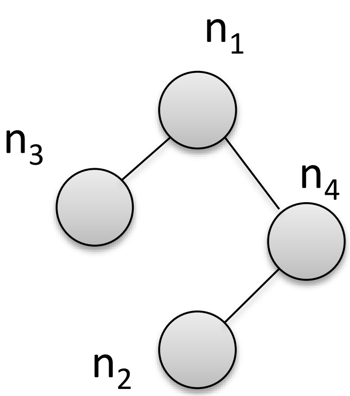
- Not uniformly random graphs
- Clusters with few inter-cluster edges
- ***Clustered*** low-rank approximation
    1. Find clusters by partitioning
    2. Use low-rank approximation for each cluster
    3. Account for inter-cluster edges

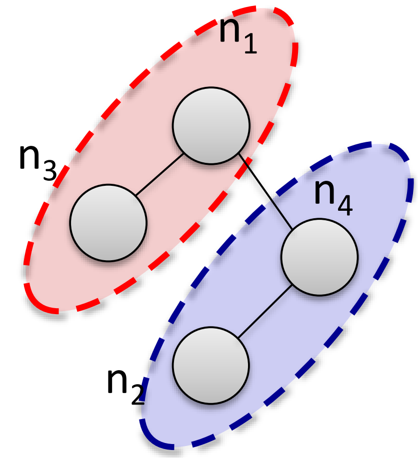


Our Contributions

- A new parallel partitioning algorithm for social networks
  - Easy to parallelize
  - Compared to ParMetis
    - Faster and scales better
    - Generates similar quality partitions when ParMetis succeeds
- First parallel implementation of clustered low-rank approximation
- Application to link prediction of very large graphs

