HPCGraph: Benchmarking Massive Graph Analytics on Supercomputers

George M. Slota[1], Siva Rajamanickam[2], Kamesh Madduri[3]

[1]Rensselaer Polytechnic Institute
[2]Sandia National Laboratories[a]
[3]The Pennsylvania State University
www.gmslota.com slotag@rpi.edu

11 October 2016

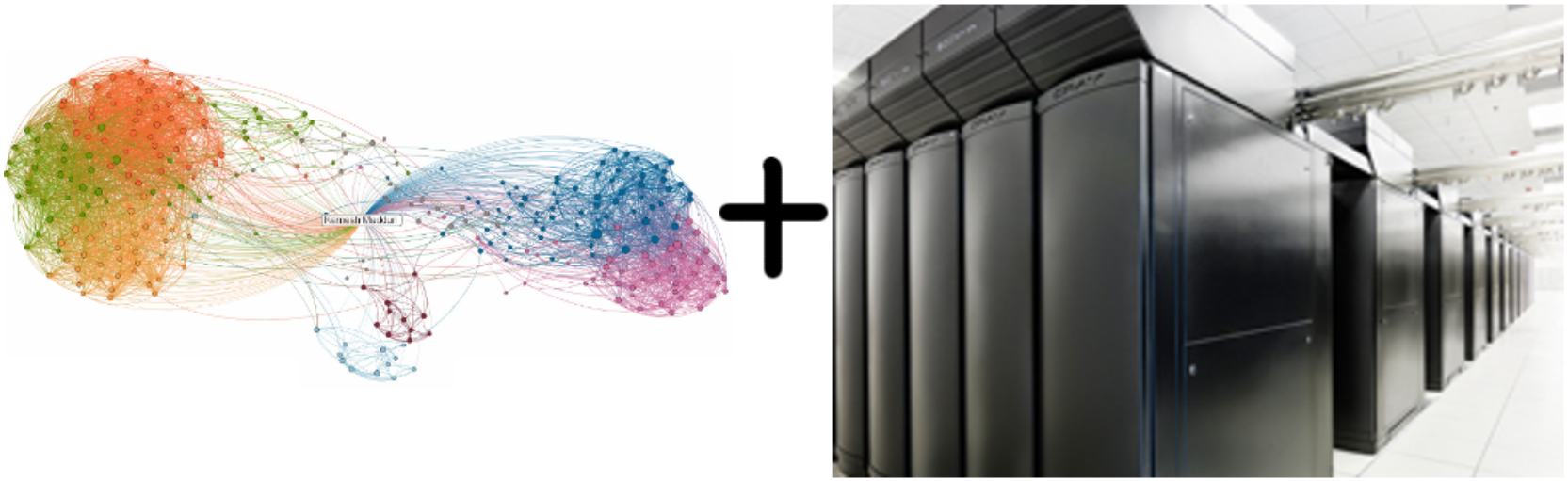
[a]Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Presentation Overview

- Motivating massive-scale distributed-memory analytics and benchmarking
- Approach for massive-scale analytics
- HPCGraph: Benchmark using 6 graph kernels
  - PageRank, Harmonic centrality, SCC decomposition, WCC decomposition, K-core computation, Label Propagation
- Performance results on the Blue Waters supercomputer

What?

Graph Analytics and HPC



Or, given modern extreme-scale graph-structured datasets (e.g. web crawls, brain graphs, human interaction networks) and modern high performance computing systems (e.g. Blue Waters), how should we develop generalized approaches to efficiently study such datasets on such systems? How would we go about evaluating such approaches?

Why?

Why do want to study these large graphs?

Human Interaction Graphs:

- Finding hidden communities, individuals, malicious actors
- Observe how information and knowledge propagates

Brain Graphs:

- Study the topological properties of neural connections
- Finding latent computational substructures, similarities to other information processing systems

Web Crawls:

- Identifying trustworthy/important sites
- Spam networks, untrustworthy sites

Why do we want to benchmark these systems?

