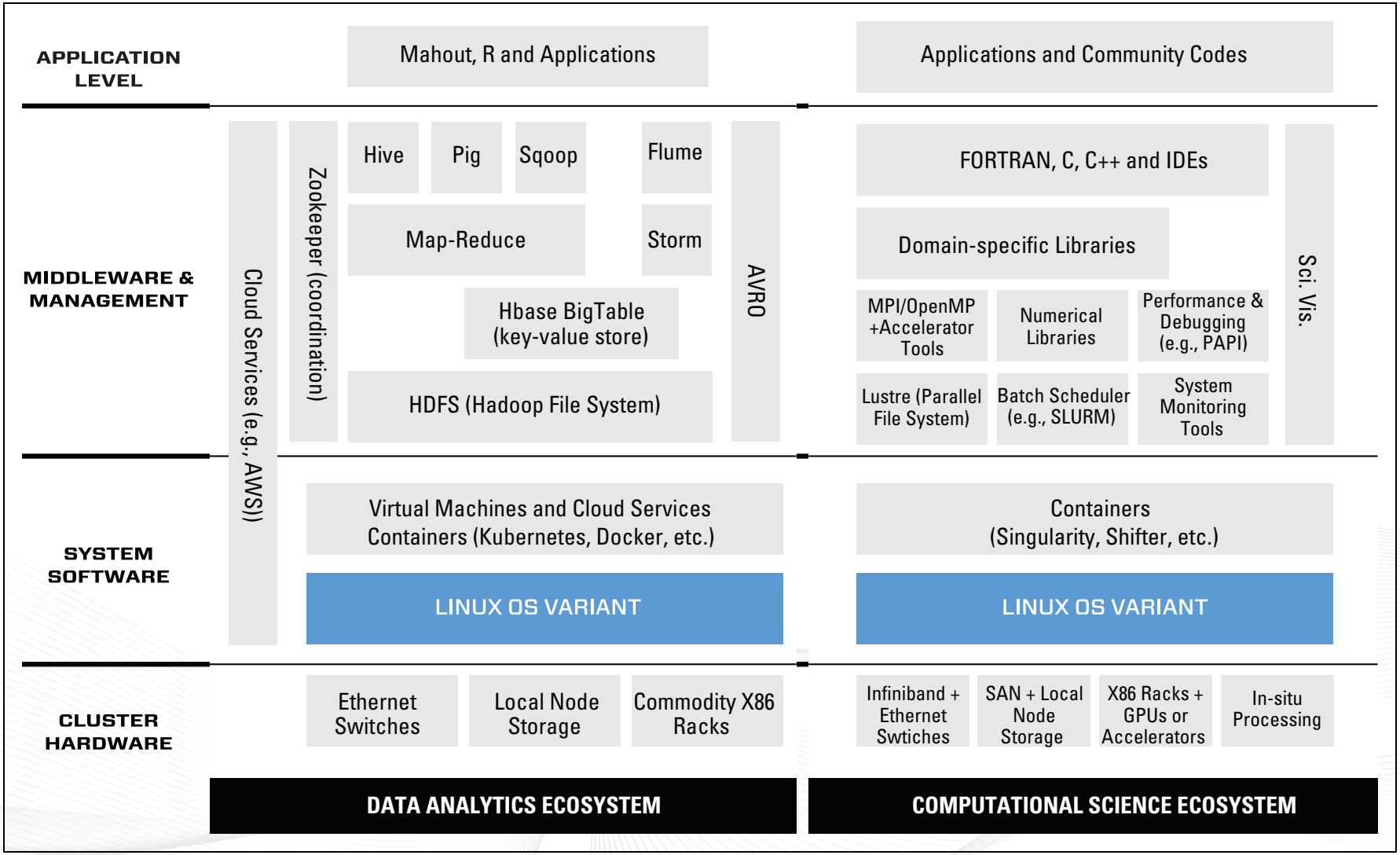
PGAS for graph analytics: can one sided communications break the scalability barrier ?

Johannes Langguth

Simula Research Laboratory, Oslo, Norway

Central Topic Today: Convergence of HPC and Big Data/AI

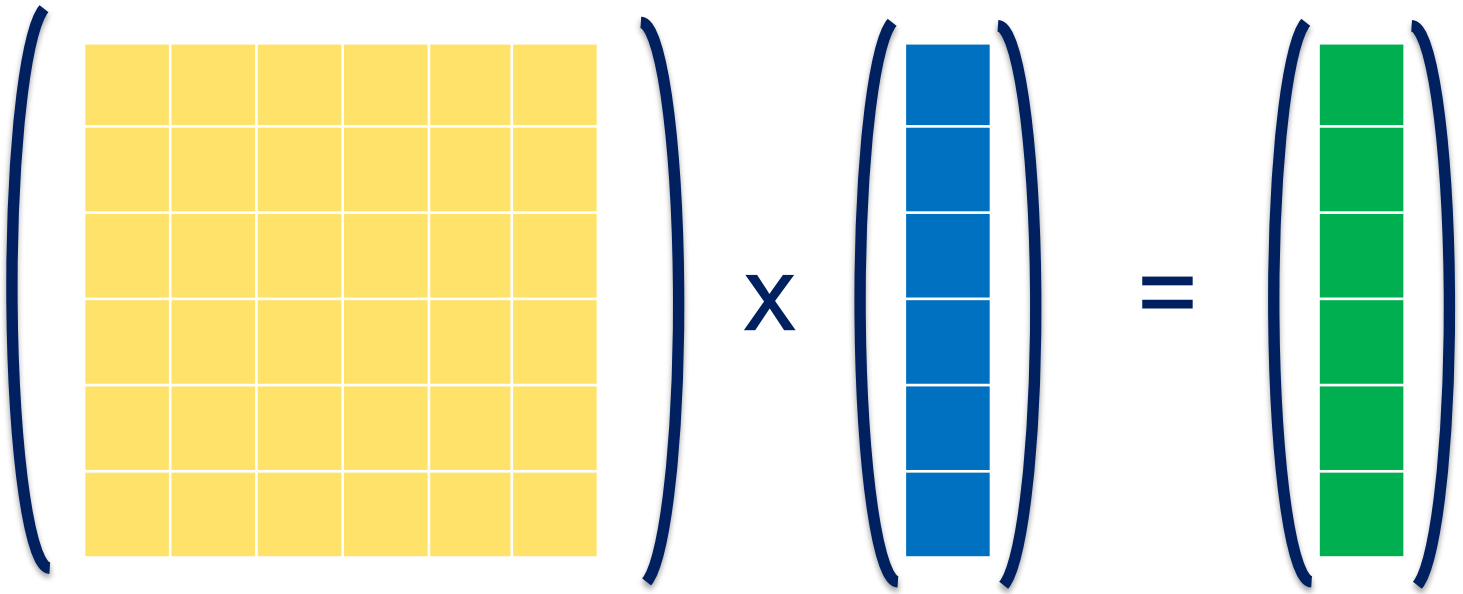


Credit: Reed and Dongarra (2015). HDA:high-end data analysis.

Convergence of HPC and Big Data/AI: Consequences

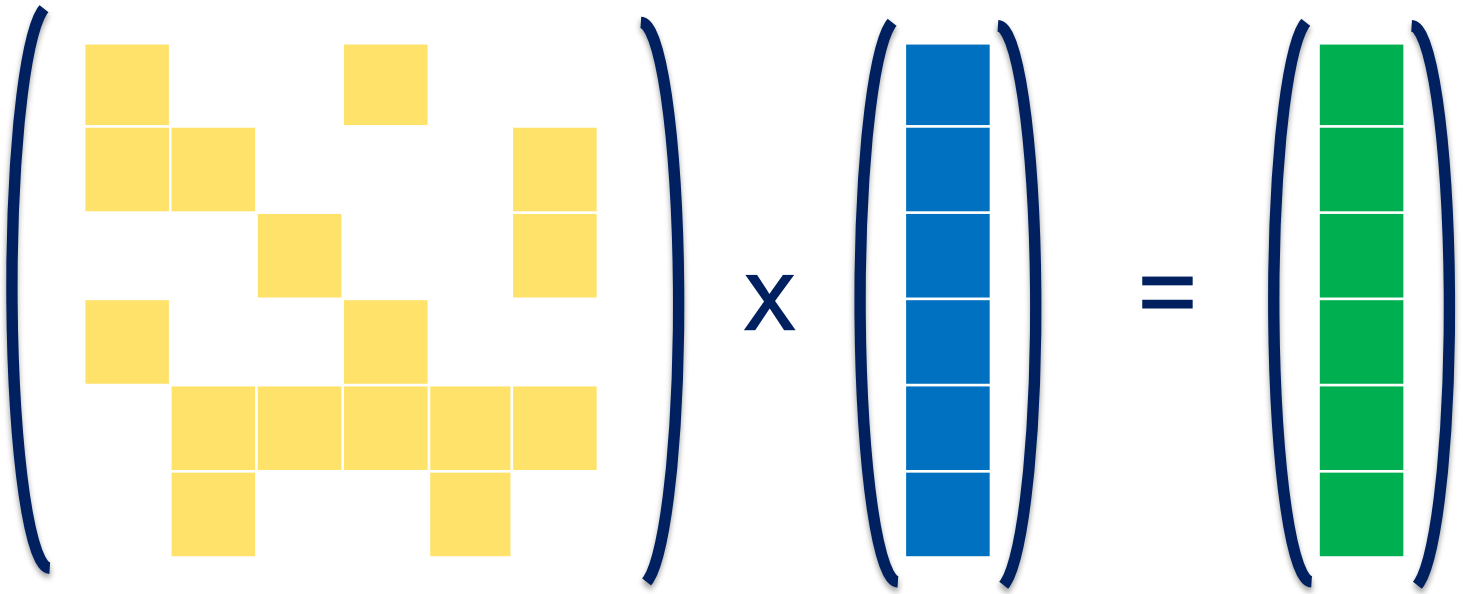
- Computing platforms are are converging
- Adoption of HPC hardware and midlleware in clouds
- Adoption of cloud technologies is HPC
- Relevant question: how do underlying computations differ ?
- Use repeated matrix-vector multiply as a generalizable example

