A Performance and Recommendation System for Parallel Graph Processing Implementations

Work-In-Progress, ICPE 2019

Samuel D. Pollard, Sudharhsan Srinivasan, Boyana Norris

Computer and Information Science
https://hpcl.cs.uoregon.edu

April 11, 2019

HPCL

Steps to (Compute a Graph Property Quickly)

1. Find a package
2. Nail down dependencies
3. Reformat graph input file
4. Learn how to use CLI/API
5. Run gamut of experiments
6. Measure performance of experiments
7. Combine, analyze, and decide
8. Link with existing workflow

Steps to Quickly (Compute a Graph Property Quickly)

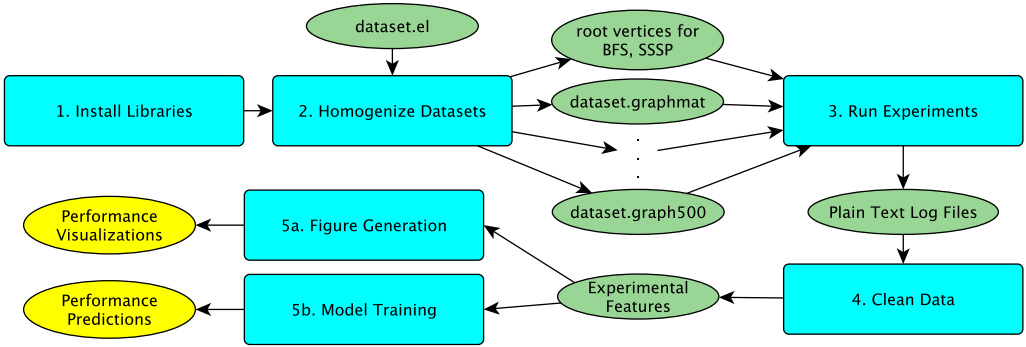
1. ~~Find a package~~
2. ~~Nail down dependencies~~
3. ~~Reformat graph input file~~
4. ~~Learn how to use CLI/API~~
5. Run gamut of experiments
6. Measure performance of such experiments
7. ~~Combine, analyze, and decide~~
8. Link with existing workflow

Parallel Graph Algorithms

HPCL

- Social network analysis [1]
  - Twitter has its own graph processing package (GraphJet)
  - Google has its own language (Pregel)
- Bioinformatics [4]
  - Tend to have more complex analysis of smaller datasets
  - e.g. protein interaction networks
  - non-e.g. Needleman-Wunsch algorithm for global sequence alignment

Workflow


dataset.el
root vertices for BFS, SSSP
1. Install Libraries
2. Homogenize Datasets
dataset.graphmat
3. Run Experiments
dataset.graph500
Plain Text Log Files
Performance Visualizations
5a. Figure Generation
Performance Predictions
5b. Model Training
Experimental Features
4. Clean Data

HPCL

Before Reproducibility...

- Ensure we're actually measuring the right times!
- File read time was mistakenly used as performance measurement

- Abridged log file from GraphMat
  - finished file read. time: **2.65211**
  - load graph: 5.91229 sec
  - initialize engine: 8.32081e-05 sec
  - run algorithm 1 (count degree): 0.0555639 sec
  - run algorithm 2 (compute PageRank): 0.149445 sec
  - print output: 0.0641179 sec
  - deinitialize engine: 0.00022006 sec

File Formats

- Binary can be much faster (factor of at least 3)
  - No parsing, can just store into array
  - Less portable
  - May be serialization of internal data structures
- 0-indexed or 1-indexed?
  - May not be interchangeable

this?

```
55555 44444
44444 22222
11111 66666
```

or this?

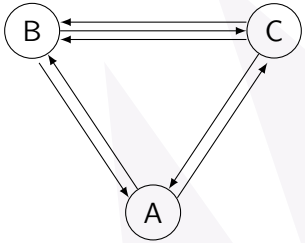
```
5 4
4 2
1 6
```

or this?

```
4 3
3 1
0 5
```

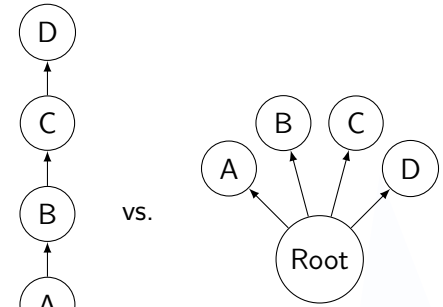
Differing Results

- PageRank stopping criterion
  - $||p_t - p_{t-1}||_1$
  - $||p_t - p_{t-1}||_\infty$
  - Stop when no weights change (machine ϵ)
- Triangle counting
  - Count both directions of triangle?
  - One, two, or three triangles?