- Graph processing - data/memory intensive computations - traditional benchmarks don't accurately capture system performance (e.g. LINPACK)
- Insight into how to frame graph problems to develop generalized and scalable approaches
- Influence future hardware design

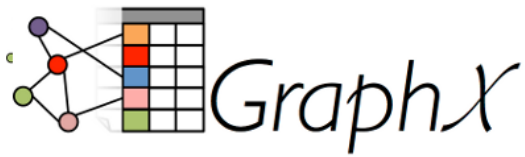
Graph500:

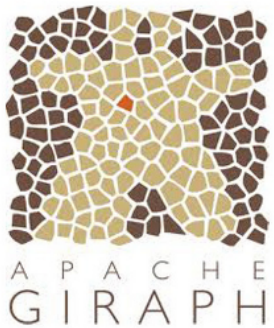
- Existing, well-known, highly optimized
- Current spec (v1.2) representative enough of real-world performance? E.g. synthetic graph, single kernel (BFS)
  - Future: SSSP and max independent set?

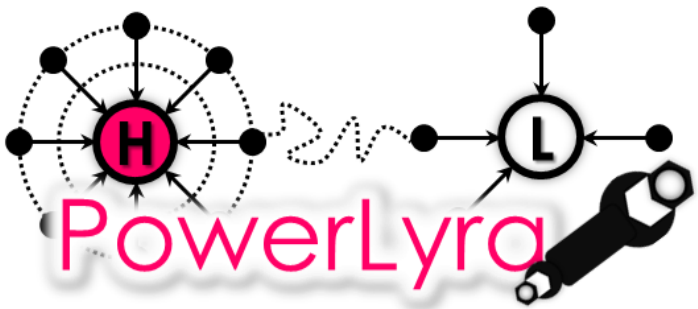
Prior Approaches - Generalized Graph Processing

Can we use them to analyze large graphs on HPC?

- Some limited by shared-memory and/or specialized hardware
- Some run in distributed memory but graph scale is still limited
- Others, graph scale isn't limiting factor but performance can be







Graph analytics on HPC

So why do we want to run graph analytics on HPC?

- Scalability for analytic performance and graph size
  - Efficient implementations should be limited only by distributed memory capacity
  - Graph500 - demonstration of performance achievable for irregular computations
- Relative availability of access in academic/research communities
  - Private clusters of various scales, shared supercomputers
  - Access for domain experts, those using analytics on real-world graphs

Can we create an approach that is as simple to use as the aforementioned frameworks but runs on common cluster hardware and gives state-of-the-art performance?

Challenges

Scale

- We consider "extreme-scale" graphs – billion+ vertices and up to trillion+ edges
- Processing these graphs requires at least hundreds to thousands of compute nodes or tens of thousands of cores
- Graph analytic algorithms are generally memory-bound; in the distributed space, this results in a ratio of communication versus computation that increases with core/node count
- **What real-world graph instances can we use?**

Complexity

- Real-world extreme-scale graphs have similar characteristics: small-world nature with skewed degree distributions
- Small-world graphs are difficult to partition for distributed computation or to optimize in terms of cache due to "too much locality"
- Skewed degree distributions make efficient parallelization and load balance difficult to achieve
- Multiple levels of cache/memory and increasing reliance on wide parallelism for modern HPC systems compounds the above challenges
- **How should we develop generalized kernels to address these challenges?**

Approach

Identifying Communication Patterns

Observation: many iterative graph algorithms have similar communication patterns - our approach should capture as many of these patterns as possible

- (Vanilla) *BFS-like*: frontier expansion, information *pushed* from vertices to adjacencies, volume of data exchanged is **variable** or fixed across iterations
- (Vanilla) *PageRank-like*: information *pulled* from incoming arcs, either **fixed** or variable communication pattern in every iteration

We develop optimized skeleton code for these four patterns, and can use it to fill in analytic-specific details

BFS-like Algorithmic Pattern

```
1:  procedure BFS-LIKE(G(V, E))                          ▷ Task Parallel
2:      for all v ∈ V do                                 ▷ Thread Parallel
3:          D(v) ← init()
4:          if addToQ(v) then
5:              Q_next ← ⟨v, D(v)⟩
6:      while any Q_next ≠ ∅ do
7:          ⟨Q, D⟩ ← AllToAllExchange(Q_next)            ▷ Thread Parallel
8:          Q_next ← ∅
9:          for all v ∈ Q do                             ▷ Thread Parallel
10:             for all ⟨v, u⟩ ∈ E do
11:                 D(u) ← update()
12:                 if addToQ(u) then
13:                     Q_next ← ⟨u, D(u)⟩
14:     return D
```