Easy Case: Dense Matrix Dense Vector



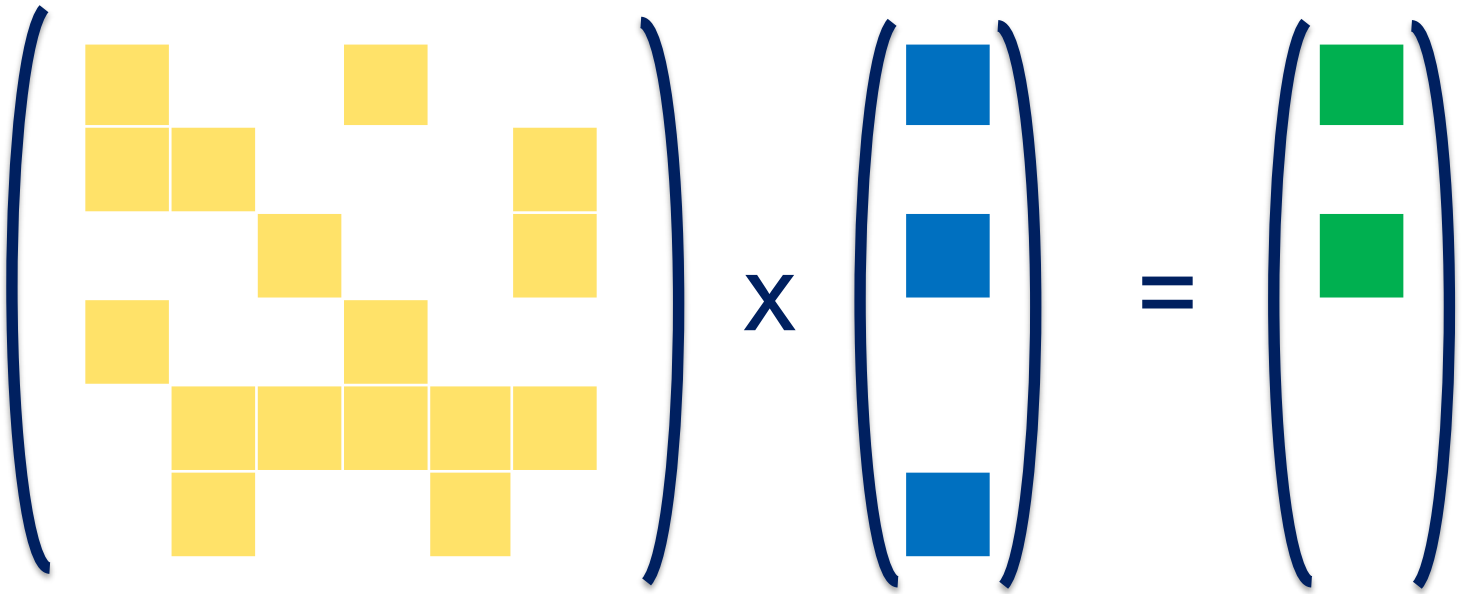
- Common pattern in scientific computing, deep learning, ML
- Data access pattern completely regular
- Batches, tiling, blocking, etc. to run from cache
- Often compute bound
- Balanced communication pattern for distributed memory

Intermediate Case: Sparse Matrix Dense Vector



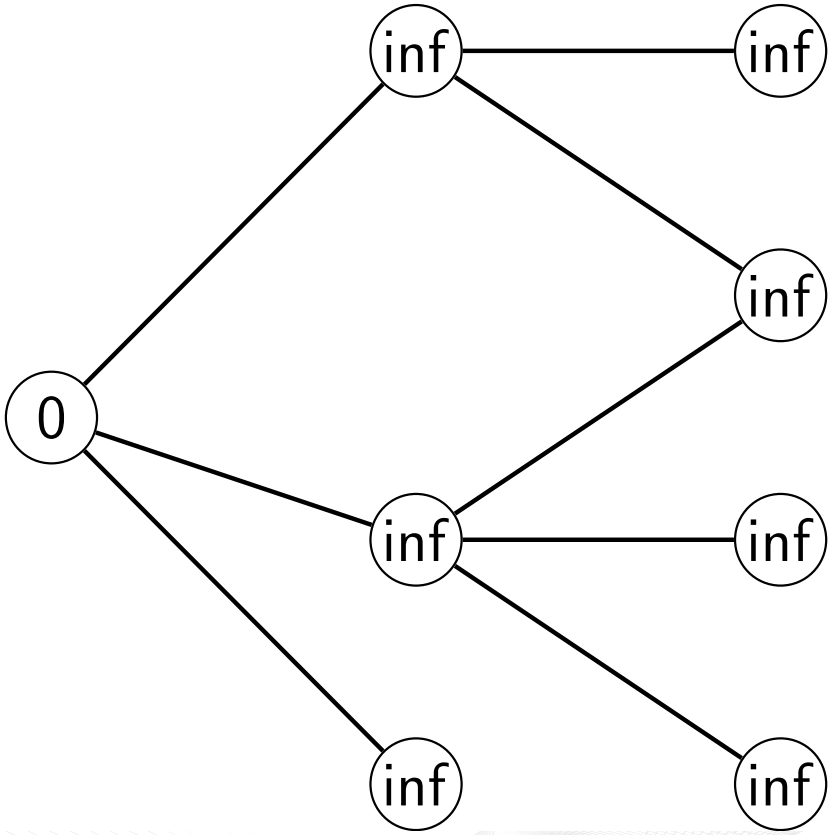
- Unstructured meshes in scientific computing, PageRank
- Data access pattern irregular but static
- Reordering techniques improve cache usage
- Typically memory bandwidth bound
- Unbalanced communication pattern in distributed memory

Challenging Case: Sparse Matrix Sparse Vector



- Graph algorithms, GNNs, data dependent computation paths
- Data access pattern irregular and dynamic
- Possibility of cache reuse is questionable
- Often latency bound
- Unbalanced communication pattern in distributed memory

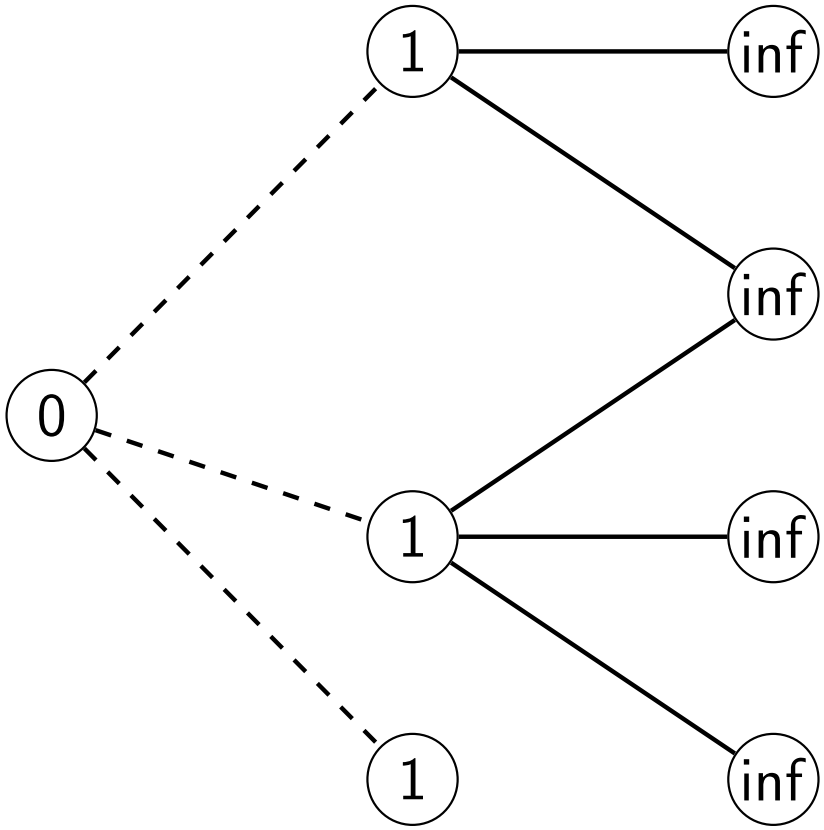
The Most Basic Graph Algorithm: BFS



- Basic kernel of the Graph500

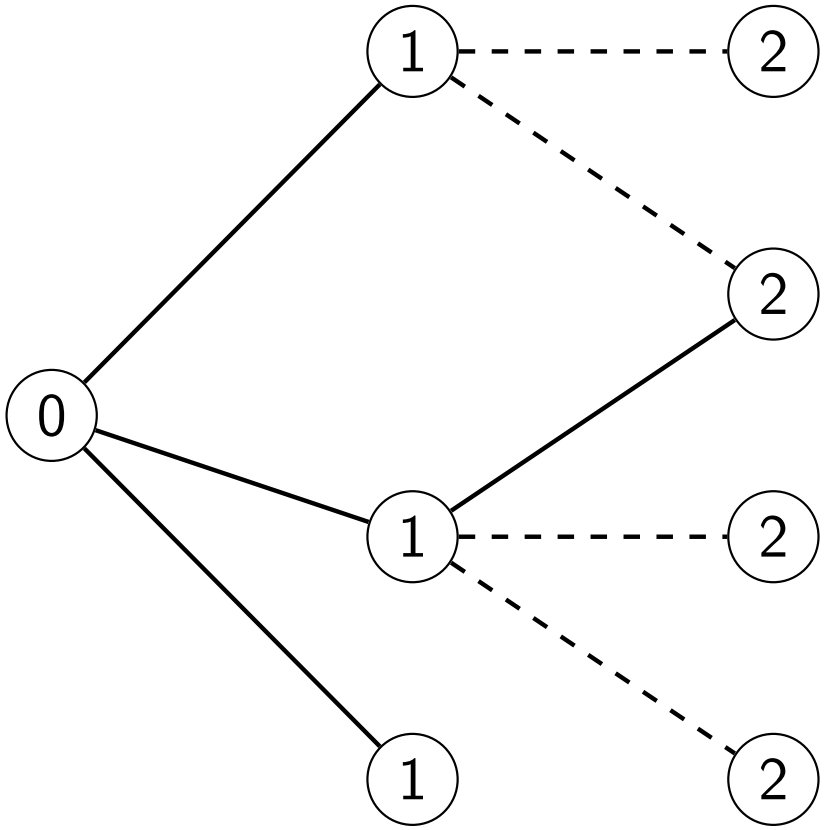
- Sequential algorithm trivial
- Efficient parallel implementation difficult

