Scalable and Performant Graph Processing on GPU using Approximate Computing

Somesh Singh

CS14D406

Department of Computer Science and Engineering

Indian Institute of Technology Madras

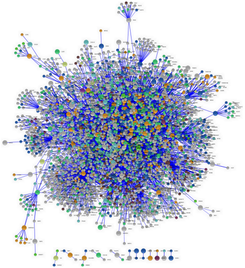
June 9, 2021

Programming Languages, Architecture and Compilers Education Laboratory

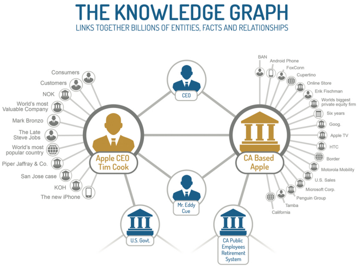
Timeline

July 14, 2014	Joined for PhD.
August 27, 2015	Comprehensive examination.
June 22, 2017	Seminar 1.
October 28, 2019	Seminar 2.
July 15, 2020	Synopsis.
June 9, 2021	PhD viva voce.

Graphs are Ubiquitous



Biological Network



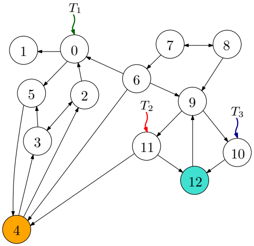
Knowledge Network

Social Network

Road Network

Image Source: Google Images

Challenges in Parallel Graph Processing



dest

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	5	0	3	2	5	2	3	4	0	4	9	6	8	7	9	10	11	12	4	12	9

src

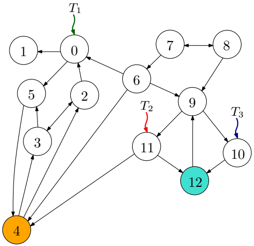
0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	2	2	4	6	8	9	12	14	16	18	19	21	22

dist

0	1	2	3	4	5	6	7	8	9	10	11	12

CSR representation

Challenges in Parallel Graph Processing



dest

1	5	0	3	2	5	2	3	4	0	4	9	6	8	7	9	10	11	12	4	12	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

src

0	2	2	4	6	8	9	12	14	16	18	19	21	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

dist

0	1	2	3	4	5	6	7	8	9	10	11	12

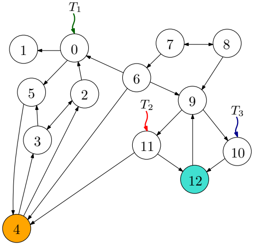
CSR representation

Assumptions

- vertex-centric model of parallelization
- propagation-based graph kernels

```
Graph G(V,E) = read_input(); // CPU
transfer_input(); // CPU → GPU
v.dist = ∞ ∀ v ∈ V;
source.dist = 0;
Worklist wl = {source};
do {
 changed = false;
 forall Node u : wl do {
  for Node v : G.neighbors(u) do {
   newVal = dist[u] + e_uv.wt();
   if(newVal < dist[v]) {
    oldVal = atomicMin(&dist[v], newVal);
    if(newVal < oldVal) {
     wl.push(v);
     changed = true;
    } } } }
} while(changed);
transfer_output(); // GPU → CPU
```

Challenges in Parallel Graph Processing



dest

1	5	0	3	2	5	2	3	4	0	4	9	6	8	7	9	10	11	12	4	12	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

src

0	2	2	4	6	8	9	12	14	16	18	19	21	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

dist

0	1	2	3	4	5	6	7	8	9	10	11	12

CSR representation

Assumptions

- vertex-centric model of parallelization
- propagation-based graph kernels

```
Graph G(V,E) = read_input(); // CPU
transfer_input(); // CPU → GPU
v.dist = ∞ ∀ v ∈ V;
source.dist = 0;
Worklist wl = {source};
do {
 changed = false;
 forall Node u : wl do {
  for Node v : G.neighbors(u) do {
   newVal = dist[u] + e_uv.wt();
   if(newVal < dist[v]) {
    oldVal = atomicMin(&dist[v], newVal);
    if(newVal < oldVal) {
     wl.push(v);
     changed = true;
    } } } }
} while(changed);
transfer_output(); // GPU → CPU
```

- Irregular accesses: The indirection "dist[dest[id]]".
- Memory-latency bound.
- Load imbalance: Skew in vertex degrees.

Our Approach

- Combine *parallelization* with *approximate computing* to make graph processing more efficient at the expense of accuracy.
- Provide tunable knobs to control the performance-accuracy trade-off.

Our Approach

- Combine *parallelization* with *approximate computing* to make graph processing more efficient at the expense of accuracy.
- Provide tunable knobs to control the performance-accuracy trade-off.

