Computing the Pseudoinverse of a Graph's Laplacian using GPUs

Nishant Saurabh
Vrije Universiteit, Amsterdam.
nishants.prmitr7@gmail.com

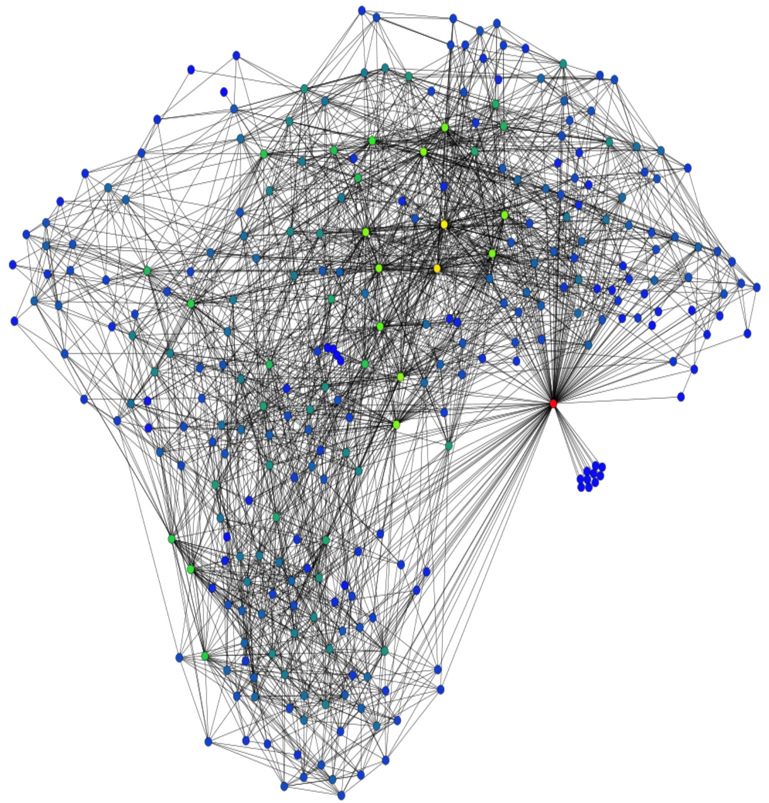
Dr. Ana Lucia Varbanescu
University of Amsterdam, Amsterdam.
a.l.varbanescu@uva.nl

Dr. Gyan Ranjan
Symantec, CA, USA.
gyan_ranjan@symantec.com

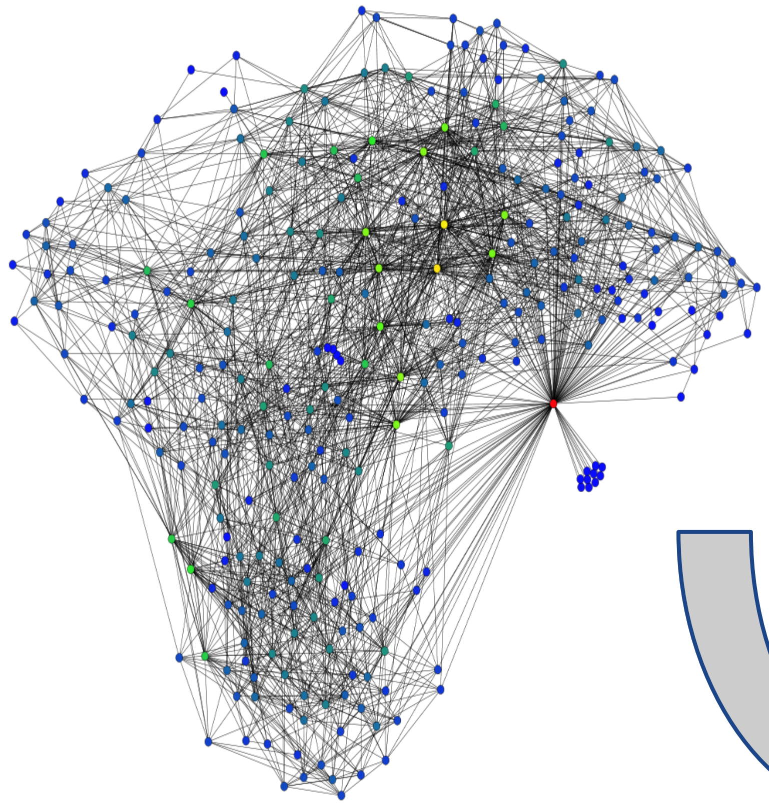
Workshop on Large-Scale Parallel Processing
IEEE International Parallel and Distributed Processing Symposium

May 25th - 29th, 2015

Graphs are Everywhere !



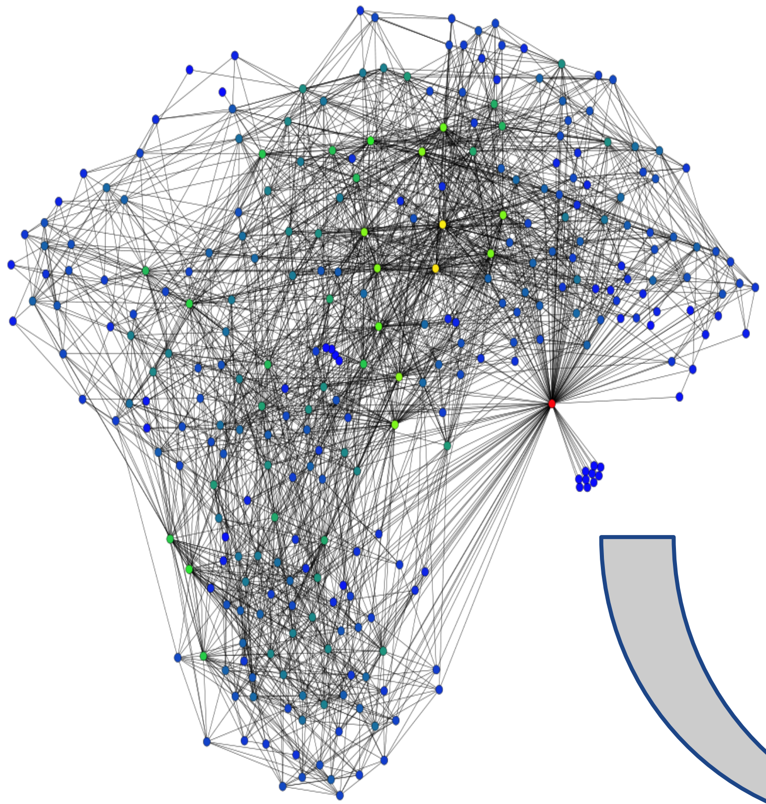
Graphs are Everywhere !



Small World Characteristics

- ➢ Large Scale
- ➢ Local World
- ➢ Degree Distribution
- ➢ Sparse

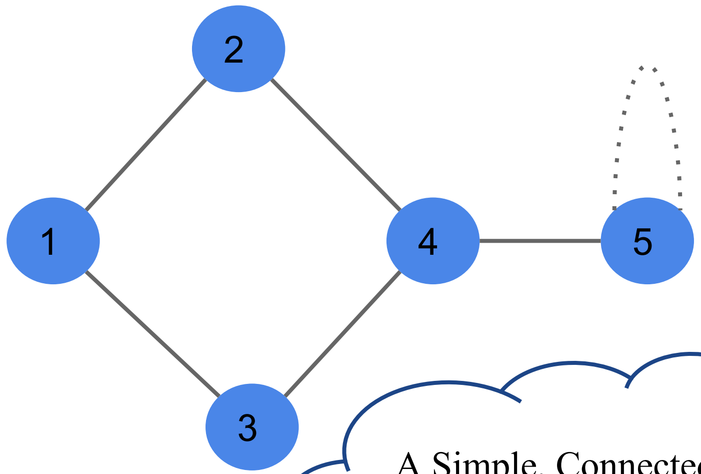
Complex Networks : Define Real World Graphs



Why ?

- ➢ Topological Characteristics
- ➢ Behavioral Predictability

Graph Formalization & Background

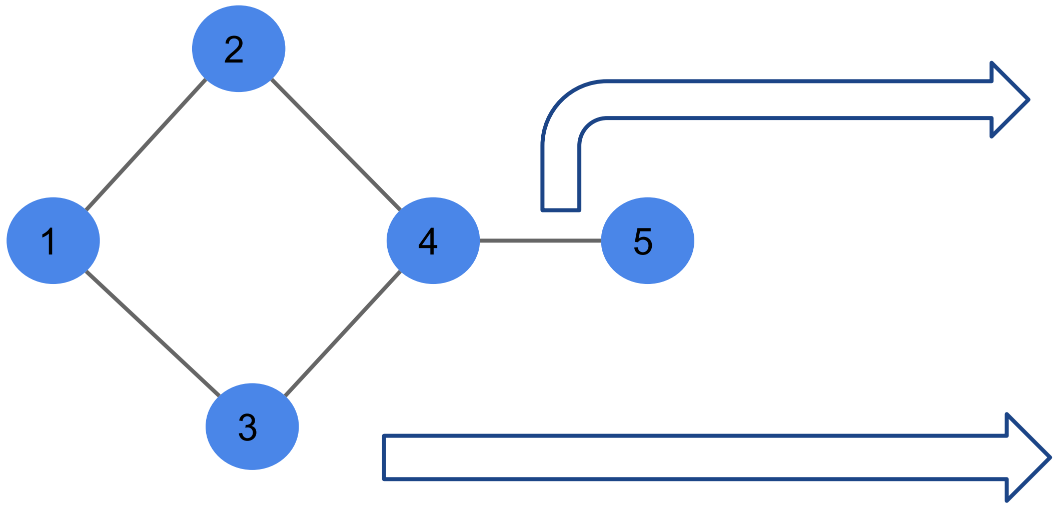


A Simple, Connected, Undirected, Unweighted Graph G(V, E) .

n = |V| is the order of the graph, i.e., the number of vertices.

m = |E| is the size of the graph, i.e., the number of edges.

