Big Graph Processing: Challenges and Opportunities

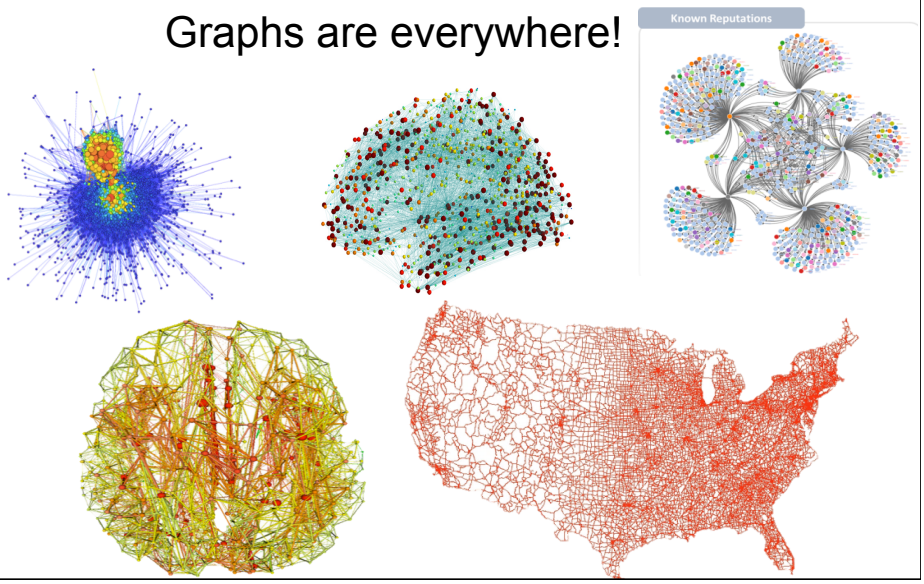
Ling Liu
Distributed Data Intensive Systems Lab (DiSL)
School of Computer Science
Georgia Institute of Technology

Georgia Tech College of Computing

Joint work with Yang Zhou, Kisung Lee, Qi Zhang

Why Graph Processing?

Graphs are everywhere!



Graphs

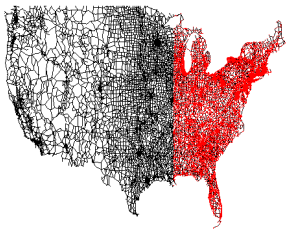
- Real information networks are represented as graphs and analyzed using graph theory
  - Internet
  - Road networks
  - Utility Grids
  - Protein Interactions
- A graph is
  - a collection of binary relationships, or
  - networks of pairwise interactions, including social networks, digital networks.

3

Why Distributed Graph Processing?

They are getting bigger!

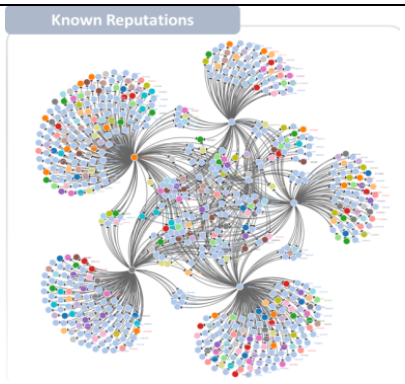
Road Scale



>24 million vertices
>58 million edges
*Route Planning in Road Networks - 2008

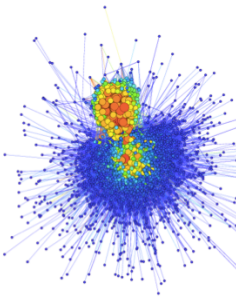
Social Scale

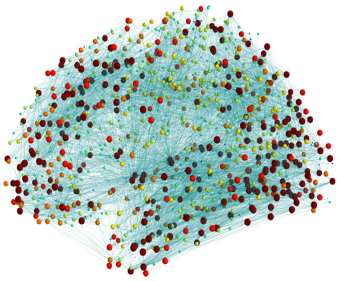
>1 billion vertices
~1 trillion edges
*Facebook Engineering Blog



