A FINE-GRAINED HYBRID CPU-GPU ALGORITHM FOR BETWEENNESS CENTRALITY COMPUTATIONS

[1]Ashirbad Mishra, [1]Sathish Vadhiyar, [2]Rupesh Nasre, [3,4]Keshav Pingali

[1]Computational and Data Sciences, Indian Institute of Science, Bangalore, India
[2]Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India
[3]Institute for Computational Engineering and Sciences, University of Texas at Austin, USA
[4]Department of Computer Science, University of Texas at Austin, USA
ashirbad@grads.cds.iisc.ac.in, vss@cds.iisc.ac.in (http://mars.serc.iisc.ernet.in/), rupesh@cse.iitm.ac.in, pingali@cs.utexas.edu

MOTIVATION AND CONTRIBUTION

Introduction

- Centrality metrics like Betweenness quantify importance of a node with respect to the entire network.
- It is used on social networks, studying structures in biological networks and for identifying hubs in transportation networks.
- Computing exact Betweenness of a network is computationally expensive.

Our Method

- We developed a novel fine-grained CPU-GPU hybrid algorithm for Betweenness Centrality that harnesses the multi-source property of BC computations.
- Our hybrid approach gives 80% improvement in performance, and 80-90% less CPU-GPU communications w.r.t Totem's hybrid strategy based on the BSP approach.

BETWEENESS AND BRANDES ALGORITHM

Betweenness Score Calculation

- For a graph *G = (V,E),* where *V* is the set of vertices and *E*, the set of edges. Let σ_{st} denote the number of shortest path from vertex *'s'* to vertex *'t'*, where $s \neq t$, and $\sigma_{st}(v)$ *denote the number of such paths* passing through vertex *'v'*.
- So $\delta_{st}(v) = \sigma_{st}(v) / \sigma_{st}$, where $\delta_{st}(v)$ denotes the pair-wise dependency between of the pair *'s'* and *'t'* on the vertex *'v.'*
- The Betweenness Centrality score of the vertex is given by

$$BC(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v)$$

Brandes Algorithm

- The **forward phase** consists of a BFS or SSSP traversal with *'s'* as the source. For each vertex the number of shortest paths and predecessor list is calculated.
  - The number of shortest paths for 'v' from source 's' is denoted as σ_{sv}.
  - 'u' is a predecessor of 'v' if 'u' lies on the shortest path from 's' to 'v'.
- **The backward phase** traverses the vertices in descending order of their distance from *'s'*. The dependency $\delta_s(v)$ is calculated by

$$\delta_s(v) = \sum_{u: v \in Ps(u)} \frac{\sigma_{sv}}{\sigma_{su}} (1 + \delta_s(u))$$

, where $P_s(u)$ is the predecessor's list of *u*:

- At the end of backward phase, the Betweenness score of each vertex 'v' is calculated by

$$BC[v] = \sum_{s \neq v \in V} \delta_s(v)$$

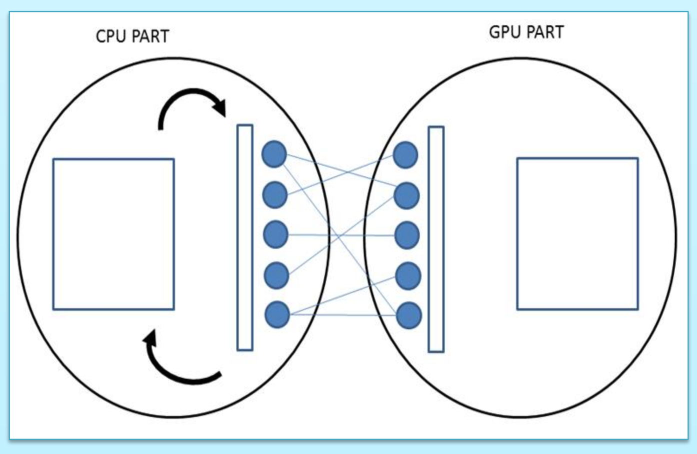
HYBRID CPU-GPU ALGORITHM

We try to find the shortest paths that lie across the partitions. The rest of shortest paths can be found out asynchronously in each partition.