Approximate computing techniques

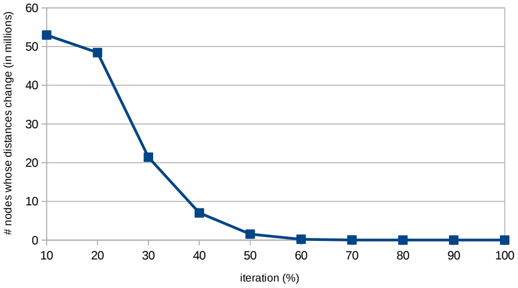
1. algorithm- and architecture-independent : *Graprox*
2. algorithm-independent but architecture-specific : *Graffix*
3. algorithm-specific but architecture-independent : *ParTBC*

Graprox : Techniques for Approximate Parallel Graph Processing

1. Reduced Execution
   - cut-short the number of outerloop iterations.
2. Partial Graph Processing
   - process only a subset of the edges in each outerloop iteration.
3. Approximate Graph Representation
   - merge nodes with overlapping neighbors, based on Jaccard's similarity.
4. Approximating Attribute Values
   - round-off numeric attribute values to *discrete* values.

1 Reduced Execution

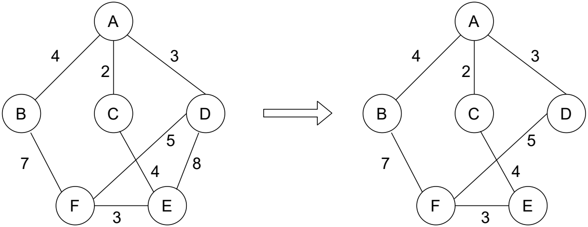
- Cut-short the number of outerloop iterations based on online stopping criteria.
- Helpful when majority of work gets done in the initial iterations.
- Results in fewer barriers in iterative parallel graph processing.



SSSP computation on `rmat26` graph

2 Partial Graph Processing

- Process only a subset of the edges in each outerloop iteration.
- At each node, select the edges to be processed; ignore others.
- Helps improve performance since the work done per iteration (measured as number of edges traversed) is less.
- Processing fewer edges translates to lesser *synchronization* per iteration.

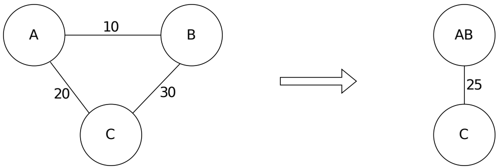


For SSSP computation

- Lossy graph compression by merging nodes with overlapping neighbors.
- Jaccard's coefficient J_{ij}, for vertices v_i and v_j with sets of neighbors $N(v_i)$ and $N(v_j)$ respectively, is:

$$J_{ij} = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$$

- If there is a triangle a-b-c and a-b get merged:



- The compressed graph is fed as input to the exact parallel implementation of the algorithm.

4 Approximating Attribute Values

- Numeric attribute values (such as vertex attributes or edge weights) rounded-off to *discrete* values.
- Helps in reaching the termination criteria faster, in fewer rounds.
- For MST computation, rounding-off the edge-weights to the closest power of 2 helps reach the termination threshold in fewer iterations.
- SSSP transformed to BFS with careful discretization of edge weights for computing approximate shortest distance.
  - level-synchronous BFS can be implemented without explicit atomic instructions, reducing the synchronization.

Graffix : GPU-Specific Approximate Computing Techniques

1. Improving Memory Coalescing
   - make the graph layout more *structured* to improve locality.
   - *renumber* the graph vertices and *replicate* a select set of vertices.
2. Reducing Memory Latency
   - process *well-connected* sub-graphs, iteratively, inside shared memory.
3. Reducing Thread Divergence
   - normalize degrees across nodes assigned to a warp.

1 Improving Memory Coalescing

Vertex Renumbering

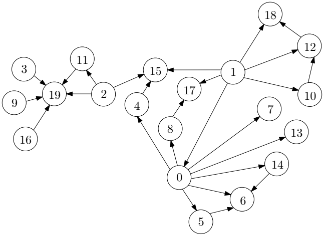
- Assign *nearby* ids to vertices to be accessed by warp-threads.

Vertex Renumbering

- Assign *nearby* ids to vertices to be accessed by warp-threads.