Challenges of Parallel BFS



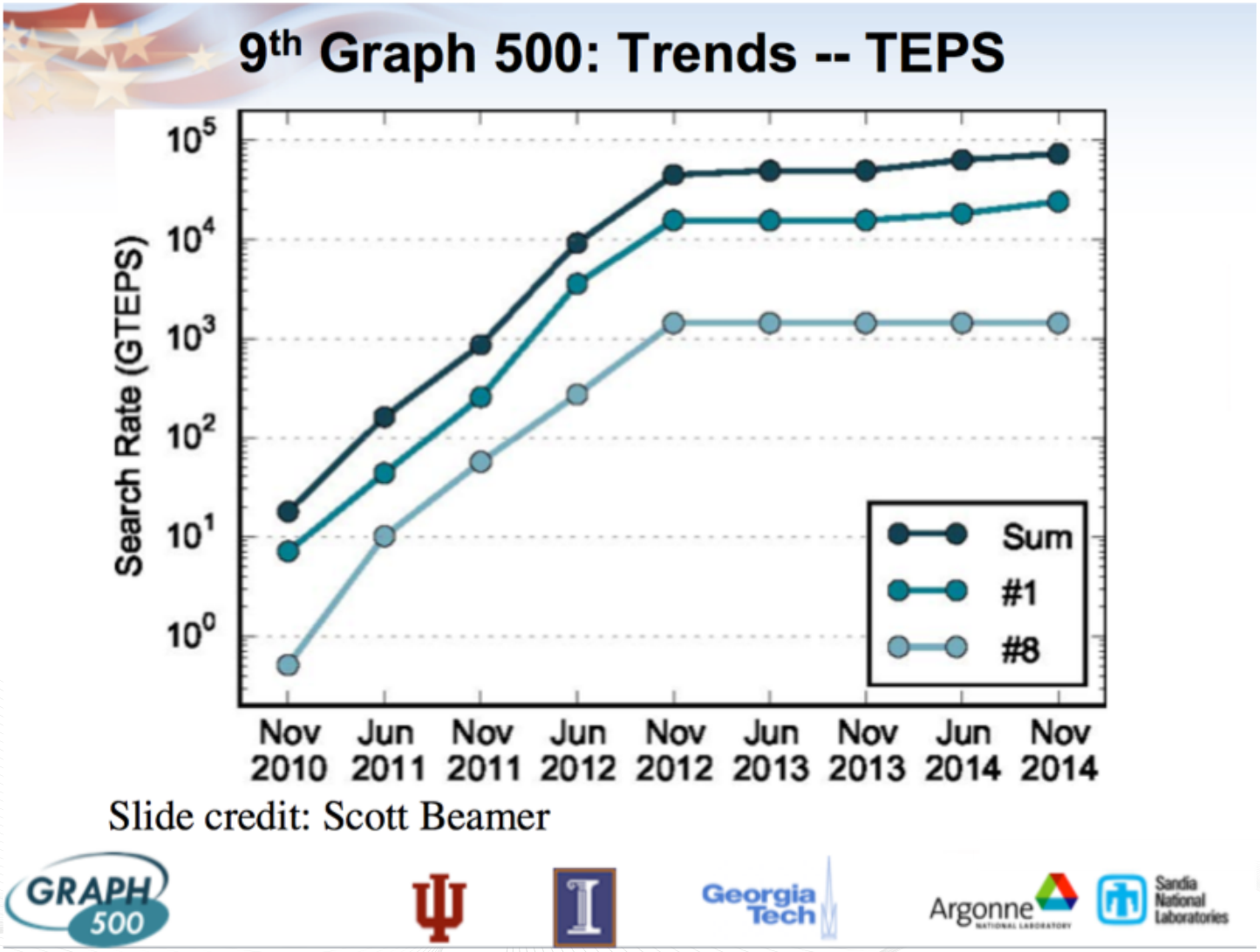
- Extremely communication heavy
- Load balance not predictable
- Communication pattern changes every round
- Communication volume changes every round

Challenges of Parallel BFS can be Overcome



- Rounds impose clear structure on algorithm
- Number of rounds bounded by graph
- Large-message all-to-all for small world graphs
- Ultimately, BFS is a simple graph problem

BFS: Successful Parallelization



Why are Parallel Graph Algorithms still Difficult ?

Scalability! But at what COST?

Frank McSherry
Unaffiliated

Michael Isard
Unaffiliated*

Derek G. Murray
Unaffiliated†

Abstract

We offer a new metric for big data platforms, COST, or the Configuration that Outperforms a Single Thread. The COST of a given platform for a given problem is the hardware configuration required before the platform outperforms a competent single-threaded implementation. COST weighs a system's scalability against the overheads introduced by the system, and indicates the actual performance gains of the system, without rewarding systems that bring substantial but parallelizable overheads.

We survey measurements of data-parallel systems recently reported in SOSP and OSDI, and find that many systems have either a surprisingly large COST, often hundreds of cores, or simply underperform one thread for all of their reported configurations.

1 Introduction

> "You can have a second computer once you've shown you know how to use the first one."
>
> –Paul Barham

The published work on big data systems has fetishized scalability as the most important feature of a distributed data processing platform. While nearly all such publications detail their system's impressive scalability, few directly evaluate their absolute performance against reasonable benchmarks. To what degree are these systems truly improving performance, as opposed to parallelizing overheads that they themselves introduce?

Contrary to the common wisdom that effective scaling is evidence of solid systems building, any system can scale arbitrarily well with a sufficient *lack* of care in its implementation. The two scaling curves in Figure 1

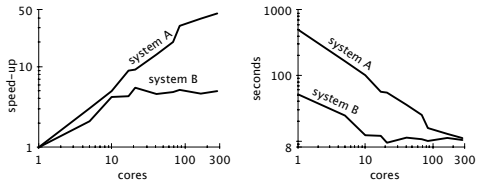


Figure 1: Scaling and performance measurements for a data-parallel algorithm, before (system A) and after (system B) a simple performance optimization. The unoptimized implementation "scales" far better, despite (or rather, because of) its poor performance.

While this may appear to be a contrived example, we will argue that many published big data systems more closely resemble system A than they resemble system B.

1.1 Methodology

In this paper we take several recent graph processing papers from the systems literature and compare their reported performance against simple, single-threaded implementations on the same datasets using a high-end 2014 laptop. Perhaps surprisingly, many published systems have *unbounded* COST—i.e., no configuration outperforms the best single-threaded implementation—for all of the problems to which they have been applied.

The comparisons are neither perfect nor always fair, but the conclusions are sufficiently dramatic that some concern must be raised. In some cases the single-threaded implementations are more than an order of magnitude faster than published results for systems using hundreds of cores. We identify reasons for these gaps: some are intrinsic to the domain, some are entirely avoid-

Scalability at what COST ?

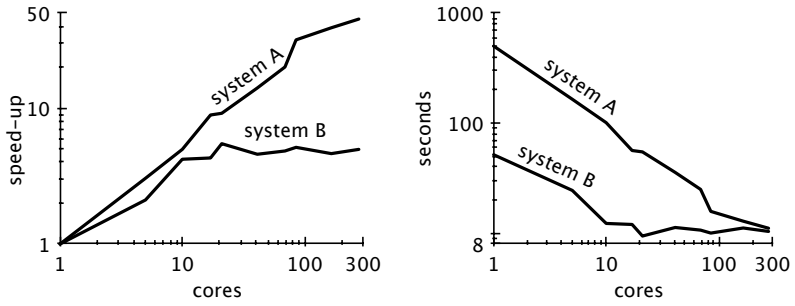
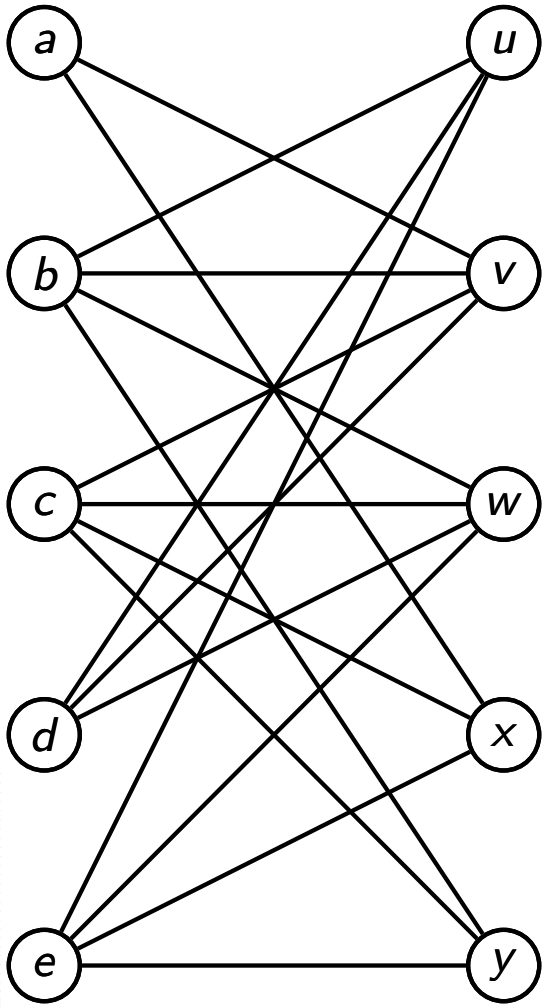


Figure 1: Scaling and performance measurements for a data-parallel algorithm, before (system A) and after (system B) a simple performance optimization. The unoptimized implementation "scales" far better, despite (or rather, because of) its poor performance.

