XtraPuLP

Partitioning Trillion-edge Graphs in Minutes

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

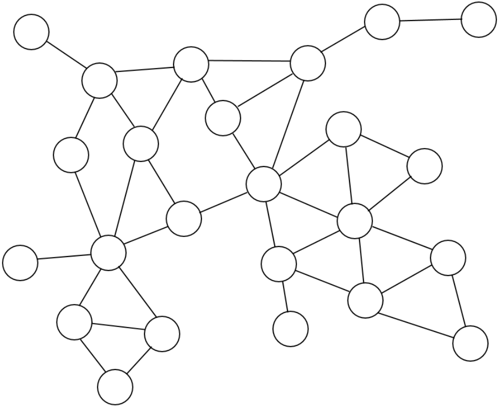
IPDPS17 31 May 2017

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: `https://github.com/HPCGraphAnalysis/PuLP` - interface also exists in Zoltan2 Trilinos package
- Slides: `http://gmslota.com/pres/PuLP-IPDPS17.pdf`

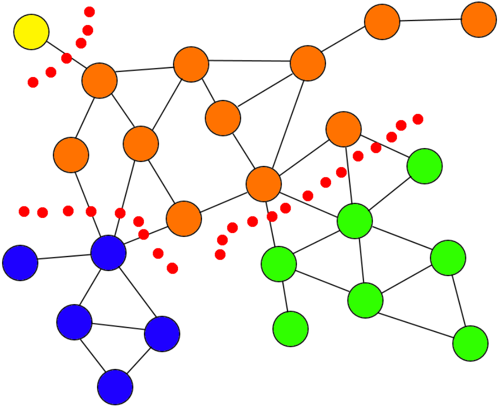
Graph Partitioning

- Input graph for some distributed computation on 4 tasks



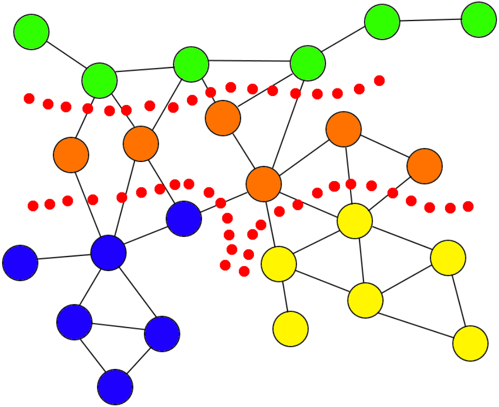
Graph Partitioning

- Input graph for some distributed computation on 4 tasks
- Vertex-disjoint partition - low cut but very imbalanced



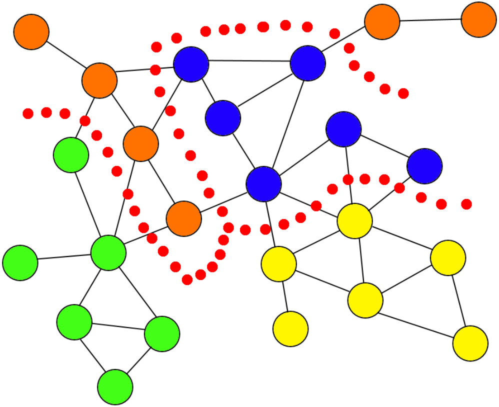
Graph Partitioning

- Input graph for some distributed computation on 4 tasks
- Vertex-disjoint partition - low cut but very imbalanced
- Balanced part sizes but high cut



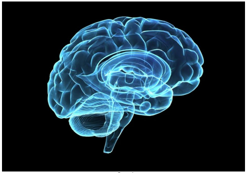
Graph Partitioning

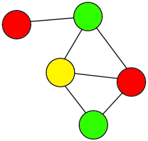
- Input graph for some distributed computation on 4 tasks
- Vertex-disjoint partition - low cut but very imbalanced
- Balanced part sizes but high cut
- Good balance and cut

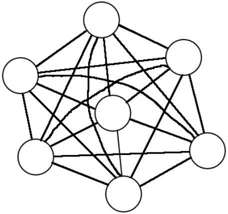


Why do we need scalable graph partitioners?

- Current and forthcoming massive scale datasets
  - Web crawls, social networks, brain graphs and other bio. networks
- Memory intensive graph computations
  - Dynamic programming-based algorithms - e.g. color-coding [Alon et al., 1995]
- High complexity graph computations
  - Subgraph, clique, path, etc., enumeration


source: forbes.com



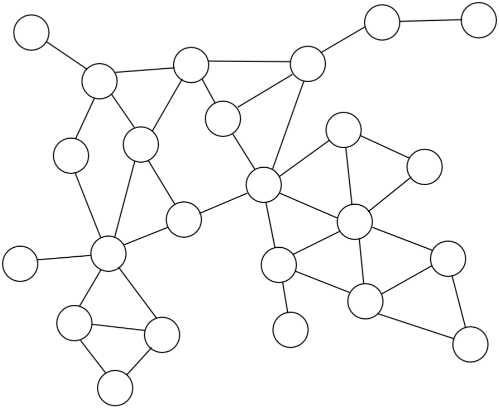


Label Propagation

(PuLP = Partitioning Using Label Propagation)