Approach

- Perform BFS from a highest outdegree node.
- Assign ids level-by-level; incrementally in a round-robin fashion at a level.



edges

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
4	5	6	7	8	13	14	0	10	12	15	17	18	11	15	19	19	15	6	17	19	12	19	18	6	19

offset

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	7	13	16	17	18	19	19	19	20	21	22	23	24	24	25	25	26	26	26	26

node attributes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

edge attributes

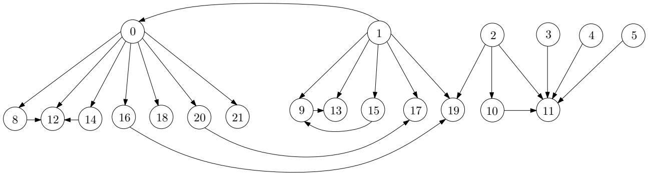
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Original graph

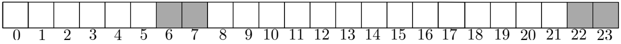
Approach

- Start a level at a multiple of $k \mid 1 \leq k \leq$ warp-size $\rightarrow$ creates *holes*.
- Divide the node array (after renumbering) into chunks of size k.

Renumbered graph



Creation of *holes* after renumbering



$k = 8$

Improving Memory Coalescing

Vertex Replication

- A node occurs exactly once, so it cannot be nearby all its neighbors even after the renumbering.
- Replication brings such a node *close* to its otherwise *far* neighbors.

Improving Memory Coalescing

Vertex Replication

- A node occurs exactly once, so it cannot be nearby all its neighbors even after the renumbering.
- Replication brings such a node *close* to its otherwise *far* neighbors.

Approach

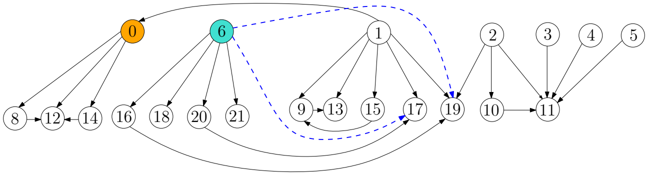
- If a node is well-connected to a chunk, replicate the node in a chunk in the previous BFS level.

$$\boxed{\text{connectedness}_{\text{chunk}}^{\text{node}} \triangleq \left(\frac{\#\ \text{edges to chunk from a node}}{\#\ \text{non-hole nodes in chunk}}\right) \geq \textit{threshold}}$$

- Distribute the outgoing edges of a node among its copies.
- Add edges from node's replica to its 2-hop neighbors inside the chunk.
- Perform a merge operation on the values of the replicas after each iteration.

Improving Memory Coalescing

- Node 0 is well-connected to the chunk 16..23
- $\text{connectedness}^{0}_{16..23} = \frac{4}{6} = 0.67 \geq 0.5$ (*threshold*)
- Node 0 is replicated in the chunk 0..7; its replica is assigned id 6.



edges

8	12	14	0	9	13	15	17	19	10	11	19	11	11	11	16	17	18	19	20	21	12	13	11	12	9	19	17
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

offset

0	3	9	12	13	14	15	21	21	22	23	24	24	24	24	25	26	27	27	27	27	28	28	28	28
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

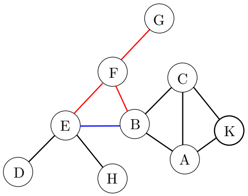
node attributes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Modified graph

- Clustering-coefficient measures the degree to which nodes in a graph tend to “cluster”.
- Local clustering-coefficient (LCC) of a node, X :

$$\mathrm{LCC}_X = \frac{\#\text{ pairs of X's neighbors that are neighbors}}{\#\text{ pairs of X's neighbors}}$$



$\#$ of pairs of F's neighbors that are neighbors $= 1$
$\#$ of pairs of F's neighbors $= \binom{3}{2} = 3$
$\mathrm{LCC}_F = \frac{1}{3}$

Reducing Memory Latency using Shared Memory

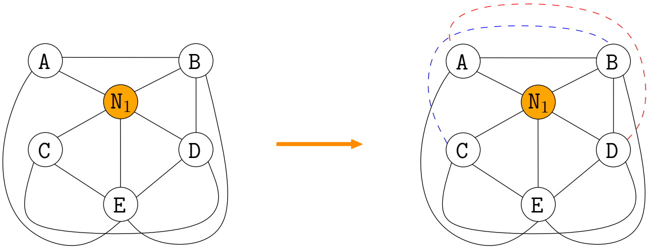
Vertices with *local clustering coefficient* (LCC) $\geq$ *threshold* are more frequently accessed in iterative processing.

Reducing Memory Latency using Shared Memory

Vertices with *local clustering coefficient* (LCC) $\geq$ *threshold* are more frequently accessed in iterative processing.

Approach

- Increase LCC of node if LCC $\leq$ *threshold* and LCC $\sim$ *threshold*.
- Boost LCC of node if LCC $\geq$ *threshold*.
- Cap on the total number of additional edges added in the graph.



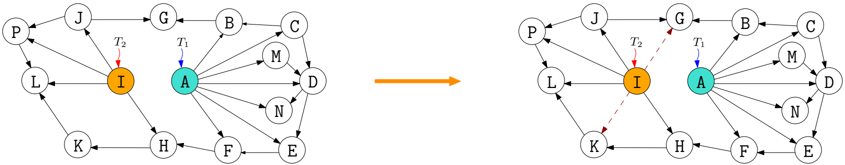
3 Reducing Thread Divergence

Make node degrees *nearly* uniform within each warp.