~41 million vertices
>1.4 billion edges
*Twitter Graph- 2010

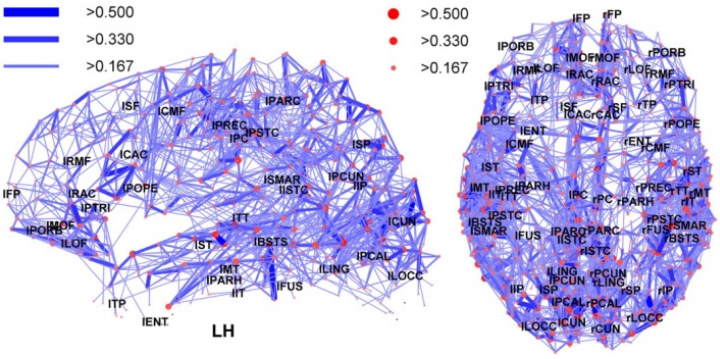
Web Scale





>50 billion vertices
>1 trillion edges
*NSA Big Graph Experiment- 2013

Brain Scale



>100 billion vertices
>100 trillion edges
*NSA Big Graph Experiment- 2013

Classifications of Graphs

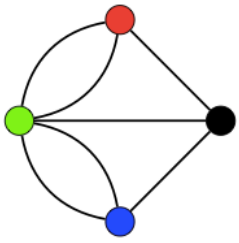
- Simple Graphs v.s. Multigraphs
  - **Simple graph**: allow only one edge per pair of vertices
  - **Multigraph**: allow multiple parallel edges between pairs of vertices
- Small Graphs v.s. Big Graphs
  - Small graph: can fit whole graph and its processing in main memory
  - Big Graph: cannot fit the whole graph and its processing in main memory

9

Scale of the first graph

Nearly 300 years ago the first graph problem consisted of 4 vertices and 7 edges—*Seven Bridges of Königsberg* problem.

[Paul Burkhardt, Chris Waring 2013]



Crossing the River Pregel

Is it possible to cross each of the Seven Bridges of Königsberg exactly once?

Can easily fit in "memory"

10

Scale of Today's Graphs

[Paul Burkhardt, Chris Waring 2013]

- **Scale of Graphs studied in current CS literature**: on the order of billions of edges, tens of GBs

Popular graph datasets in current literature

	n (vertices in millions)	m (edges in millions)	size
AS-Skitter	1.7	11	142 MB
LJ	4.8	69	337.2 MB
USRD	24	58	586.7 MB
BTC	165	773	5.3 GB
WebUK	106	1877	8.6 GB
Twitter	42	1470	24 GB
YahooWeb	1413	6636	120 GB

AS by Skitter (AS-Skitter) — internet topology in 2005 (n = router, m = traceroute)
LiveJournal (LJ) — social network (n = members, m = friendship)
U.S. Road Network (USRD) — road network (n = intersections, m = roads)
Billion Triple Challenge (BTC) — RDF dataset 2009 (n = object, m = relationship)
WWW of UK (WebUK) — Yahoo Web spam dataset (n = pages, m = hyperlinks)
Twitter graph (Twitter) — Twitter network[1] (n = users, m = tweets)
Yahoo! Web Graph (YahooWeb) — WWW pages in 2002 (n = pages, m = hyperlinks)

11

Scale of Big Graphs

[Paul Burkhardt, Chris Waring 2013]

- **Social Scale:** 1 billion vertices, 100 billion edges
  - adjacency matrix: $>10^8$ GB
  - adjacency list: $>10^3$GB
  - edge list: $>10^3$GB
- **Web Scale:** 50 billion vertices, 1 trillion edges
  - adjacency matrix: $>10^{11}$ GB
  - adjacency list: $> 10^4$ GB
  - edge list: $> 10^4$ GB
- **Brain Scale:** 100 billion vertices, 100 trillion edges
  - adjacency matrix: $>10^{20}$ GB
  - adjacency list: $> 10^6$ GB
  - edge list: $> 10^6$ GB

1 terabyte (TB) =1,024GB ~10^3GB
1 petabyte (PB) =1,024TB ~ 10^6GB
1 exabyte (EB) =1,024PB~10^9GB
1 zettabyte (ZB) =1,024EB~10^{12}GB.
1 yottabyte (YB) =1,024ZB~10^{15}GB.

12

How hard it can be?

- **Difficult to parallelize (data/computation)**
  - irregular data access increases latency
  - skewed data distribution creates bottlenecks
    - giant component
    - high degree vertices, highly skewed edge weight distribution
- **Increased size imposes greater challenge . . .**
  - Latency
  - resource contention (i.e. hot-spotting)
- **Algorithm complexity really matters!**
  - Run-time of $O(n^2)$ on a trillion node graph is not practical!

13

How do we store and process graphs?

- Conventional approach is to store and compute in-memory.
  - SHARED-MEMORY
    - Parallel Random Access Machine (PRAM)
    - data in globally-shared memory
    - implicit communication by updating memory
    - fast-random access
  - DISTRIBUTED-MEMORY
    - Bulk Synchronous Parallel (BSP)
    - data distributed to local, private memory
    - explicit communication by sending messages
    - easier to scale by adding more machines

[Paul Burkhardt, Chris Waring 2013]

14

Larger systems, greater latency. . .