Graph Formalization & Background

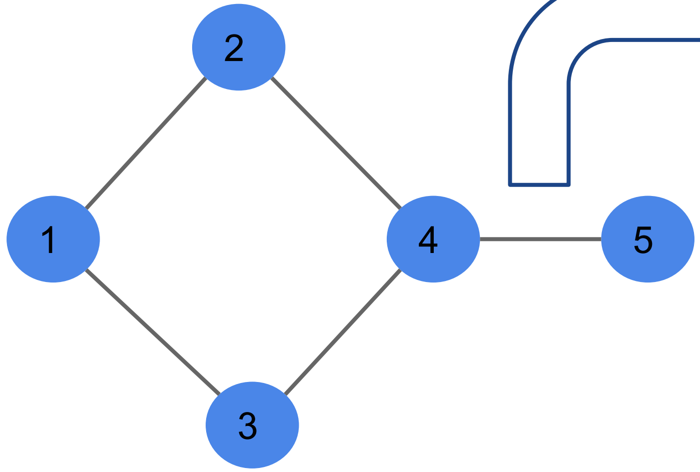


$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

A is Adjacency Matrix , D is Degree Matrix.

Graph Formalization & Background



Degree matrix (D) - Adjacency Matrix (A) = Laplacian Matrix (L)

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 \\ -1 & 0 & 3 & -1 & -1 \\ 0 & -1 & -1 & 2 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

Inverse of Laplacian matrix and Eigenspace

The inverse of Laplacian matrix L is L^{-1} such that :

$$L\,L^{-1} = I$$

where I is Identity matrix and L is a square matrix.

Inverse of Laplacian matrix and Eigenspace

The inverse of Laplacian matrix L is L^{-1} such that :

$$L\,L^{-1} = I$$

where I is Identity matrix and L is a square matrix.

Not Every matrix is invertible.

Inverse of Laplacian matrix and Eigenspace

The inverse of Laplacian matrix L is L^{-1} such that :

$$L\,L^{-1} = I$$

where I is Identity matrix and L is a square matrix.

Not Every matrix is invertible.

The Eigenspace can be formulated as :

$$Lv = \lambda\, v$$

where L is Laplacian Matrix, $\boldsymbol{v}$ is Eigenvector and $\boldsymbol{\lambda}$ is Eigenvalue.

Inverse of Laplacian matrix and Eigenspace

The inverse of Laplacian matrix L is L^{-1} such that :

$$L\,L^{-1} = I$$

where I is Identity matrix and L is a square matrix.

Not Every matrix is invertible.

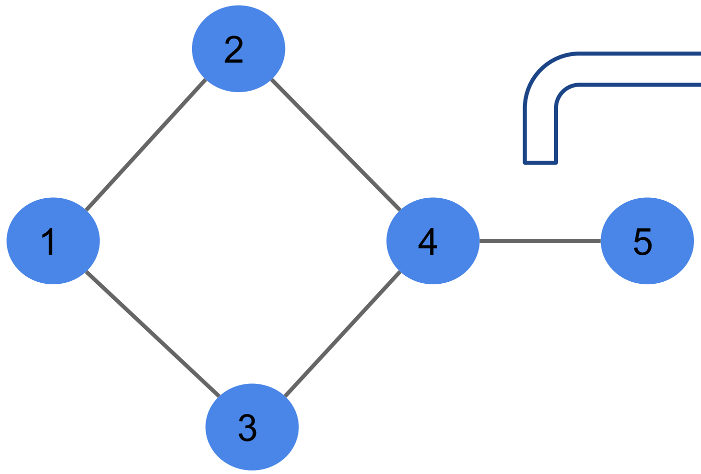
The Eigenspace can be formulated as :

$$Lv = \lambda\, v$$

where L is Laplacian Matrix, $\boldsymbol{v}$ is Eigenvector and $\boldsymbol{\lambda}$ is Eigenvalue.

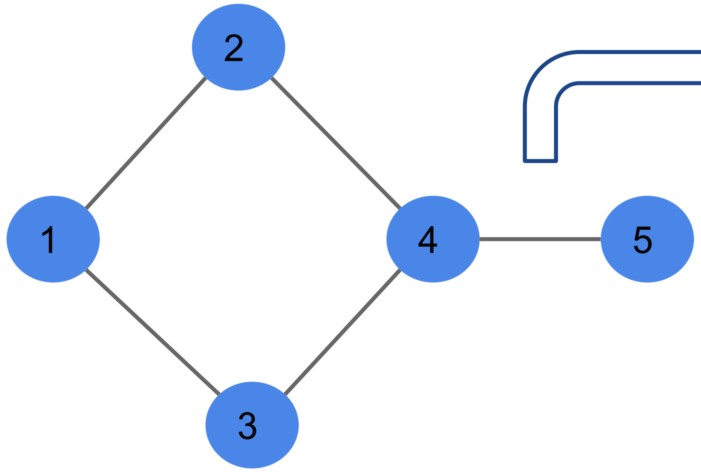
A matrix is not invertible, if any corresponding value of $\boldsymbol{\lambda}$ is 0.

Eigenvalues & Eigenvector



$$L = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 \\ -1 & 0 & 3 & -1 & -1 \\ 0 & -1 & -1 & 2 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

Eigenvalues & Eigenvector



$$L = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 \\ -1 & 0 & 3 & -1 & -1 \\ 0 & -1 & -1 & 2 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

Non - Invertible !

$$\lambda = \left\{ \boxed{0}\ ,\ 0.82\ ,\ 2\ ,\ 2.68\ ,\ 4.4812 \right\}$$

Moore Penrose pseudo-inverse

To calculate the inverse for a rank deficient matrix (L = laplacian matrix, of order n):

Moore Penrose pseudo-inverse

To calculate the inverse for a rank deficient matrix (L = laplacian matrix, of order n):

$$L^{+} = (L + 1/n)^{-1} - 1/n$$

Moore Penrose pseudo-inverse

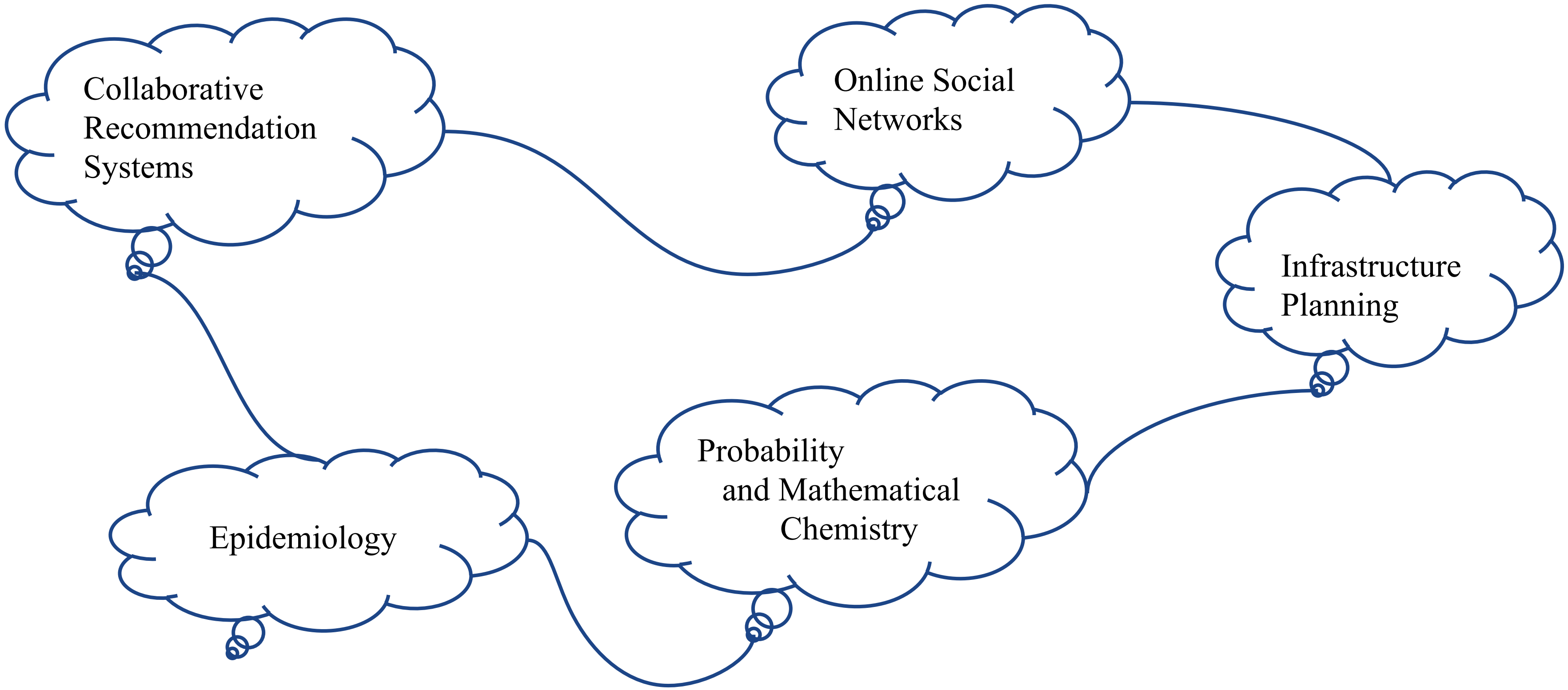
To calculate the inverse for a rank deficient matrix (L = laplacian matrix, of order n):

