XtraPuLP

Partitioning Irregular Graphs at the Trillion-Edge Scale

George M. Slota[1] Sivasankaran Rajamanickam[2]
Kamesh Madduri[3] Karen Devine[2]

[1]Rensselaer Polytechnic Institute, [2]Sandia National Labs, [3]The Pennsylvania State University
slotag@rpi.edu, srajama@sandia.gov, madduri@cse.psu.edu, kddevin@sandia.gov

SIAM CSE17 28 Feb 2017

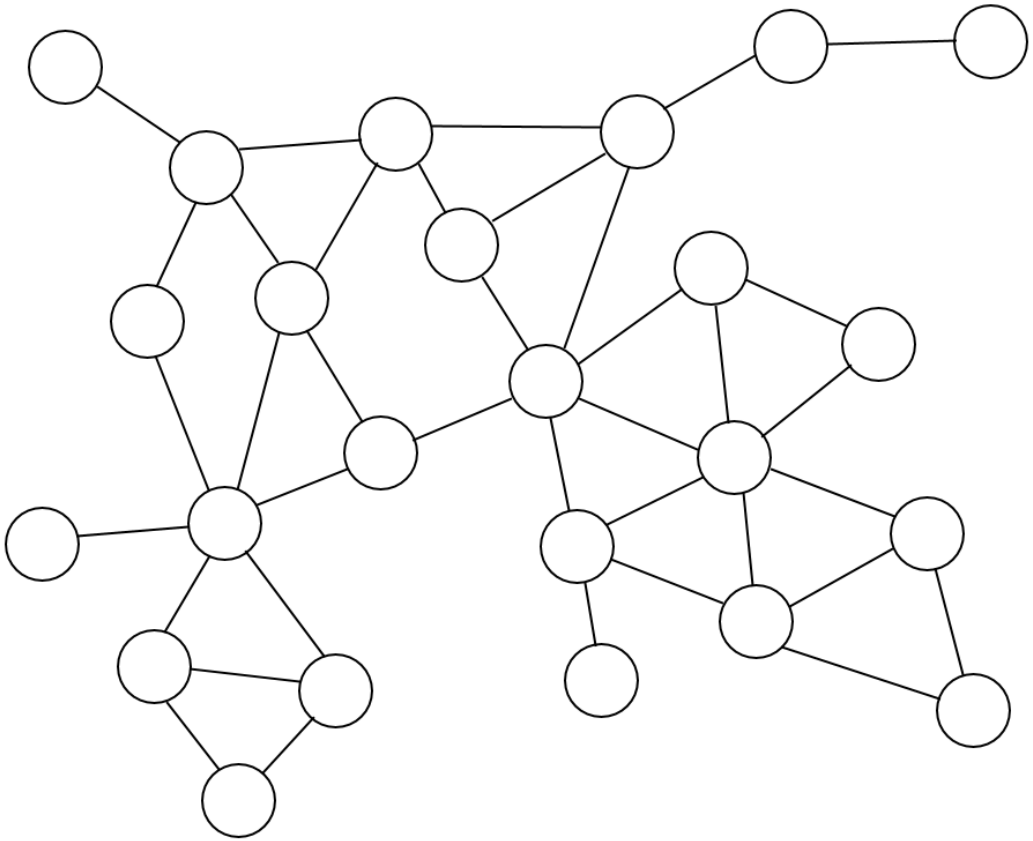
Highlights

- We present XtraPuLP, a multi-constraint multi-objective distributed-memory partitioner based on PuLP, a shared-memory label propagation-based graph partitioner
- Scales to 17 billion vertices and 1.1 trillion edges - **several orders-of-magnitude larger than any in-memory partitioner is able to process**; partitions these graphs on **131,072 cores** of *Blue Waters* in minutes
- Cut quality within small factor of state-of-the-art
- Code available: `https://github.com/HPCGraphAnalysis/PuLP` - interface also exists in Zoltan2 Trilinos package
- Paper to appear in IPDPS 2017