- Increasing memory can increase latency.
  - traverse more memory addresses
  - larger system with greater physical distance between machines
- Latency causes significant inefficiency in new CPU architectures.
- Disk capacity not unlimited but higher than memory.
  - Disk is not enough—applications will still require memory for processing

15

Top Super Computer Installations

[Paul Burkhardt, Chris Waring 2013]

Largest supercomputer installations do not have enough memory to process the Brain Graph (3 PB)!

- Titan Cray XK7 at ORNL — #1 Top500 in 2012
  - 0.5 million cores
  - 710 TB memory
  - 8.2 Megawatts[3]
  - 4300 sq.ft. (NBA basketball court is 4700 sq.ft.)
- Sequoia IBM Blue Gene/Q at LLNL — #1 Graph500 in 2012
  - 1.5 million cores
  - 1 PB memory
  - 7.9 Megawatts[3]
  - 3000 sq.ft.

16

Big Data Makes Bigger Challenges to Graph Problems

- Increasing volume, velocity, variety of Big Data are posting significant challenges to scalable graph algorithms.
- Big Graph Challenges:
  - How will graph applications adapt to big data at petabyte scale?
  - Ability to store and process Big Graphs impacts typical data structures.
- Big graphs challenge our conventional thinking on both algorithms and computing architectures

17

Existing Graph Processing Systems

- Single PC Systems
  - **GraphLab** [Low et al., UAI'10]
  - **GraphChi** [Kyrola et al., OSDI'12]
  - **X-Stream** [Roy et al., SOSP'13]
  - **TurboGraph** [Han et al., KDD'13]
  - **GraphLego** [Zhou, Liu et al., ACM HPDC'15]
- Distributed Shared Memory Systems
  - **Pregel** [Malewicz et al., SIGMOD'10]
  - **Giraph/Hama** – Apache Software Foundation
  - **Distributed GraphLab** [Low et al., VLDB'12]
  - **PowerGraph** [Gonzalez et al., OSDI'12]
  - **SPARK-GraphX** [Gonzalez et al., OSDI'14]
  - **PathGraph** [Yuan et.al, IEEE SC 2014]
  - **GraphMap** [LeeLiu et al. IEEE SC 2015]

18

Parallel Graph Processing: Challenges

- Structure driven computation
  - Storage and Data Transfer Issues
- Irregular Graph Structure and Computation Model
  - Storage and Data/Computation Partitioning Issues
  - Partitioning v.s. Load/Resource Balancing

19

Parallel Graph Processing: Opportunities

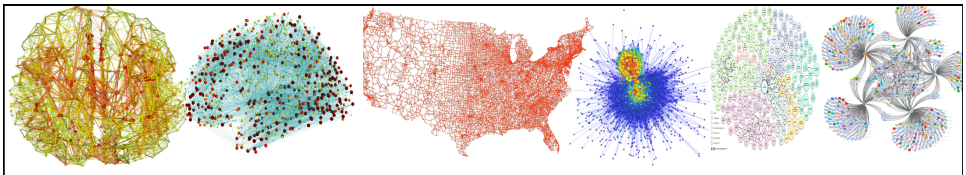
- Extend Existing Paradigms
  - Vertex centric
  - Edge centric
- **BUILD NEW FRAMEWORKS!**
  - Centralized Approaches
    - GraphLego [ACM HPDC 2015] / GraphTwist [VLDB2015]
  - Distributed Approaches
    - GraphMap [IEEE SC 2015], PathGraph [IEEE SC 2014]

20

Parallel Graph Processing

- Part I: Single Machine Approaches
  - Can big graphs be processed on a single machine?
  - How to effectively maximize sequential access and minimize random access?
- Part II: Distributed Approaches
  - How to effectively partition a big graph for graph computation?
  - Are large clusters always better for big graphs?
- Part III Graph Queries (Subgraph Marching)

21

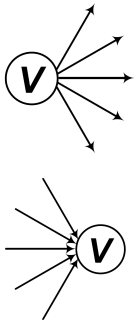


Part I:
Parallel Graph Processing
(single machine)

Georgia Tech College of Computing

Vertex-centric Computation Model

- Think like a vertex
- vertex_scatter(vertex *v*)
  - send updates over outgoing edges of *v*
- vertex_gather(vertex *v*)
  - apply updates from inbound edges of *v*



- repeat the computation iterations
  - for all vertices *v*
    - vertex_scatter(*v*)
  - for all vertices *v*
    - vertex_gather(*v*)

Edge-centric Computation Model (X-Stream)

