Heterogeneous Parallel Programming COMP4901D

Betweenness Centrality Computation

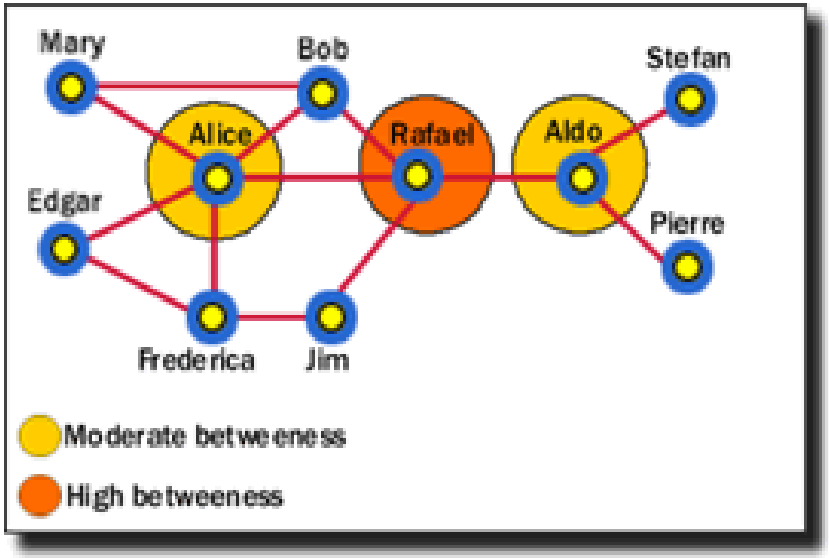
On Heterogeneous Processors

Overview

- Betweenness Centrality measure
- Sequential algorithms
- Existing parallel algorithms for weighted and unweighted graphs
- Our GPU-based algorithms

BC Measure

Centrality Metric



- importance of individual nodes
- wide range of applications

Betweenness Centrality Measure

Freeman [1]

Brandes [2]

- Denote σ_{st}

shortest paths between s and t

- Denote $\sigma_{st}(v)$

 # shortest paths between s and t through v

$$BC(v) = \sum_{s \neq t \neq v} \delta_{st}(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

$$\delta_s(v) = \sum_t \delta_{st}(v)$$

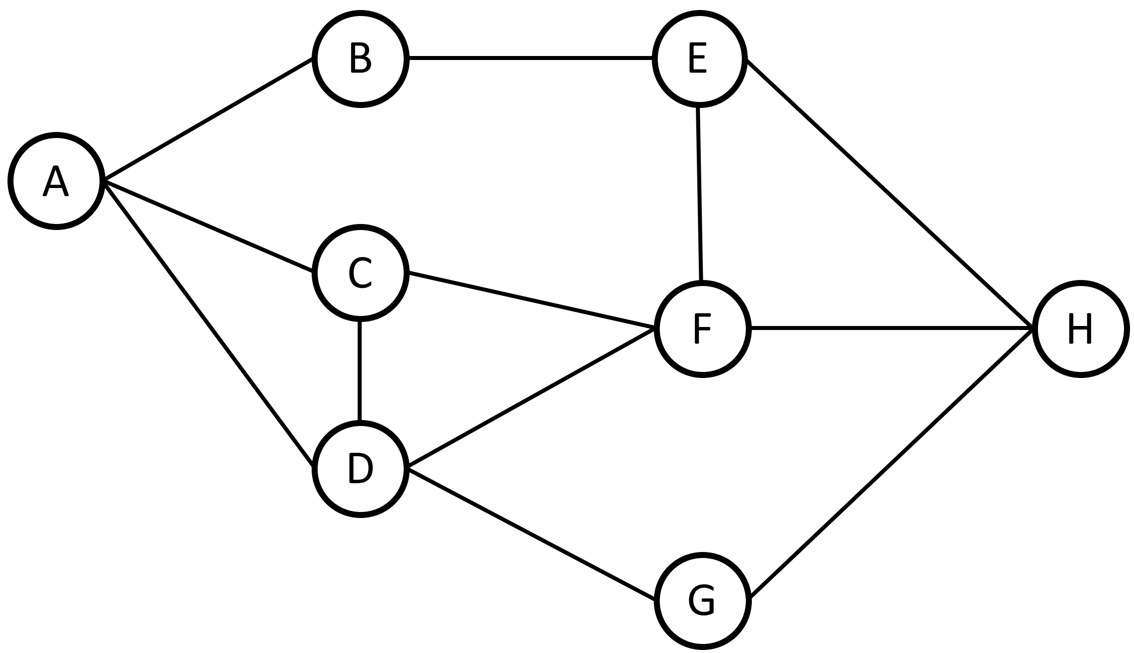
$$= \sum_{w: v \in prev(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}} \left(1 + \delta_s(w)\right)$$

$$BC(v) = \sum_{s \neq v} \delta_s(v)$$

[1] L. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40:35–41, 1977.

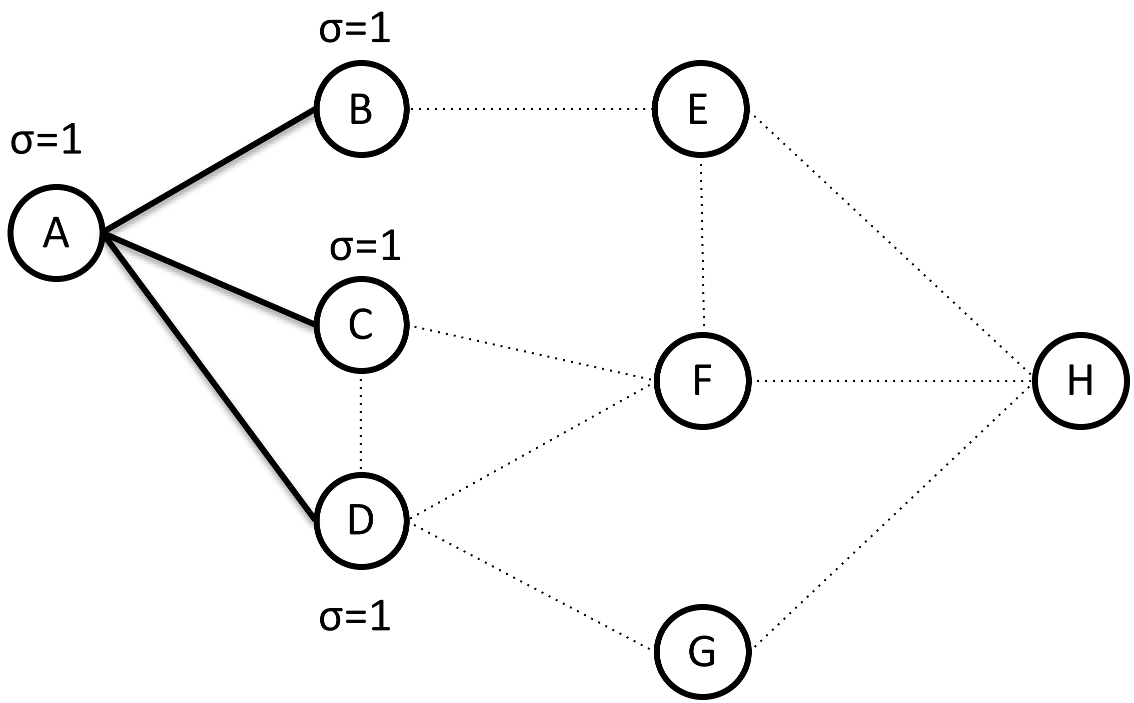
[2] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