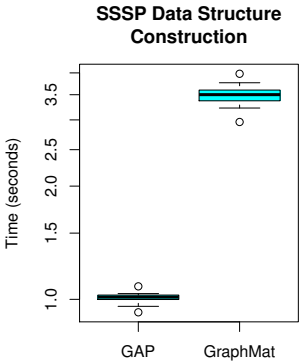
Starting (Root) Vertices

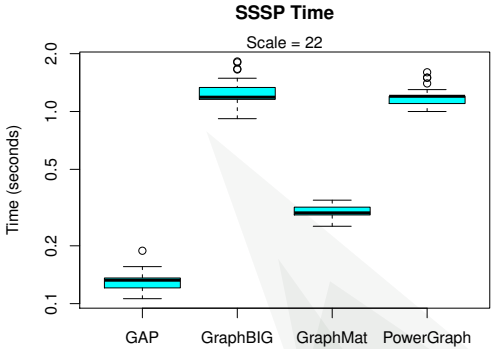
- Performance of BFS and SSSP depends on where you start
- More reachable vertices



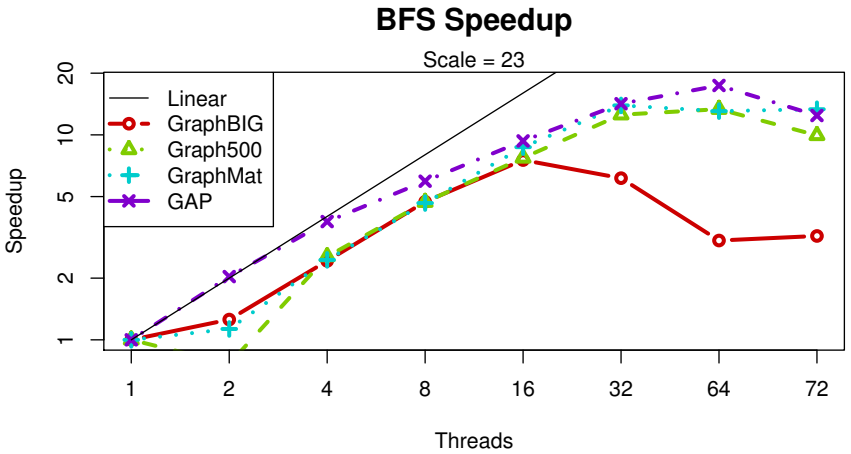
Early Results

- Data structure construction time sometimes not split





Early Results



$\approx 16 * 2^{23}$

$\approx$ 135M

edges

Be Careful with Speedup

- What if I naïvely wrote serial implementation?
- Graph500 has Serial, OpenMP, and MPI implementations

HPCL

Recommendation and Ranking

- Pick the best graph package for your hardware and graph
  1. Compute features of a graph
     - This may be expensive
  2. Train a model based on these features
- Apply work in linear solver recommendations [5] to graph processing packages

Computing Features

- 12 features (the ones computed on SNAP)
- e.g. # vertices, # edges, diameter, clustering coefficient