- Think like an edge (source vertex and destination vertex)
- edge_scatter(edge *e*)
  - send update over *e* (from source vertex to destination vertex)
- update_gather(update *u*)
  - apply update *u* to *u.destination*

- repeat the computation iterations
  - for all edges *e*
    - edge_scatter(*e*)
  - for all updates *u*
    - update_gather(*u*)

Challenges of Big Graphs

- **Graph size v.s. limited resource**
  - Handling big graphs with billions of vertices and edges in memory may require hundreds of gigabytes of DRAM
- **High-degree vertices**
  - In uk-union with 133.6M vertices: the maximum indegree is 6,366,525 and the maximum outdegree is 22,429
- **Skewed vertex degree distribution**
  - In Yahoo web with 1.4B vertices: the average vertex degree is 4.7, 49% of the vertices have degree zero and the maximum indegree is 7,637,656
- **Skewed edge weight distribution**
  - In DBLP with 0.96M vertices: among 389 coauthors of Elisa Bertino, she has only one coauthored paper with 198 coauthors, two coauthored papers with 74 coauthors, three coauthored papers with 30 coauthors, and coauthored paper larger than 4 with 87 coauthors

Real-world Big Graphs

Graph	Type	#Vertices	#Edges	AvgDeg	MaxIn	MaxOut
Yahoo	directed	1.4B	6.6B	4.7	7.6M	2.5K
uk-union	directed	133.6M	5.5B	41.22	6.4M	22.4K
uk-2007-05	directed	105.9M	3.7B	35.31	975.4K	15.4K
Twitter	directed	41.7M	1.5B	35.25	770.1K	3.0M
Facebook	undirected	5.2M	47.2M	18.04	1.1K	1.1K
DBLPS	undirected	1.3M	32.0M	40.67	1.7K	1.7K
DBLPM	undirected	0.96M	10.1M	21.12	1.0K	1.0K
Last.fm	undirected	2.5M	42.8M	34.23	33.2K	33.2K

Graph Processing Systems: Challenges

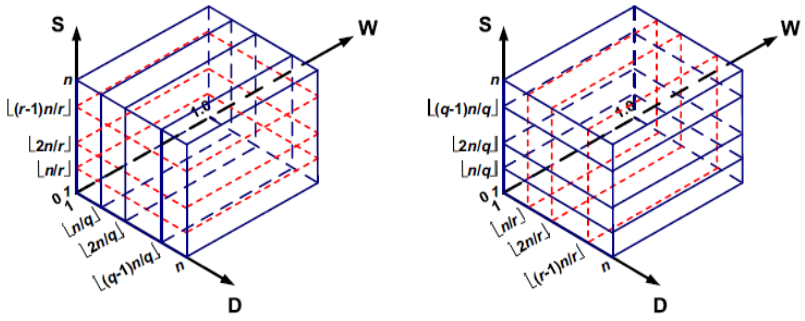
- **Diverse types of processed graphs**
  - Simple graph: not allow for parallel edges (multiple edges) between a pair of vertices
  - Multigraph: allow for parallel edges between a pair of vertices
- **Different kinds of graph applications**
  - Matrix-vector multiplication and graph traversal with the cost of $O(n^2)$
  - Matrix-matrix multiplication with the cost of $O(n^3)$
- **Random access**
  - It is inefficient for both access and storage. A bunch of random accesses are necessary but would hurt the performance of graph processing systems
- **Workload imbalance**
  - The time of computing on a vertex and its edges is much faster than the time to access to the vertex state and its edge data in memory or on disk
  - The computation workloads on different vertices are significantly imbalanced due to the highly skewed vertex degree distribution.

Solution Approach GraphLego@Georgia Tech

- **Flexible multi-level hierarchical graph parallel abstractions**
  - Model a large graph as a 3D cube with source vertex, destination vertex and edge weight as the dimensions
  - Partitioning a big graph by: `slice, strip, dice` based graph partitioning
- **Access Locality Optimization**
  - Dice-based data placement: store a large graph on disk by minimizing non-sequential disk access and enabling more structured in-memory access
  - Construct partition-to-chunk index and vertex-to-partition index to facilitate fast access to slices, strips and dices
  - implement partition-level in-memory gzip compression to optimize disk I/Os
- **Optimization for Partitioning Parameters**
  - Build a regression-based learning model to discover the latent relationship between the number of partitions and the runtime