Sequential Algorithm



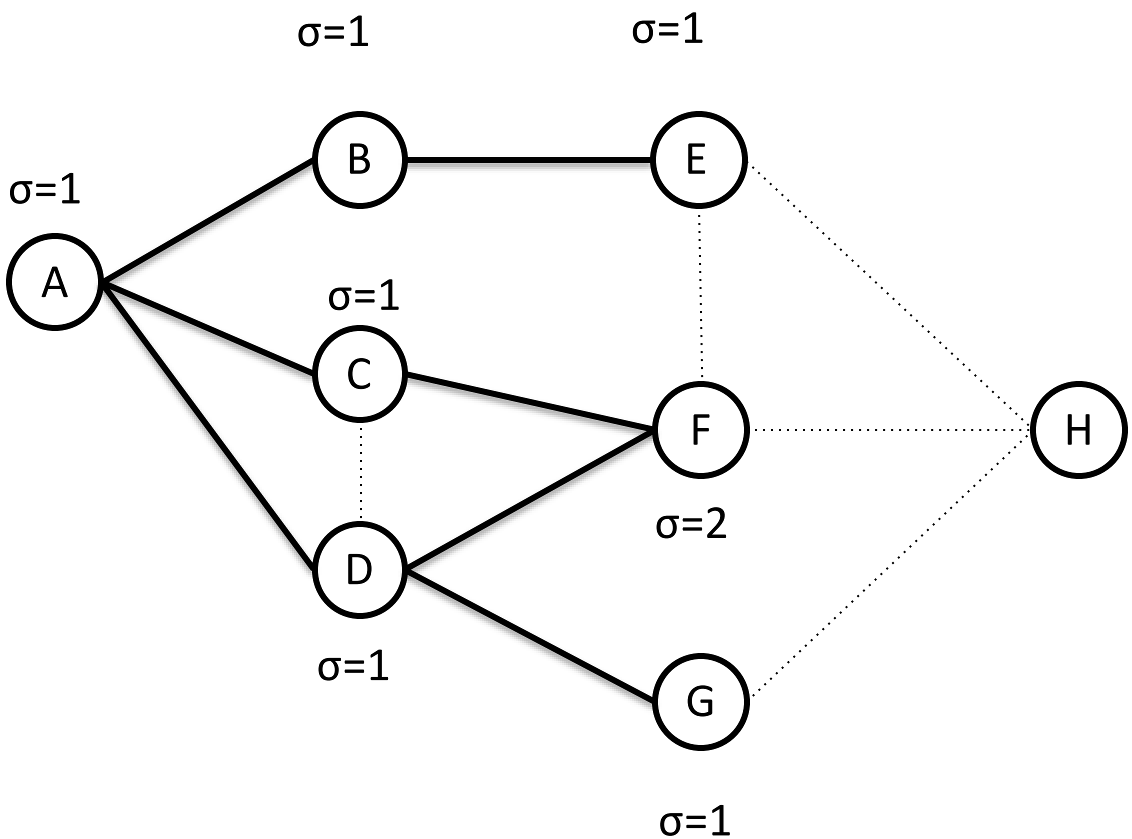
An example of BC computation rooted at A on unweighted graph

Sequential Algorithm



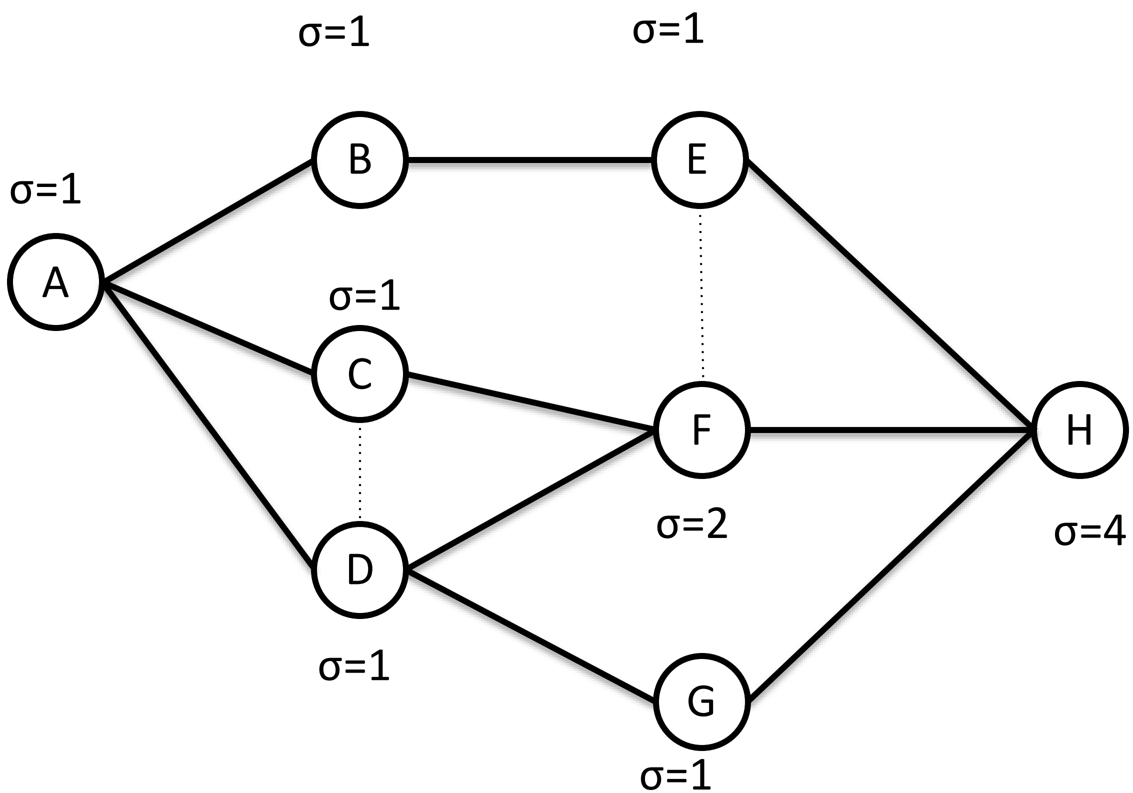
An example of BC computation rooted at A on unweighted graph

Sequential Algorithm



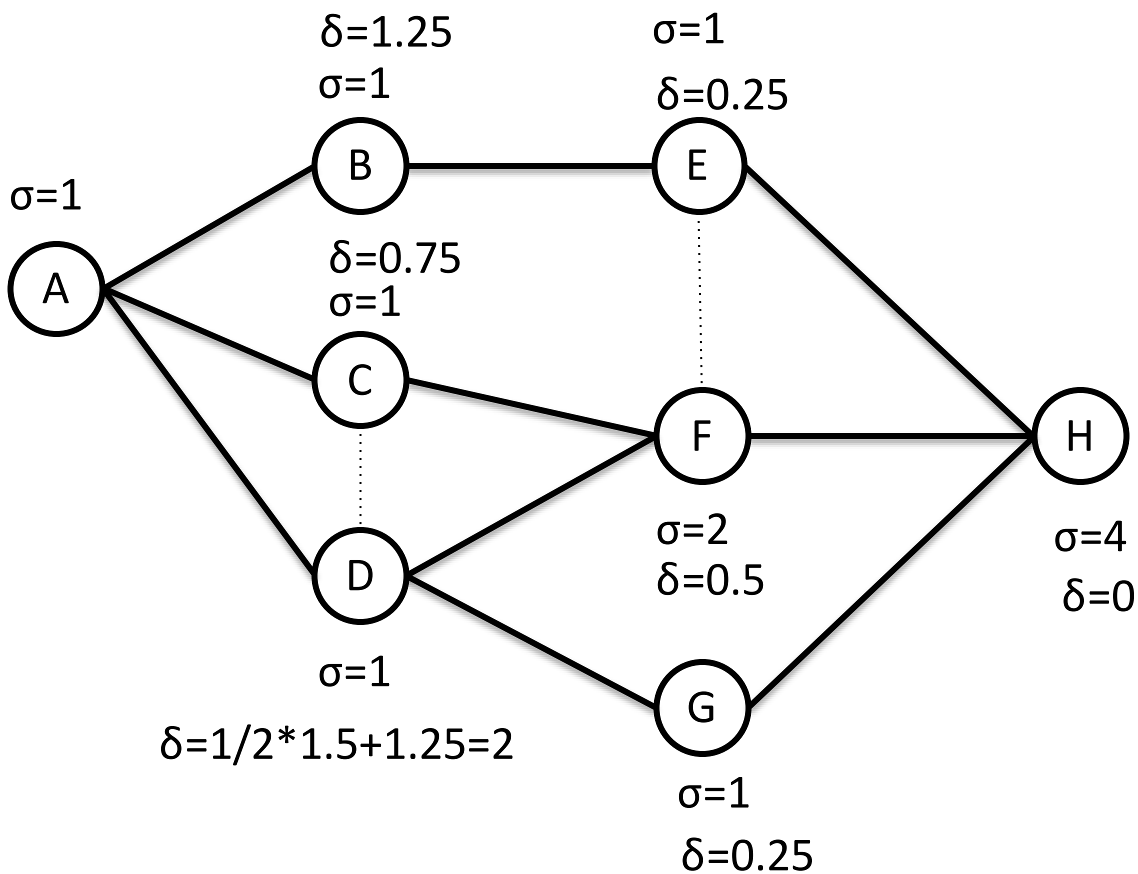
An example of BC computation rooted at A on unweighted graph