Initialization phase:

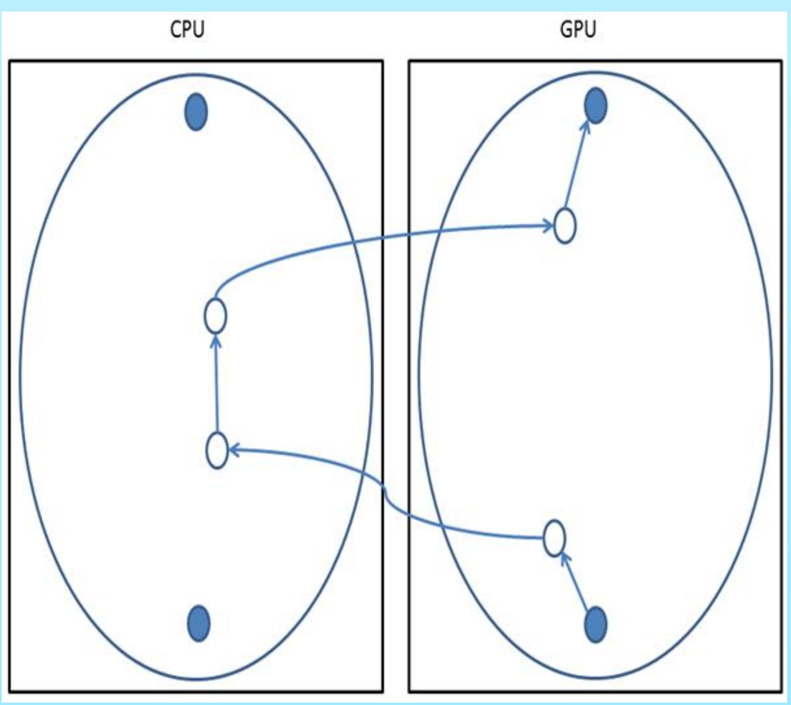
- The graph is partitioned between CPU and GPU in a flexible ratio based on their processing power.
- A border matrix is computed for each part which stores the relative distance between each pair of border vertex in that partition.

Forward phase:



- **Initial step**: A source 's' is selected; BFS/SSSP is performed in the same partition as 's'. A border vector of distance values is stored.
- **Iterative step**: The border vector and the border matrix from the initialization phase are used to find the exact distances values.
- **Relaxation step**: The distances of the border vertices is used to asynchronously compute the distances of all the vertices in each partition without communication.

Backward phase:



- In each partition the CPU/GPU will start the backward phase from the minimum level in the partition.
- Each partition will wait if it finds a border vertex at the current level which requires data in the other part.
- The partition will then request the data for that vertex without hindering the execution of the other partition.