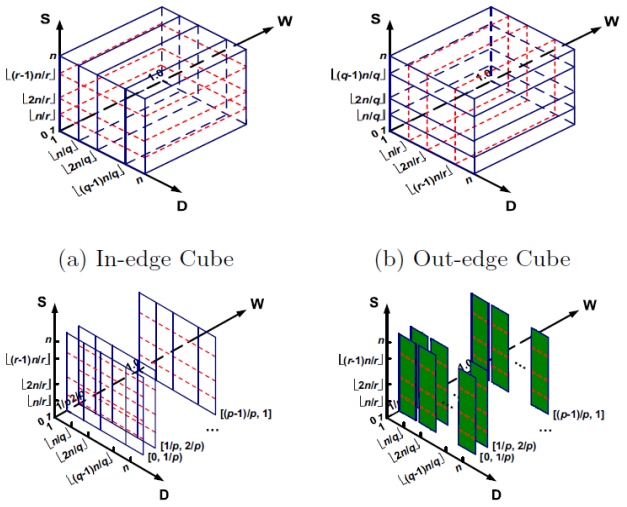
Modeling a Graph as a 3D Cube

- Model a directed graph $G=(V,E,W)$ as a 3D cube $I=(S,D,E,W)$ with source vertices ($S=V$), destination vertices ($D=V$) and edge weights (W) as the three dimensions



(a) In-edge Cube (b) Out-edge Cube

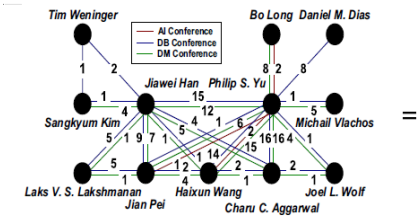
Multi-level Hierarchical Graph Parallel Abstractions



(a) In-edge Cube (b) Out-edge Cube

(c) In-edge Slice (d) In-edge Strip

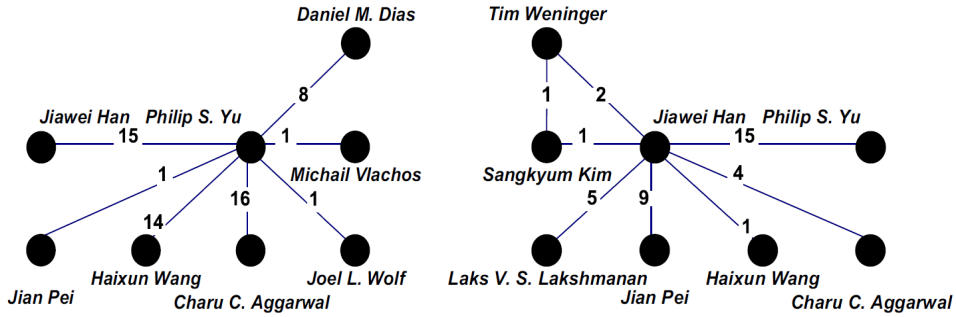
Slice Partitioning: DBLP Example



=

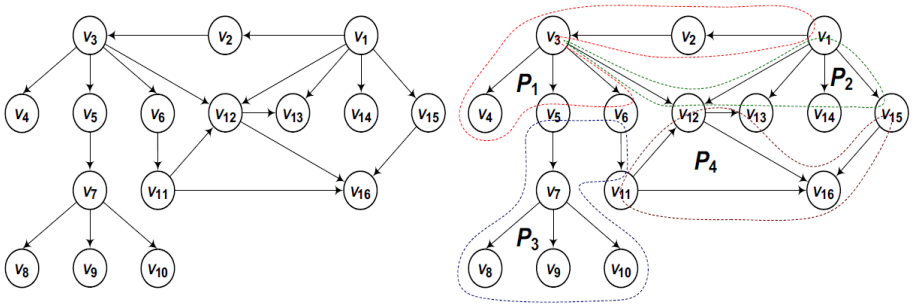
(a) Coauthor Multigraph

Strip Partitioning of DB Slice



(a) Strip 1 (b) Strip 2

Dice Partitioning: An Example



(a) Original Graph (b) Dice Partitioning

SVP: v1,v2,v3, v5,v6,v7, v11, v12, v15

DVP: v2,v3,v4,v5,v6, v7,v8,v9,v10,v11, v12,v13,v14,v15.v16

Dice Partition Storage (OEDs)

Vertex Table

SVP 1: V_1 V_2 V_3
SVP 2: V_5 V_6 V_7
SVP 3: V_{11} V_{12} V_{15}

DVP 1: V_2 V_3 V_4 V_5 V_6
DVP 2: V_7 V_8 V_9 V_{10} V_{11}
DVP 3: V_{12} V_{13} V_{14} V_{15} V_{16}

P_1: *SVP* 1 ∪ *DVP* 1
P_2: *SVP* 1 ∪ *DVP* 3
P_3: *SVP* 2 ∪ *DVP* 2
P_4: *SVP* 3 ∪ *DVP* 3

Edge Table

P_1	P_2
(V_1, V_2)	(V_1, V_{12})
(V_2, V_3)	(V_1, V_{13})
(V_3, V_4)	(V_1, V_{14})
(V_3, V_5)	(V_1, V_{15})
(V_3, V_6)	(V_3, V_{12})