Matrix View of Clustering

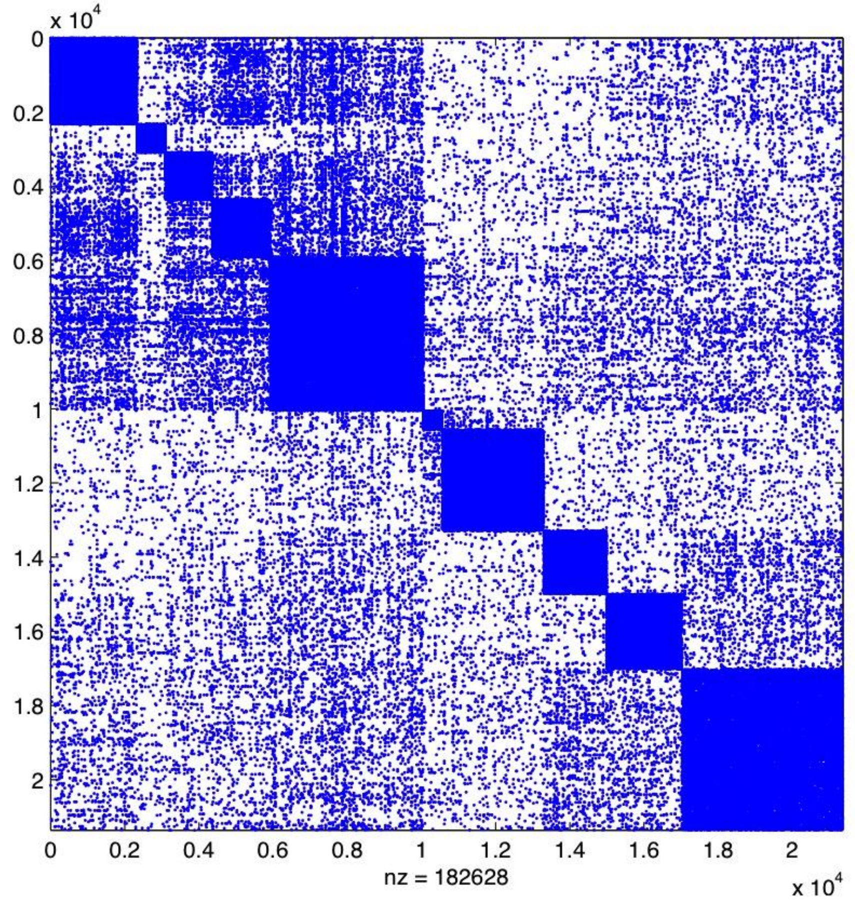




	n_1	n_2	n_3	n_4
n_1	1	0	1	1
n_2	0	1	0	1
n_3	1	0	1	0
n_4	1	1	0	1

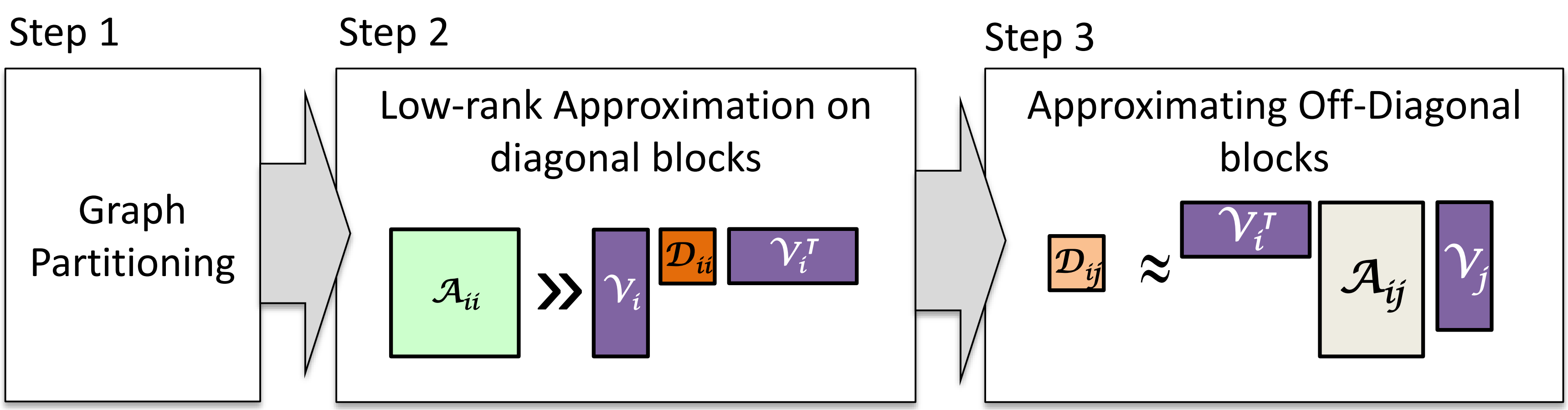
	n_1	n_3	n_2	n_4
n_1	1	1	0	1
n_3	1	1	0	0
n_2	0	0	1	1
n_4	1	0	1	1