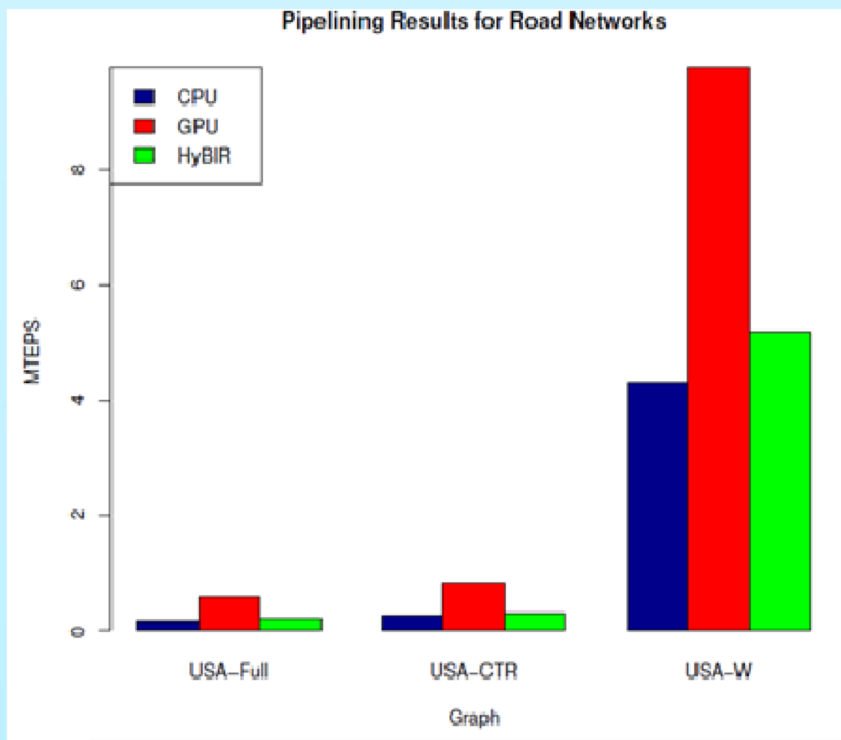
EXPERIMENTATION AND RESULTS

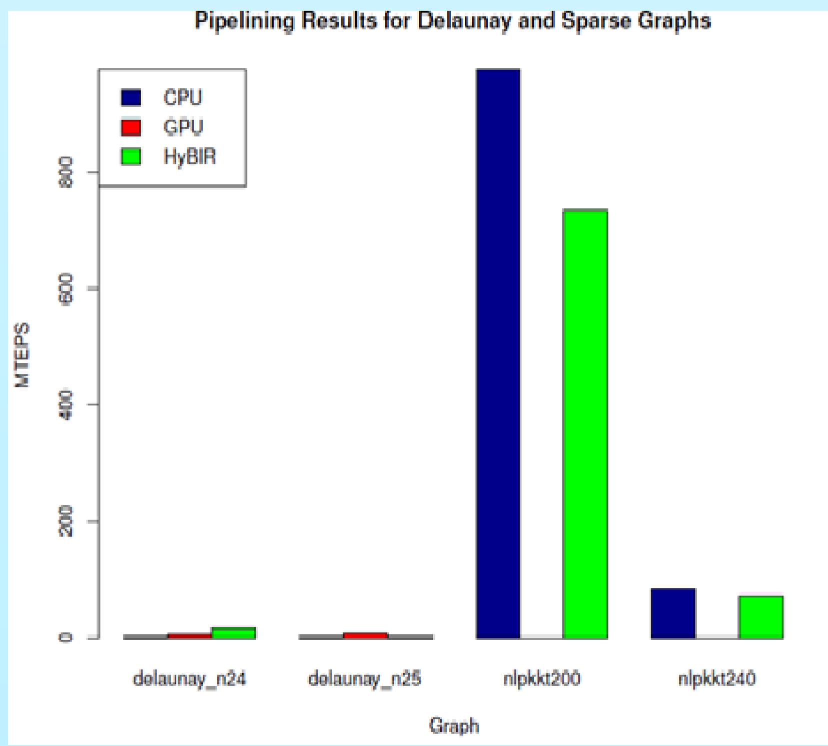
Setup

- We perform experiment on a heterogeneous system with a dual octo-core Intel Xeon and a NVIDIA Kepler K20.
- We used directed graphs from datasets of 10th Dimacs challenge, The University of Florida Sparse Matrix Collection and the Stanford Network Analysis Platform (SNAP). E.g USA-Full (|V|=23M,|E|=57M), delaunay_n25(|V|=33M, E|=167M),web-edu(|V|=10M, E|=55M), etc.
- The CPU forward phase is based on frontier-based vertex-parallel algorithm of Madduri et al.[3]. The GPU part is based on frontier-based edge-parallel BFS code of LoneStar-GPU version 2.0 [4].
- We compared our implementation(**HyBIR**) with
  - ✓ CPU standalone and GPU standalone codes based on our hybrid algorithm.
  - ✓ **Totem[2]**, a hybrid framework which follows with CPU standalone and GPU standalone codes based on our hybrid algorithm.
- We have implemented a variable partitioning technique which partitions the graphs based on the CPU-GPU processing capability.