PageRank-like Algorithmic Pattern

```
1:  procedure PageRank-like(G(V, E))                    ▷ Task Parallel
2:      for all v ∈ V do                                ▷ Thread Parallel
3:          D(v) ← init()
4:          if addToQ(v) then
5:              Q_next ← ⟨v, D(v)⟩
6:      while any Q_next ≠ ∅ do
7:          ⟨Q, D⟩ ← AllToAllExchange(Q_next)           ▷ Thread Parallel
8:          Q_next ← ∅
9:          for all v ∈ Q do                            ▷ Thread Parallel
10:             for all ⟨v, u⟩ ∈ E do
11:                 D(v) ← update()
12:             if addToQ(v) then
13:                 Q_next ← ⟨v, D(v)⟩
14:     return D
```

Implementation Considerations

Choices, choices, choices ...

Tradeoffs (ease of implementation vs. scalability):

- **1D** (vertex-based) vs. 2D (edge-based) partitioning and graph layout
- **Bulk-synchronous** vs. asynchronous communication
- Programming language and parallel programming model
  - High-level language (e.g., Scala) vs. **C/C++**
  - High-level model (e.g., Spark) vs. MPI-only vs. **MPI+OpenMP**

Other considerations:

- In-memory graph representation
  - **Vanilla CRS-like** vs. compressed (e.g., with RLE) adjacencies
- Partitioning strategy (with 1D layout)
  - **Vertex-balanced, Edge-balanced, Random** vs. Explicit partitioning (PuLP)

Benchmark Kernel Details

BFS-like:

- **WCC**: 1st stage of MULTISTEP-WCC (BFS-based)
- **SCC**: 1st/2.2 stage of MULTISTEP-SCC ([Fleischer et al., 2000])
- **Harmonic Centrality**: Routine for calculating centrality value of any given vertex

PageRank-like:

- **PageRank**: Iterative algorithm
- **Label Propagation**: Community detection algorithm ([Raghavan et al., 2007])
- **WCC**: 2nd stage of MULTISTEP-WCC (Color-propagation)
- **SCC**: Stage 2.1 of MULTISTEP-SCC ([Orzan, 2004])
- **K-core**: Output all vertex corenesses ([Montresor et al., 2013])

Plus:

- Parallel methods for I/O or edge list generation
- Parallel methods for graph creation
- Parallel methods for partitioning (PULP, other methods)

Performance Results on Blue Waters

Experimental Setup

Test system, Graphs

- *Blue Waters*: Dual-socket AMD Interlagos 6276 - 16 cores and 64 GB memory per node
- Primary real-world instance: Web Data Commons 2012 crawl
- Strong scaling on WDC12 and random graphs
- Weak scaling also evaluated on various random graphs

Graph	n	m	D_{avg}	Source
WDC12	3.6 B	129 B	36	[Meusel et al., 2015]
R-MAT	3.6 B	129 B	36	[Chakrabarti et al., 2004]
Rand-ER	3.6 B	129 B	36	Erdös-Rényi
R-MAT	2^{23}-2^{34}	2^{29}-2^{40}	64	[Chakrabarti et al., 2004]
Rand-ER	2^{23}-2^{34}	2^{29}-2^{40}	64	Erdös-Rényi
Rand-HD	2^{25}-2^{34}	2^{29}-2^{40}	64	High Diameter

WDC12 on 256 node of Blue Waters

How can we improve?