$L^+ = (L + 1/n)^{-1} - 1/n$

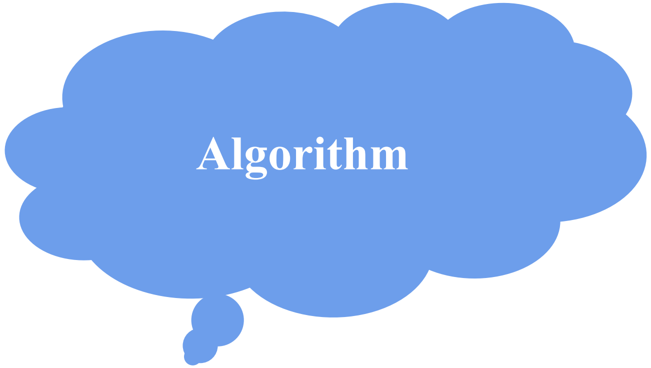
$L^+ =$ pinv(L) in Matlab

$L^+ =$ numpy.linalg.pinv(L) in Python

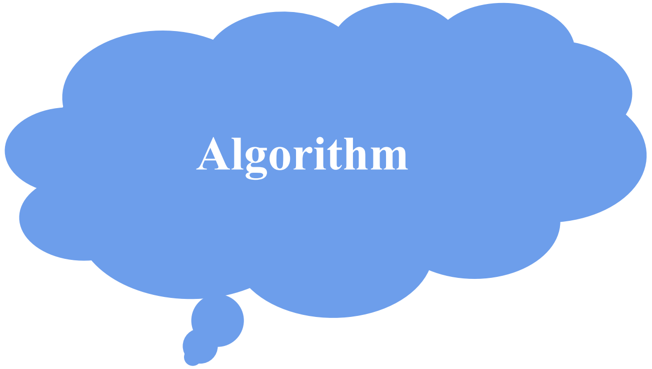
Applications to Computation of L^+


Collaborative Recommendation Systems
Online Social Networks
Infrastructure Planning
Epidemiology
Probability and Mathematical Chemistry

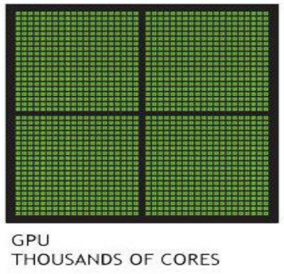
The Goal



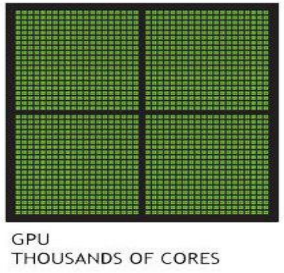
The Goal



+



The Goal

Algorithm + 

$=$ **Speedup ?**

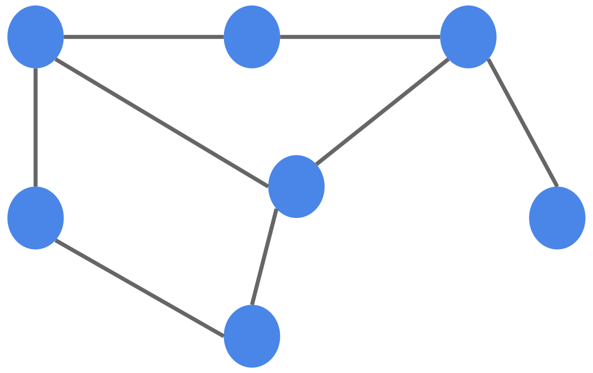
Divide and Conquer Approach to compute L^+

Divide and Conquer Approach to compute L^+

Three Steps

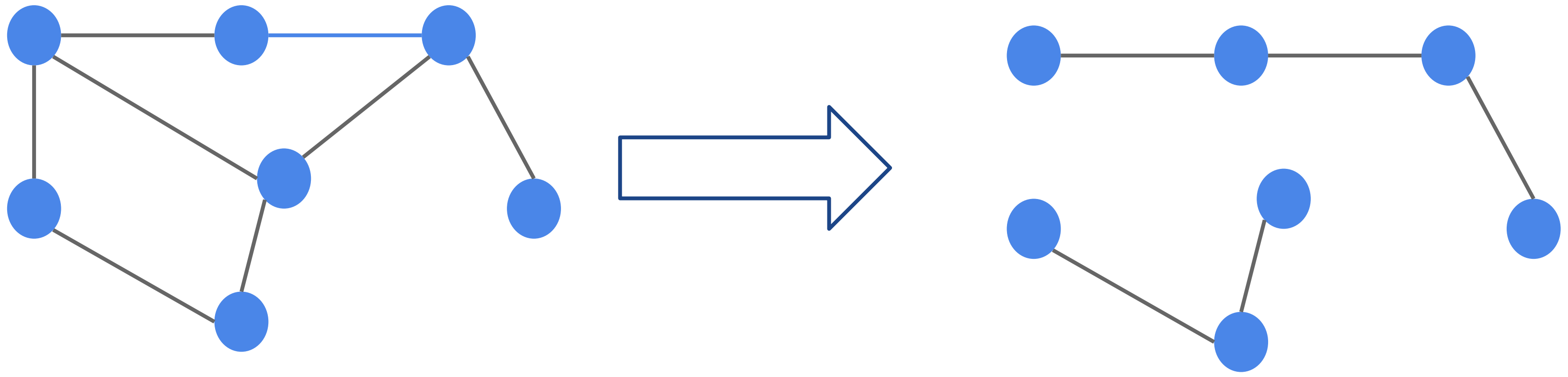
- Partition

Divide : Partition



A simple, connected,unweighted, undirected Graph G.

Divide : Partition



A simple, connected,unweighted, undirected Graph G.

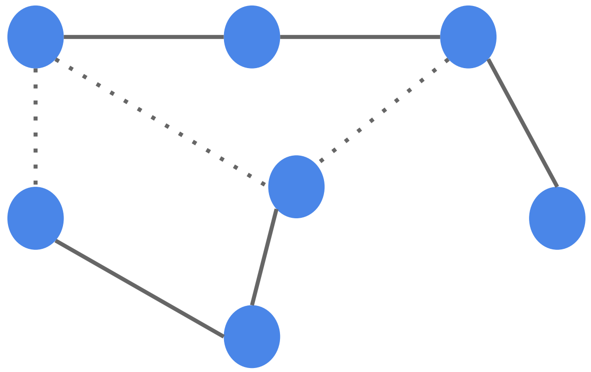
Connected Subgraph G_1 and G_2
Compute L^+_{G1} and L^+_{G2}

Divide and Conquer Approach to compute L^+

Three Steps

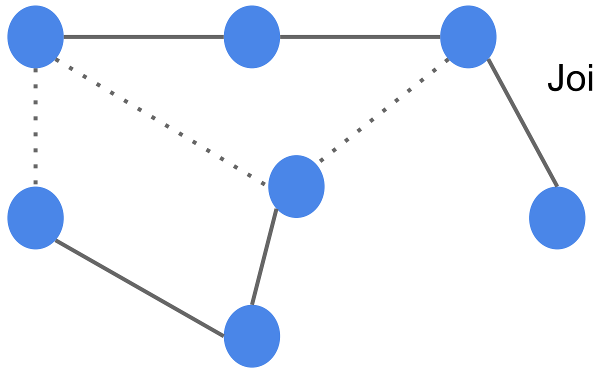
- Partition
- First Join

Conquer : First Join

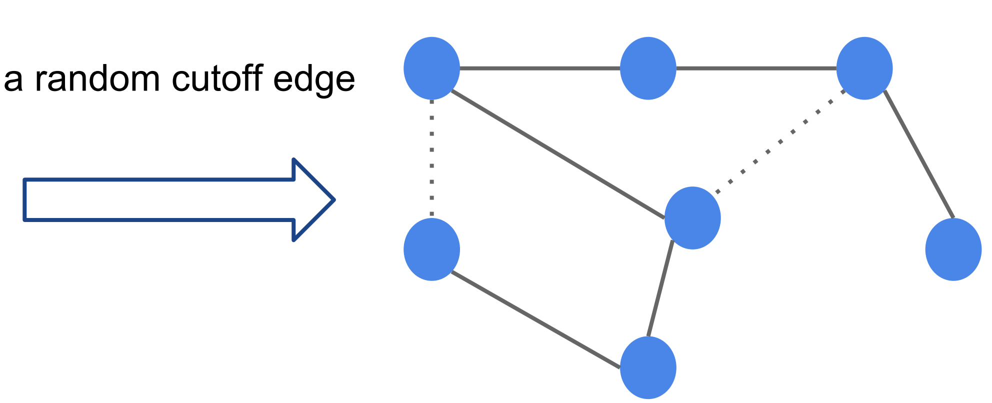


Dotted lines represent minimized cutoff edges during Partition.

Conquer : First Join



Dotted lines represent minimized cutoff edges during Partition.



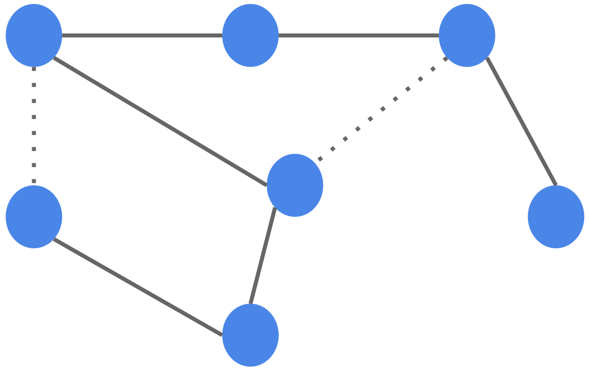
We get a new connected Graph G_3
Using L^+_{G1} and L^+_{G2} , compute L^+_{G3}