Make node degrees *nearly* uniform within each warp.

Approach

- Add edges to the nodes that are deficient in their connectivity.
- Add edges between 2-hop neighbors for faster convergence.
- Increase the degree of the candidate nodes to be close to α% of max. degree (e.g., 85%); α is tunable.



ParTBC : Approximate top-k Betweenness Centrality Computation

Betweenness centrality (BC) is a metric that measures the significance of a vertex by the number of shortest paths leading through it.

Algorithm 1: Brandes' algorithm

Input: An unweighted graph $G(V, E)$
Output: Vertex betweenness centrality

$bc(v) = 0 \quad \forall v \in V$
for $s \in V$ **do**
    // Forward Pass: form BFS DAG D
    forall $v : Node \in G$ **do**
        compute σ_{sv}
        compute $pred(s, v)$
    // Backward Pass: backward traverse DAG D
    forall $v : Node \in D$ **do**
        compute $\delta_s(v)$
        $bc(v) \mathrel{+}= \delta_s(v)$
    // Reset graph attributes

Formally: $\mathrm{bc}(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st(v)}}{\sigma_{st}}$

- σ_{sv} : # shortest paths from s to v.
- $pred(s, w)$: immediate predecessors of w in $s \rightsquigarrow w$ path.
- $\delta_s(v)$: *dependency* of v w.r.t. s.

  $$\delta_s(v) = \sum_{w \mid v \in pred(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_s(w))$$
- $bc(v) = \sum_{s \neq v \in V} \delta_s(v)$

ParTBC : Approximate top-k Betweenness Centrality Computation

Algorithm 2: ParTBC's approximate top-k computation

Input: An undirected, unweighted graph $G(V, E)$

$bc(v) = 0 \quad \forall v \in V$ // initialization
// Phase-I
$\boxed{G'(V, E) = \text{graphReordering}(G)}$ // vertex renumbering ($G' \cong G$)
// Phase-II
nextIter = *true*
while *nextIter* **do**
  nextIter = *false*
  $\boxed{s = \text{getSource}()}$ // pick source vertex
  // Forward Pass: form BFS DAG D
  forall $v : Node \in G'$ **do**
    compute σ_{sv}
    compute $pred(s, v)$
  Let D be the DAG formed by the forward pass
  // Backward Pass: backward traverse DAG D
  forall $v : Node \in D$ **do**
    compute $\delta_s(v)$
    $bc(v)$ += $\delta_s(v)$
  // Reset graph attributes
  if *stopping criteria not met* **then**
    nextIter = *true*;

ParTBC : Approximate top-k Betweenness Centrality Computation

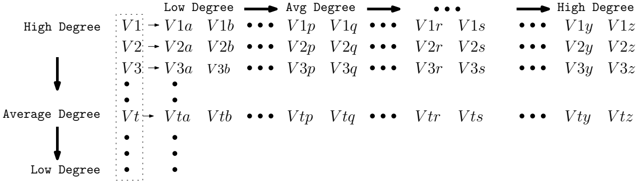
Observations

- Source does not contribute to its own BC source.
- High BC vertices are: i) high degree nodes, ii) nodes lying between large well-connected clusters.

Heuristics for selection of source vertices

1. Random Selection
2. Selection in Ascending and Descending Degree Order
3. Restricted Round Robin (RRR)
4. Dynamic (Dyn)
5. Dynamic Round Robin (DynRR)

Dynamic Round Robin (DynRR)



Arrangement of vertices for DynRR

- Select source nodes in a round-robin fashion for the initial 5% iterations.
- In subsequent iterations, pick the node with the least BC score as the next source.

Experimental Setup

CPU	Intel Xeon E5-2650 v2 (32 cores, 2.6 GHz, 96 GB RAM).
GPU	Nvidia (Kepler) Tesla K40C (15 SMXs, 2880 cores, 745 MHz, 12 GB global memory).
Software	CentOS 6.5, gcc 4.8.2, CUDA 8.0

Machine Configuration

Graph	$\|V\|$ $\times 10^6$	$\|E\|$ $\times 10^6$	**Graph type**
LiveJournal	4.8	68.9	Social network, small diameter
USA-road	23.9	57.7	Road network, large diameter
twitter	41.6	1468.3	Twitter graph 2010 snapshot
rmat26	67.1	1073.7	Synthetic scale-free graph
random26	67.1	1073.7	Synthetic random graph