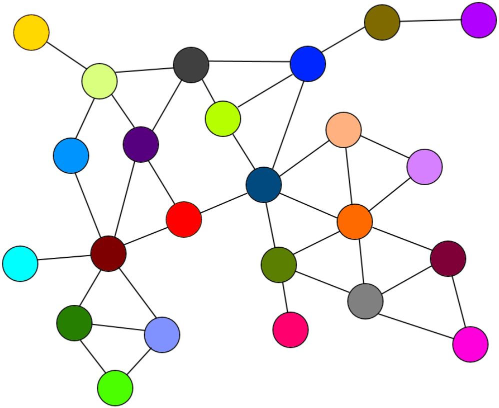
Label Propagation
Algorithm progression

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



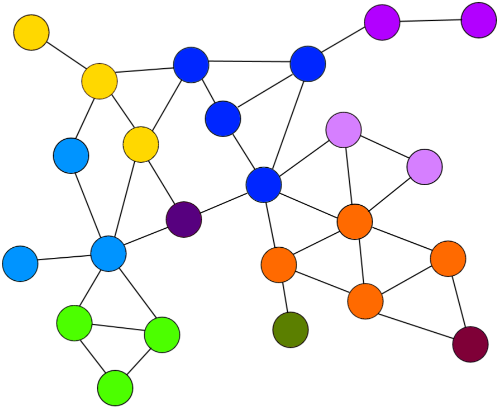
Label Propagation
Algorithm progression

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



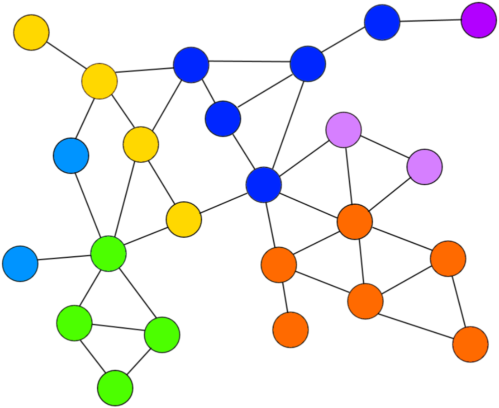
Label Propagation
Algorithm progression

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



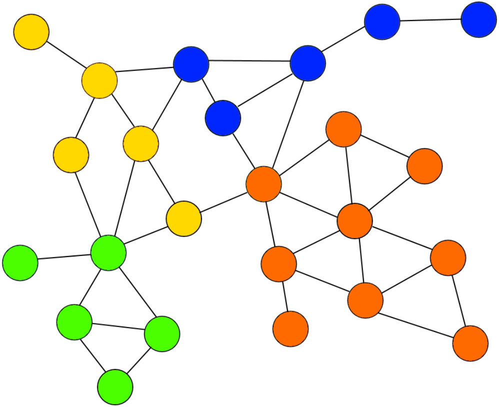
Label Propagation
Algorithm progression

- 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: Possible 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?**

PuLP
Algorithm overview

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

Initialize p partitions
for Some number of iterations **do**
    Label propagation balancing for first constraint
        and optimizing for first objective
    Refine partitions
for Some number of iterations **do**
    Label propagation balancing for second constraint
        and optimizing for second objective
    Refine partitions

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 $O(p)$ tasks during each of $O(100)$ LP iterations
- Real-time tracking of global part assignment updates infeasible
- Parallelization methodology should account for degree skew and imbalance of large irregular graphs

Methodology: Graph and Data Layout

- Assume maximum of $O(\frac{n}{p} + \frac{m}{p})$ storage per task/node
- We use the HPCGRAPH Framework as baseline ([Slota et al., 2016], IPDPS16)
  - Supplies low overhead 1D distributed representation
  - Access to graph structure and associated data fast and efficient
- Each task only stores their local partitioning data
  - Might introduce some complexities - to be explained

Methodology: Processing

Again, we use our IPDPS16 framework as baseline

- *PageRank-like* BSP processing pattern
  - Information pulled for per-vertex updates; i.e., vertex v updates its part assignment $P(v)$ to reflect some optimal given known local information combined with assumed global information
- Parallelization: MPI+OpenMP
  - Everything from I/O to pre-processing to XtraPuLP algorithm itself is fully task and thread-parallel
  - (except for MPI calls)

BSP means that global updates are only available at the end of each iteration!