Divide and Conquer Approach to compute L^+

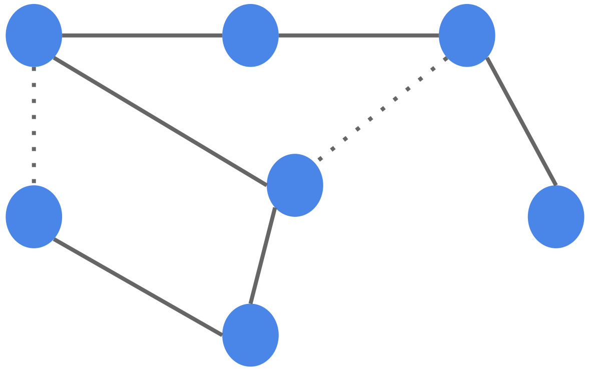
Three Steps

- ➢ Partition
- ➢ First Join
- ➢ Edge Firing

Conquer : Edge Firing

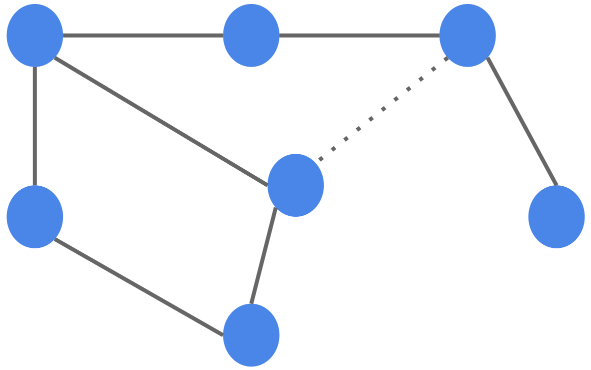


Conquer : Edge Firing

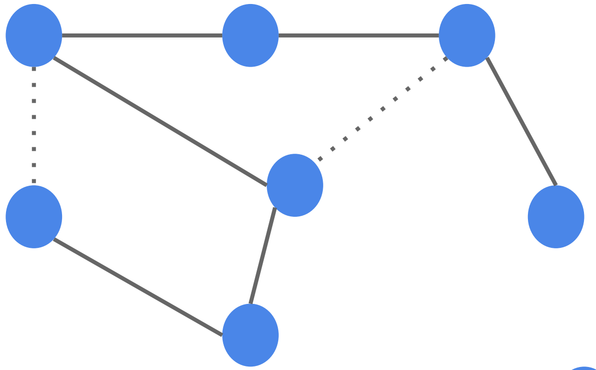


Fire first cutoff Edge

G_4, Compute L^{+}_{G4}

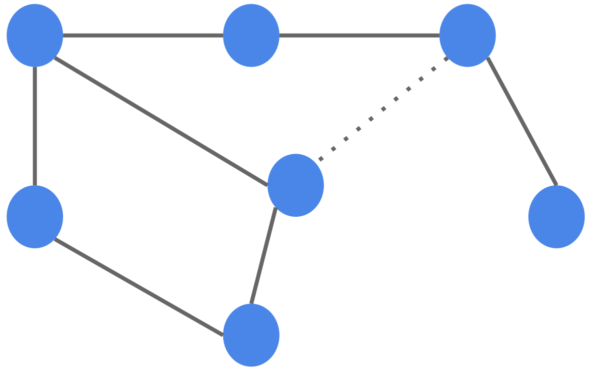


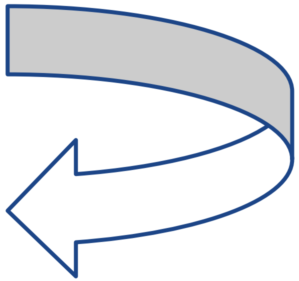
Conquer : Edge Firing



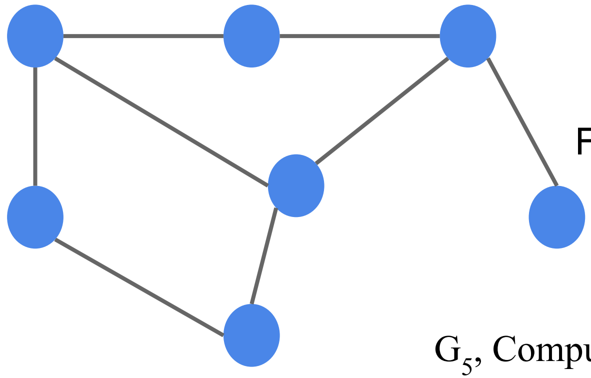
Fire first cutoff Edge

G_4, Compute L^+_{G4}





Fire Second cutoff Edge

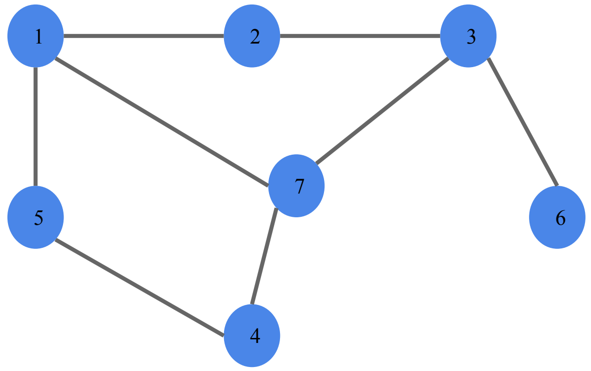


G_5, Compute L^+_{G5}

Methodology

- ➢ Abstract a simple Partition Approach.
- ➢ Use GPU to compute L^+ of the subgraphs.
- ➢ Apply element-wise computation on First Join and Edge Firing using GPU.
- ➢ Minimize Data Transfer.

Implementation Approach - Representation

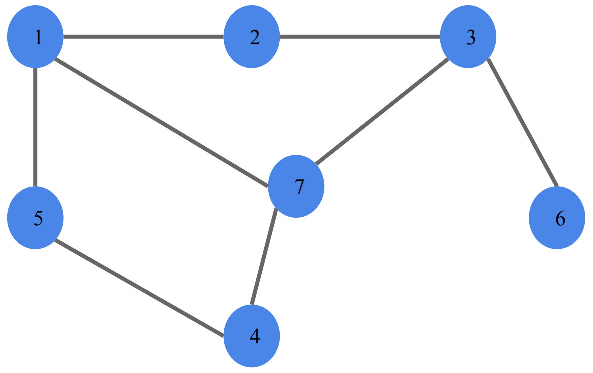


A simple, connected,unweighted, undirected Graph G.

$$\begin{pmatrix} 1, & 2, & 1 \\ 2, & 1, & 1 \\ 1, & 5, & 1 \\ 5, & 1, & 1 \\ 1, & 7, & 1 \\ 7, & 1, & 1 \\ 2, & 3, & 1 \\ 3, & 2, & 1 \\ 3, & 6, & 1 \\ 6, & 3, & 1 \\ 3, & 7, & 1 \\ 7, & 3, & 1 \\ 4, & 7, & 1 \\ 7, & 4, & 1 \\ 4, & 5, & 1 \\ 5, & 4, & 1 \end{pmatrix}$$

Sparse Representation

Implementation Approach - Partition

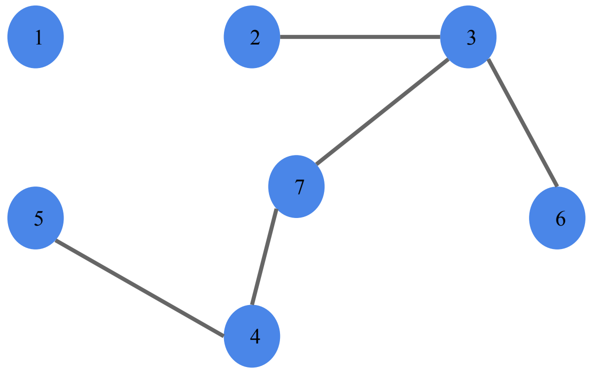


A simple, connected,unweighted, undirected Graph G.

Node	Degree
1	3
3	3
7	3
2	2
4	2
5	2
6	1

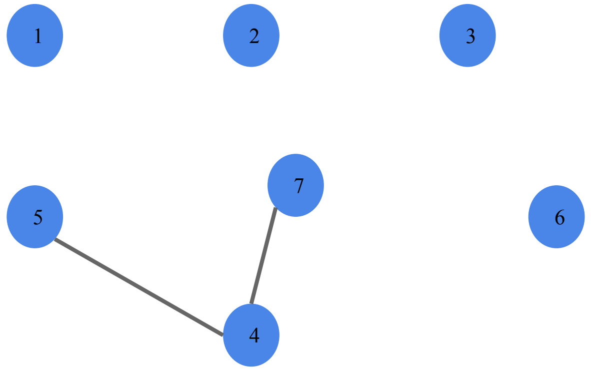
Implementation Approach - Partition

Remove edges of node 1



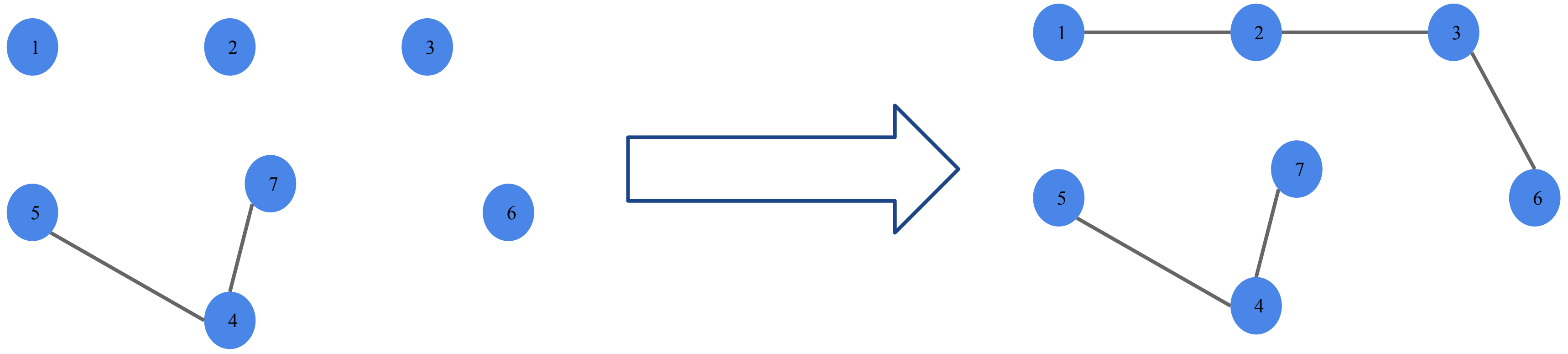
Largest Component has order 6

Remove edges of node 3



Largest Component has order 3 , which is <= n / 2 (n = 7). But we wanted two connected components, and there are some isolated ones.