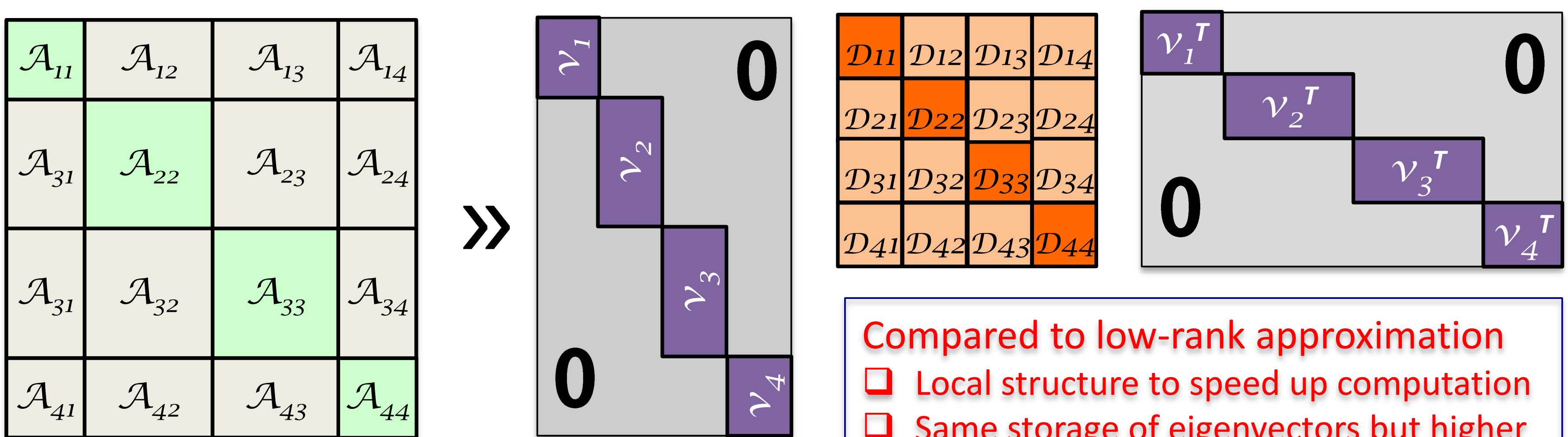
Example: arXiv Network



21,363 vertices and 91,314 edges

Clustered Low-rank Approximation



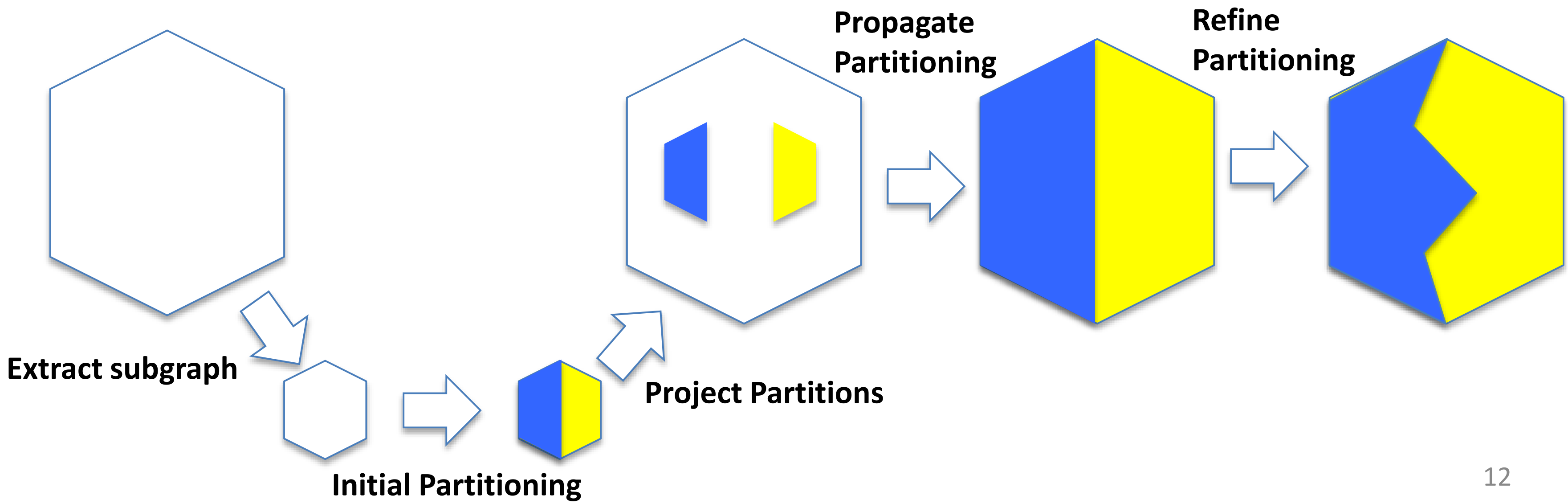


Compared to low-rank approximation

- ❑ Local structure to speed up computation
- ❑ Same storage of eigenvectors but higher rank

1. PEK: A new graph partitioning algorithm for social networks

- Intuition: High degree vertices capture the high level structure of such graphs
- PEK Algorithm:
  - Extract a small representative sub-graph(high degree vertices and their edges)
  - Partition this sub-graph
  - Propagate partitioning to entire graph
  - Refine with weighted kernel K-Means



Extract a Representative Sub-Graph

- Extract a small number of high-degree vertices and the edges between them
  - Graph is randomly and evenly distributed across processes
  - Each process selects its local vertices with degree larger than a threshold
  - Those vertices and the edges between them form the representative sub-graph

Partition Sub-graph

- Use ParMetis to partition sub-graph
  - Takes a small fraction of time
- Project partitions of vertices in sub-graph to original graph
  - **Projected vertices** assigned to partitions
  - **Un-projected vertices** are not assigned

Propagate Partitioning(1)

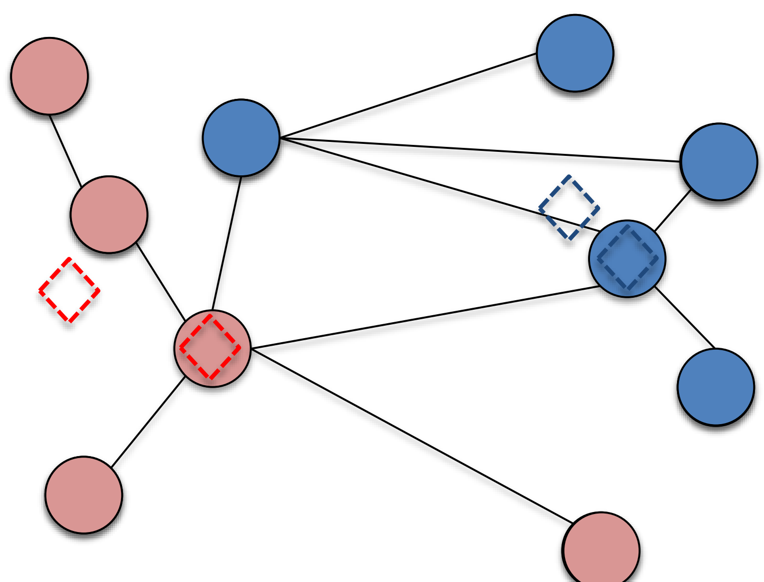
- Each partition has a virtual center point(centroid)
  - Initially computed based on the partitions of projected vertices
- Distance between a vertex to the centroid of a partition
  - Measure how close a vertex to the partition
  - Computed based on the partition size, #edges of the vertex to the partition, #edges within the partitions, etc.