Input Graphs for Graprox and Graffix

Experimental Setup

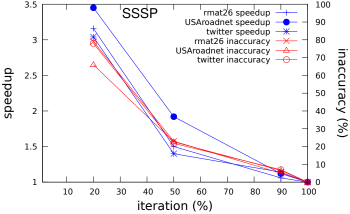
Graprox and Graffix

Graph Algorithms

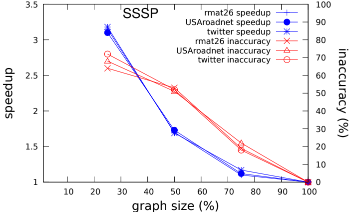
- Single Source Shortest Path computation (SSSP)
- PageRank computation (PR)
- Strongly Connected Component computation (SCC)
- Minimum Spanning Tree computation (MST)
- Betweenness Centrality computation (BC)

Baselines

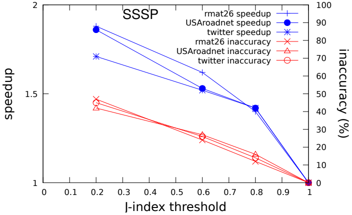
- Baseline I:
  - SSSP, MST from LonestarGPU;
  - SCC by Devshatwar et al.;
  - BC by McLaughlin and Bader;
  - Our exact parallel version of PR.
- Baseline II: SSSP, PR, BC from Tigr.
- Baseline III: SSSP, PR, BC from Gunrock.



Effect of reduced execution on performance and inaccuracy.



Effect of partial graph processing on performance and inaccuracy.



Effect of varying J-index threshold on performance and inaccuracy.

Takeaway:

Approximate computation of graph algorithms is a robust way of dealing with irregularities.

Graprox Results

	Technique	Mean Speedup	Mean Inaccuracy
SSSP	Reduced execution	1.34×	6.07%
	Partial processing of graph	1.38×	16.19%
	Approx. graph representation	1.22×	13.87%
	Approx. attributes	1.92×	17.64%
MST	Reduced execution	1.18×	16.05%
	Partial processing of graph	1.65×	17.44%
	Approx. graph representation	1.44×	15.17%
	Approx. attributes	1.48×	19.07%
SCC	Reduced execution	1.25×	18.26%
	Partial processing of graph	1.32×	19.61%
	Approx. graph representation	1.45×	20.11%
	Approx. attributes	–	–
PR	Reduced execution	2.03×	2.75%
	Partial processing of graph	1.43×	15.74%
	Approx. graph representation	1.37×	13.70%
	Approx. attributes	–	–
BC	Reduced execution	1.74×	18.07%
	Partial processing of graph	1.42×	16.73%
	Approx. graph representation	1.33×	14.35%
	Approx. attributes	–	–

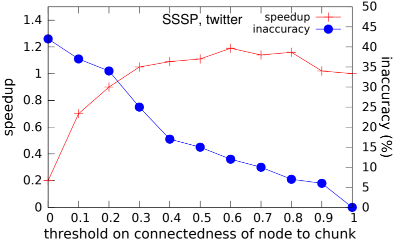
Takeaway:

Approximate computing techniques are consistently helpful in improving the execution performance of graph analytics in exchange for inaccuracy.

Graffix Results

Improving Memory Coalescing

	Baseline I	**Baseline II**	**Baseline III**
Mean Speedup	1.16×	1.10×	1.14×
Mean Inaccuracy	10%	9%	9%

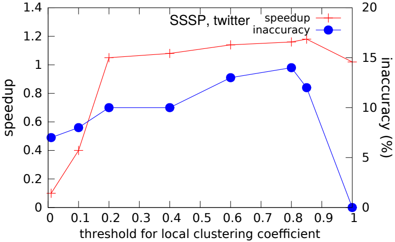


Effect of varying the threshold for node replication on memory coalescing. (Chunk size is set to 16.)

Takeaway:

Desired accuracy and performance for an algorithm – input graph pair can be achieved by tuning the chunk size and the threshold for node replication.

	Baseline I	Baseline II	Baseline III
Mean Speedup	1.20×	1.19×	1.19×
Mean Inaccuracy	13%	12%	12%

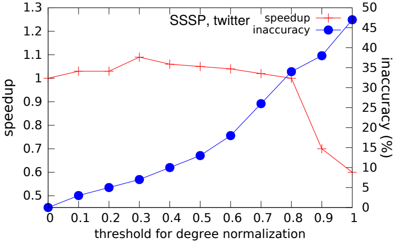


Effect of varying the LCC threshold on memory latency.

Takeaway:

Appreciable speedup, with low inaccuracy, can be achieved by processing well-connected subgraphs inside shared memory.

	Baseline I	**Baseline II**	**Baseline III**
Mean Speedup	1.07×	1.03×	1.07×
Mean Inaccuracy	8%	8%	8%



Effect of varying the threshold for degree normalization.