Sequential Algorithm



An example of BC computation rooted at A on unweighted graph

Sequential Algorithm

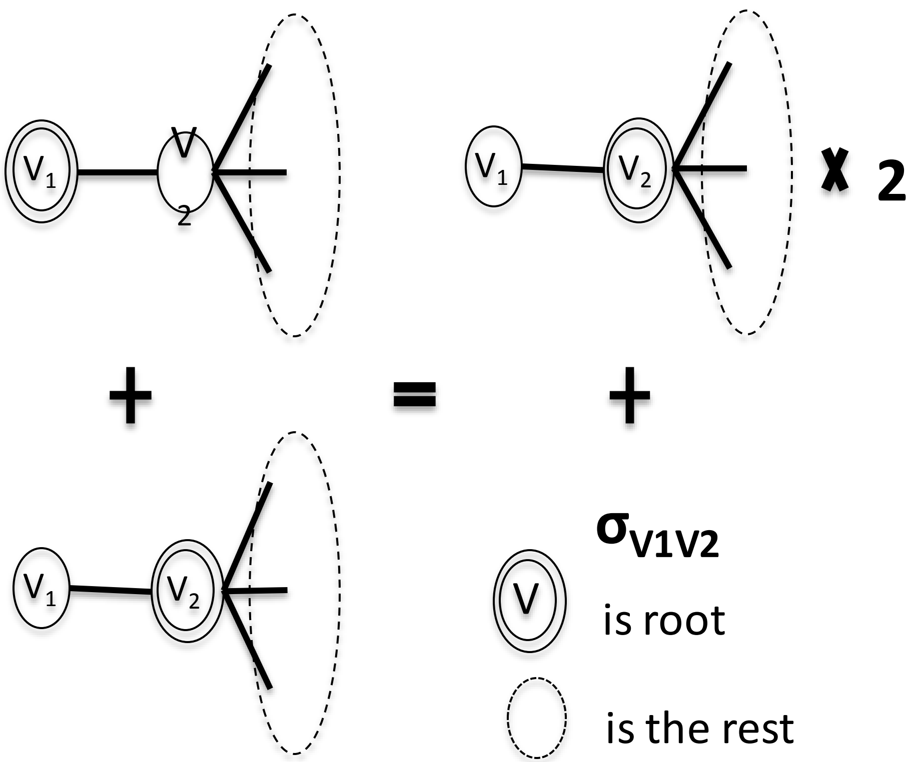


An example of BC computation rooted at A on unweighted graph

Algorithmic Redundancy in BC

one-degree vertex

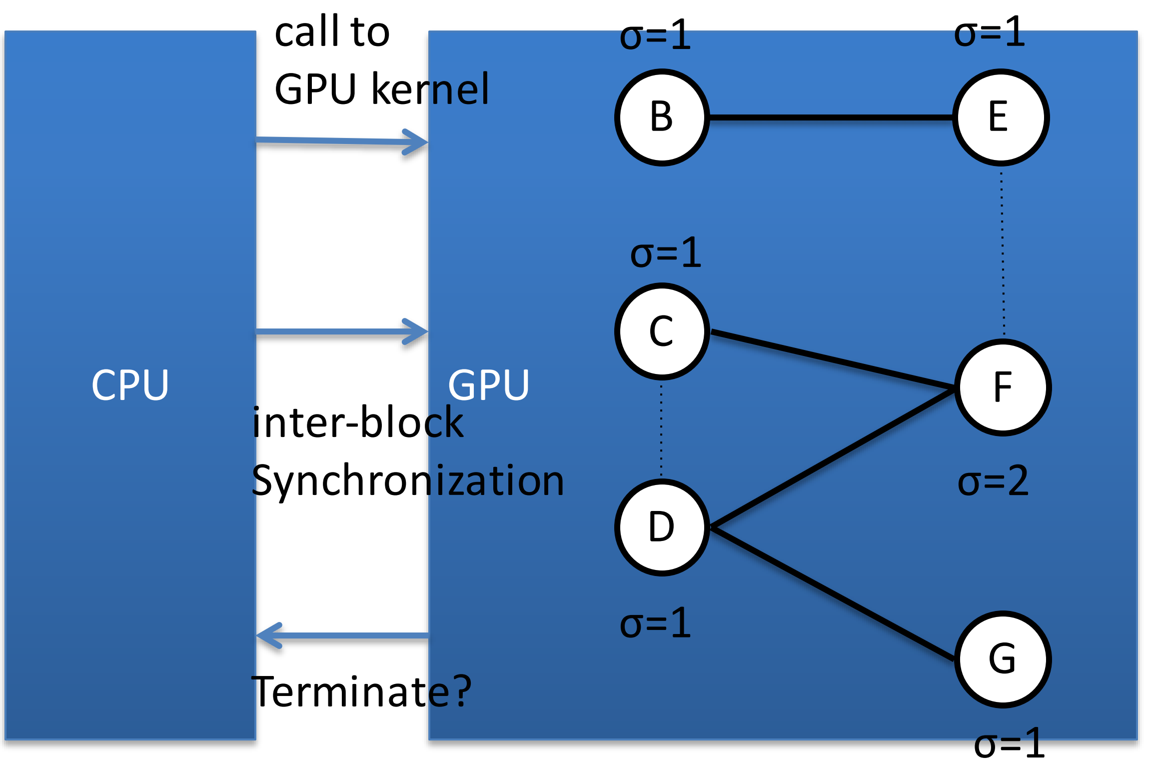
two-degree vertex




```
for each v∈V &v≠s do
  d1[v] ← d1[v] + weight1
  d2[v] ← d2[v] + weight2
  if d1[v] < d2[v] then
          ds[v] ← d1[v],
          σs[v] ← σ1[v]
  if d1[v] = d2[v] then
          ds[v] ← d1[v],
          σs[v] ← σ1[v] + σ2[v]
  if d1[v] > d2[v] then
          ds[v] ← d2[v]
          σs[v] ← σ2[v]
```

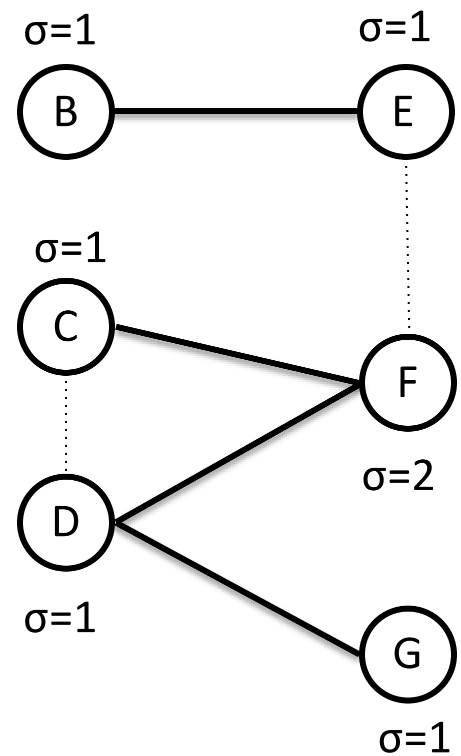
Parallel BC on Unweighted Graphs

- Level-synchronization mechanism



Parallel BC on Unweighted Graphs

- Vertex-oriented BC[1]
- Edge-oriented BC [2]
- Virtualization-based BC [3]