- Most graph algorithms are more difficult than BFS
- Programming models are not designed for graph applications
- Latency cannot be overcome inside the application

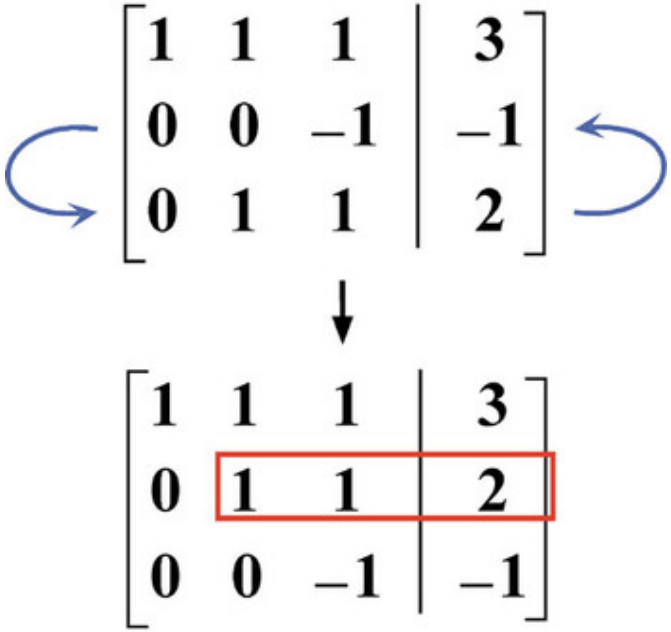
Contrary to the common wisdom that effective scaling is evidence of solid systems building, any system can scale arbitrarily well with a sufficient *lack* of care in its implementation.

Much more challenging: The Bipartite Matching Problem



- Given a bipartite graph
- Find a maximum set of pairweise non-incident edges
- Useful for many interesting applications (e.g. Shapley & Roth, 2012)
- Main motivation: pivoting in sparse direct solvers

Pivoting in Sparse Direct Solvers



Gaussian elimination requires nonzero diagonal

Pivoting

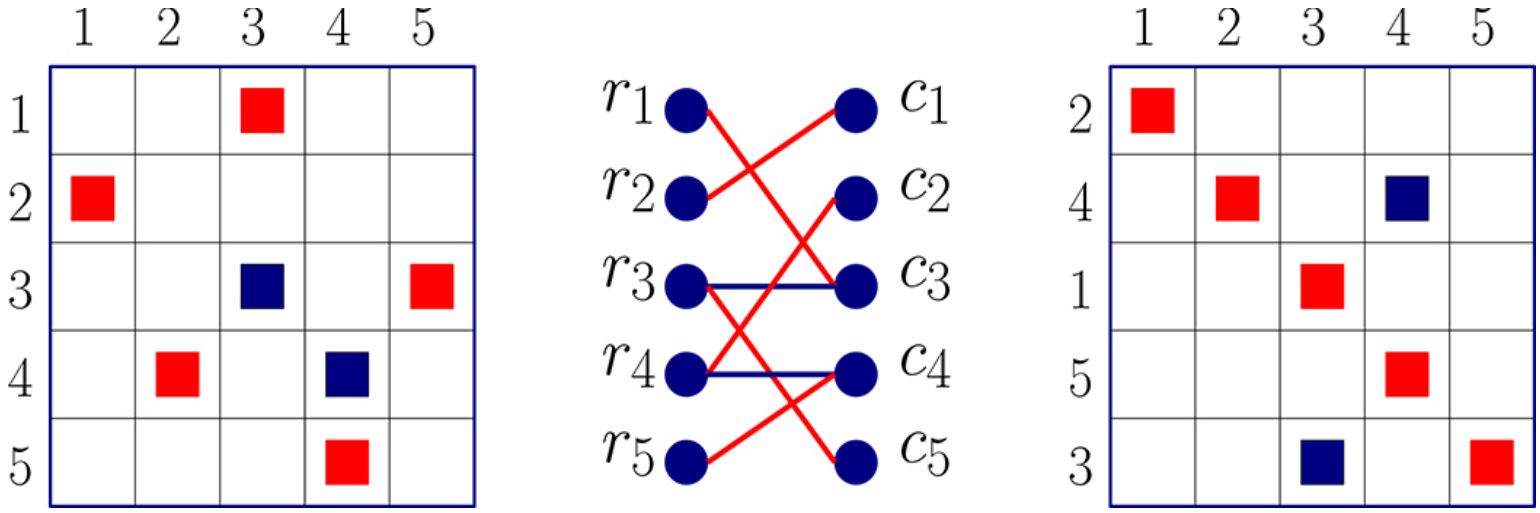
Static Pivoting in Distributed Sparse Direct Solvers

Why do static pivoting ?

- SuperLU_DIST, 2003:
  "the main advantage of static pivoting over classical partial pivoting is that it permits a priori determination of data structures and communication patterns"

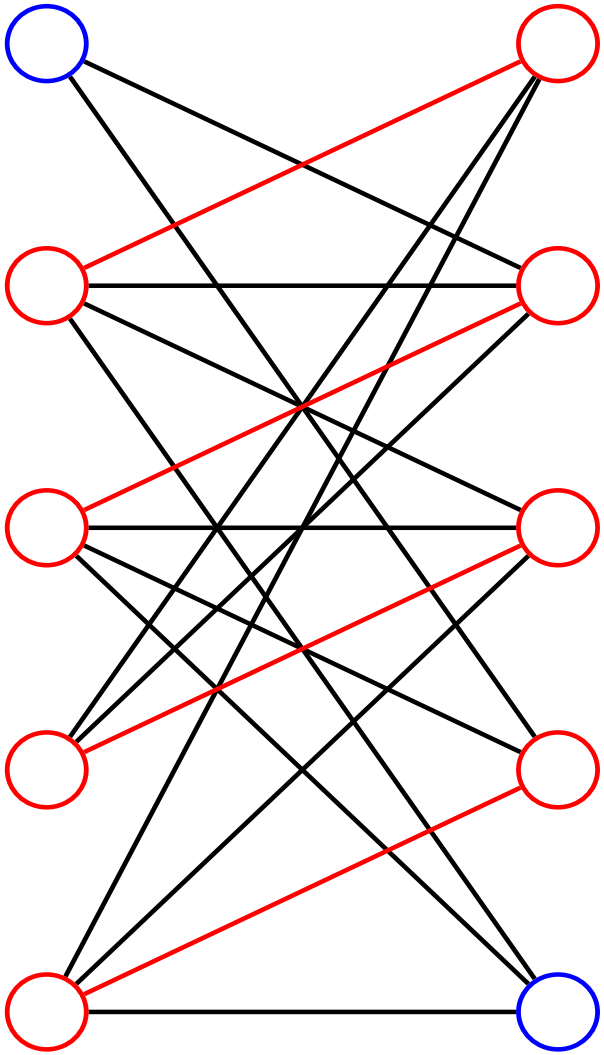
  (SuperLU DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, X. Li, J. W. Demel)

Transversals



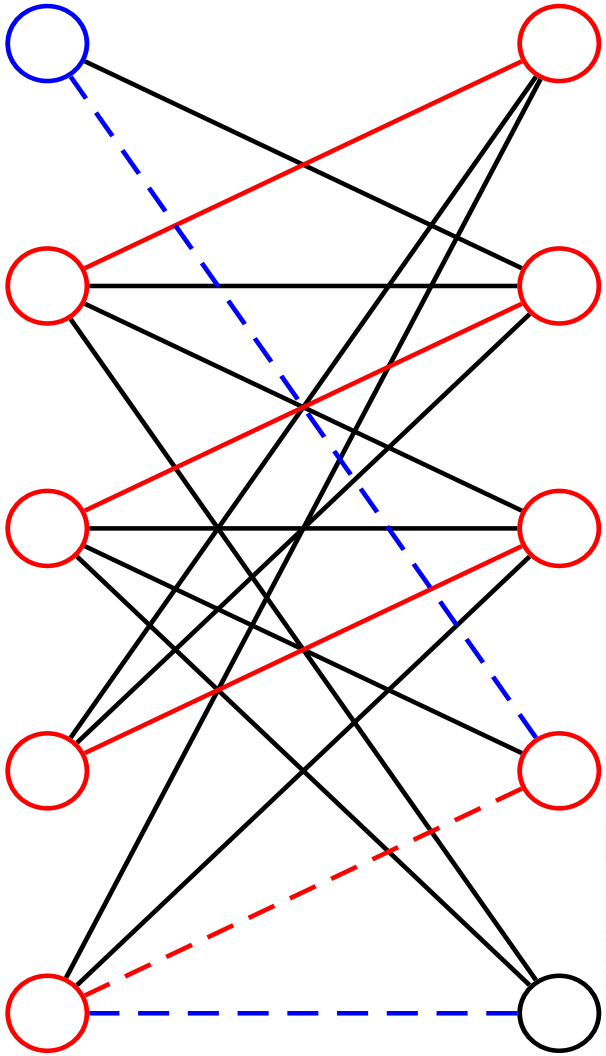
- Every sparse matrix A can be represented as a weighted bipartite graph G_A
- A perfect matching in G_A implies a permutation
- Permuted A has a full transversal (nonzero diagonal)