Implementation Approach - Recombine



We get a bipartition , with two simple, connected components

Parallel Implementation - First Join & Edge Firing

1. ***CPU Based Parallel Implementation*** :

★ Parallelisation using pthreads
★ *inverse (L + 1/n) - 1/n*
★ *dgetri.f* Blas routine
★ First Join and Edge Firing
★ 4 threads generated

Parallel Implementation - First Join & Edge Firing

1. ***CPU Based Parallel Implementation*** **:**

★ Parallelisation using pthreads
★ *inverse (L + 1/n) - 1/n*
★ *dgetri.f* Blas routine
★ First Join and Edge Firing
★ 4 threads generated

2. ***Matlab based GPU Implementation*** **:**

★ Parallel Computing Toolbox
★ *inv( )* , GPU enabled function
★ First Join and Edge Firing
★ *bsxfun( )*

Parallel Implementation - First Join & Edge Firing

1. ***CPU Based Parallel Implementation* :**

- ★ Parallelisation using pthreads
- ★ *inverse (L + 1/n) - 1/n*
- ★ *dgetri.f* Blas routine
- ★ First Join and Edge Firing
- ★ 4 threads generated

2. ***Matlab based GPU Implementation* :**

- ★ Parallel Computing Toolbox
- ★ *inv( )* , GPU enabled function
- ★ First Join and Edge Firing
- ★ *bsxfun( )*

3. ***CUDA based GPU Implementation* :**

- ★ Thrust Library
- ★ *inverse* cuBlas library routine
- ★ First Join - Three Device Kernels
- ★ Edge Firing - Single Kernel
- ★ 256 threads generated per block

Parallel Implementation - First Join & Edge Firing

1. ***CPU Based Parallel Implementation*** **:**

★ Parallelisation using pthreads
★ *inverse (L + 1/n) - 1/n*
★ *dgetri.f* Blas routine
★ First Join and Edge Firing
★ 4 threads generated

2. ***Matlab based GPU Implementation*** **:**

★ Parallel Computing Toolbox
★ *inv( )* , GPU enabled function
★ First Join and Edge Firing
★ *bsxfun( )*

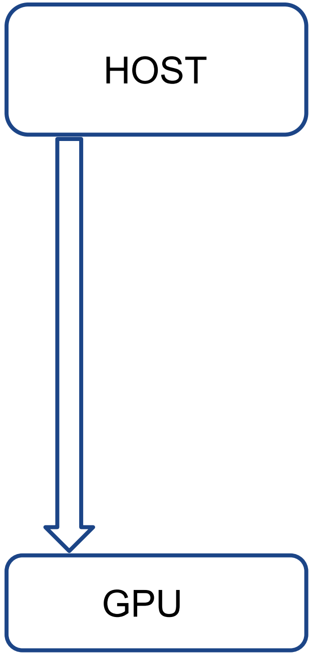
3. ***CUDA based GPU Implementation*** **:**

★ Thrust Library
★ *inverse* cuBlas library routine
★ First Join - Three Device Kernels
★ Edge Firing - Single Kernel
★ 256 threads generated per block

4. ***cuBlas Implementation -Baselining the performance*** **:**

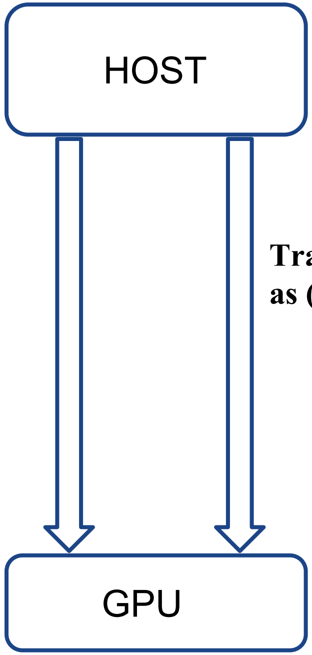
★ *inverse (L + 1/n) - 1/n*
★ *cublas<t>getriBatched* routine
★ *cublas<t>getrfBatched* routine

Implementation - First Join Using GPU



Transfers , the sub-graphs as full Matrix of order n1 & n2.
Here n1 = 2 and n2 = 3

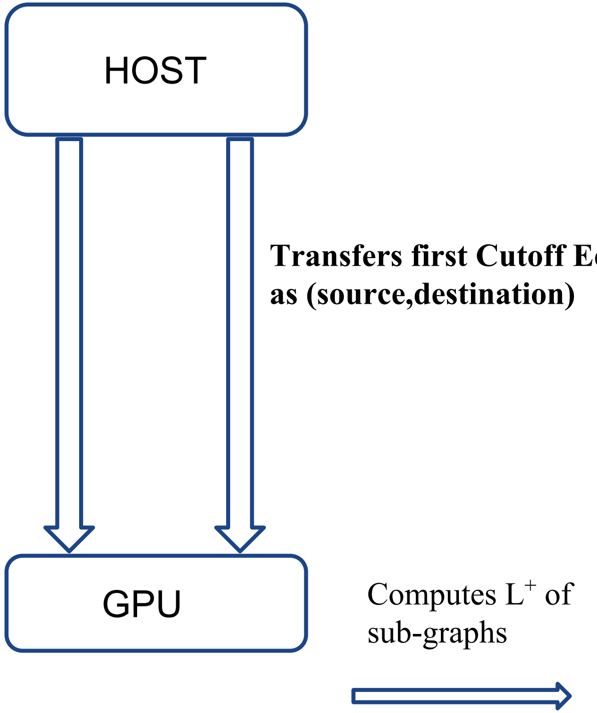
Implementation - First Join Using GPU



Transfers , the sub-graphs as full Matrix of order n1 & n2.
Here n1 = 2 and n2 = 3

Transfers first Cutoff Edge as (source,destination)

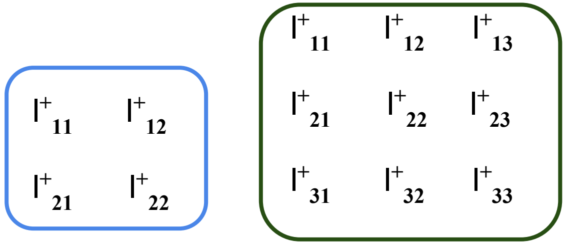
Implementation - First Join Using GPU



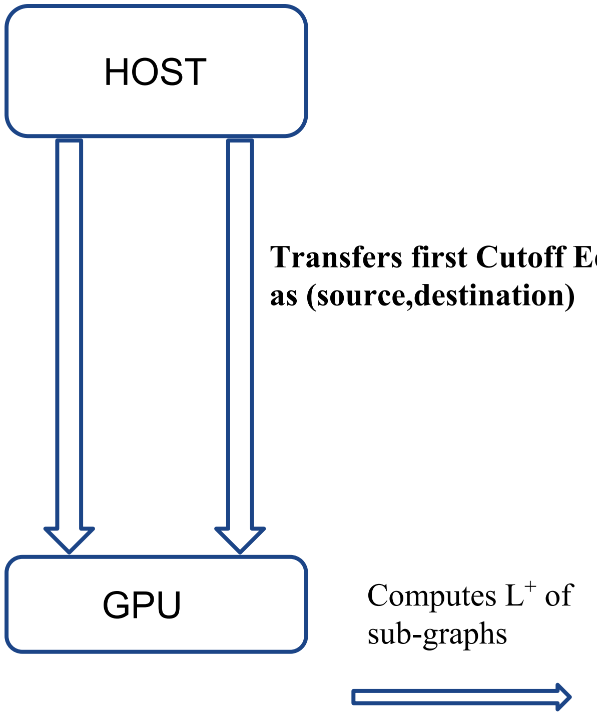
Transfers , the sub-graphs as full Matrix of order n1 & n2. Here n1 = 2 and n2 = 3

Transfers first Cutoff Edge as (source,destination)

Computes L^+ of sub-graphs


l^+_{11} l^+_{12}
l^+_{21} l^+_{22}
l^+_{11} l^+_{12} l^+_{13}
l^+_{21} l^+_{22} l^+_{23}
l^+_{31} l^+_{32} l^+_{33}

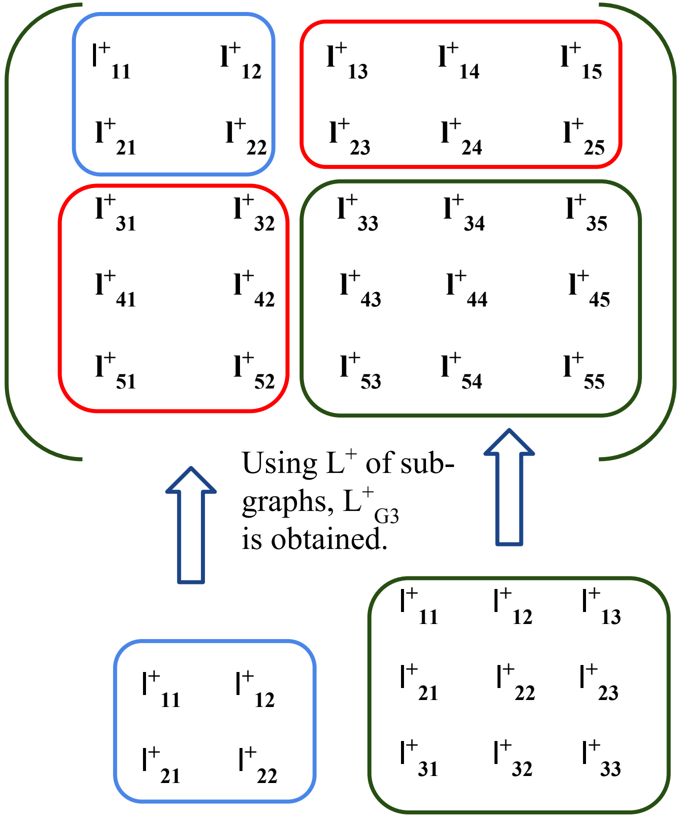
Implementation - First Join Using GPU



Transfers , the sub-graphs as full Matrix of order n1 & n2. Here n1 = 2 and n2 = 3