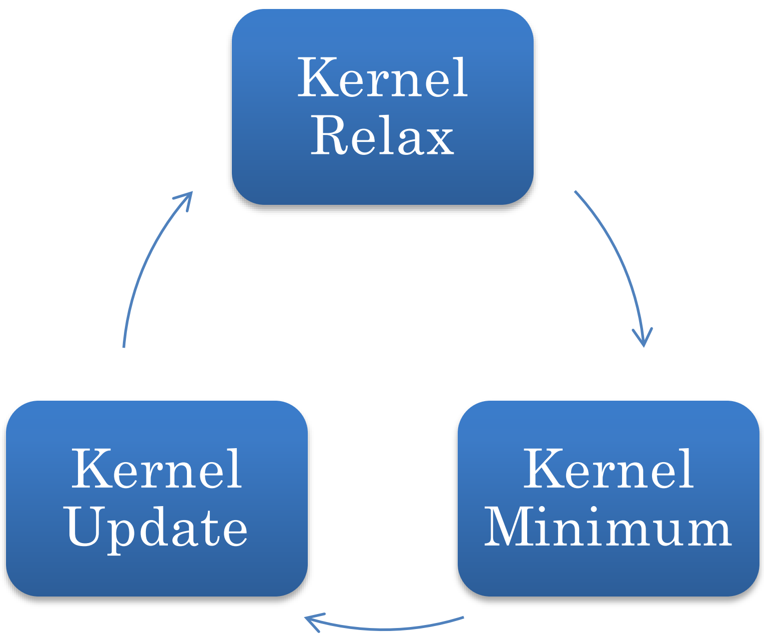
[1] Pawan Harish and P. J. Narayanan. Accelerating Large Graph Algorithms on the GPU using CUDA. In Srinivas Aluru, Manish Parashar, Ramamurthy Badrinath, and Viktor K. Prasanna, editors, *HiPC*, volume 4873 of *Lecture Notes in Computer Science*, pages 197–208. Springer, 2007

[2] Zhiao Shi and Bing Zhang 0003. Fast network centrality analysis using gpus. *BMC Bioinformatics*, 12:149, 2011.

[3] Ahmet Erdem Sariyu ce, Kamer Kaya, Erik Saule, and Umit VC atalyu rek. Betweenness centrality on gpus and heterogeneous architectures. In *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, pages 76–85. ACM, 2013.

Parallel Shortest-Path Search on Weighted Graphs

- Distance-sensitive GPU-based BC



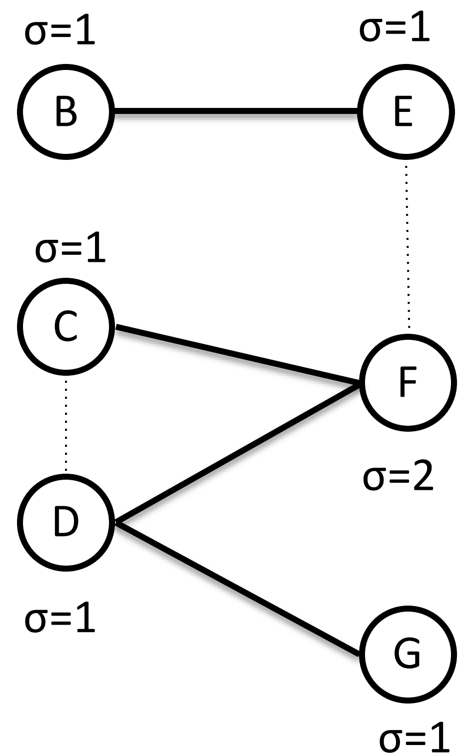
U_i: the unsettled nodes in *i*th iteration

Limited value Δ_i: the shortest distance of vertices in U_i plus the minimum of edge weights

F_i: the set of frontier nodes in *i*th iteration

Parallel BC on Unweighted Graphs

- Edge-oriented BC with filter
  - CPU-based filter
  - GPU-based filter
- Dynamic parallel BC

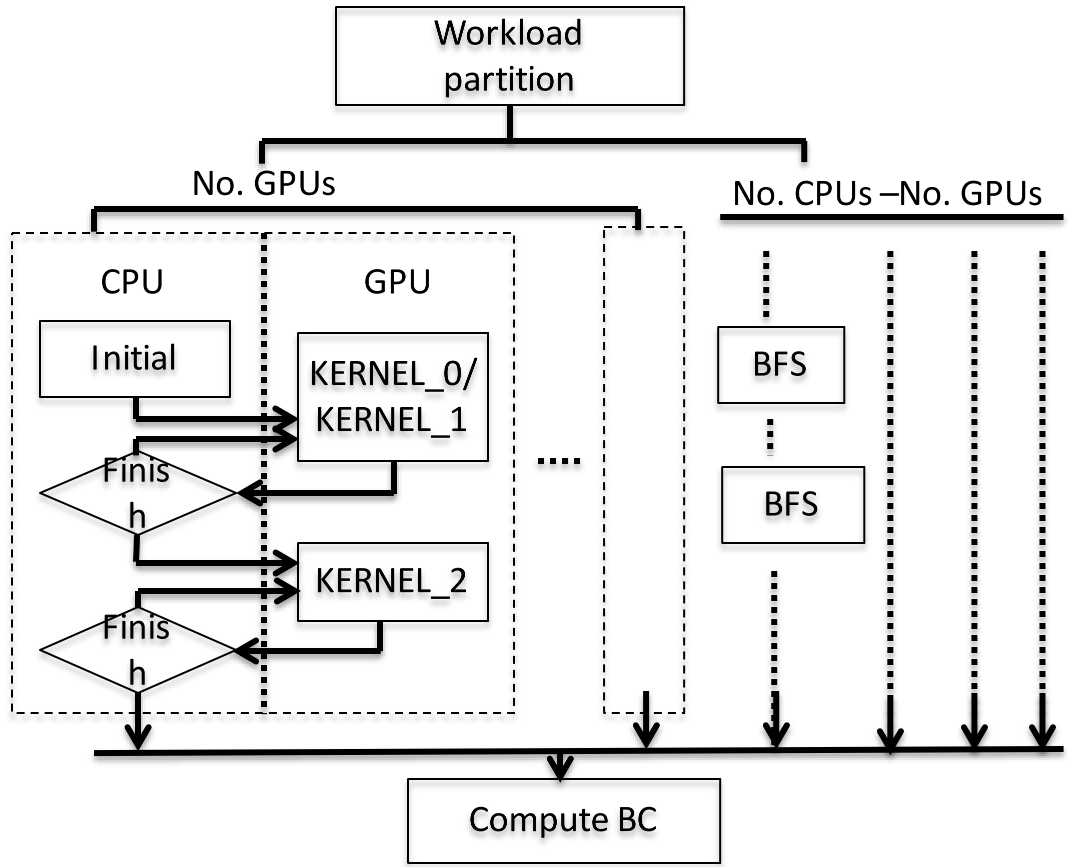


Parallel BC on Unweighted Graphs

- GPU-based implementation issues
  - Maximize memory throughput
    - Segmented sorting end array
    - Mark array
  - Reduce thread workload
    - Special first F-STEP kernel
    - Vertex-backward
    - DAG-traversal merging