The Bipartite Matching Problem



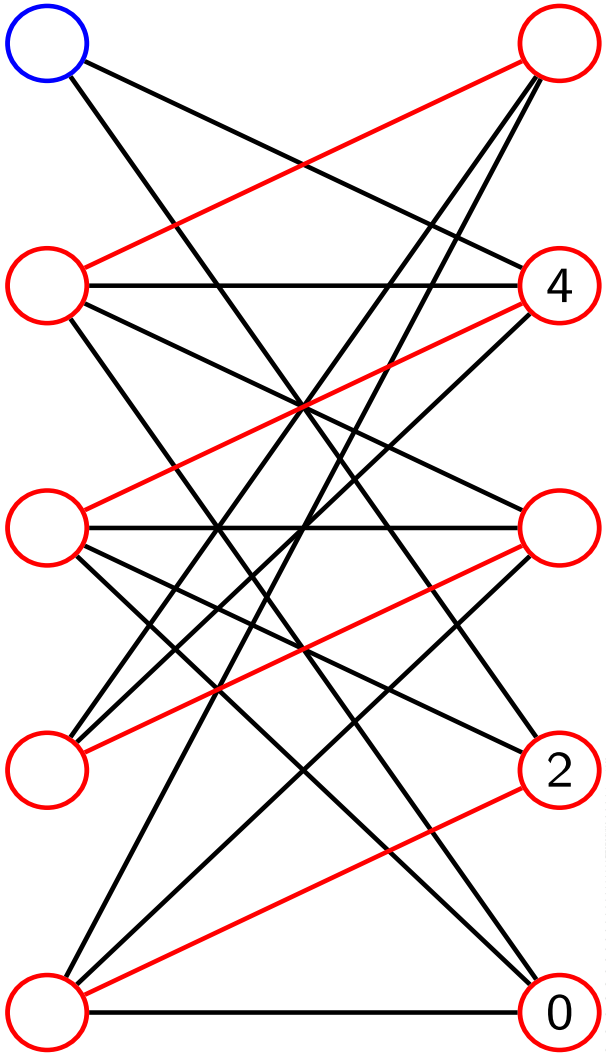
- Greedy solution will match most vertices
- Matching remaining vertices not trivial
- Need to find a sequence of exchanges
- Many sequential algorithms for this problem

The Bipartite Matching Problem



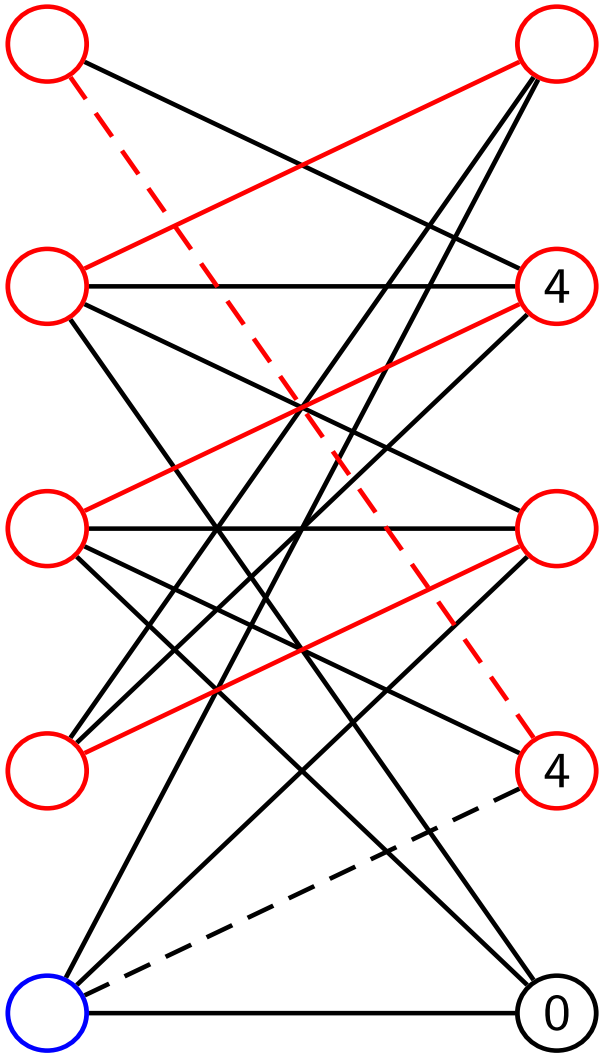
- Alternating path suggests exchanges
- Augmenting path suggest cardinality increase
- Use BFS/DFS to find augmenting paths
- Search can be parallelized

Push-Relabel Algorithms



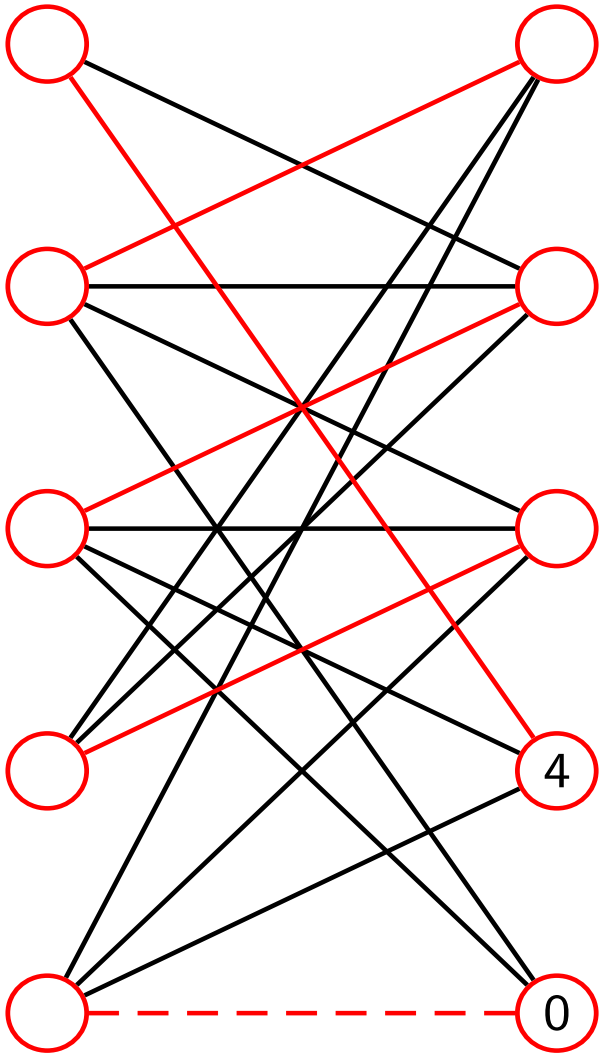
- Break down searches into individual steps
- Use distance labeling to guide searches
- Based on local decisions

Push-Relabel Algorithms



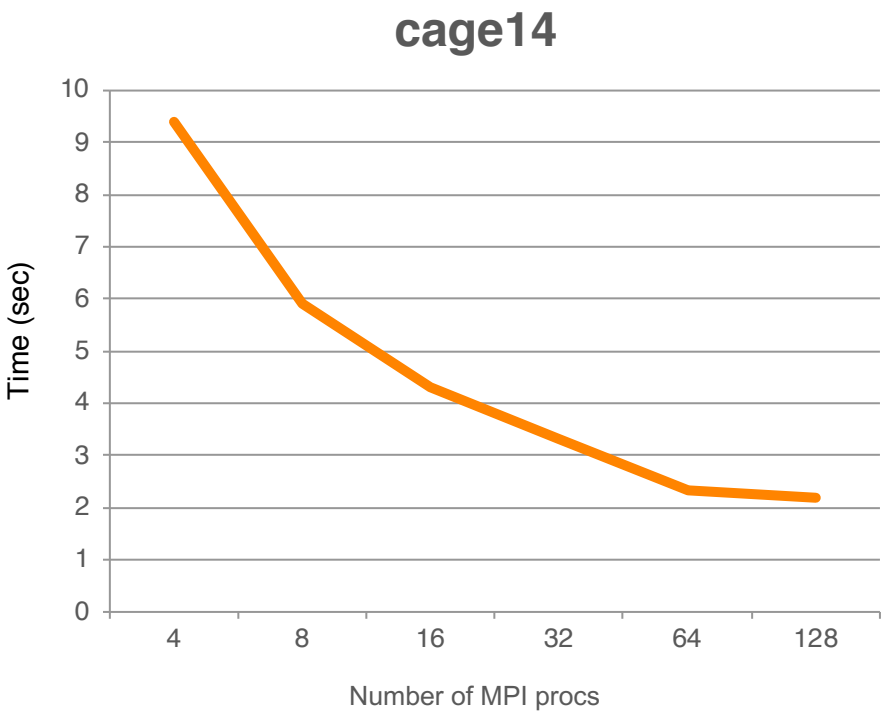
- Operation of swapping edges is called *push*
- Pushes can happen in parallel and asynchronously
- Hard to implement in the message passing model