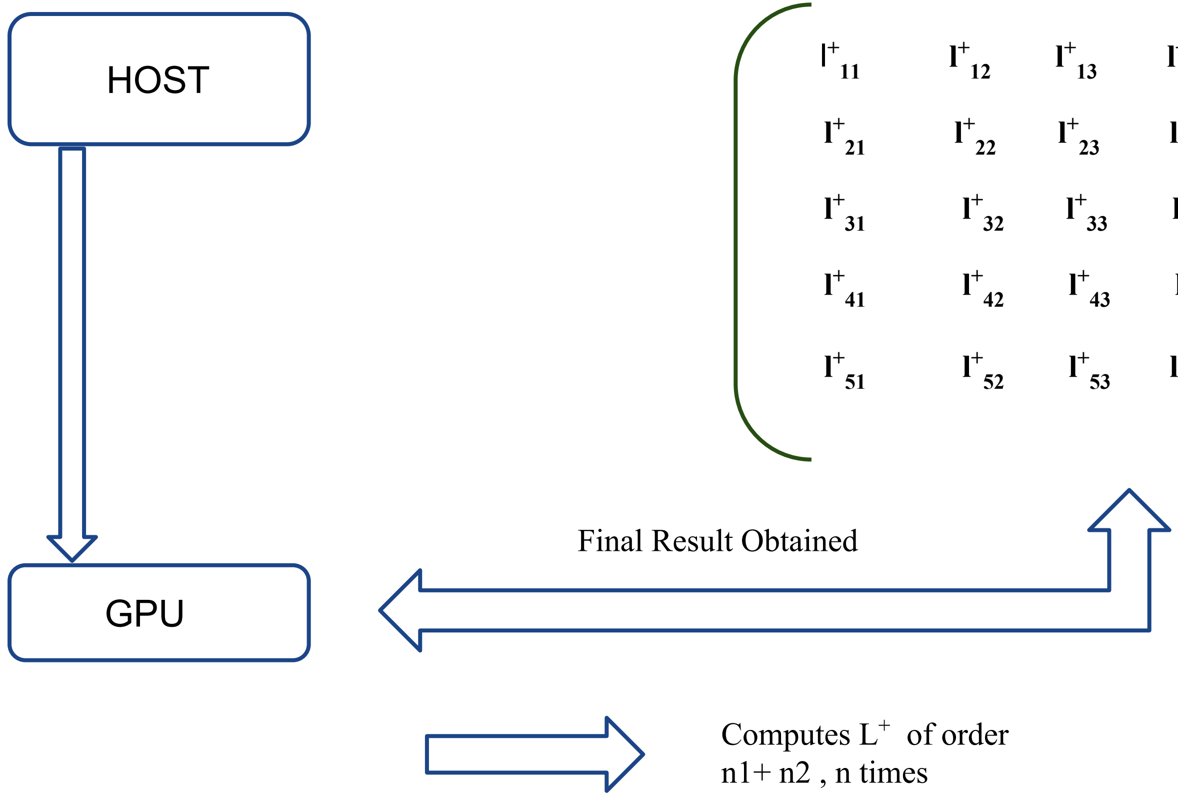
Transfers first Cutoff Edge as (source,destination)

Computes L^+ of sub-graphs


l^+_{11} l^+_{12} l^+_{13} l^+_{14} l^+_{15}
l^+_{21} l^+_{22} l^+_{23} l^+_{24} l^+_{25}
l^+_{31} l^+_{32} l^+_{33} l^+_{34} l^+_{35}
l^+_{41} l^+_{42} l^+_{43} l^+_{44} l^+_{45}
l^+_{51} l^+_{52} l^+_{53} l^+_{54} l^+_{55}
l^+_{11} l^+_{12}
l^+_{21} l^+_{22}
l^+_{11} l^+_{12} l^+_{13}
l^+_{21} l^+_{22} l^+_{23}
l^+_{31} l^+_{32} l^+_{33}

Using L^+ of sub-graphs, L^+_{G3} is obtained.

Implementation - Edge Firing Using GPU



Remaining cutoff edges transferred as (source, destination), N number of times. N is the number of edges fired after first Join.

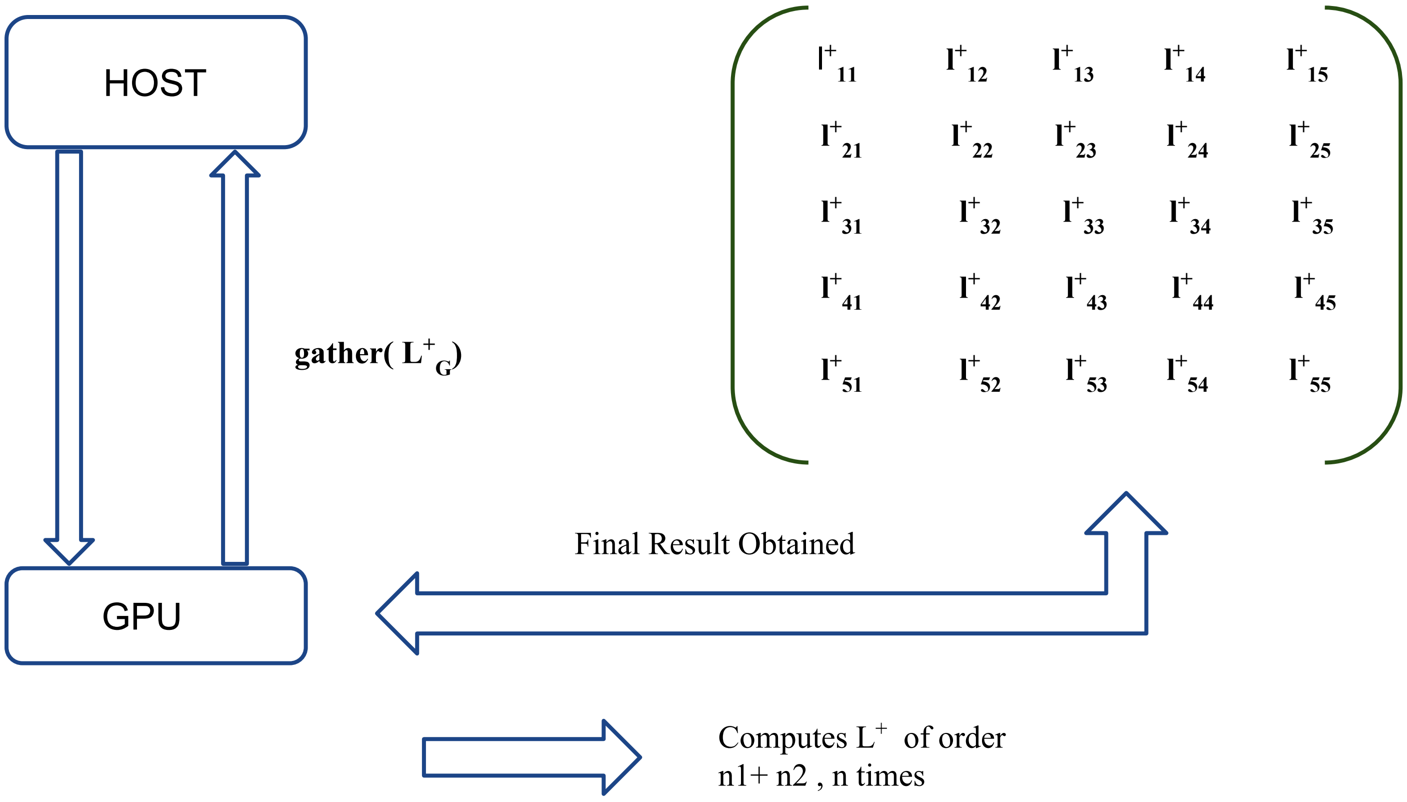
$$\begin{pmatrix} l^{+}_{11} & l^{+}_{12} & l^{+}_{13} & l^{+}_{14} & l^{+}_{15} \\ l^{+}_{21} & l^{+}_{22} & l^{+}_{23} & l^{+}_{24} & l^{+}_{25} \\ l^{+}_{31} & l^{+}_{32} & l^{+}_{33} & l^{+}_{34} & l^{+}_{35} \\ l^{+}_{41} & l^{+}_{42} & l^{+}_{43} & l^{+}_{44} & l^{+}_{45} \\ l^{+}_{51} & l^{+}_{52} & l^{+}_{53} & l^{+}_{54} & l^{+}_{55} \end{pmatrix}$$

Final Result Obtained

Computes L^{+} of order n1+ n2 , n times

Implementation - Edge Firing Using GPU

Remaining cutoff edges transferred as (source, destination), N number of times. N is the number of edges fired after first Join.



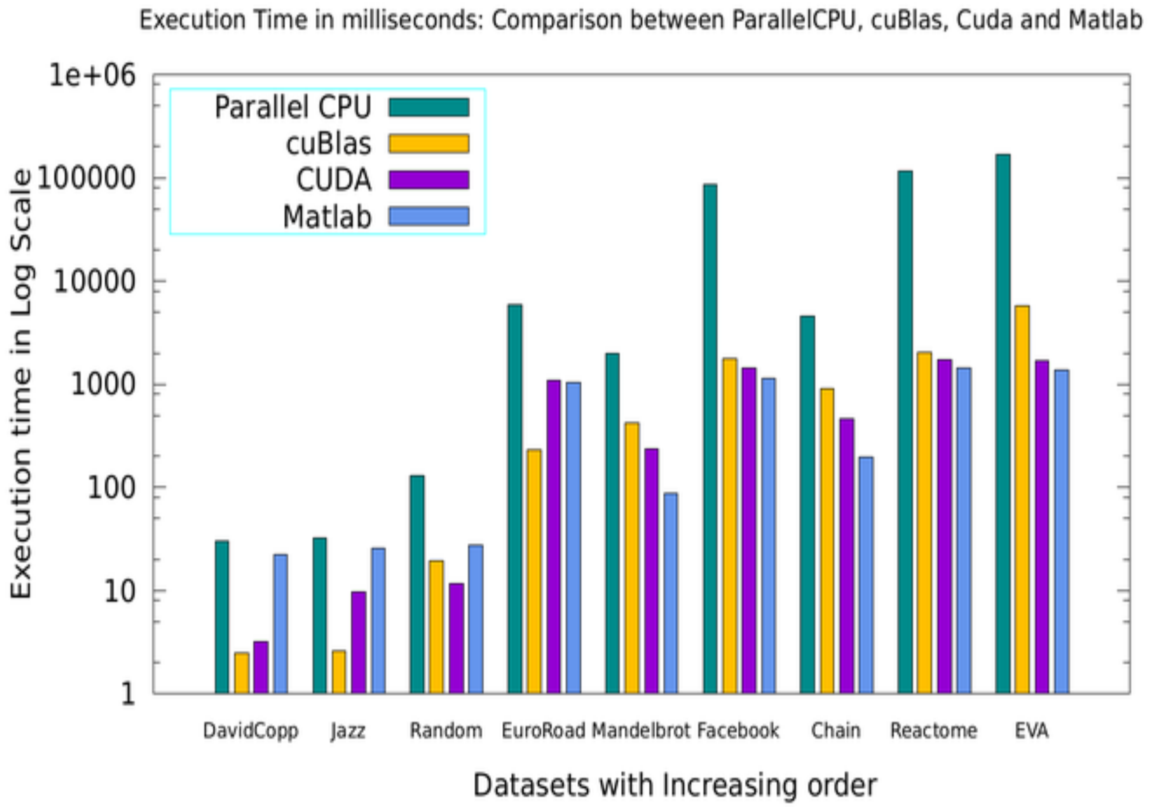
$$\begin{pmatrix} l^{+}_{11} & l^{+}_{12} & l^{+}_{13} & l^{+}_{14} & l^{+}_{15} \\ l^{+}_{21} & l^{+}_{22} & l^{+}_{23} & l^{+}_{24} & l^{+}_{25} \\ l^{+}_{31} & l^{+}_{32} & l^{+}_{33} & l^{+}_{34} & l^{+}_{35} \\ l^{+}_{41} & l^{+}_{42} & l^{+}_{43} & l^{+}_{44} & l^{+}_{45} \\ l^{+}_{51} & l^{+}_{52} & l^{+}_{53} & l^{+}_{54} & l^{+}_{55} \end{pmatrix}$$