Hetero-BC for Unweighted Graphs

1. discard the one-degree vertices
2. assign the computation of the two-degree vertices and their neighbors to the GPUs
3. partition the rest of the work on heterogeneous processors

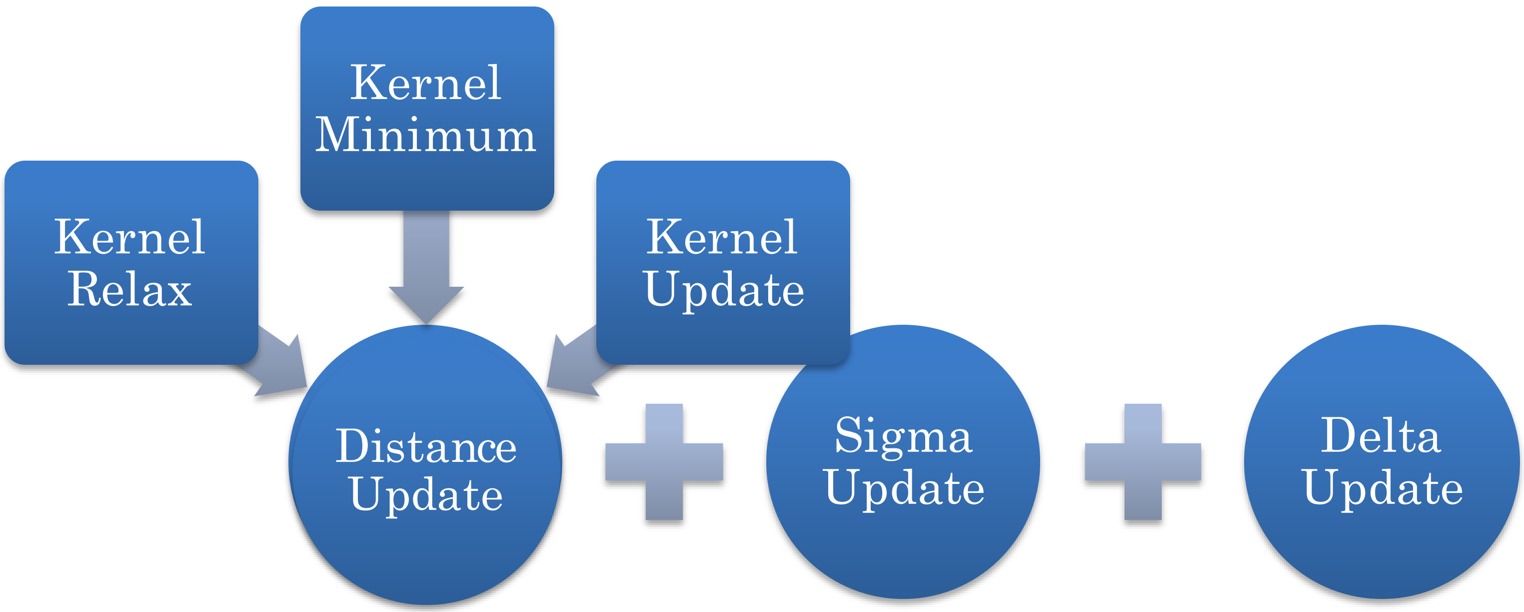


Parallel BC on Weighted Graphs

- Distance-sensitive GPU-based BC
- Distance-insensitive GPU-based BC
- Hetero-BC for Weighted Graphs

Parallel BC on Weighted Graphs

- Distance-sensitive GPU-based BC



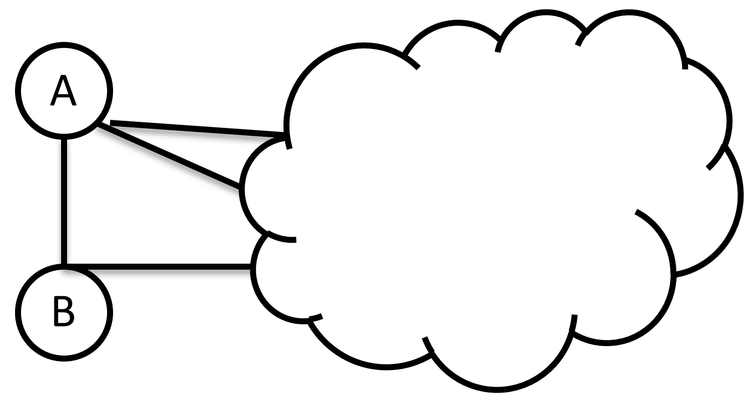
Parallel BC on Weighted Graphs

- Distance-insensitive GPU-based BC
  - cons
    - redundant relaxation in distance update kernel
  - pros
    - remove Kernel Minimum & Update
    - fewer iterations in level-synchronization framework

Parallel BC on Weighted Graphs

- reduce write conflict

Hetero-BC for Weighted Graphs



w←head[getThreadID]
v←end[getThreadID]
if level[w] = current level then
  if distance[v] >= distance[w] + weight[w][v] then
    atomicMin d[v] ← distance[w] + weight[w][v]
    level[v] ← current level
    d cont ← TRUE