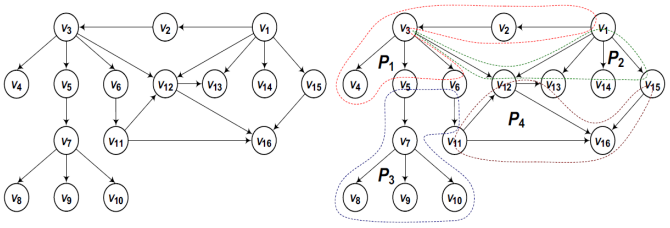
P_3	P_4
(V_5, V_7)	(V_{11}, V_{12})
(V_6, V_{11})	(V_{11}, V_{16})
(V_7, V_8)	(V_{12}, V_{13})
(V_7, V_9)	(V_{12}, V_{16})
(V_7, V_{10})	(V_{15}, V_{16})

Vertex Map

V_1	P_1, P_2
V_2	P_1
V_3	P_1, P_2
V_4	P_1
V_5	P_1, P_3
V_6	P_1, P_3
V_7	P_3
V_8	P_3
V_9	P_3
V_{10}	P_3
V_{11}	P_3, P_4
V_{12}	P_2, P_4
V_{13}	P_2, P_4
V_{14}	P_2
V_{15}	P_2, P_4
V_{16}	P_4

Advantage: Multi-level Hierarchical Graph Parallel Abstractions

- Choose smaller subgraph blocks such as dice partition or strip partition to balance the parallel computation efficiency among partition blocks

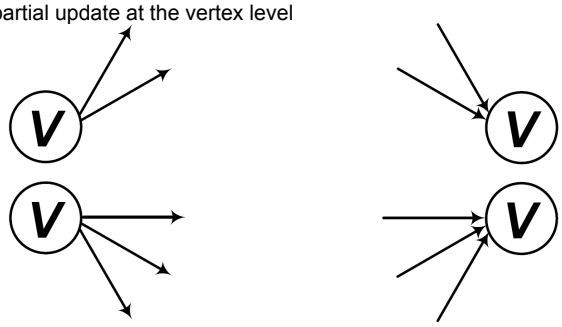


(a) Original Graph (b) Dice Partitioning

- Use larger subgraph blocks such as slice partition or strip partition to maximize sequential access and minimize random access

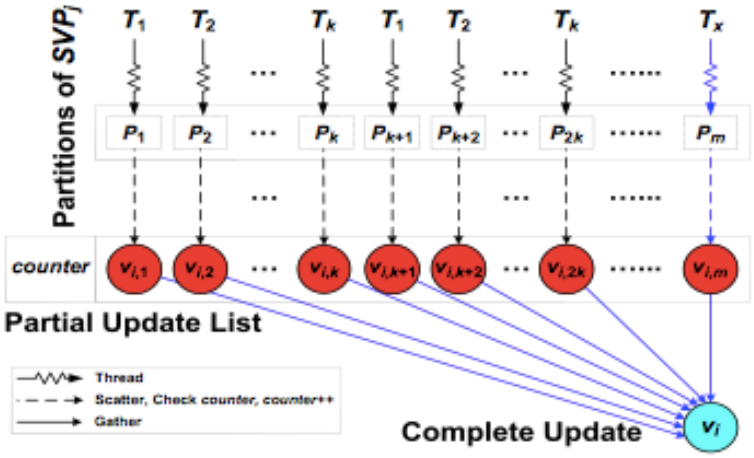
Partial Aggregation in Parallel

- Aggregation operation
  - Partially scatter vertex updates in parallel
  - Partially gather vertex updates in parallel
- Two-level Graph Computation Parallelism
  - Parallel partial update at the subgraph partition level (slice, strip or dice)
  - Parallel partial update at the vertex level



Scatter Gather

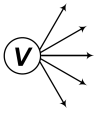
Multi-threading Update and Asynchronization



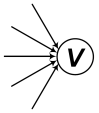
Programmable Interface

- vertex centric programming API, such as *Scatter* and *Gather*.