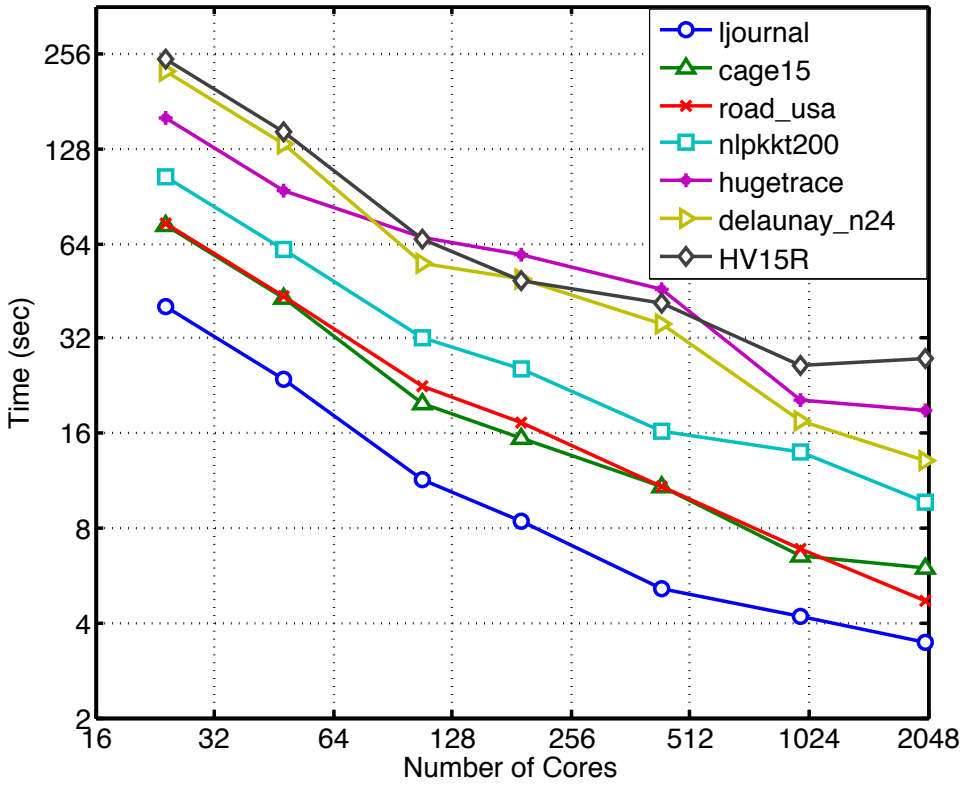
Push-Relabel Algorithms



- Algorithms are latency-bound
- Distributed memory latency is always high
- Synchronous operation slows intra-node pushes to distributed latency
- Asynchronous operation can overcome this

Old vs New, Locality vs no Locality





Hexagon, Cray XT 4, 2008, cage 14, sloppy programming, flat, **locality**

Edison, Cray XC 30, 2014, cage 15 CombBlas, hybrid, **no locality**

How can UPC++ Help ?

1. **One sided messaging**
   - Reduce synchronization overhead
   - Reduce communication latency

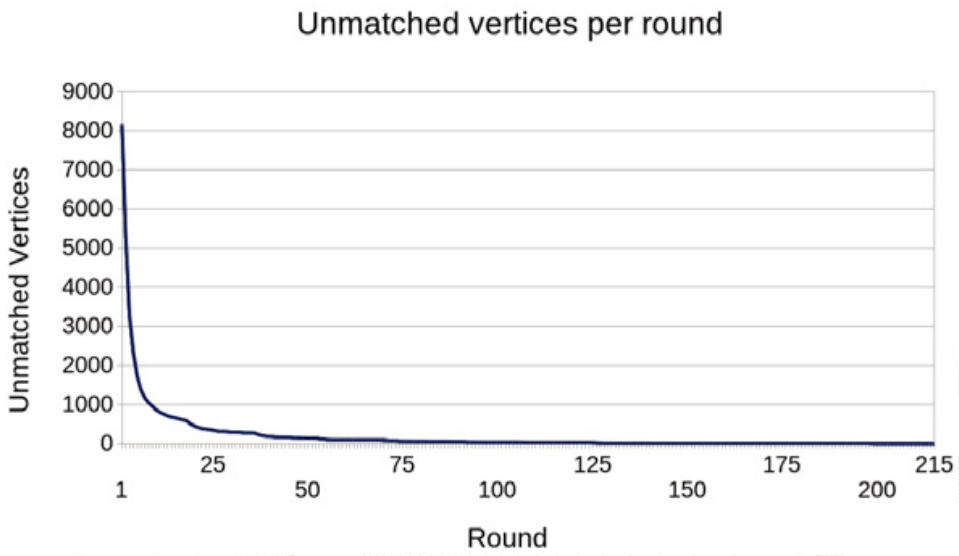
2. **Remote procedure calls**
   - Reduce complexity of 2D graph distribution

One Sided Messaging

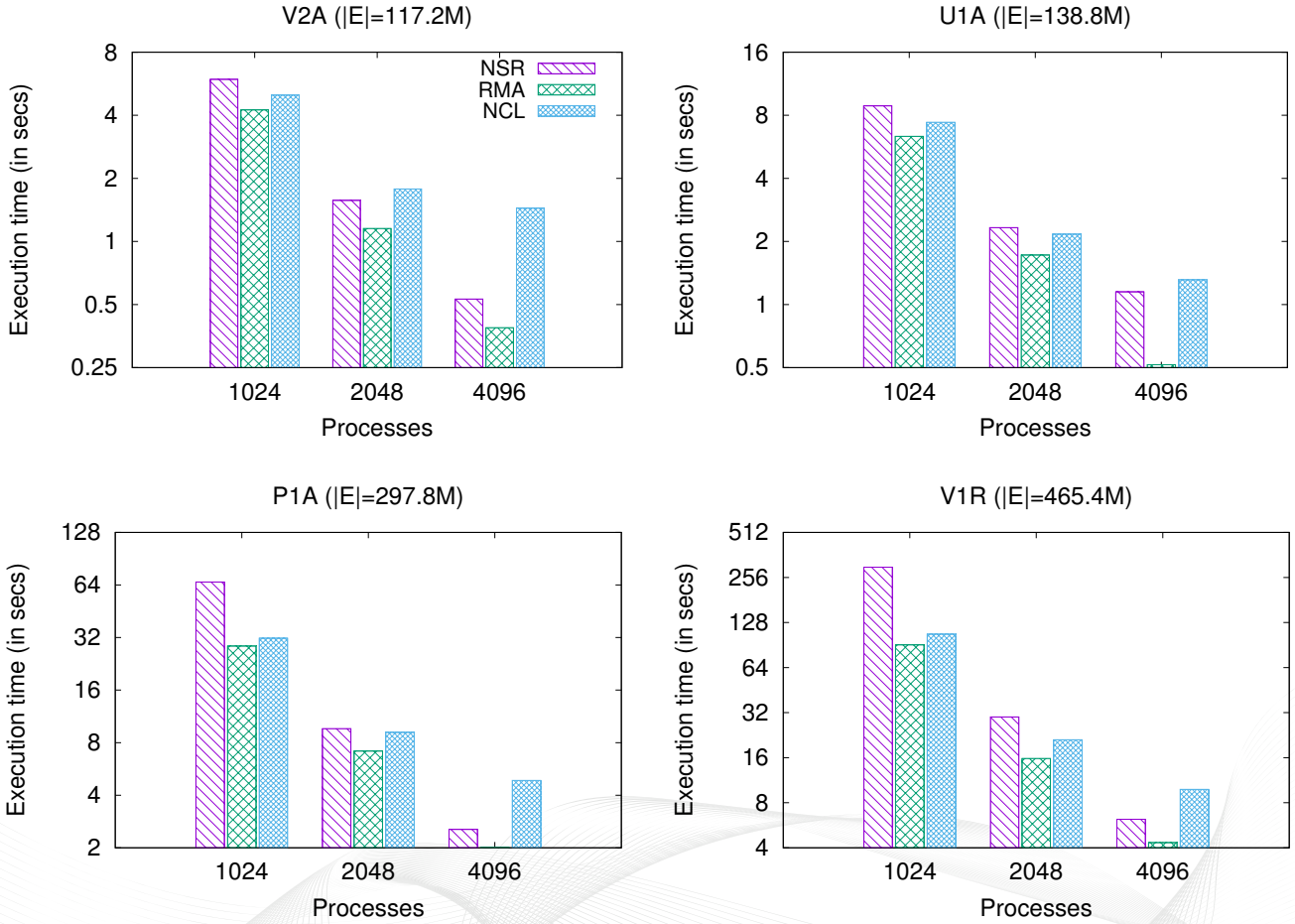
Standard BSP-like message passing operation:

- Send all-to-all to send message sizes
- Use all-to-all to communicate pushes

Inefficient because most messages are empty



Potential Payoff: Some Evidence from MPI-RMA



Message passing vs. One-sided vs. Neighborhood collectives

Exploring MPI Communication Models for GraphApplications Using Graph Matching as a Case Study
Sayan Ghosh∗, Mahantesh Halappanavar†, Ananth Kalyanaraman∗, Arif Khan†, Assefaw H. Gebremedhin