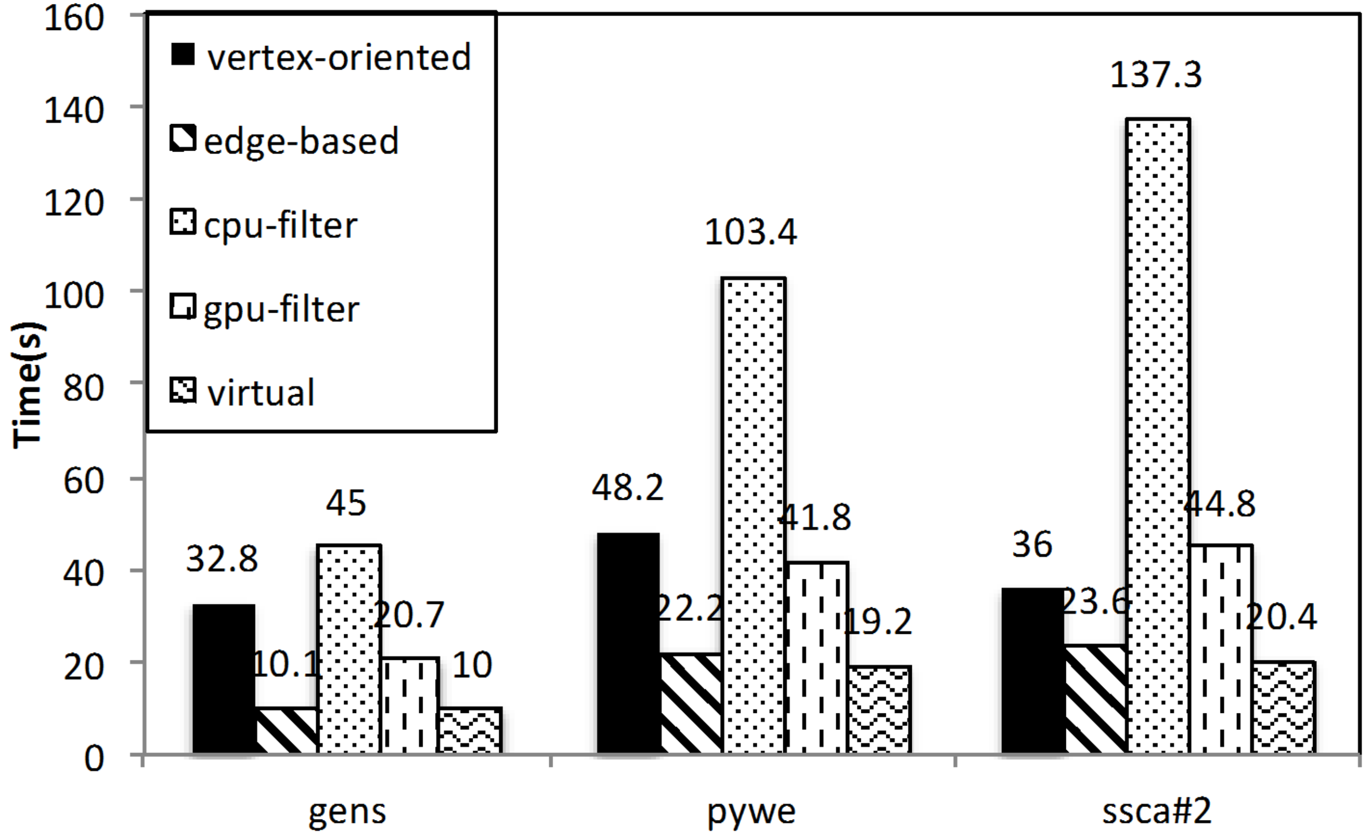
Experimental Evaluation

- Setup

Processor	Intel E5-2650	Tesla M2090	Tesla K20m
No. cores	8	512	2496
Processor clock	2 GHz	1.3 GHz	3.52 GHz
Cache size	20 M	L2: 768 KB	L2: 1.3 MB
Memory size	768 GB	6 GB	5 GB
Memory bandwidth	51.2 GB/s	138 GB/s	208 GBy/s
Bandwidth between CPU and GPU	2973 MB/s		

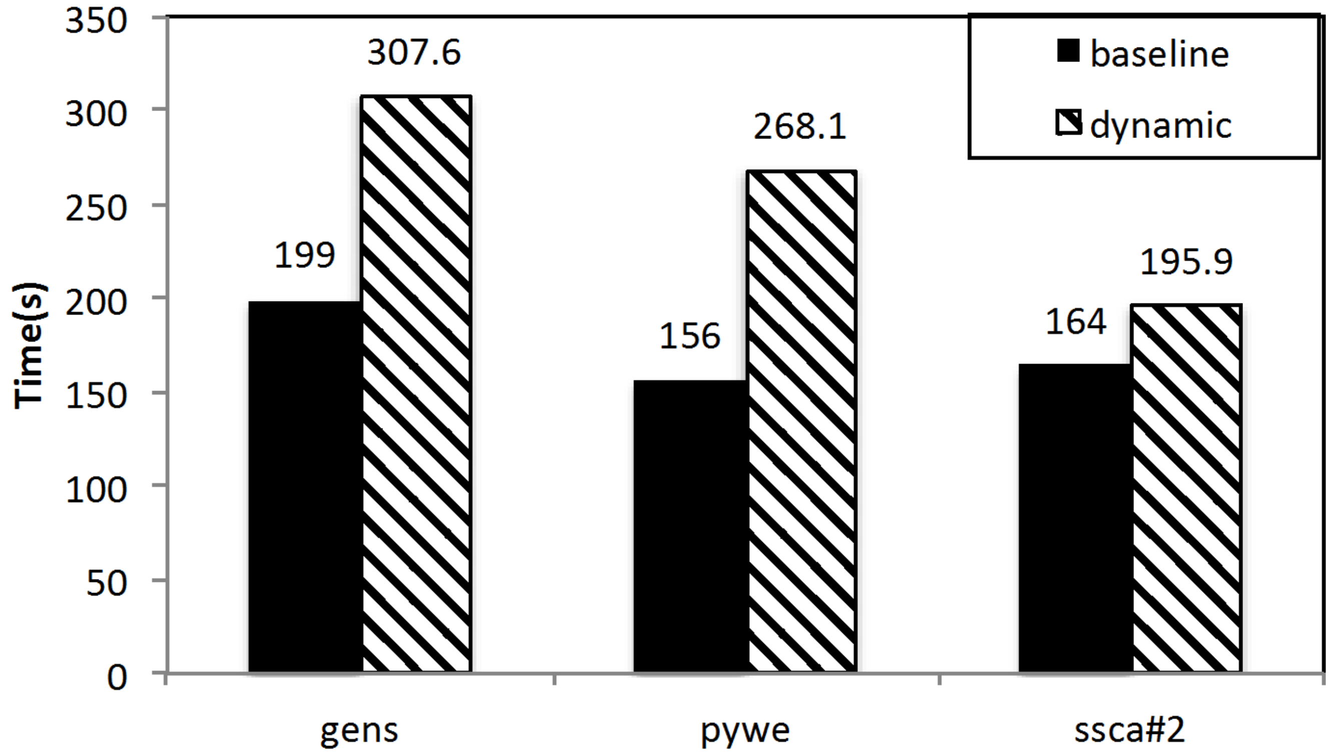
Software :CentOS 6.3 64-bit with gcc 4.4.6 and CUDA Toolkit 4.2.9

Experimental Evaluation



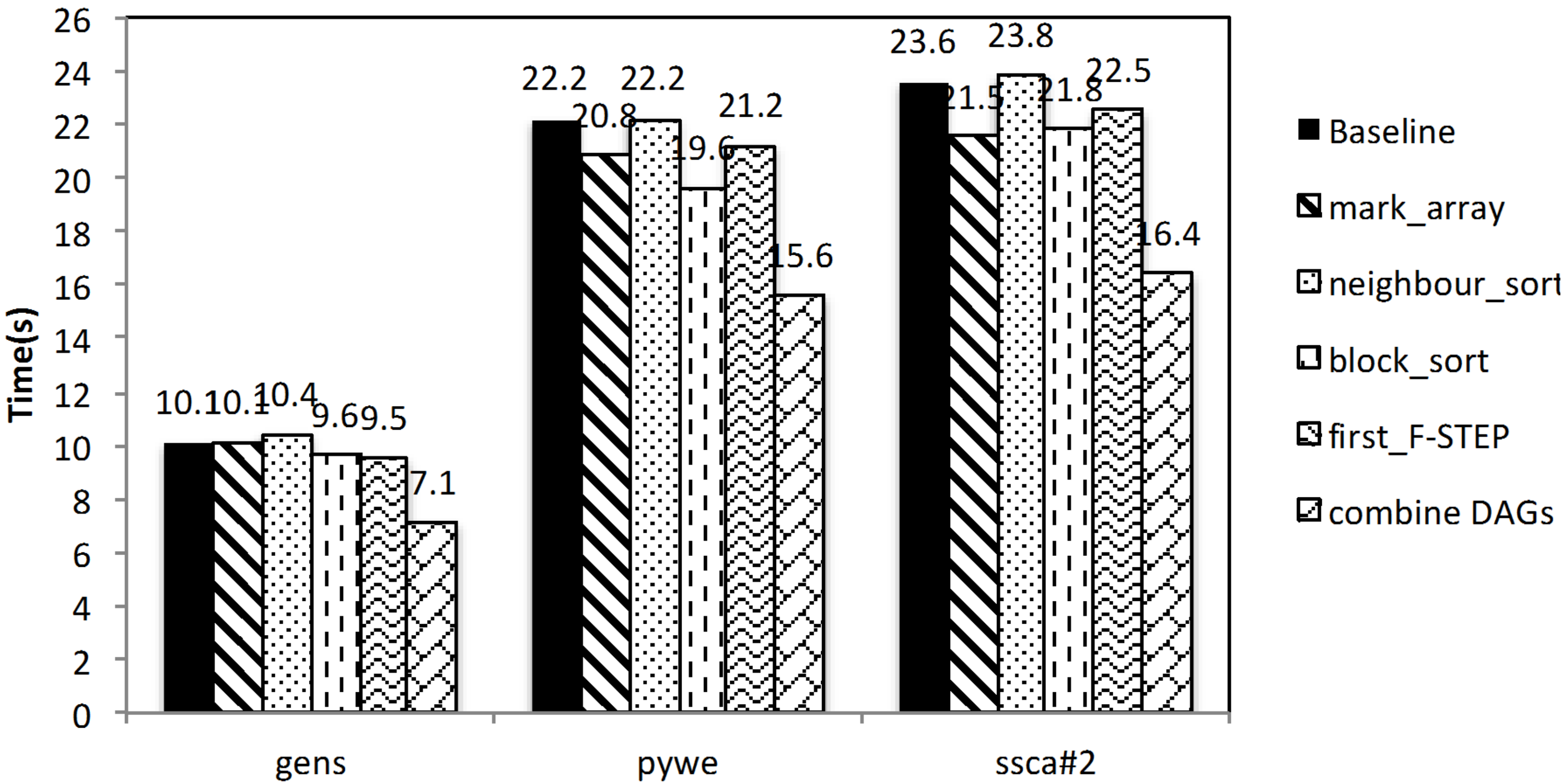
Performance of GPU-based BC on unweighted graphs with five basic F-STEP kernel programs