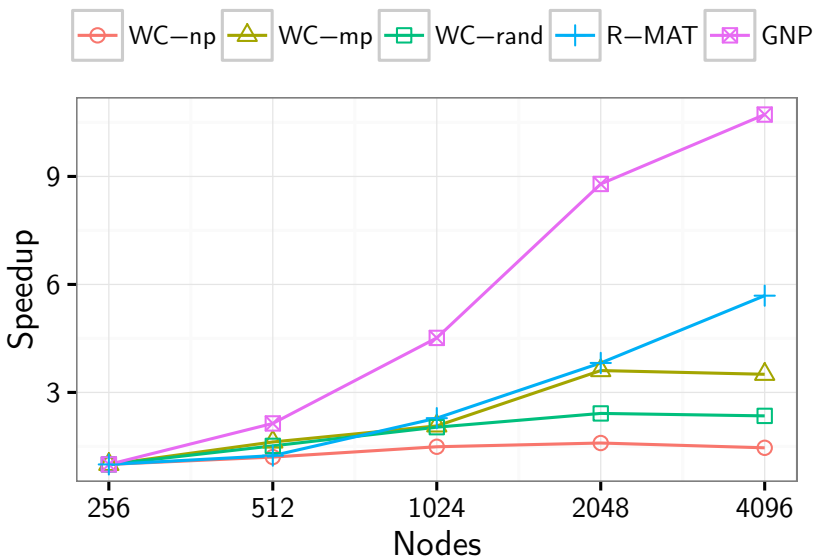
Analytic	Partitioning Strategy			
	PuLP	1DVert	1DEdge	1DRand
+PuLP	105	-	-	-
SCC	181	184	**108**	184
WCC	**39**	88	63	112
PR	**55**	87	111	227
HC	54	54	**46**	101
KC	375	445	**363**	583
LP	**59**	400	435	367
Total	**868**	1258	1126	1574

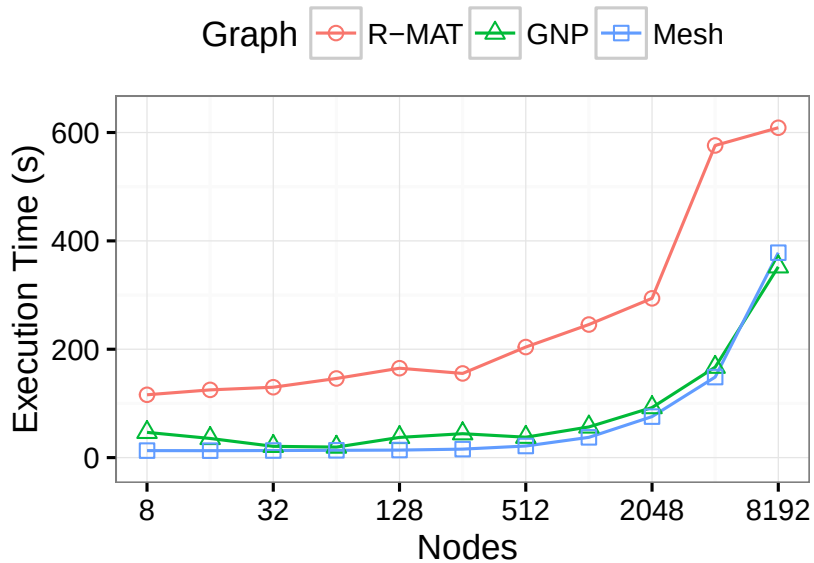
Overall performance: about 10% of Graph500 in edges processed per second

Weak and Strong Scaling

Label propagation-based analytics

- Strong scaling on *Blue Waters* for label propagation community detection with WC and random graphs
- Weak scaling on *Blue Waters* for label propagation-based algorithm on random graphs