Label Propagation

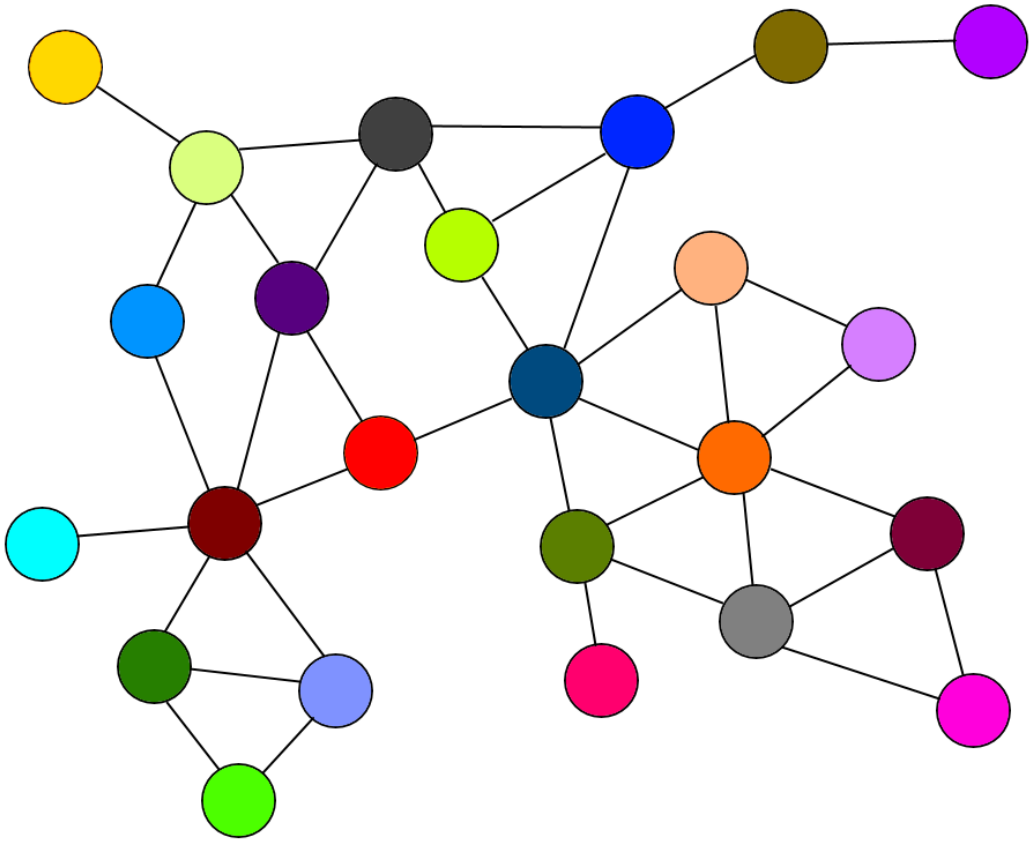
Label Propagation
Algorithm progression

- Randomly label with $n = \#verts$ labels



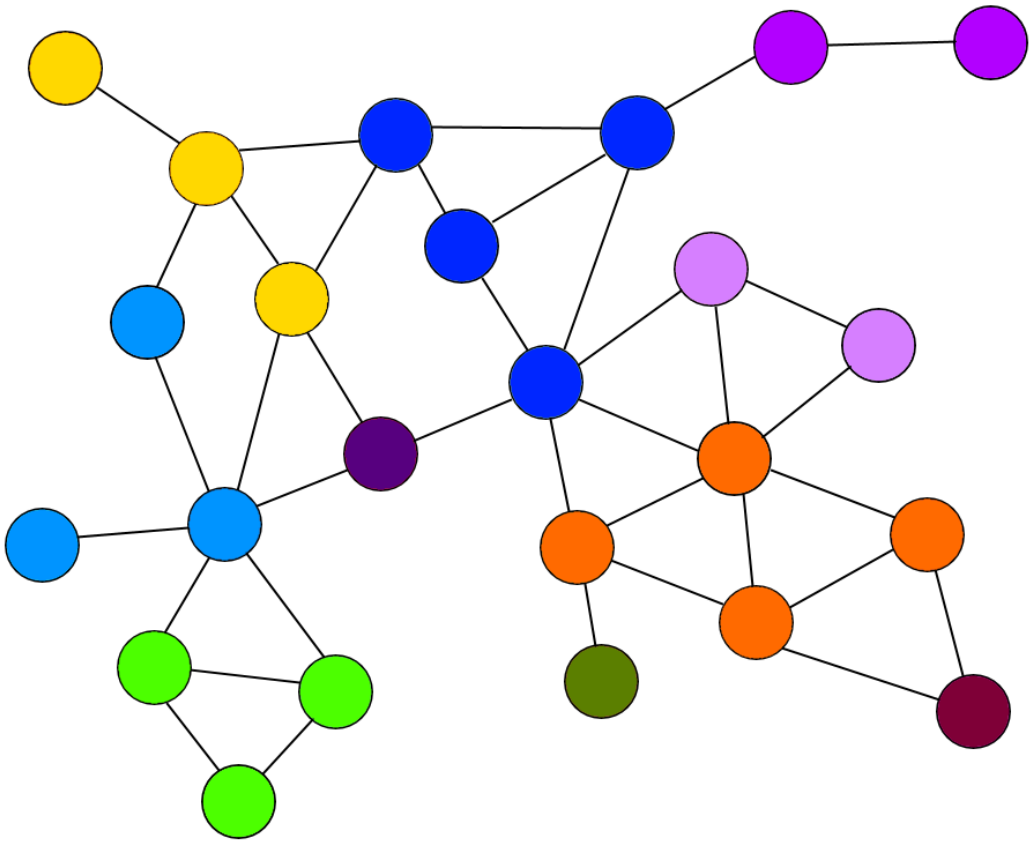
Label Propagation
Algorithm progression

- Randomly label with $n = \#verts$ labels



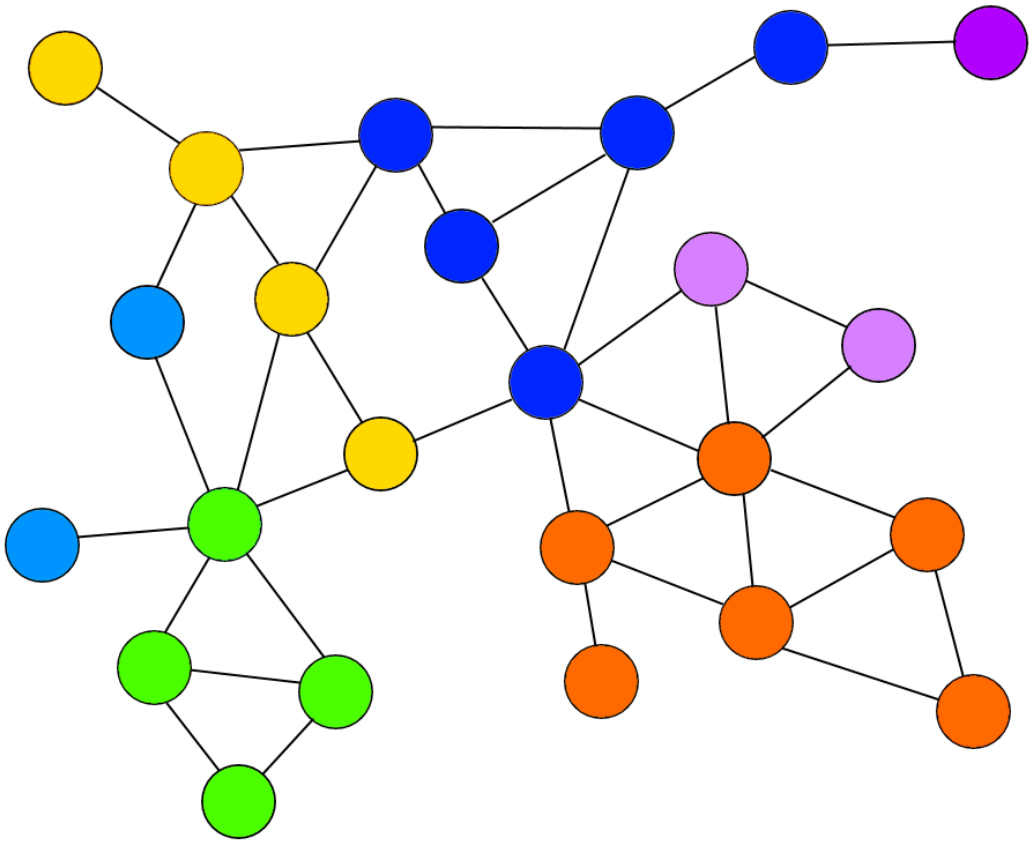
Label Propagation
Algorithm progression