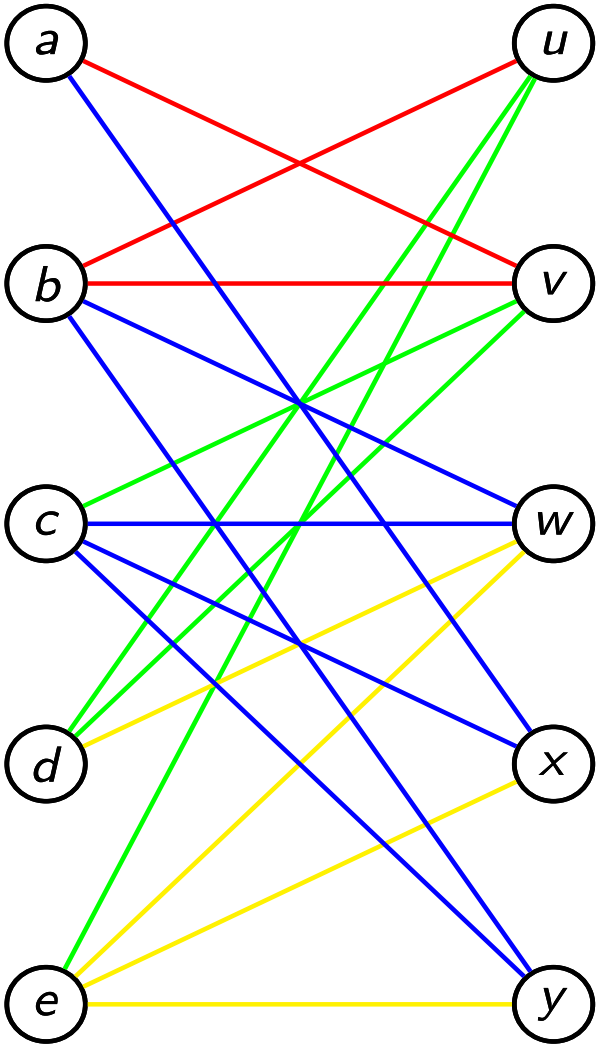
One Sided Messaging

Locality aware algorithm has load imbalance

- Traditional communication rounds force idling
- Asynchronous messages can overcome this

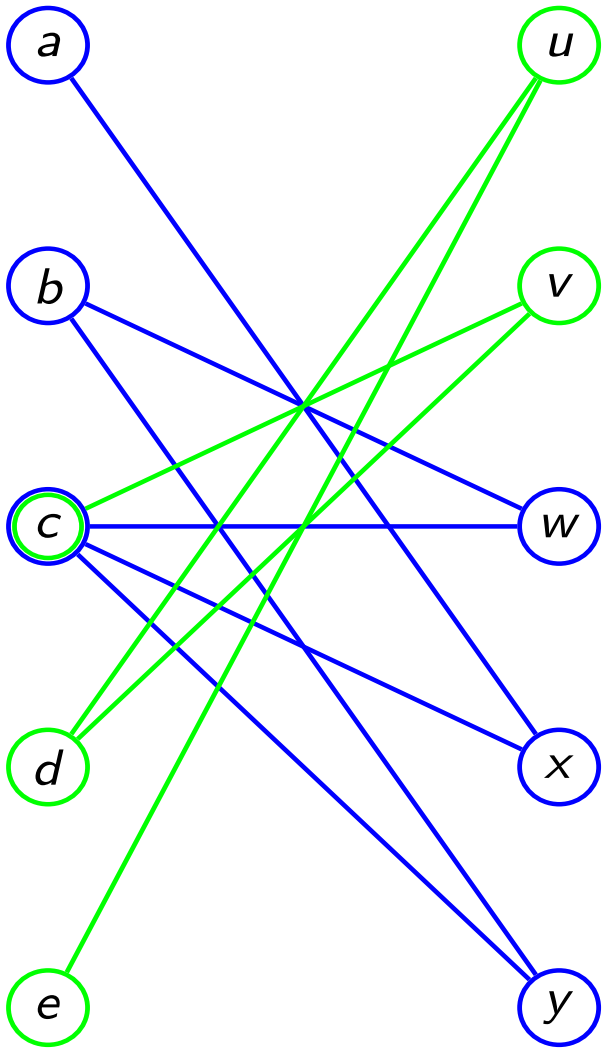
UPC++ enables using the flexibility of the algorithm

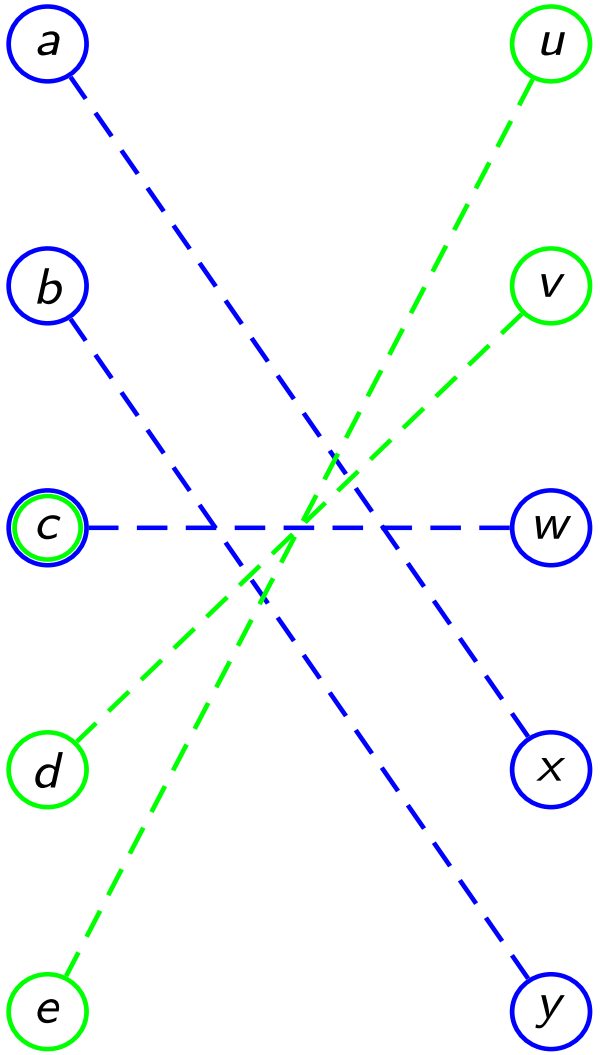
RPCs for 2D Data distribution



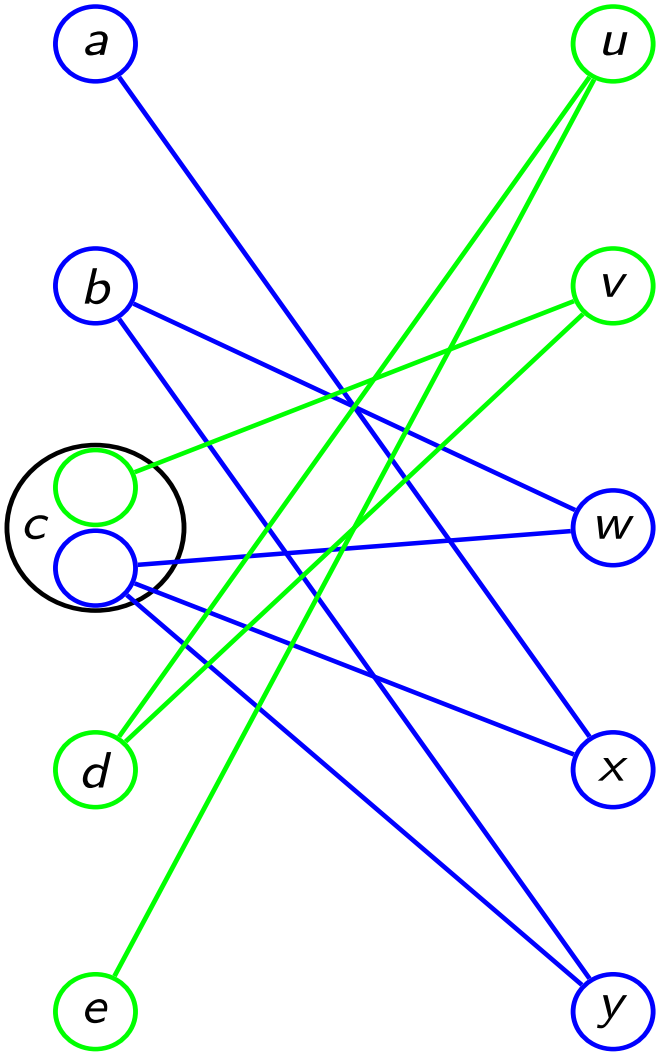
- 2D (edge) distribution
- Vertices are shared among processors
- Requires communication to find push targets
- RPC can simplify this