Propagate Partitioning(2)

- Visit un-projected vertices in breadth-first order
  - Start from projected vertices
- For each un-projected vertex :
  - Assign to partition with the closest centroid
  - Update the centroid
- Each process has its own copy of all centroids
  - Do not synchronize updates of centroids
  - No impact on partition quality

Centroid of $Part_1$

Centroid of $Part_2$



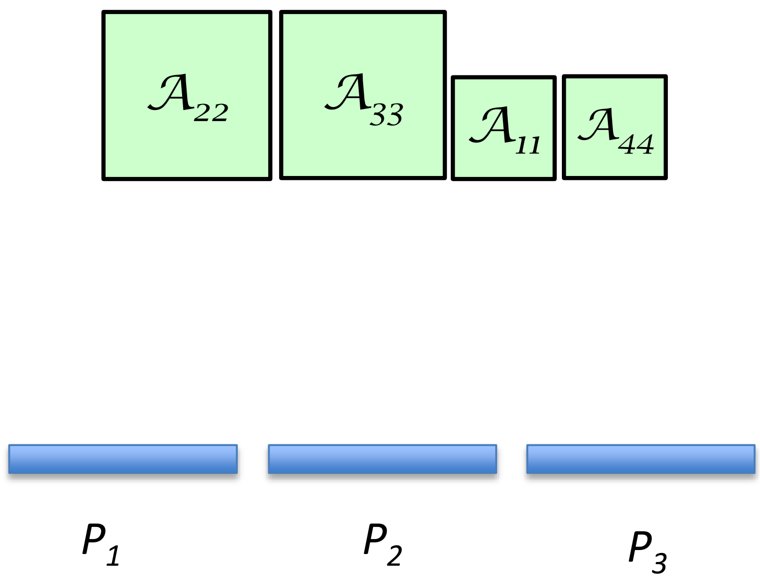
Refine Partitions

- Iteratively improve initial partitioning
- On every iteration, each process:
  - Visits its local vertices **on the partition boundary**
  - For each boundary vertex *v*:
    - Moves *v* from partition $Part_i$ to $Part_j$ if *v* is closer to $Part_j$
    - If moved, update the old and new centroids
- Processes synchronize updates of centroids once every iteration
  - Less communication
  - Does not degrade quality

2. Approximating Diagonal Blocks

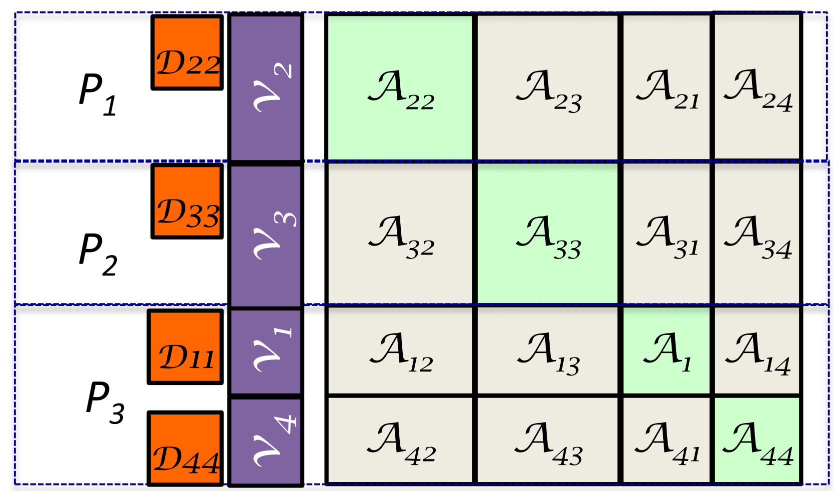
- Assigns partitions to processes
- Each process computes low rank approximation on partitions independently