- Randomly label with $n = \#verts$ labels
- Iteratively update each $v \in V(G)$ with max per-label count over neighbors with ties broken randomly



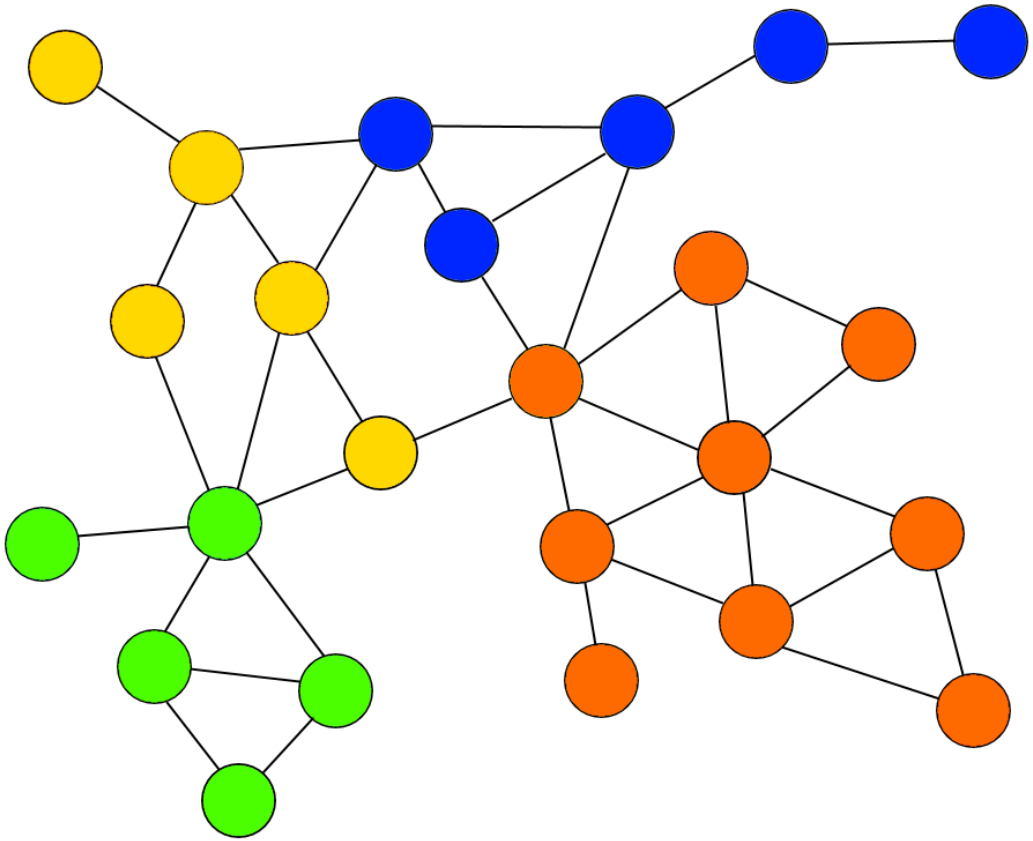
Label Propagation
Algorithm progression

- Randomly label with $n = \#verts$ labels
- Iteratively update each $v \in V(G)$ with max per-label count over neighbors with ties broken randomly