Experimental Evaluation



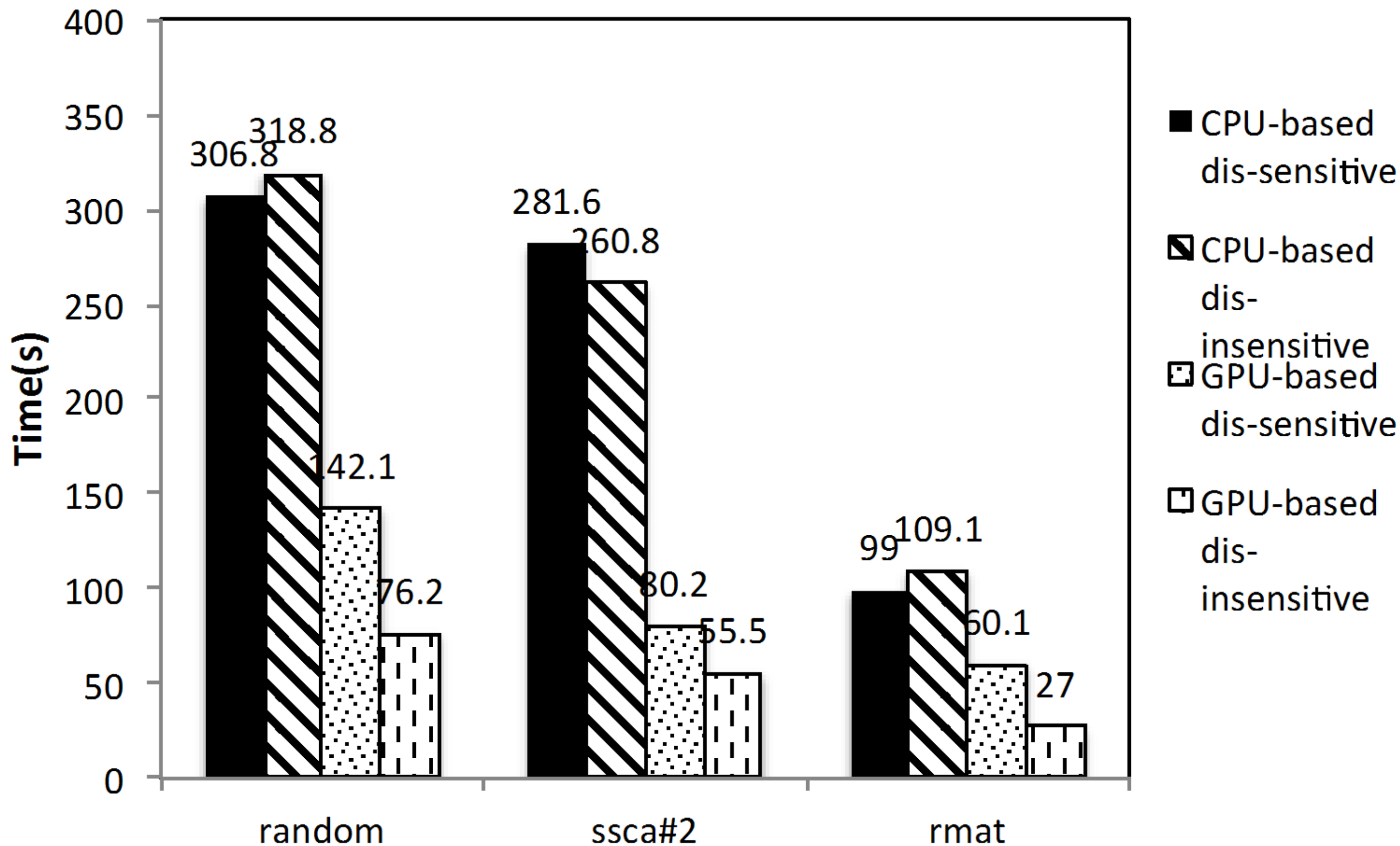
Performance of dynamic Parallel BC on unweighted graphs

Experimental Evaluation



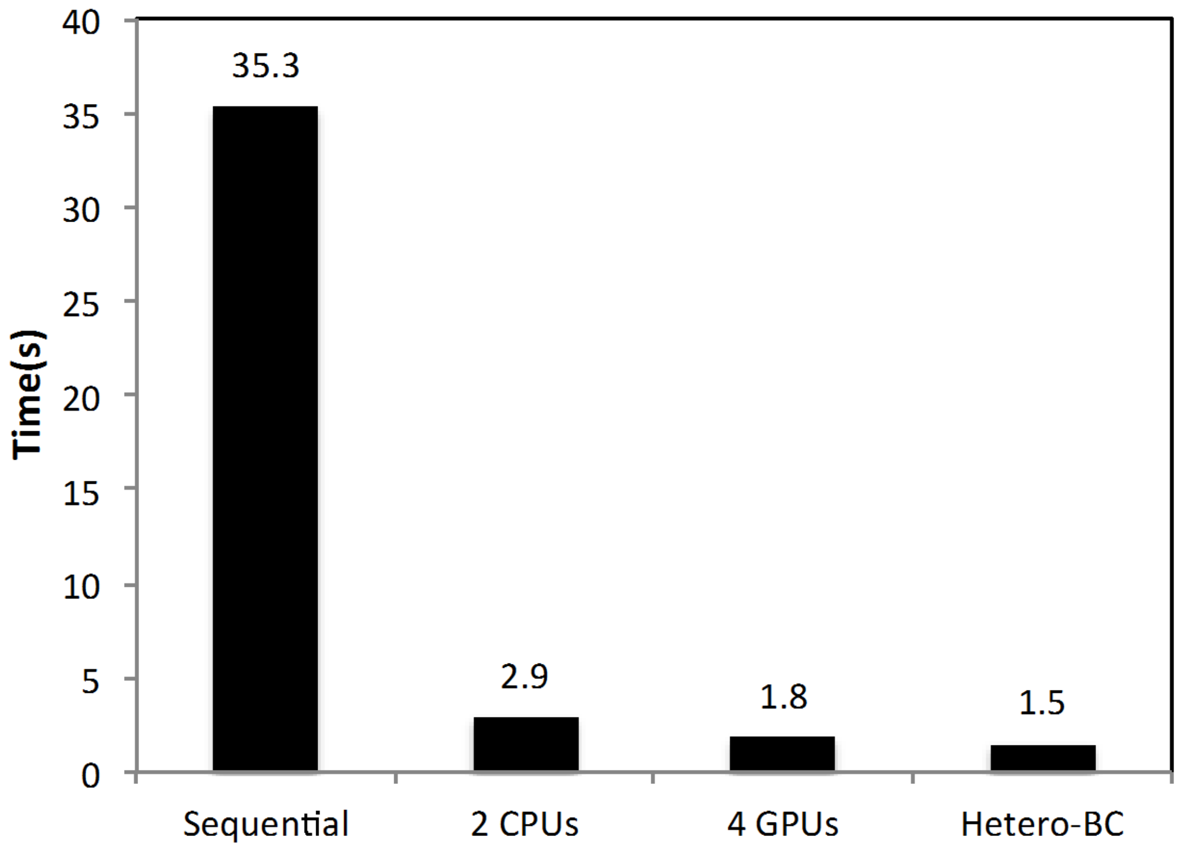
Performance of GPU-based BC on unweighted graphs with various optimizations