Sorted by the weights: #nonzero(A_{ii}) x rank



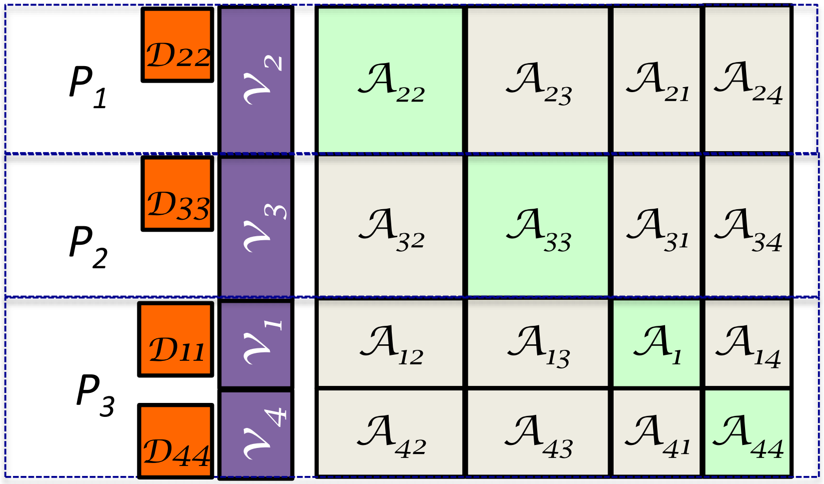
Assigns partition to the process currently having the least weights

Graph is reorganized after assignment



3. Approximating Off-Diagonal Blocks

- Undirected graph => symmetric adjacency matrix *A*
  - Only one of A_{ij} and A_{ji} needs to be approximated
- A job, $J_{i,j}$, $_{i<j}$, *denotes* approximating either A_{ij} or A_{ji}
  - Private jobs of process P_i, *e.g.* $J_{1,4}$
    - Can be finished by P_i without communication
  - Shared jobs between *Pi* and *Pj, e.g.* $J_{2,3}$
    - Either P_i or P_j can finish it
    - Communication is needed between P_i and P_j
- Processes first finish its private jobs
- Dynamic load balancing for scheduling shared jobs



Experimental Setting

- Machine: Ranger(Texas Advanced Computing Center)
  - Each node has a *4 x 4*-core AMD Opteron *2.2GHz* CPU and 32GB memory.
  - InfiniBand networks with *5GB/s* point-to-point bandwidth
- Libraries: Intel ICC *10.1*, OpenMPI *1.3*, ARPACK++, GotoBLAS *1.3* and Elemental *1.7*
- Assign one process per node