Label Propagation
Algorithm progression

- Randomly label with $n = \#verts$ labels
- Iteratively update each $v \in V(G)$ with max per-label count over neighbors with ties broken randomly
- Algorithm completes when no new updates possible; in large graphs, fixed iteration count



Label Propagation Partitioning
Prior Work

Multilevel methods:

- [Wang et al., 2014] - label prop to coarsen, METIS to partition
- [Meyerhenke et al., 2015] - label prop to coarsen, KaFFPaE to partition
- Benefits: High relative quality
- Drawbacks: Overheads of multilevel framework

Single level methods:

- [Ugander and Backstrom, 2013] - direct partition via constrained label prop
- [Vaquero et al.] - dynamic partitioning via constrained label prop
- Benefits: Low overhead, high scalability
- Drawbacks: Low relative quality

Our original PuLP implementation [Slota et al., 2014] showed quality near the former and scalability higher than the latter. **How do we further scale for processing current and forthcoming massive-scale datasets (e.g. brain graphs, web crawls, social networks)?**

PuLP
Algorithm overview

- XtraPuLP algorithm follows outline of original PuLP
- Constrain: vertices and edges per part
- Minimize: global cut and cuts per part
- Iterate between satisfying various balance constraints and objectives

Initialize p partitions
for Some number of iterations **do**
    Label propagation constraining vertices per part
    Refine partitions to minimize edge cut
for Some number of iterations **do**
    Label propagation constraining edges and cut edges per part
    Refine partitions to minimize edge cut

XtraPuLP

Challenges
Partitioning at the Trillion Edge Scale

- No longer can assume graph or part assignments fit in shared-memory; graph structure and part assignments need to be fully distributed
- MPI communication methods need to be as efficient as possible; up to $O(m)$ data exchanged all-to-all among $O(nprocs)$ tasks during each of $O(100)$ iterations
- Parallelization methodology should account for degree skew and imbalance of large irregular graphs
- Real-time tracking of updates infeasible. At no point does any processing task have accurate global part information. Therefore, need to carefully control part assignment updates as tasks independently relabel their owned vertices

Implementation Methodology

- Storage and communication: we use the HPCGraph Framework [Slota et al., 2016] as a baseline; supplies efficient and scalable 1D graph representation
- Modify and optimize code for PageRank-like processing pattern; information is pulled in - i.e. vertex v updates its part assignment $P(v)$ to reflect some optimal given known neighborhood information combined with assumed global information
- MPI+OpenMP parallelization; everything is done thread-parallel except for MPI calls
- But still need to address: How to control label propagation updates to balance communication requirements/accuracy/quality/etc?

Controlling Part Assignment

- Our weighted label propagation algorithms update part assignments $P(v)$ using a weighting function $W(p)$, where v is more likely to join part p if p is underweight

$$P(v) = \max_p(W(p) \times |u \in N(v)| \text{ where } P(u) = p)$$
$$W(p) \propto \max(S_{max} - S(p), 0)$$

- Algorithms require knowledge of global part sizes $S(p)$ for balance/refinement - real-time global updates not feasible
- Instead, we *approximate* current sizes as $A(p)$ using known global sizes, and per-task changes observed $C(p)$ scaled by dynamic multiplier $mult$

$$A(p) = S(p) + mult \times C(p)$$

Controlling Part Assignment - mult

- Consider $mult$ to be the inverse of each task's *share* of allowed updates before part p becomes imbalanced
- A larger $mult$ means each task will compute $A(p)$ to be relatively larger, less likely to assign new vertices to p
- E.g. if $mult = numTasks$, each task can add $\frac{S_{max} - S(p)_{cur}}{numTasks}$ new vertices/edges/cut edge to part p
- We use two parameters X and Y to dynamically adjust $mult$ as iterations progress; Y controls initial size of $mult$ and X controls final size of $mult$

$$mult \leftarrow nprocs \times ((X - Y)(\frac{Iter_{\text{cur}}}{Iter_{\text{tot}}}) + Y)$$

We use $Y = 0.25$ and $X = 1.0$; each task can alter a part by $4\times$ its *share* of updates initially but only by $1\times$ its *share* finally

XtraPuLP using PageRank-like Algo Pattern

1: **procedure** PuLPAlg(Graph $G(V, E)$, Parts P) ▷ Task Parallel
2: $S, C, A,$ **mult** $\leftarrow$ **init**(P) ▷ Initialize per-task part size data
3: $\langle S \rangle \leftarrow$ **AllToAllExchange**(S) ▷ Initial global part sizes
4: $Q \leftarrow V$
5: **while** $Q \neq \varnothing$ **or** $Iter_{\text{cur}} < Iter_{\text{tot}}$ **do**
6: **for all** $v \in Q$ **do** ▷ Thread Parallel
7: **for all** $\langle v, u \rangle \in E$ **do**
8: **if** $P(v) = p \leftarrow$ update() **then**
9: $C(p) \leftarrow$ **update()** ▷ Atomic update
10: $A(p) \leftarrow$ **update()**
11: $Q_{next} \leftarrow \langle v, P(v) \rangle$
12: $\langle P \rangle \leftarrow$ AllToAllExchange(Q_{next}) ▷ Update ghost parts
13: $\langle S \rangle \leftarrow$ **AllToAllExchange**(C) ▷ Update part sizes
14: $C \leftarrow 0, A \leftarrow S$ ▷ Reset per-task changes
15: $mult \leftarrow$ **update()**
16: $Q_{next} \leftarrow \varnothing$
17: **return** D