Experimental Evaluation



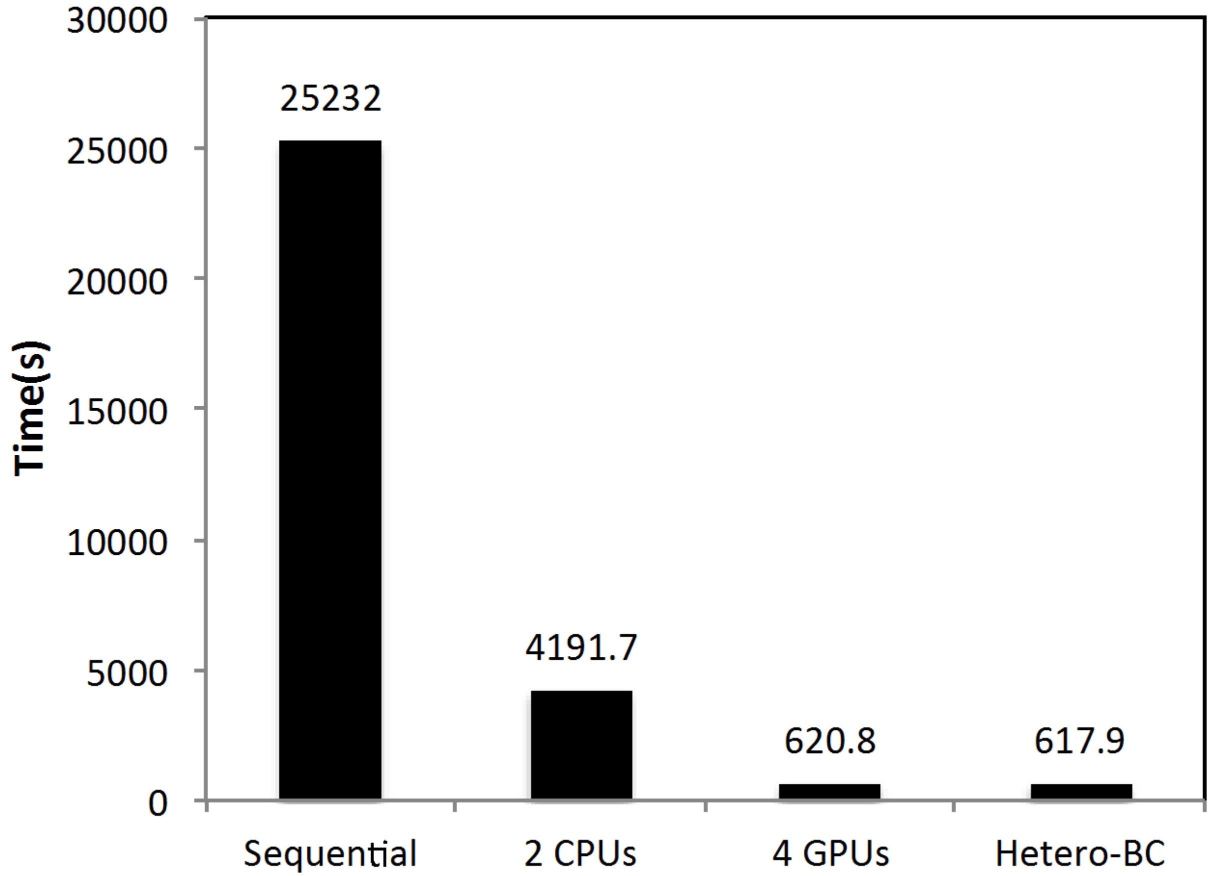
Comparison of distance-sensitive/insensitive CPU-based and GPU-based BC on weighted graphs

Experimental Evaluation



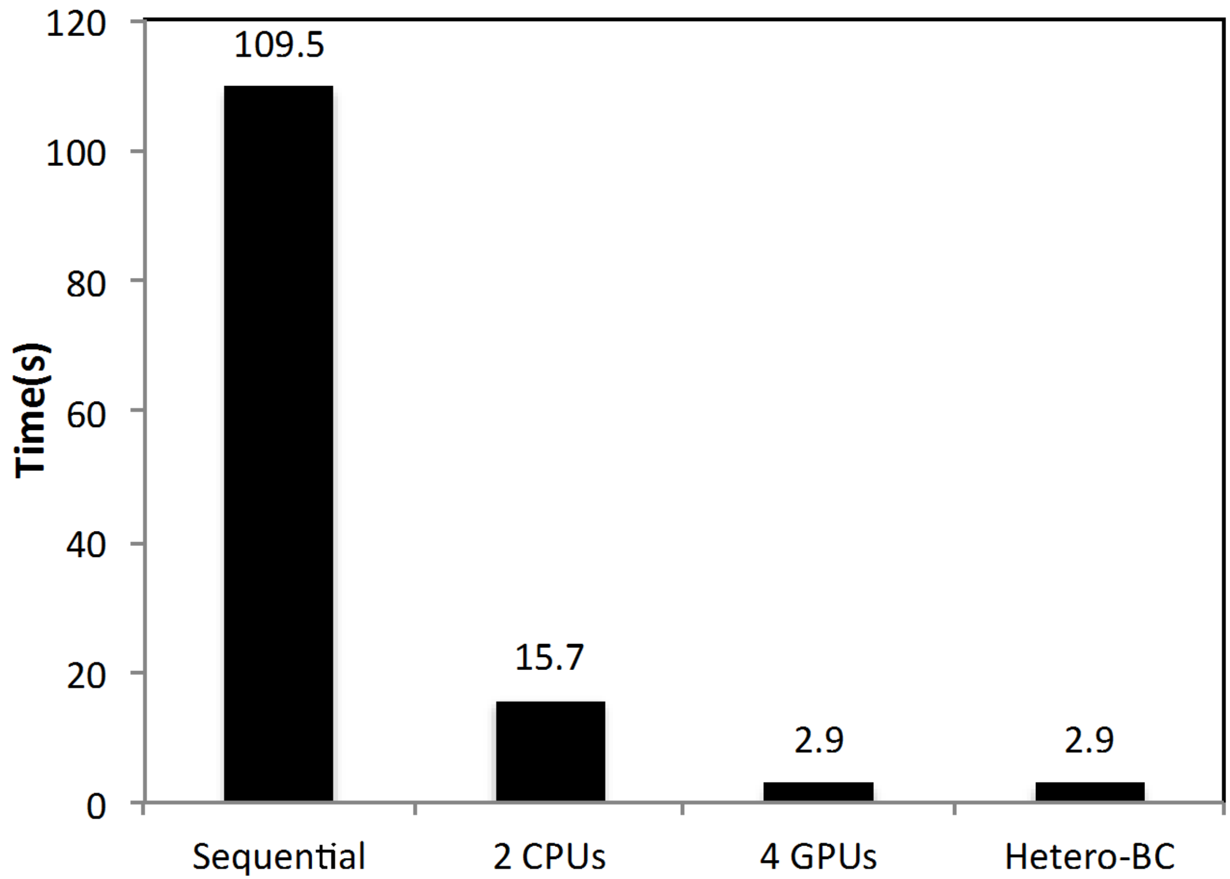
Overall performance on Protein interaction network (unweighted)

Experimental Evaluation



Overall performance on Co-author and Citation Networks (unweighted)

Experimental Evaluation



Overall performance on Scientific collaboration network (weighted)

Summary

- We design a GPU-based BC computation algorithm for unweighted graphs, where the BFS is parallelized by edges for small scale graphs or by virtual vertices for large scale graphs.
- We design a novel distance-insensitive edge-based BC algorithm for weighted graphs on the GPU.
- We perform general optimizations such as for low-degree nodes as well as GPU-specific optimizations such as splitting different computation stages into separate kernel programs.