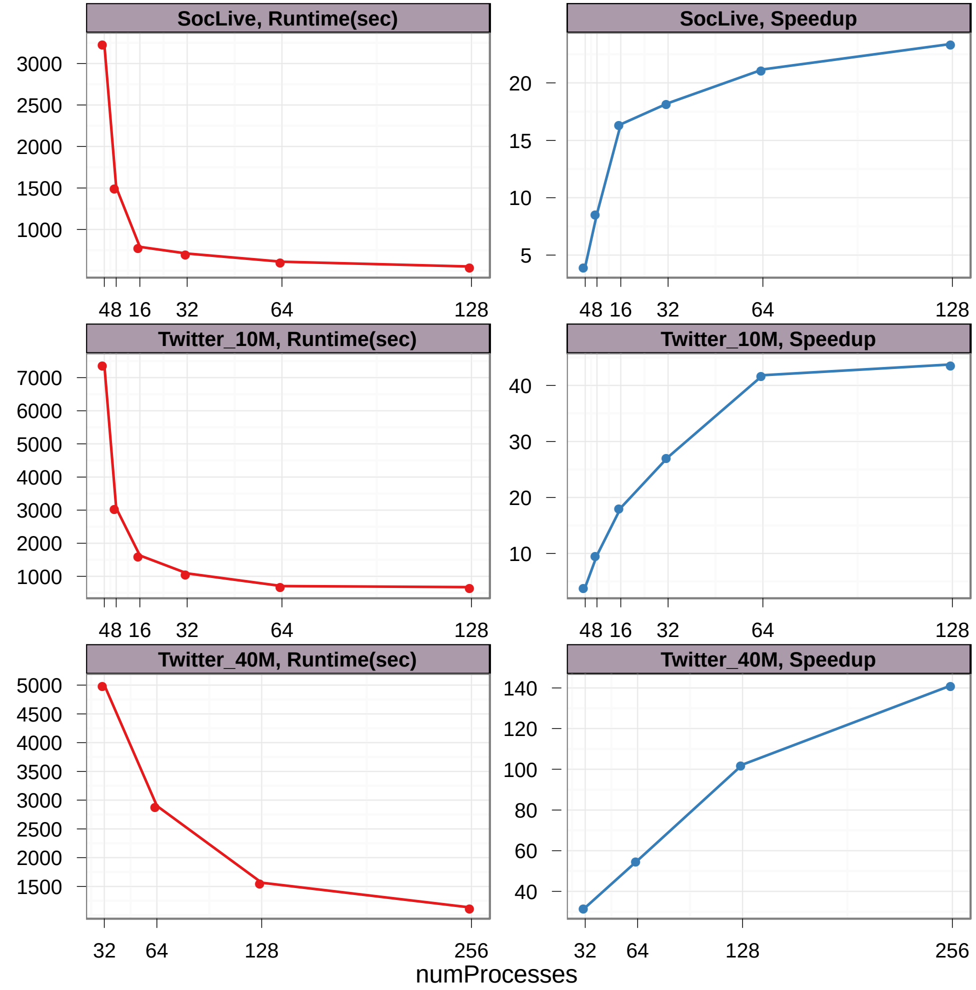
Datasets

- Converted the graphs to undirected graphs, the table shows the statistics of graphs after conversion

Name	#Vertices	#Edges	Description
SocLive	3,828,682	39,870,459	LiveJournal online social network
Twitter_10M	11,316,799	63,555,738	Twitter social network
Twitter_40M	41,652,230	1,202,513,046	Twitter social network

Runtime and Speedup of Parallel Clustered Low-rank Approximation

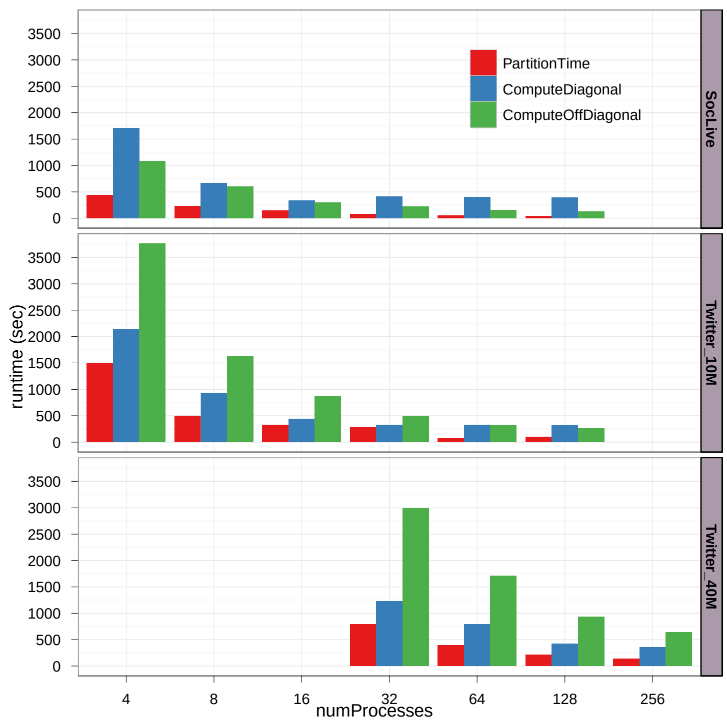
- #Partitions
  - SocLive: *500*
  - Twitter_10M: *500*
  - Twitter_40M: *1000*
- Rank for Diagonal Phase
  - SocLive: *100*
  - Twitter_10M: *100*
  - Twitter_40M: *100*