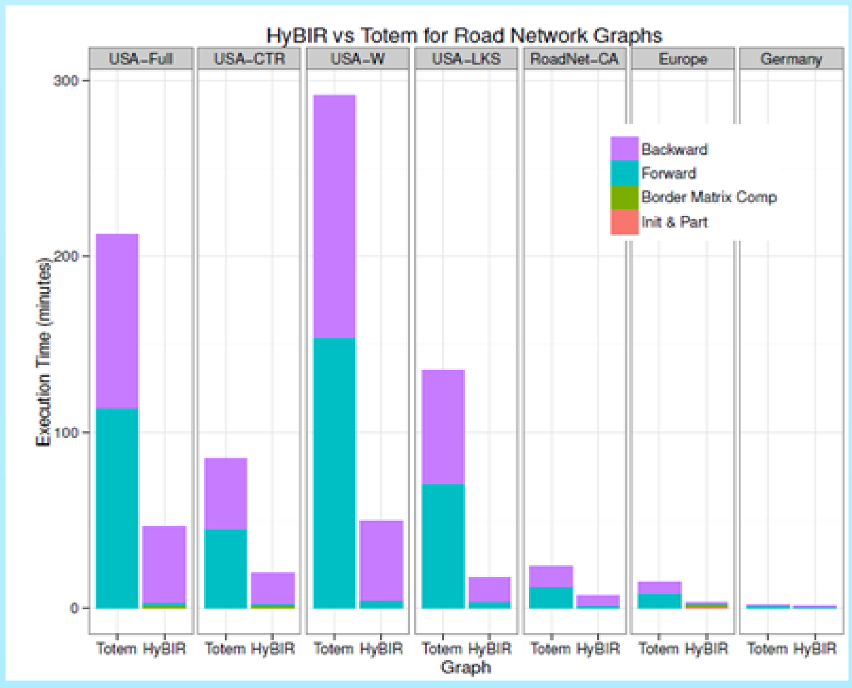
Results

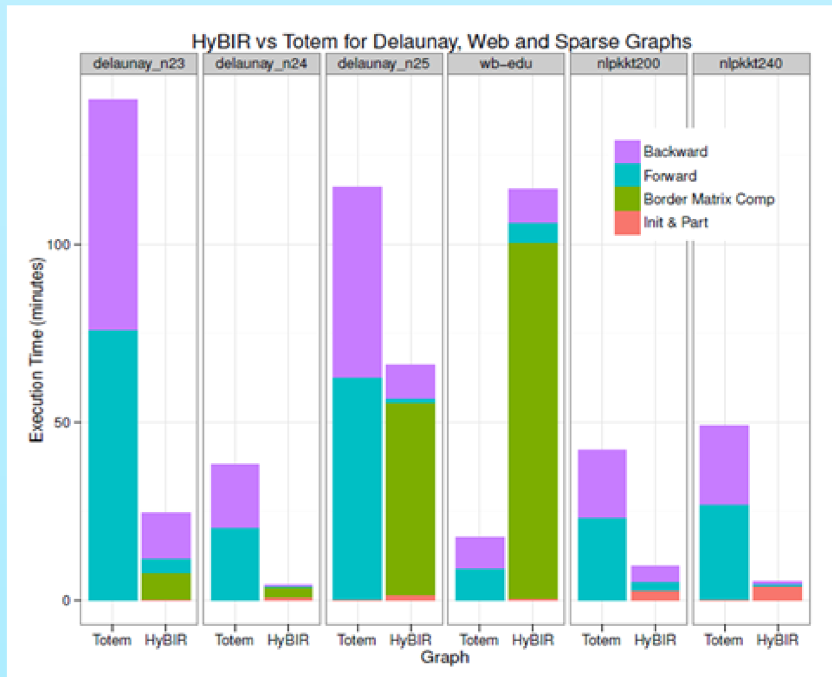
- In all the graphs HyBIR out performs the CPU standalone (except nlpkkt(s)). However performs worse than GPU standalone(except for graphs which GPU can't execute).