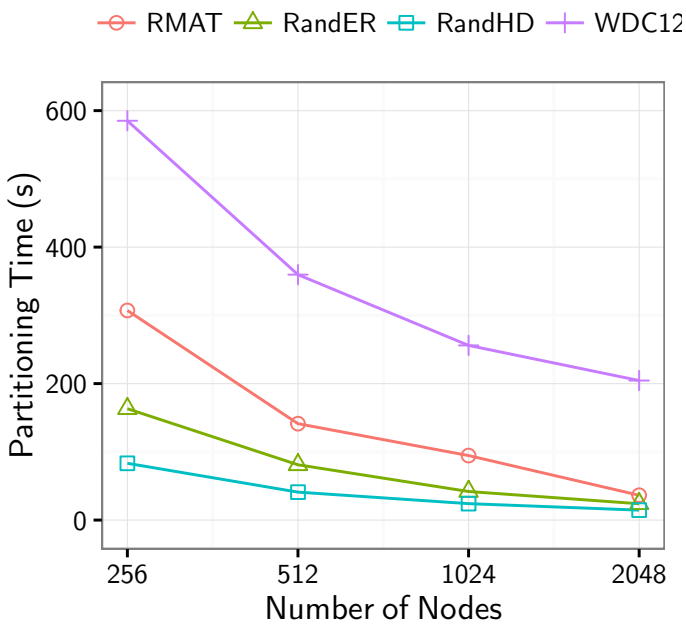
Experimental Results

Test Environment and Graphs

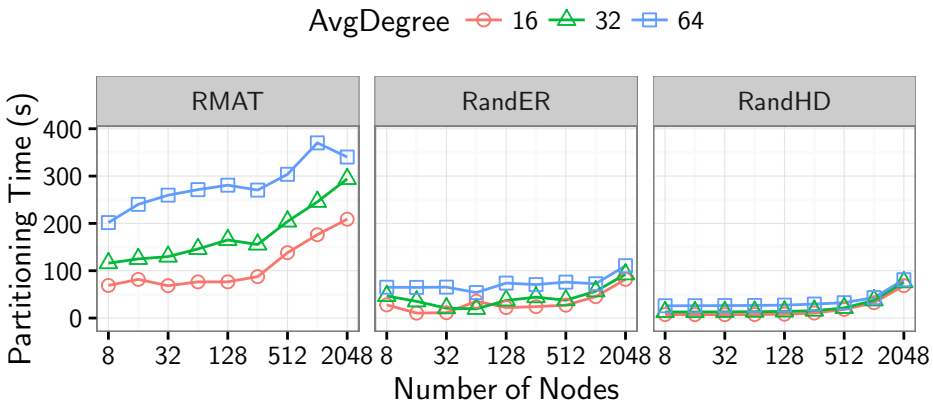
- Test systems:
  - *Blue Waters*: 2x AMD 6276 Interlagos, 16 cores, 64 GB memory, up to 8192 nodes
  - *Compton*: 2x Intel Xeon E5-2670 (Sandy Bridge), 16 cores, 64 GB memory, up to 16 nodes
- Test graphs:
  - UF Sparse Matrix, 10th DIMACS, SNAP, Max Planck Inst., Koblenz, Web Data Commons 2012 (WDC12) - social networks, web crawls, and meshes up to 128 billion edges
  - R-MAT, Erdős-Rényi (ER), High Diameter (HD) - up to 1.1 trillion edges
- Test Algorithms:
  - PuLP- multi objective and multi constraint
  - XtraPuLP- multi objective and multi constraint
  - ParMETIS - single objective and multi constraint
  - KaHIP - single objective and single constraint

Large Scale - Strong and Weak Scaling

256 - 2048 nodes of *Blue Waters*



- Strong scaling on 256 parts of WDC12, R-MAT, ER, HD (left)
- Weak scaling on random graphs - 2^{22} vertices per node (below)