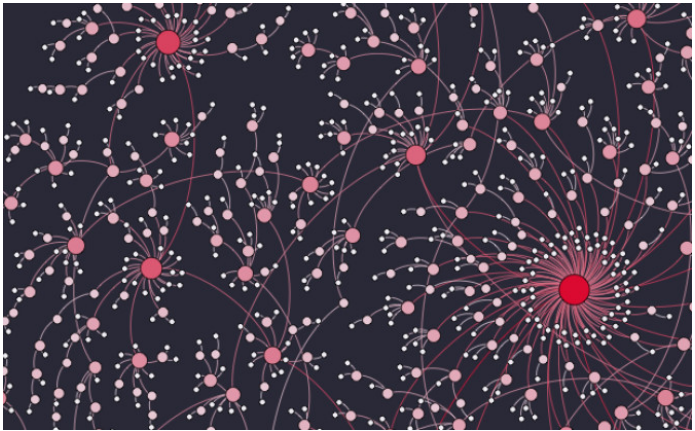
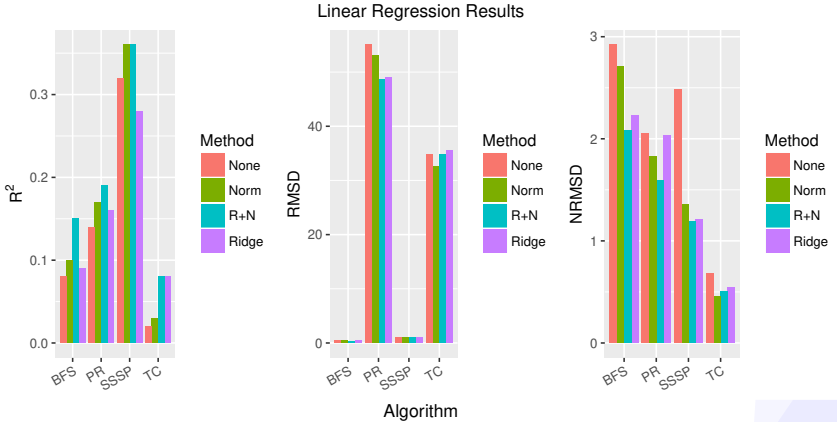


Figure: A scale free network has low diameter [3]

Training Models



- Worst case: BFS with factor of 2
- Best case: TC with factor of 0.5

Classification

Random Forest

		good	bad
TC	good	43	1
	bad	0	118
		good	bad
BFS	good	110	0
	bad	0	147
		good	bad
PR	good	75	0
	bad	1	193
		good	bad
SSSP	good	51	11
	bad	9	198

- Columns are predictions
- Rows are observations

HPCL

Parameter Tuning

- BFS - α and β (direction-optimizing). This is a combination of:
  - bottom-up: unvisited nodes searching for visited parents
  - top-down: visited nodes searching for unvisited children
- SSSP - Δ-stepping; order nodes using ranges of size Δ to distribute work

More More More!

- Experiments
  - O(10,000)
  - Probably some will fail
  - Storage concerns
- Packages
  - Difficult to learn different APIs, CLIs, and ensure consistent measurement
  - Leaderboard (in the spirit of Graph500) could motivate package authors
- Algorithms
  - Packages may not provide reference implementations
  - Rolling our own may not be as efficient as expert users

Containerization

- Singularity [2] an attractive, HPC-focused option

Conclusion

- Automated performance collection of parallel graph processing implementations
  - 6 packages
  - 4 algorithms
- Ran experiments and generated models to predict performance for a given (hardware, graph) pair
- Speedup over random selection ranges from 7% (PageRank) to 700% (BFS)

References I

[1] Kang, U., Meeder, B., and Faloutsos, C.
Spectral analysis for billion-scale graphs: Discoveries and implementation.
In Advances in Knowledge Discovery and Data Mining (Berlin, 2011), vol. 6635 of Lecture Notes in Computer Science, Springer.

[2] Kurtzer, G. M., Sochat, V., and Bauer, M. W.
Singularity: Scientific containers for mobility of compute.
PLOS ONE 12, 5 (May 2017), 1–20.

[3] Lamberson, P. J.
Scale-free network.
Available at http://social-dynamics.org/scale-free-network/.

[4] Pavlopoulos, G. A., Secrier, M., Moschopoulos, C. N., Soldatos, T. G., Kossida, S., Aerts, J., Schneider, R., and Bagos, P. G.
Using graph theory to analyze biological networks.
In BioData Mining (Bethesda, MD, 2011), PubMed Central.

References II

[5] Sood, K., Norris, B., and Jessup, E.
Comparative performance modeling of parallel preconditioned krylov methods.
In IEEE 19th International Conference on High Performance Computing and Communications; 15th International Conference on Smart City; 3rd International Conference on Data Science and Systems (Dec. 2017), HPCC/SmartCity/DSS, pp. 26–33.