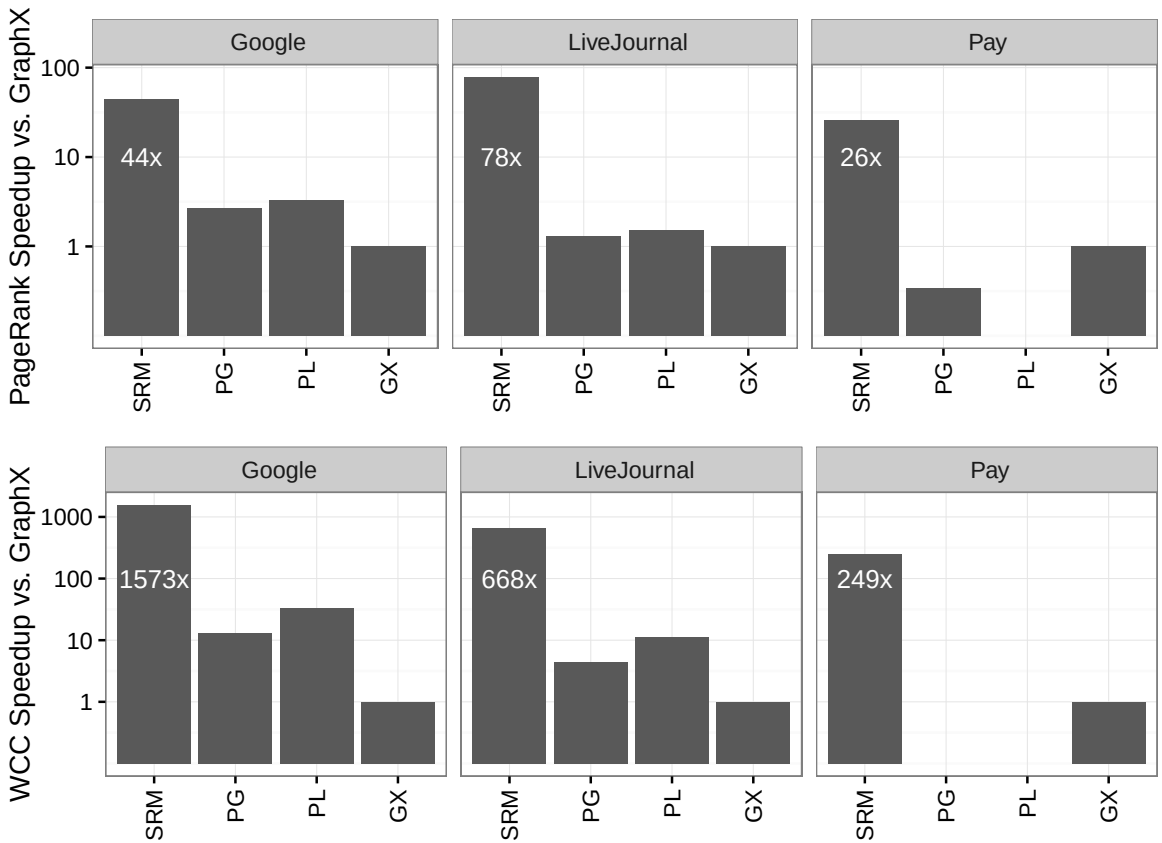




Comparison to Distributed Graph Frameworks

Our approach vs. GraphX, PowerGraph, PowerLyra

- Compared GraphX (GX), PowerGraph (PG), and PowerLyra (PL) on 16 nodes to our code (SRM)
- Google (5M edges) and LiveJournal (70M edges) crawls from SNAP; Pay-level domain of WDC12 (600M edges)
- About 38× faster on average for PageRank (top), 201× faster for WCC (bottom) against distributed memory frameworks



Acknowledgments

- **Sandia and FASTMATH**
  - This research is supported by NSF grants CCF-1439057 and the DOE Office of Science through the FASTMath SciDAC Institute. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.
- **Blue Waters Fellowship**
  - This research is part of the Blue Waters sustained petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070, ACI-1238993, and ACI-1444747) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana Champaign and its National Center for Supercomputing Applications.
- **Kamesh Madduri's CAREER Award**
  - This research was also supported by NSF grant ACI-1253881.

Possible Future Extensions

- Processing **quadrillion-edge** (petascale) graphs?
- Identify and implement additional analytics that fit push/pull/fixed/variable communication patterns
- Evaluate generalizable performance optimizations? Asynchronous communication, graph compression, other partitioning strategies, push-pull hybrid methods
- Open-source code:
  - Contact slotag@rpi.edu for current version

Bibliography I

Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In *Proc. Int'l. Conf. on Data Mining (SDM)*, 2004.

L. K. Fleischer, B. Hendrickson, and A. Pinar. On identifying strongly connected components in parallel. In *Parallel and Distributed Processing*, volume 1800 of *LNCS*, pages 505–511. Springer Berlin Heidelberg, 2000.

Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, and Christian Bizer. The graph structure in the web - analyzed on different aggregation levels. *J. Web Sci.*, 1(1):33–47, 2015.

Alberto Montresor, Francesco De Pellegrini, and Daniele Miorandi. Distributed k-core decomposition. *IEEE Transactions on parallel and distributed systems*, 24(2):288–300, 2013.

S. Orzan. *On Distributed Verification and Verified Distribution*. PhD thesis, Vrije Universiteit, 2004.

U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.