Takeaway:

Small speedup with low inaccuracy can be achieved using a low threshold for degree normalization.

Experimental Setup
ParTBC

Graph	$|V|$	$|E|$	Graph type
fb-Friendships (FB)	63,731	817,035	Facebook friendship graph
usroad48 (RNUS)	102,615	147,656	Continental US road network
rmat17 (RMT)	130,977	2,091,451	Synthetic scale-free graph
random17 (RNM)	131,072	2,096,902	Synthetic random graph
roadnetSF (RNSF)	174,424	221,802	San Francisco road network
loc-Gowalla (LG)	196,591	950,327	Location-based social network
soc-Pokec (SP)	1,632,803	30,622,564	Online social network

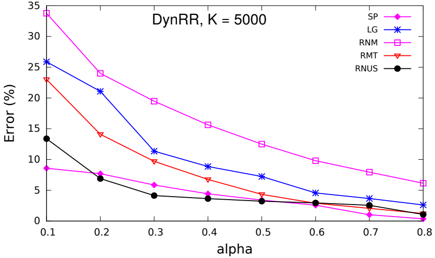
Input Graphs for ParTBC

Baseline: Exact GPU-parallel implementation of betweenness centrality.

ParTBC Results

Graph	Speedup w.r.t. exact parallel	Speedup breakdown (w.r.t. exact parallel)	
		VR	DynRR
fb-Friendships	2.80×	1.22×	2.29×
usroad48	2.68×	1.13×	2.37×
rmat17	2.65×	1.19×	2.22×
random17	1.67×	1.04×	1.61×
roadnetSF	2.71×	1.13×	2.40×
loc-Gowalla	2.48×	1.18×	2.10×
soc-Pokec	2.76×	1.20×	2.30×
geomean	**2.5**×	**1.15**×	**2.17**×

Performance of ParTBC w.r.t. exact parallel Brandes' algorithm for error $\sim 6\%$ (VR: vertex renumbering)



Effect of the number of sources chosen on error in top-k for DynRR

Takeaway:

The performance-accuracy trade-off can be controlled by carefully choosing the source nodes in top-k computation.

Conclusions

1. Parallel graph processing is challenging due to *irregularity* in the data-access, control-flow, and communication patterns.
2. We proposed
   - algorithm- and architecture-independent : Graprox
   - algorithm-independent but architecture-specific : Graffix
   - algorithm-specific but architecture-independent : ParTBC

   techniques for approximate parallel graph processing.
3. The techniques provide *tunable knobs* to control the performance-accuracy trade-off in graph applications.
4. The techniques are generally applicable to a large class of parallel graph algorithms and input graphs of varying characteristics.
5. Approximate computing combined with parallelization promises to make heavy-weight graph computation practical, as well as scalable.

Publications

Conferences

1. Somesh Singh and Rupesh Nasre. "Graffix: Efficient Graph Processing with a Tinge of GPU-Specific Approximations." In Proceedings of the *49th International Conference on Parallel Processing (ICPP 2020)*. 23:1 – 23:11. `https://doi.org/10.1145/3404397.3404406`
2. Somesh Singh and Rupesh Nasre. "Optimizing Graph Processing on GPUs Using Approximate Computing: Poster". In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming (PPoPP 2019)*. 395 – 396. `https://doi.org/10.1145/3293883.3295736`

Journals

1. Somesh Singh and Rupesh Nasre. "Scalable and Performant Graph Processing on GPUs Using Approximate Computing". *IEEE Transactions on Multi-Scale Computing Systems (TMSCS)* 4, 3 (2018), 190 – 203. `https://doi.org/10.1109/TMSCS.2018.2795543`

Publications

Conferences

1. Somesh Singh and Rupesh Nasre. "Graffix: Efficient Graph Processing with a Tinge of GPU-Specific Approximations." In Proceedings of the *49th International Conference on Parallel Processing (ICPP 2020)*. 23:1 – 23:11. https://doi.org/10.1145/3404397.3404406
2. Somesh Singh and Rupesh Nasre. "Optimizing Graph Processing on GPUs Using Approximate Computing: Poster". In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming (PPoPP 2019)*. 395 – 396. https://doi.org/10.1145/3293883.3295736

Journals

1. Somesh Singh and Rupesh Nasre. "Scalable and Performant Graph Processing on GPUs Using Approximate Computing". *IEEE Transactions on Multi-Scale Computing Systems (TMSCS)* 4, 3 (2018), 190 – 203. https://doi.org/10.1109/TMSCS.2018.2795543

Thank You

Backup Slides

Graprox

Exact execution time

Graph	Exact Time (sec)					
	SSSP	MST	SCC	Color	PR	BC
rmat26	37	8996	21	14	12	15223
random26	29	10087	23	18	16	13127
LiveJournal	2	3424	7	5	1	1711
USA-road	152	82	12	10	1	2043
twitter	231	10943	37	29	18	21462