Scatter vertex updates to outgoing edges



Gather vertex updates from neighbor vertices and incoming edges



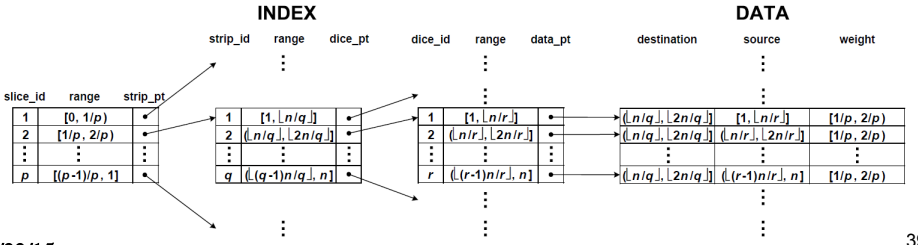
- Compile iterative algorithms into a sequence of internal function (routine) calls that understand the internal data structures for accessing the graph by different types of subgraph partition blocks

Algorithm 2 PageRank

```
1: Initialize(v)
2:     v.rank = 1.0;
3:
4: Scatter(v)
5:     msg = v.rank/v.degree;
6:     //send msg to destination vertices of v's out-edges
7:
8: Gather(v)
9:     state = 0;
10:    for each msg of v
11:    //receive msg from source vertices of v's in-edges
12:        state += msg; //summarize partial vertex updates
13:    v.rank = 0.15+0.85*state; //produce complete vertex update
```

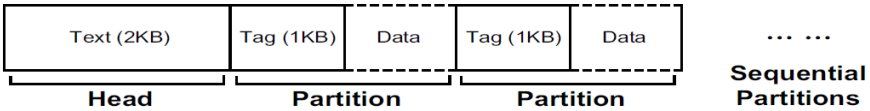
Partition-to-chunk Index
Vertex-to-Partition Index

- The dice-level index is a dense index that maps a dice ID and its DVP (or SVP) to the chunks on disk where the corresponding dice partition is stored physically
- The strip-level index is a two level sparse index, which maps a strip ID to the dice-level index-blocks and then map each dice ID to the dice partition chunks in the physical storage
- The slice level index is a three-level sparse index with slice index blocks at the top, strip index blocks at the middle and dice index blocks at the bottom, enabling fast retrieval of dices with a slice-specific condition



Partition-level Compression

- Iterative computations on large graphs incur non-trivial cost for the I/O processing
  - The I/O processing of Twitter dataset on a PC with 4 CPU cores and 16GB memory takes 50.2% of the total running time for PageRank (5 iterations)
- Apply in-memory gzip compression to transform each graph partition block into a compressed format before storing them on disk



Data Structure for Compressed Graph

Configuration of Partitioning Parameters

- **User definition**
- **Simple estimation**
- **Regression-based learning**
  - Construct a polynomial regression model to model the nonlinear relationship between independent variables p, q, r (partition parameters) and dependent variable T (runtime) with latent coefficient α_{ijk} and error term ε

$$T \approx f(p,q,r,\alpha) = \sum_{i=1}^{n_p}\sum_{j=1}^{n_q}\sum_{k=1}^{n_r}\alpha_{ijk}p^i q^j r^k + \epsilon$$

  - The goal of regression-based learning is to determine the latent α_{ijk} and ε to get the function between p, q, r and T
  - Select m limited samples of (p_l, q_l, r_l, T_l) $(1 \le l \le m)$ from the existing experiment results
  - Solve m linear equations consisting of m selected samples to generate the concrete α_{ijk} and ε
  - Utilize a successive convex approximation method (SCA) to find the optimal solution (i.e., the minimum runtime T) of the above polynomial function and the optimal parameters (i.e., p, q and r) when T is minimum

$$T_1 = \sum_{i=1}^{n_p}\sum_{j=1}^{n_q}\sum_{k=1}^{n_r}\alpha_{ijk}p_1^i q_1^j r_1^k + \epsilon$$

$$\cdots \qquad \cdots$$

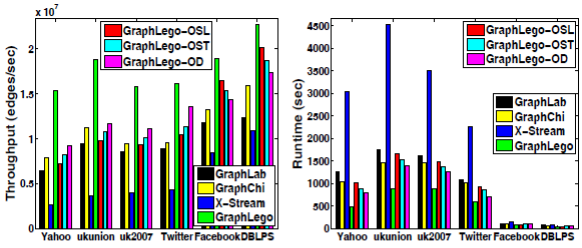
$$T_m = \sum_{i=1}^{n_p}\sum_{j=1}^{n_q}\sum_{k=1}^{n_r}\alpha_{ijk}p_m^i q_m^j r_m^k + \epsilon$$

Experimental Evaluation

- Computer server
  - Intel Core i5 2.66 GHz, 16 GB RAM, 1 TB hard drive, Linux 64-bit
- Graph parallel systems
  - **GraphLab** [Low et al., UAI'10]
  - **GraphChi** [Kyrola et al., OSDI'12]
  - **X-Stream** [Roy et al., SOSP'13]
- Graph applications