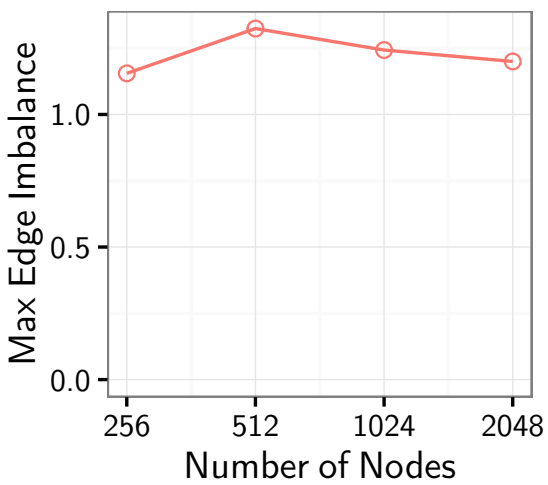


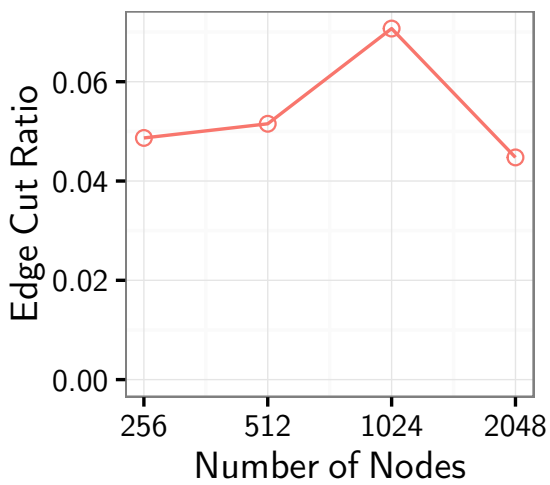
Trillion Edge Tests

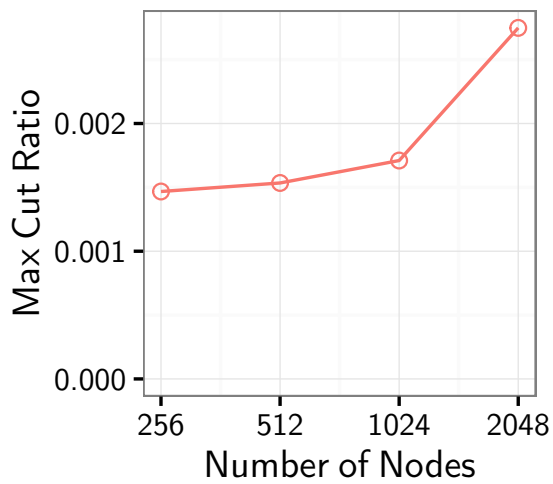
- Also attempted R-MAT, Erdős-Rényi, and high diameter graphs with 2^{34} (17 billion) vertices and 2^{40} (1.1 trillion) edges
- Ran on 8192 nodes of *Blue Waters*
- Erdős-Rényi partitioned in 380 seconds, high diameter in 357 seconds
- R-MAT graph failed; 2^{34} vertex 2^{39} edge R-MAT graph completed in 608 seconds
- No scalability bottlenecks for less skewed graphs; 1D representation limits further scalability for highly irregular graphs

Application Performance
Partitioning WDC12

- At the large scale, how does increasing processor count affect partitioning quality for a fixed number of parts?
- Edge cut ratio stays below 0.07; vs. 0.16 for vertex block and over 0.99 for random
- Note: only competing methods at this scale are block and random/hash partitioning