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

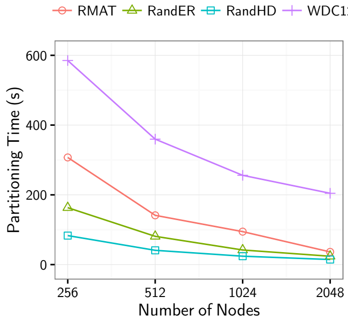
Experimental Results

Test Environment and Graphs

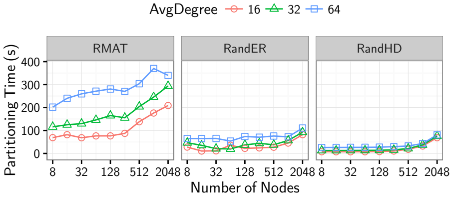
- Test systems:
  - *Blue Waters*: 2x AMD 6276 Interlagos, 16 cores, 64 GB memory, up to 8192 nodes
  - *Compton*: 2x Intel Xeon E5-2670 (Sandy Bridge), 16 cores, 64 GB memory, up to 16 nodes
- Test graphs:
  - UF Sparse Matrix, 10th DIMACS, SNAP, Max Planck Inst., Koblenz, Web Data Commons 2012 (WDC12) - 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)



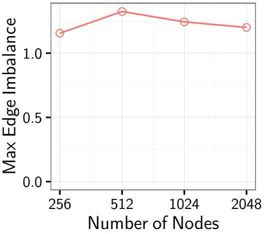
Trillion Edge Tests

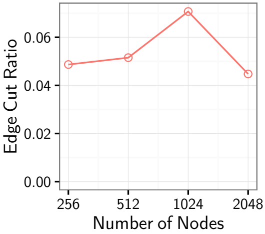
- Also attempted R-MAT, Erdős-Rényi, and high diameter graphs with 2^{34} (17 billion) vertices and 2^{40} (1.1 trillion) edges
- Ran on 8192 nodes of *Blue Waters*
- Erdős-Rényi partitioned in 380 seconds, high diameter in 357 seconds
- R-MAT graph failed; 2^{34} vertex 2^{39} edge R-MAT graph completed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

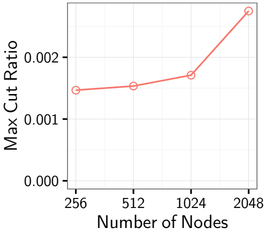
Application Performance

Partitioning WDC12

- At the large scale, how does increasing processor count affect partitioning quality for a fixed number of parts (256)?
- 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



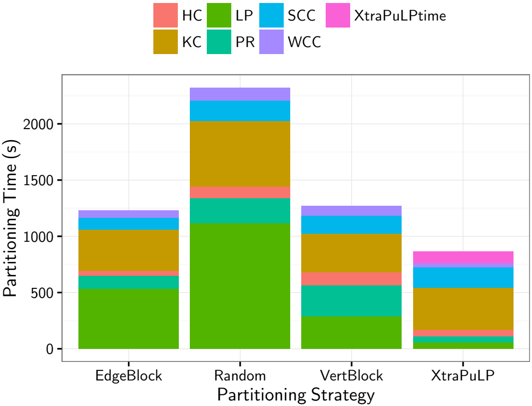




Application Performance

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

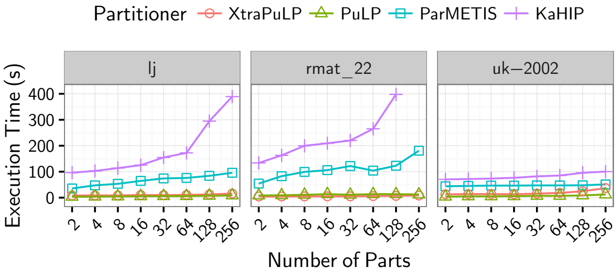
- 6 applications from HPCGraph- HC: harmonic centrality, LP: label propagation, SCC: strong connectivity, WCC: weak connectivity, PR: PageRank, KC: K-core
- 4 partitioning strategies - Edge block, vertex block, random, XtraPuLP



Comparisons to Prior Work

Performance comparisons at smaller scale

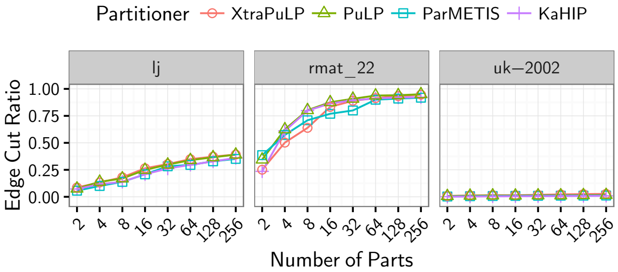
- **Multi-Constraint Scenario**: Computing 16 parts of 26 test graphs on 16 nodes, XtraPuLP is 2.5x faster than PuLP and 4.6x faster than ParMETIS; on a single node, PuLP is 1.5x faster than XtraPuLP
- **Single-Constraint Scenario**: Computing 2-256 parts of test graphs on a single node, XtraPuLP is 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



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/hybrid layouts for further scaling
  - Optimize communication and update methods
  - Explore techniques to improve quality

For papers and presentations: `www.gmslota.com`, `slotag@rpi.edu`

Bibliography I

N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

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.