Experiments & Results

Hardware Platform :

- ★ NVIDIA Tesla K20m GPU from DAS4 , a six-cluster wide-area distributed system designed and used by 5 research institutions in The Netherlands.
- ★ 5GB of GPU global memory, Memory bandwidth of 208 GB/sec, and a peak performance of 3520 GFlops (single precision).
- ★ CUDA 5.5 , Matlab R2014a
- ★ CPU experiments (sequential and parallel) - performed on DAS4 computing node, using dual-quad-core 2.4 GHz CPU configuration and 24GB memory.

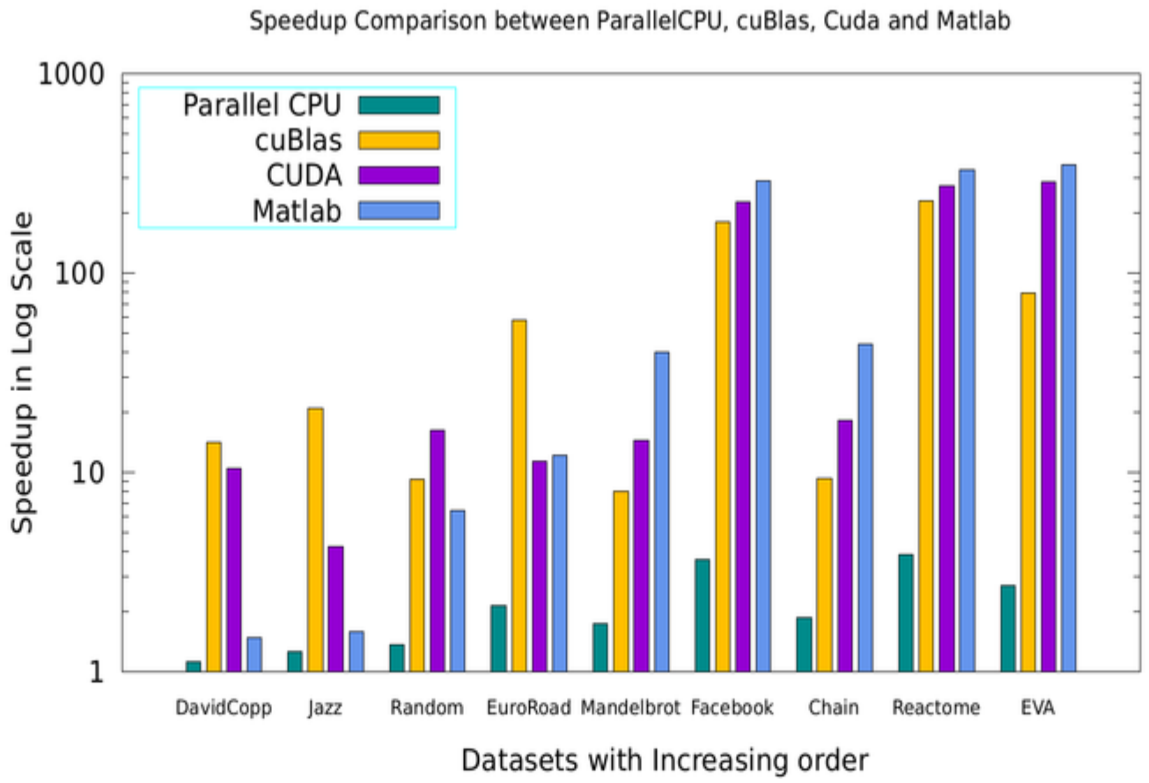
Experiments & Results



Finding :

- Different behavior for different graphs.
- cuBlas and CUDA better for smaller graphs.
- Matlab suitable for graphs of large order.
- Divide and Conquer approach versus $(L + 1/n)^{-1} - 1/n$

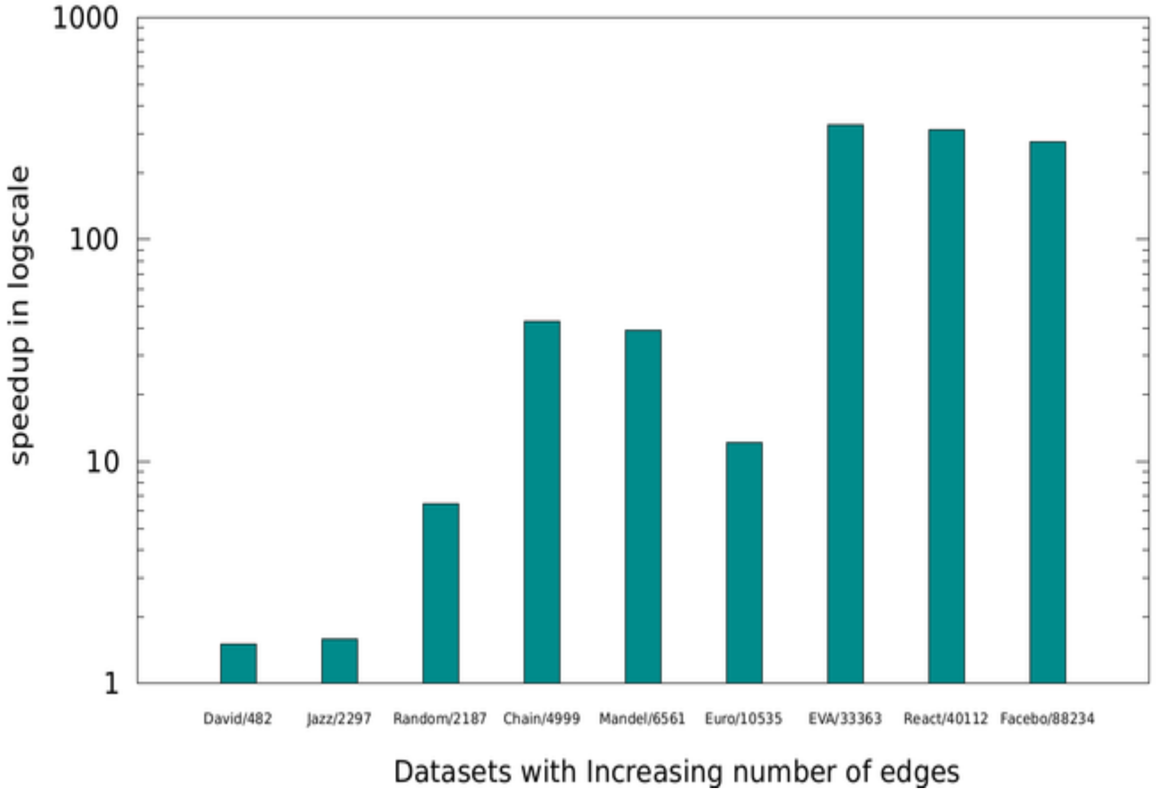
Experiments & Results



Finding :

- Speedup achieved up-to 300 times.
- Matlab - Speedup - better for large order graphs.

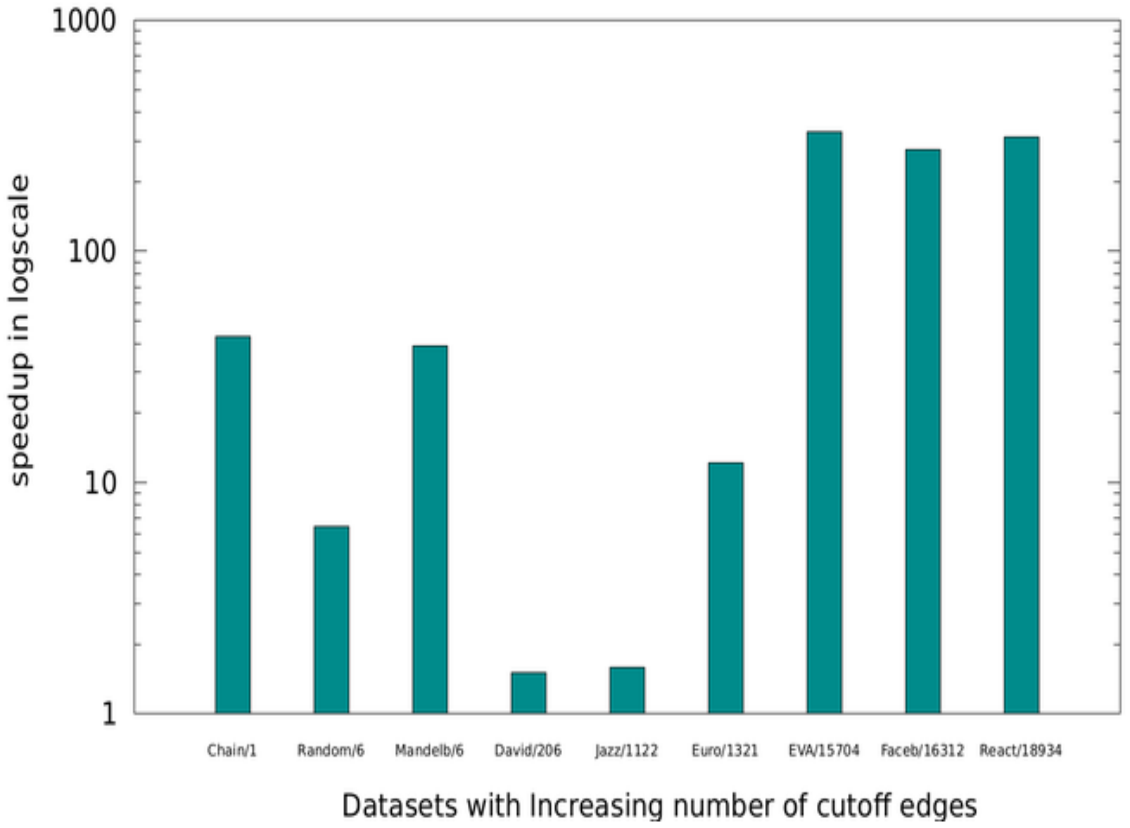
Experiments & Results



No Correlation Found !