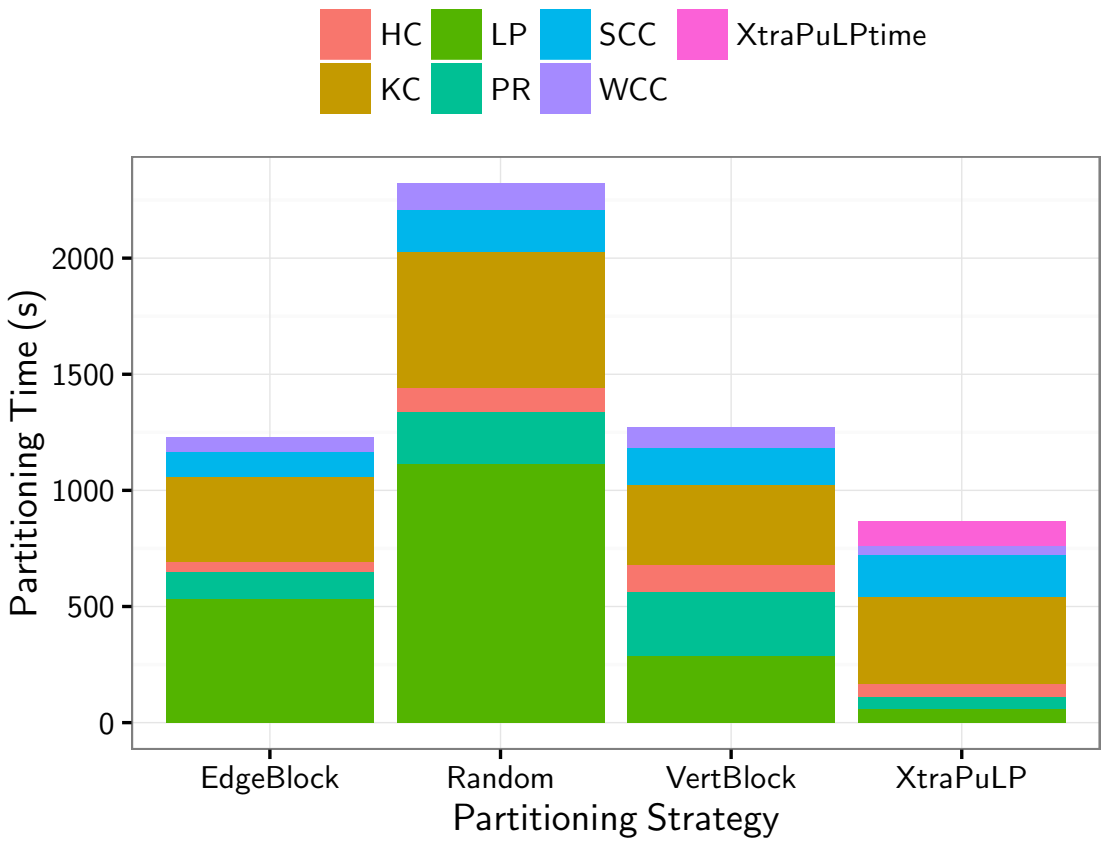




Application Performance

HPCGraph benchmark - https://github.com/HPCGraphAnalysis/HPCGraph

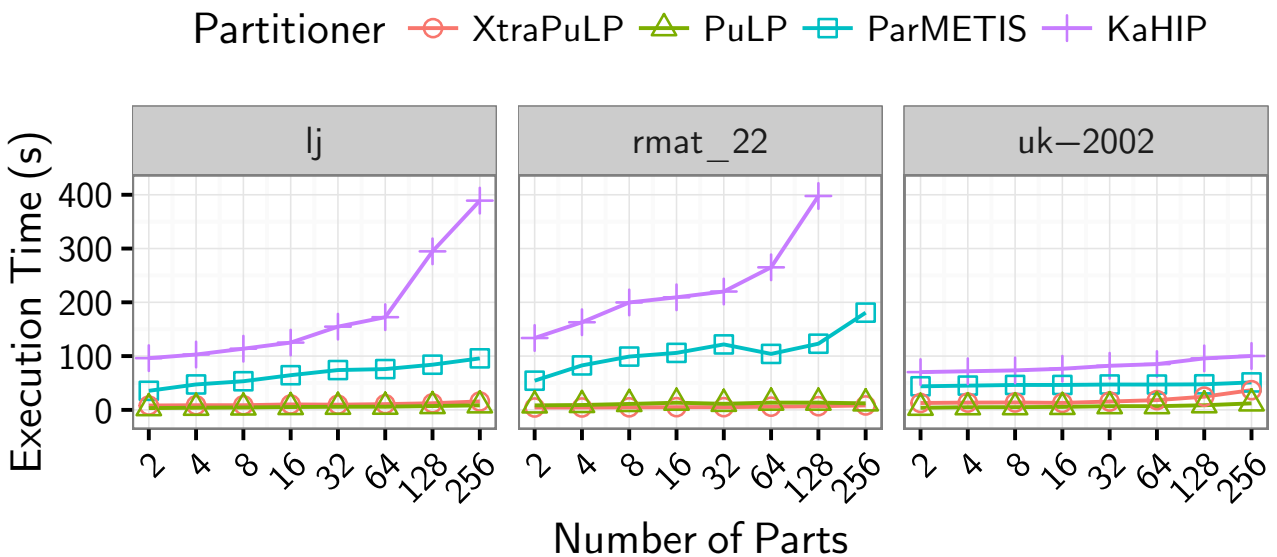
- 6 applications from HPCGraph- HC: harmonic centrality, LP: label propagation, SCC: strong connectivity, WCC: weak connectivity, PR: PageRank, KC: K-core
- 4 partitioning strategies - Edge block, vertex block, random, XtraPuLP



Comparisons to Prior Work

Performance comparisons at smaller scale

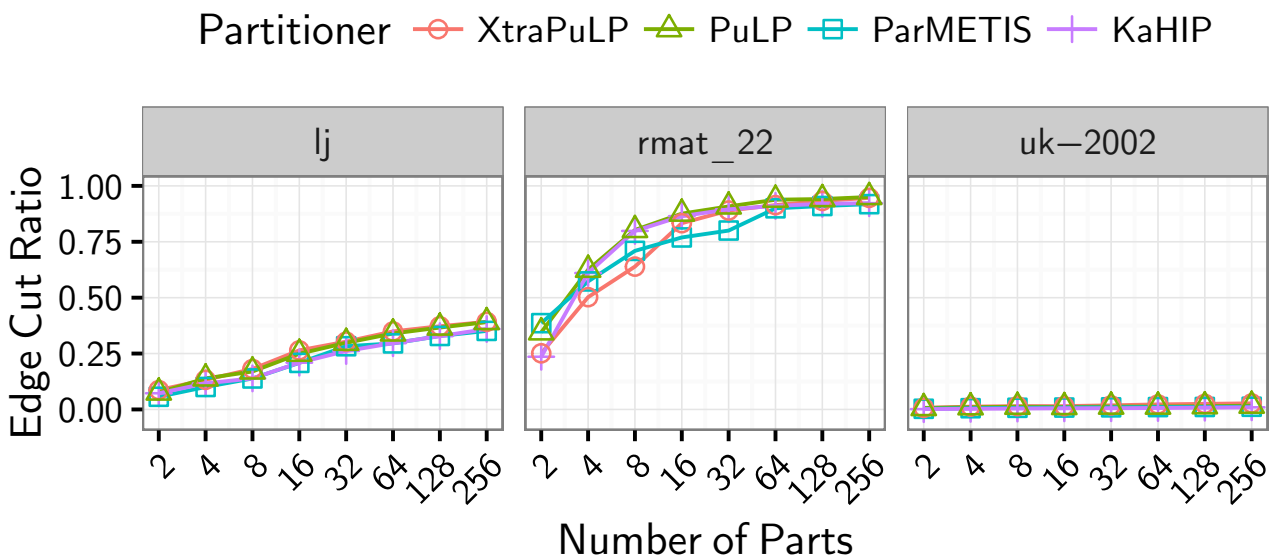
- **Multi-Constraint Scenario**: Computing 16 parts of 26 test graphs on 16 nodes, XtraPuLP is 2.5x faster than PuLP and 4.6x faster than ParMETIS; on a single node, PuLP is 1.5x faster than XtraPuLP
- **Single-Constraint Scenario**: Computing 2-256 parts of test graphs on a single node, XtraPuLP is about 1.36x slower than PuLP, 6.8x faster than ParMETIS and 15x faster than KaHIP; on more nodes, speedups versus ParMETIS and KaHIP are consistent