Execution time for exact versions of graph algorithms

Graprox
Preprocessing cost

Graph	Preprocessing Time (sec)
rmat26	26
random26	18
LiveJournal	1
USA-road	43
twitter	87

Preprocessing overhead for partial graph processing

Graph	Preprocessing Time (sec)	
	J-coeff = 0.6	J-coeff = 0.8
rmat26	81	77
random26	103	94
LiveJournal	27	18
USA-road	273	203
twitter	340	321

Preprocessing overhead for approximate graph representation

Graph	Preprocessing Time (sec)
rmat26	28
random26	19
LiveJournal	2
USA-road	43
twitter	89

Preprocessing overhead for approximate attribute values

Graprox

Average execution time of the approximate versions of graph algorithms

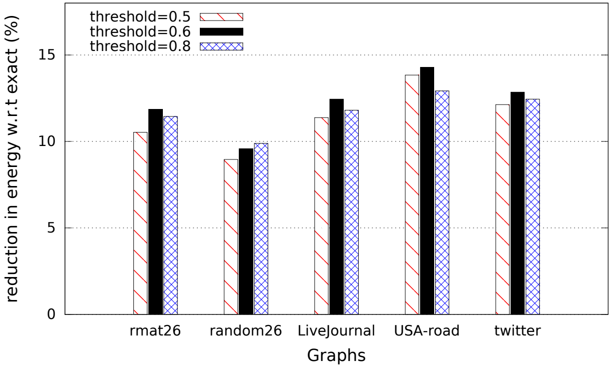
Algo.	Technique	Mean Time (sec)
SSSP	Outer-loop iterations	60.54
	Partial processing of graph	61.36
	Approx. graph representation	71.02
	Approx. attributes	46.98
MST	Outer-loop iterations	5497.05
	Partial processing of graph	3854.25
	Approx. graph representation	4298.97
	Approx. attributes	4531.35
SCC	Outer-loop iterations	15.87
	Partial processing of graph	15.15
	Approx. graph representation	13.79
	Approx. attributes	–
Color	Outer-loop iterations	10.48
	Partial processing of graph	11.87
	Approx. graph representation	11.18
	Approx. attributes	–
PR	Outer-loop iterations	4.73
	Partial processing of graph	5.27
	Approx. graph representation	6.23
	Approx. attributes	–
BC	Outer-loop iterations	6157.01
	Partial processing of graph	7544.51
	Approx. graph representation	8055.04
	Approx. attributes	7598.01

Graffix

Preprocessing cost

Technique	Graph	Preprocessing overhead	
		Time (sec)	Additional space
improving coalescing	rmat26	18	9%
	random26	21	11%
	LiveJournal	8	6%
	USA-road	11	8%
	twitter	30	6%
reducing latency	rmat26	23	5%
	random26	27	8%
	LiveJournal	9	5%
	USA-road	13	4%
	twitter	46	7%
reducing thread-divergence	rmat26	6	2%
	random26	7	3%
	LiveJournal	4	2%
	USA-road	2	1.5%
	twitter	9	4%

Preprocessing overhead



Effect of coalescing on energy consumption for BC (higher is better).
Note: decrease in energy for different thresholds of *connectedness* for node replication

Graffix

Reduction in total end-to-end time for BC, MST

	Graphs	Reduction in end-to-end time
MST	rmat26	15.05%
	random26	11.29%
	LiveJournal	12.04%
	USA-road	5.28%
	twitter	14.25%
BC	rmat26	14.41%
	random26	10.55%
	LiveJournal	12.57%
	USA-road	15.42%
	twitter	12.14%

(a) Effect of memory coalescing

Graphs	Reduction in end-to-end time
rmat26	17.77%
random26	8.82%
LiveJournal	14.99%
USA-road	0.81%
twitter	13.37%
rmat26	19.20%
random26	11.30%
LiveJournal	16.83%
USA-road	19.99%
twitter	14.31%

(b) Effect of memory latency

Graphs	Reduction in end-to-end time
rmat26	0.046%
random26	0.019%
LiveJournal	0.064%
USA-road	0.058%
twitter	0.047%
rmat26	0.098%
random26	0.047%
LiveJournal	0.080%
USA-road	0.106%
twitter	0.056%