Investigate more parameters !

Experiments & Results



No Correlation Found !

Future Work : Investigate more parameters.

Contributions

★ Designed a parallel version of the Divide-and-Conquer computation of the Moore-Penrose pseudo-inverse of the Laplacian .

★ Designed a GPU-enabled version of this parallel solution, and implemented it in Matlab, with significant speedup.

★ Implemented three other parallel versions, one using CUDA, one using cuBLAS, and a pThreads-based version.

★ Empirical evidence that the performance of three GPU-enabled versions is heavily dependent on the input graph properties.

Conclusion & Future Work

Conclusion **:**

- ❑ cuBlas and CUDA - small order graphs.
- ❑ Matlab implementation outperforms cuBlas and CUDA for large order graphs.
- ❑ Divide and Conquer Approach - Large Graphs - Significant Performance.
- ❑ Matlab GPU Computing - Productivity and Performance.
- ❑ Performance Variation - Input Graph

Conclusion & Future Work

***Future Work* :**

- ★ Multiple GPU’s , Recursive Partitioning.
- ★ Spanning Trees to Compute L^+.
- ★ Investigate parameters of the graph affecting performance,

Thank you !

Any Questions ?

Nishant Saurabh
nishants.prmitr7@gmail.com

Workshop on Large-Scale Parallel Processing
IEEE International Parallel and Distributed Processing Symposium

Back-up Slides

Nishant Saurabh
Vrije Universiteit Amsterdam.

Dr. Ana Lucia Varbanescu
University of Amsterdam Amsterdam.

Dr. Gyan Ranjan
Symantec CA USA.

Workshop on Large-Scale Parallel Processing
IEEE International Parallel and Distributed Processing Symposium

May 25th - 29th, 2015

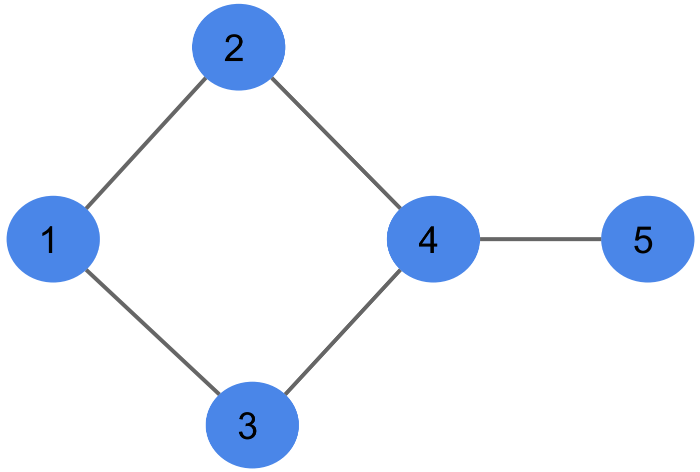
Computational Formula

Operations	**Ω**	**L⁺**
First Join	$x, y \in G_1 : \Omega^{G3}_{xy} = \Omega^{G1}_{xy}$	$l^{+(1)}_{xy} - \dfrac{n_2 n_3 (l^{+(1)}_{xi} + l^{+(1)}_{iy}) - n^2_2 (l^{+(1)}_{ii} + l^{+(2)}_{jj} + \omega_{ij})}{n^2_3}$
	$x, y \in G_2 : \Omega^{G3}_{xy} = \Omega^{G2}_{xy}$	$l^{+(2)}_{xy} - \dfrac{n_1 n_3(l^{+(2)}_{xj} + l^{+(2)}_{jy}) - n^2_1 (l^{+(1)}_{ii} + l^{+(2)}_{jj} + \omega_{ij})}{n^2_3}$
	$x \in G_1, y \in G_2 : \Omega^{G3}_{xy} = \Omega^{G1}_{xi} + \omega_{ij} + \Omega^{G1}_{jy}$	$\dfrac{n_3(n_1 l^{+(1)}_{xi} + n_2 l^{+(2)}_{jy}) - n_1 n_2(l^{+(1)}_{ii} + l^{+(2)}_{jj} + \omega_{ij})}{n^2_3}$

Computational Formula

Operations	**Ω**	**L^{+}**
Edge Firing	$\Omega^{G1}_{xy} - \dfrac{[(\Omega^{G1}_{xj} - \Omega^{G1}_{xi}) - (\Omega^{G1}_{jy} - \Omega^{G1}_{iy})]^{2}}{4(\omega_{ij} + \Omega^{G1}_{ij})}$	$l^{+(1)}_{xy} - \dfrac{(l^{+(1)}_{xi} + l^{+(1)}_{xj})(l^{+(1)}_{iy} + l^{+(2)}_{jy})}{\omega_{ij} + \Omega^{G1}_{ij}}$

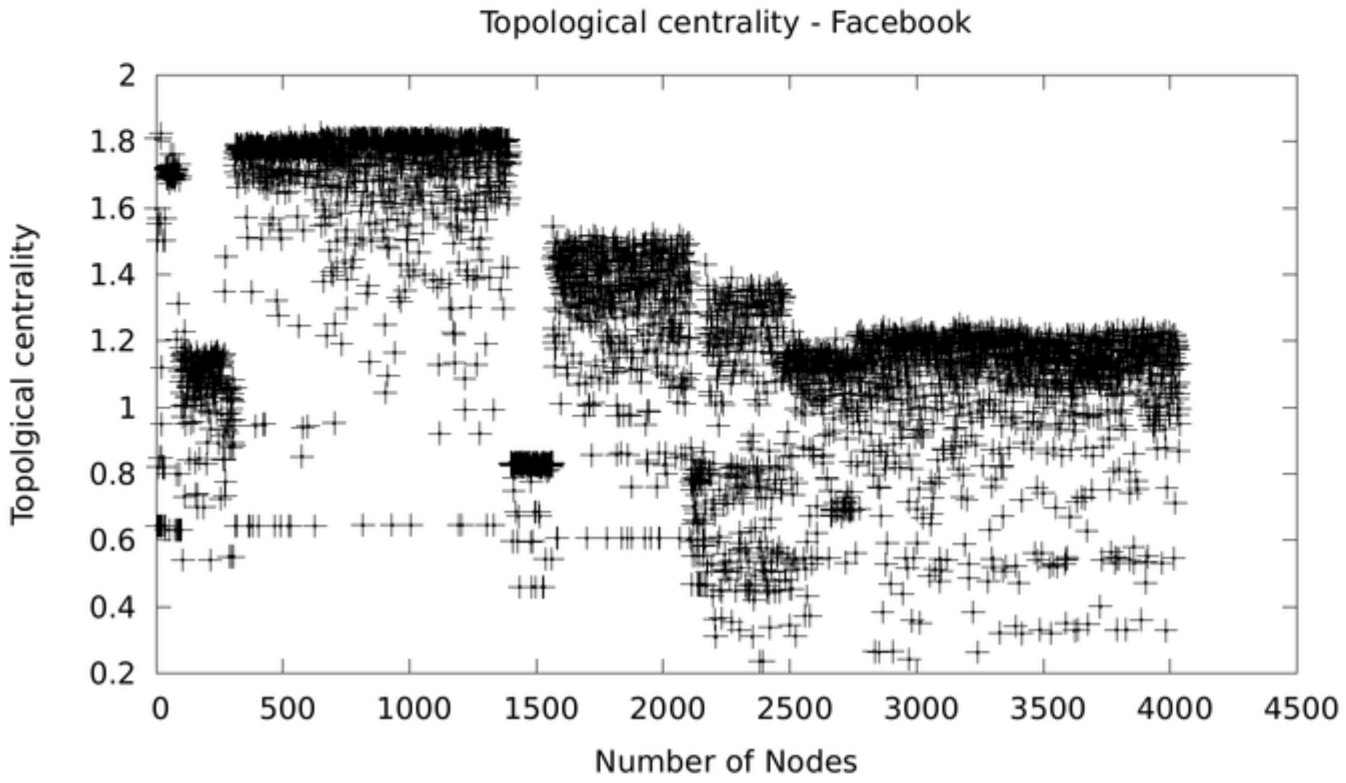
Case Study - Topological Centrality



$$\begin{pmatrix} \boxed{l^+_{11}} & l^+_{12} & l^+_{13} & l^+_{14} & l^+_{15} \\ l^+_{21} & \boxed{l^+_{22}} & l^+_{23} & l^+_{24} & l^+_{25} \\ l^+_{31} & l^+_{32} & \boxed{l^+_{33}} & l^+_{34} & l^+_{35} \\ l^+_{41} & l^+_{42} & l^+_{43} & \boxed{l^+_{44}} & l^+_{45} \\ l^+_{51} & l^+_{52} & l^+_{53} & l^+_{54} & \boxed{l^+_{55}} \end{pmatrix}$$

Topological Centrality (C^*_i) = $1/ l^+_{ii}$, where l^+_{ii} is general term of L^+ for node i

Experiments & Results


Topological centrality - Facebook
Topolgical centrality
Number of Nodes