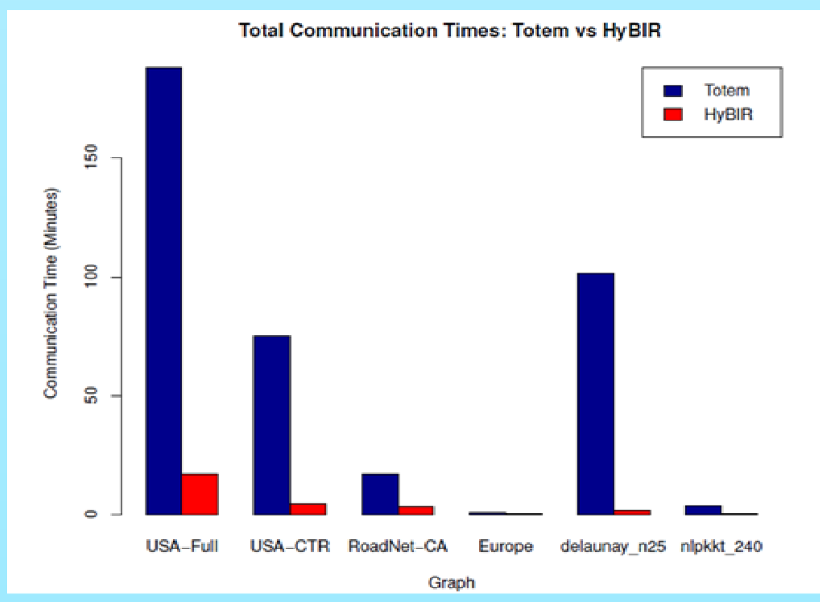


- HyBIR outperforms TOTEM for graphs due to 80-90 % less communication.





- Totem synchronizes between the parts for all the levels w.r.t source 's' for both forward and backward phase(e.g. in the graph USA-Full there are 6093 levels.).
- For HyBIR synchronization occurs for the number of edges in the edge cut for both backward and forward phase. (e.g. for the graoh USA-Full the synchronization ranges from 2 to 6 iterations).



CONCLUSIONS AND FUTURE WORK

- Our result shows up to 80% improvement w.r.t the state of art hybrid implementation, Totem.
- Our hybrid approach gives better performance than CPU-only version and explore graphs not able to fit in GPU memory.
- Our semi asynchronous divide and conquer based border aware technique can be used for exploring big data graphs with multiple partitions in an efficient manner.

REFERENCES

[1] U. Brandes, "A Faster Algorithm for Betweenness Centrality," The Journal of Mathematical Sociology, vol. 25, no. 2, pp. 163–177, 2001.

[2] A. Gharaibeh, L. B. Costa, E. Santos-Neto, and M. Ripeanu, "A Yoke of Oxen and a Thousand Chickens for Heavy Lifting Graph Processing," in International Conference on Parallel Architectures and Compilation Techniques, PACT '12, Minneapolis, MN, USA - September 19 - 23, 2012, 2012, pp. 345–354.

[3] K. Madduri, D. Ediger, K. Jiang, D. Bader, and D. Chavarria-Miranda, "A Faster Parallel Algorithm and Efficient Multithreaded Implementations for Evaluating Betweenness Centrality on Massive Datasets," in Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, ser. IPDPS '09, 2009.

[4] "Lonestargpu," http://iss.ices.utexas.edu/?p=projects/galois/lonestargpu.