Comparisons to Prior Work

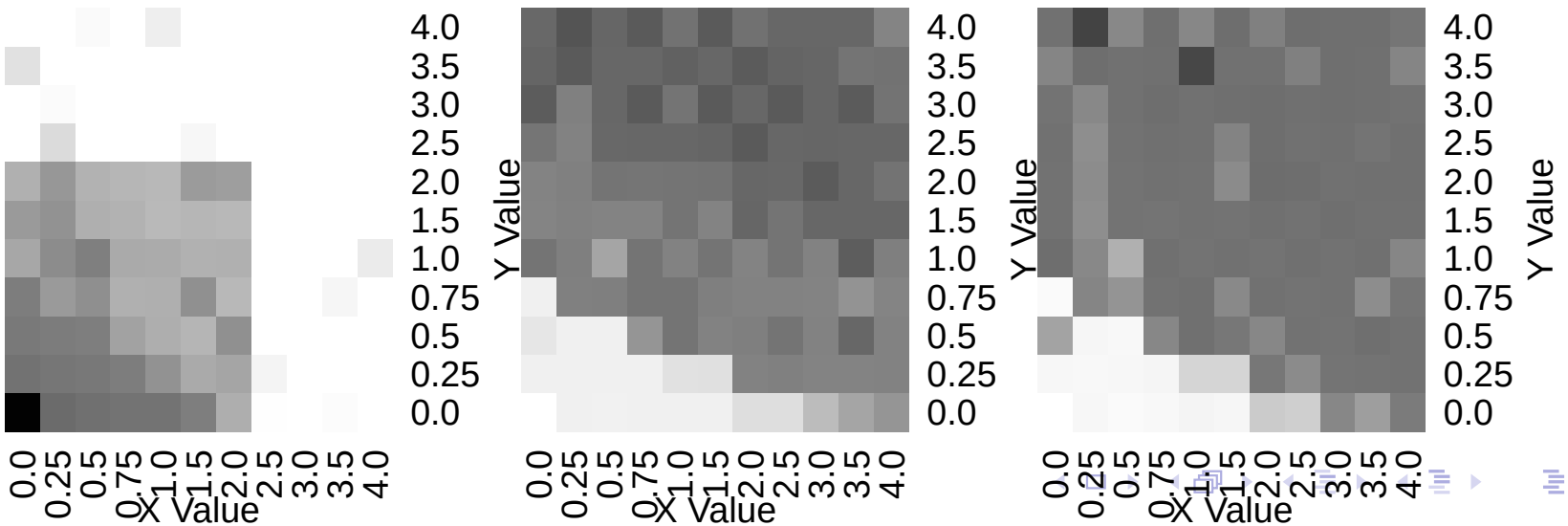
Quality comparisons at smaller scale

- **Multi-Constraint Scenario**: XtraPuLP is within 10% of PuLP and ParMETIS for both edge cut and max cut objectives
- **Single-Constraint Scenario**: XtraPuLP cuts 8% more edges than PuLP, 33% more than ParMETIS, and 50% more than KaHIP



What about X,Y parameters?

- Edge balance, edge cut, max per-part cut; values averaged across all small scale experiments; darker is worse, lighter is better
- Ideally, need to select values along *threshold* where lighter colors overlap to get both quality and balanced partitions
- Our values of $X = 1.0, Y = 0.25$ are selected empirically based on our experiments



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.

Conclusions
and future work

- XtraPuLP scales to orders-of-magnitude larger graphs than prior art
- Efficient at all scales; quality comparable to state-of-the-art for multiple network types
- XtraPuLP code available: `https://github.com/HPCGraphAnalysis/PuLP`
- Interface also exists in Zoltan2 Trilinos package: `https://github.com/trilinos/Trilinos`
- Future work:
  - Explore 2D layouts for further scaling
  - Optimize communication and update methods
  - Explore techniques to improve quality

For papers, visit: `www.gmslota.com`, or email: `slotag@rpi.edu`

Bibliography I

Henning Meyerhenke, Peter Sanders, and Christian Schulz. Parallel graph partitioning for complex networks. In *Proc. Int'l. Parallel and Distributed Processing Symp. (IPDPS)*, 2015.

G. M. Slota, K. Madduri, and S. Rajamanickam. PuLP: Scalable multi-objective multi-constraint partitioning for small-world networks. In *Proc. IEEE Int'l. Conf. on Big Data (BigData)*, 2014.

G. M. Slota, S. Rajamanickam, and K. Madduri. A case study of complex graph analysis in distributed memory: Implementation and optimization. In *Proc. IEEE Int'l. Parallel and Distributed Proc. Symp. (IPDPS)*, 2016.

J. Ugander and L. Backstrom. Balanced label propagation for partitioning massive graphs. In *Proc. Web Search and Data Mining (WSDM)*, 2013.

Luis Vaquero, Félix Cuadrado, Dionysios Logothetis, and Claudio Martella. xDGP: A dynamic graph processing system with adaptive partitioning. arXiv: 1309.1049 [cs.DC], 2013.

Lu Wang, Yanghua Xiao, Bin Shao, and Haixun Wang. How to partition a billion-node graph. In *Proc. Int'l. Conf. on Data Engineering (ICDE)*, 2014.