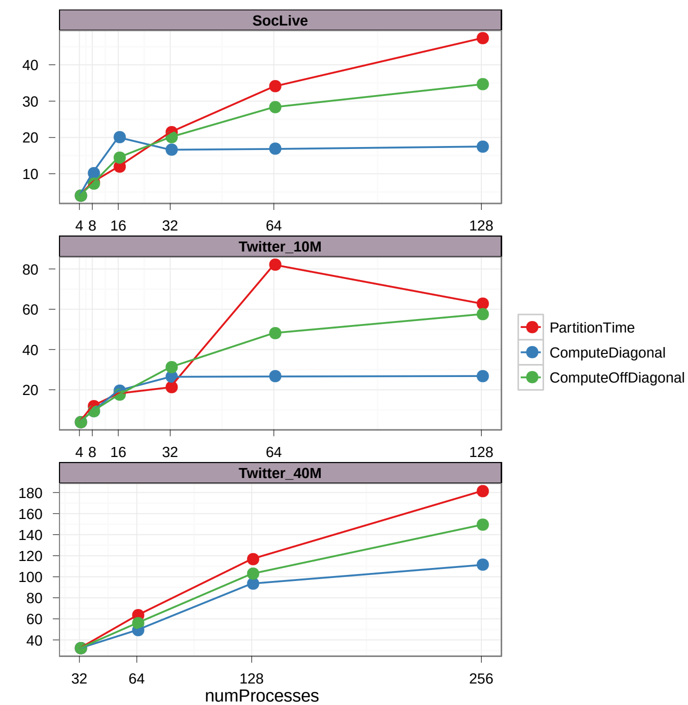
Runtime and Speedup in Each Phase

- Partitioning and offDiagonal phases scale well

Runtime in Each Phase



Speedup

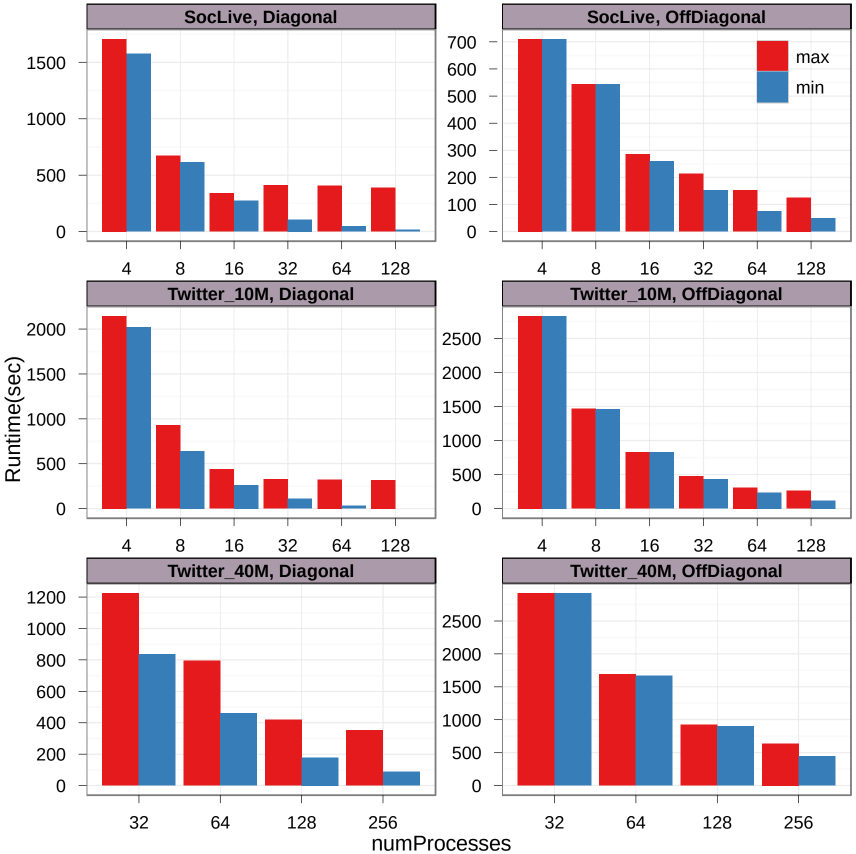


Load Balancing of diagonal and offDiagonal Phases

- #Partitions is small compared to #Processes, not enough space for load balancing in diagonal phase

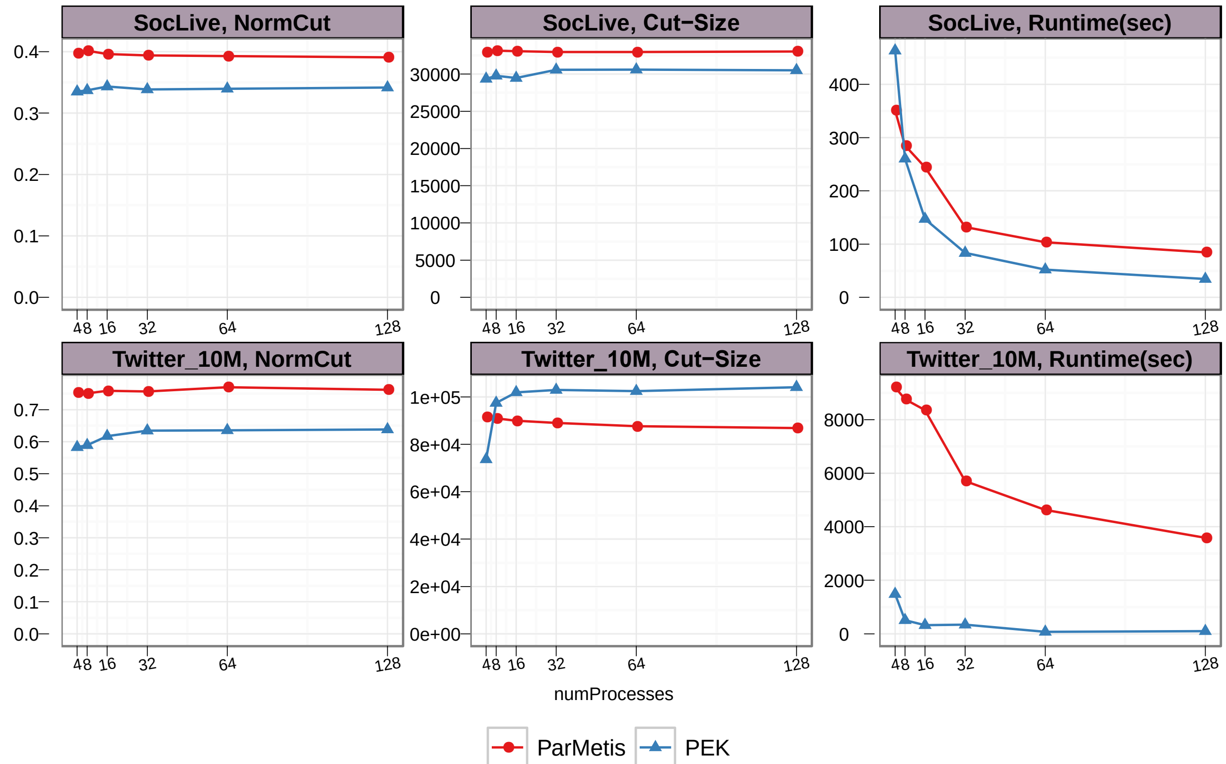
#Partitions:

- SocLive: *500*
- Twitter_10M: *500*
- Twitter_40M: *1000*



Graph Partitioning
Comparing PEK with ParMetis

- #Partitions: *500*
- Degree Threshold:
  - SocLive: *42*(*5%* vertices)
  - Twitter_10M:*200*(less than 5% vertices)
- Cut-size and NormCut
  - **Lower is Better**
  - cut-size: the edges across partitions
  - NormCut: normalized cut-size by the total degree of vertices of each partition
  - divided by the number of clusters
- ParMetis cannot partition Twitter_40M since the memory is not enough



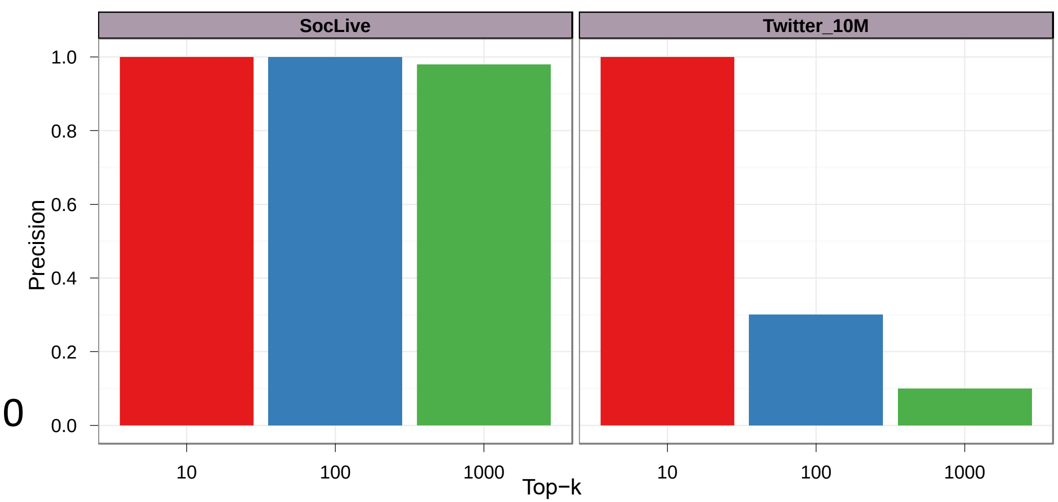
Link Prediction

- Our parallel clustered low-rank approximation enabled first ever study of Katz measure on large real-world social networks

$$Katz(v_i, v_j) = \sum_{k=1}^{\infty} b^k \left| paths_{v_i \to v_j}^{length=k} \right| \quad \text{where } \beta \text{ is damping factor}$$

- Randomly remove *30%* edges from graphs and perform link prediction on the resulting graphs.
- Precision is the ratio of correct predictions in *top-k* predictions

#partitions:500
rank of diagonal:50



#partitions:500
Rank of diagonal:100

Conclusion

- Developed a new graph partitioning algorithm for social networks
  - Fast and scales well to large number of processes
  - Faster than ParMetis and similar partition quality as ParMetis
- Parallelized clustered low-rank approximation and applied it on large real-world social networks
- Benchmark combines:
  - Irregular and regular computations
  - Dense and sparse data structures
- Approximation and Parallelization are the keys for solving large-scale social network problems