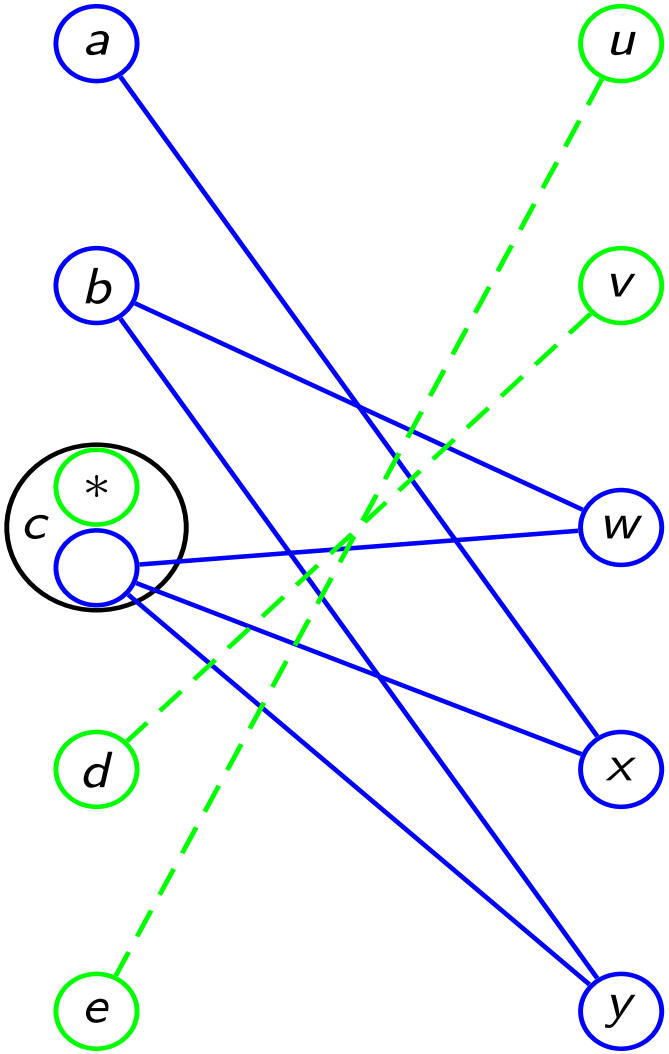
Distributed Perfect Matching



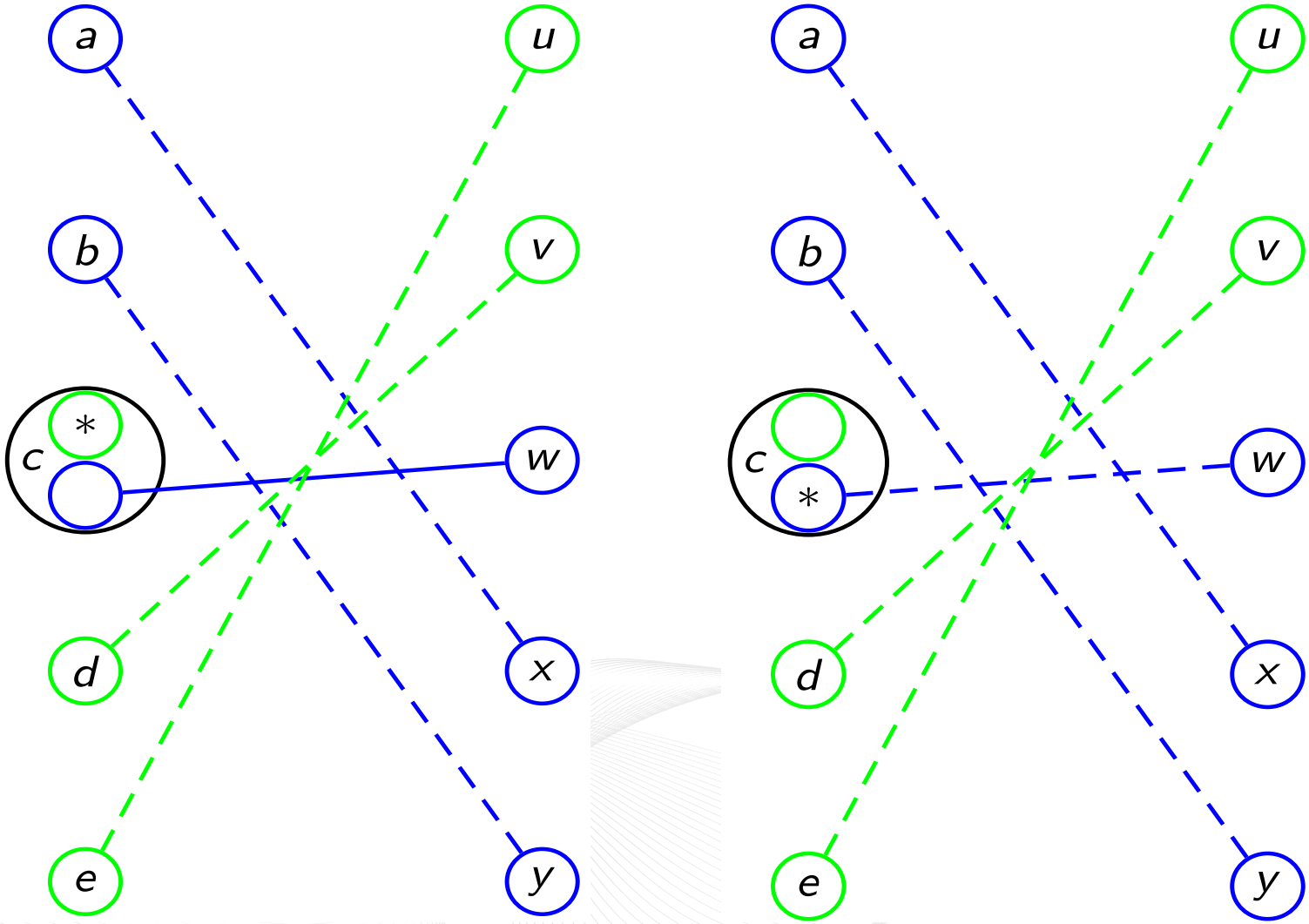


Handling Split Vertices: Connector Tokens

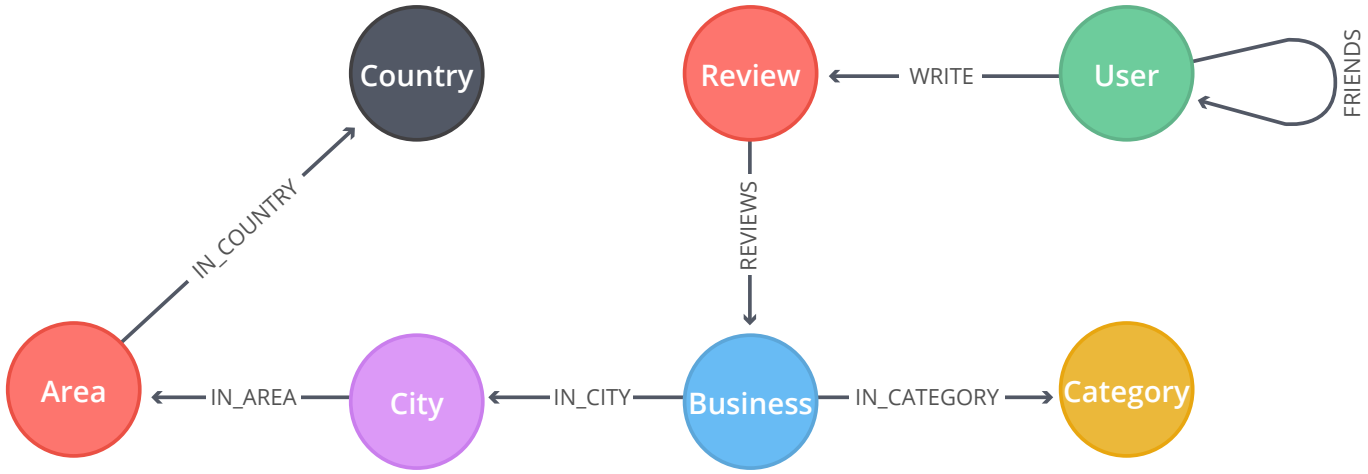




Connector Tokens: *Bids* via RPC



Further Applications: Graph Databases



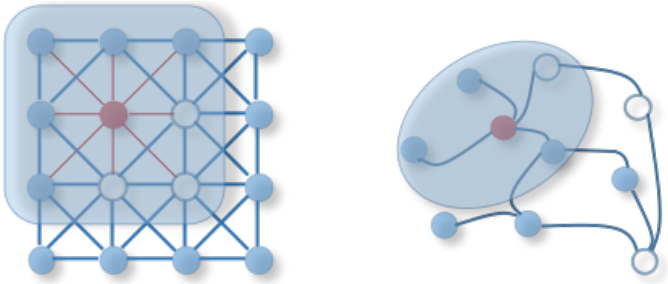
Yelp Graph Model

- Interactions between real-world objects are often unstructured
- Processing naturally imposes graph properties
- Ignoring graph sparsity leads to quadratic runtimes

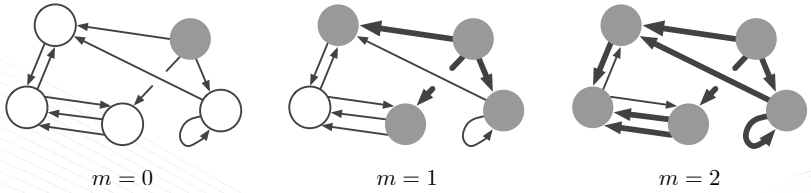
Further Applications

Challenge	Algorithm	Algorithm Type
Figure out traffic load capacity and plan distribution or logistics in an urban area	All Pairs Shortest Path	Pathfinding
Create a low-cost tour of a travel destination	Minimum Weight Spanning Tree	Pathfinding
Identify the most influential machine learning features for extraction and model updates	PageRank	Centrality
Separate the fraudsters from the legitimate users in an online auction	Weighted Degree Centrality	Centrality
Identify the bridge points that connect separate groups	Betweenness Centrality	Centrality
Determine the delivery ETA for a package	Closeness Centrality	Centrality
Find potential duplicate records	Union Find	Community Detection
Figure out dangerous interactions between prescription drugs	Label Propagation	Community Detection
Research structures in the brain	Louvain Modularity	Community Detection

Future Topics: Graph Neural Networks



Standard convolution vs graph convolution



Message-passing GNN

- Standard deep learning uses structured inputs
- Real-world entities do not follow dense structures
- GNNs overcome this limitation
- Considered by some to be the next fundamental step in AI
- Require immense graph processing capabilities

Battaglia et al.: Relational inductive biases, deep learning, and graph networks

Questions ?