(c) Effect of thread divergence

Graffix

Overall result

	Graphs	Speedup	Inaccuracy
SSSP	rmat26	1.22 ×	12%
	random26	1.13 ×	10%
	LiveJournal	1.18 ×	11%
	USA-road	1.15 ×	9%
	twitter	1.17 ×	12%
MST	rmat26	1.18 ×	13%
	random26	1.13 ×	15%
	LiveJournal	1.14 ×	12%
	USA-road	1.23 ×	11%
	twitter	1.17 ×	13%
SCC	rmat26	1.14 ×	8%
	random26	1.08 ×	14%
	LiveJournal	1.13 ×	7%
	USA-road	1.16 ×	11%
	twitter	1.15 ×	12%
PR	rmat26	1.20 ×	5%
	random26	1.15 ×	7%
	LiveJournal	1.21 ×	7%
	USA-road	1.19 ×	6%
	twitter	1.22 ×	7%
BC	rmat26	1.17 ×	9%
	random26	1.12 ×	13%
	LiveJournal	1.15 ×	10%
	USA-road	1.19 ×	12%
	twitter	1.14 ×	11%
	Mean	**1.16 ×**	**10%**

(a) Effect of memory coalescing

Graphs	Speedup	Inaccuracy
rmat26	1.26 ×	12%
random26	1.08 ×	17%
LiveJournal	1.22 ×	13%
USA-road	1.30 ×	13%
twitter	1.18 ×	12%
rmat26	1.22 ×	16%
random26	1.10 ×	18%
LiveJournal	1.18 ×	16%
USA-road	1.20 ×	19%
twitter	1.16 ×	15%
rmat26	1.20 ×	12%
random26	1.10 ×	16%
LiveJournal	1.22 ×	13%
USA-road	1.20 ×	12%
twitter	1.18 ×	13%
rmat26	1.32 ×	7%
random26	1.16 ×	11%
LiveJournal	1.26 ×	7%
USA-road	1.30 ×	5%
twitter	1.22 ×	9%
rmat26	1.24 ×	14%
random26	1.13 ×	18%
LiveJournal	1.21 ×	16%
USA-road	1.26 ×	15%
twitter	1.17 ×	13%
Mean	**1.20 ×**	**13%**

(b) Effect of memory latency

Graphs	Speedup	Inaccuracy
rmat26	1.06 ×	8%
random26	1.03 ×	9%
LiveJournal	1.07 ×	8%
USA-road	1.12 ×	7%
twitter	1.09 ×	6%
rmat26	1.05 ×	10%
random26	1.02 ×	11%
LiveJournal	1.07 ×	8%
USA-road	1.09 ×	10%
twitter	1.05 ×	9%
rmat26	1.04 ×	9%
random26	1.00 ×	7%
LiveJournal	1.04 ×	6%
USA-road	1.05 ×	9%
twitter	1.06 ×	8%
rmat26	1.10 ×	4%
random26	1.04 ×	9%
LiveJournal	1.08 ×	5%
USA-road	1.06 ×	8%
twitter	1.09 ×	8%
rmat26	1.11 ×	11%
random26	1.05 ×	14%
LiveJournal	1.09 ×	9%
USA-road	1.12 ×	7%
twitter	1.06 ×	12%
Mean	**1.07 ×**	**8%**

(c) Effect of thread divergence

ParTBC

Graph	Time (sec)		Speedup	Graph reordering
	NVR	VR	(NVR/VR)	Time(sec)
fb-Friendships	1020	836	1.22×	5
soc-Pokec	214578	174453	1.23×	32
loc-Gowalla	26760	22677	1.18×	9
roadnetSF	2900	2566	1.13×	5
usroad48	3810	3371	1.13×	4
rmat17	1550	1302	1.19×	6
random17	616	592	1.04×	7

Effect of vertex numbering on exact version
(NVR: no vertex renumbering, VR: with vertex renumbering)

ParTBC

Graph	Speedup w.r.t.		Speedup breakdown (w.r.t. exact parallel)	
	Exact Parallel	ABRA	VR	DynRR
fb-Friendships	2.80×	4.28×	1.22×	2.29×
soc-Pokec	2.76×	4.31×	1.20×	2.30×
loc-Gowalla	2.48×	4.16×	1.18×	2.10×
roadnetSF	2.71×	4.72×	1.13×	2.40×
usroad48	2.68×	4.63×	1.13×	2.37×
rmat17	2.65×	4.18×	1.19×	2.22×
random17	1.67×	1.92×	1.04×	1.61×
geomean	**2.5**×	**3.88**×	**1.15**×	**2.17**×

Performance of ParTBC w.r.t. exact parallel Brandes' algorithm and ABRA (VR: vertex renumbering) Error $\sim$ 6%.

ParTBC

Graph	Global Memory Load Efficiency (%)	
	Without Vertex Renumbering	**With Vertex Renumbering**
fb-Friendships	19.5	59.4
soc-Pokec	23.3	70.7
loc-Gowalla	24.2	64.6
roadnetSF	25.2	66.3
usroad48	26.3	57.8
rmat17	22.1	63.5
random17	13.8	39.2

Effect of vertex renumbering on global memory coalescing