Application	Propagation	Core Computation
PageRank	single graph	matrix-vector
SpMV	single graph	matrix-vector
Connected Components	single graph	graph traversal
Diffusion Kernel	two graphs	matrix-matrix
Inc-Cluster	two graphs	matrix-matrix
Matrix Multiplication	two graphs	matrix-matrix
LMF	multigraph	matrix-vector
AEClass	multigraph	matrix-vector

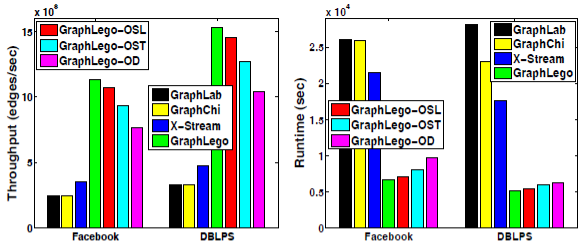
Execution Efficiency on Single Graph



(a) Throughput (b) Runtime

PageRank on Six Real Graphs

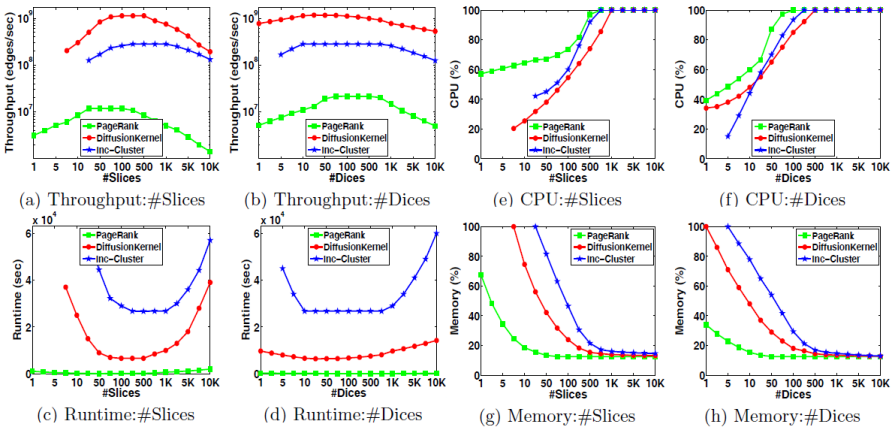
Execution Efficiency on Multiple Graphs



(a) Throughput (b) Runtime

Diffusion Kernel on Two Real Graphs

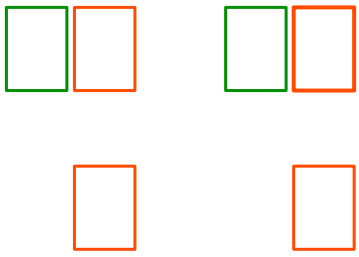
Decision of #Partitions



(a) Throughput:#Slices (b) Throughput:#Dices (e) CPU:#Slices (f) CPU:#Dices

(c) Runtime:#Slices (d) Runtime:#Dices (g) Memory:#Slices (h) Memory:#Dices

Efficiency of Regression-based Learning



GraphLego: Resource Aware GPS

- **Flexible multi-level hierarchical graph parallel abstractions**
  - Model a large graph as a 3D cube with source vertex, destination vertex and edge weight as the dimensions
  - Partitioning a big graph by: `slice, strip, dice` based graph partitioning
- **Access Locality Optimization**
  - Dice-based data placement: store a large graph on disk by minimizing non-sequential disk access and enabling more structured in-memory access
  - Construct partition-to-chunk index and vertex-to-partition index to facilitate fast access to slices, strips and dices
  - implement partition-level in-memory gzip compression to optimize disk I/Os
- **Optimization for Partitioning Parameters**
  - Build a regression-based learning model to discover the latent relationship between the number of partitions and the runtime

47

NSA Big Graph Study

[Paul Burkhardt, Chris Waring 2013]

Big Graphs challenge our conventional thinking on both algorithms and computer architecture.

New Graph500.org benchmark provides a foundation for conducting experiments on graph datasets.

Graph500 benchmark

Problem classes from 17 GB to 1 PB — many times larger than common datasets in literature.

48

Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets

Nadathur Satish, Narayanan Sundaram, Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M. Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey

49

Experimental Study: Goal

- Compares graph frameworks in terms of programming model and implementation of multiple algorithms
- Exposes performance gap (2-30X) between graph frameworks and hand-optimized native code
- Analyzes CPU usage, memory footprint, and network traffic to explain performance gap
- Shows performance gains of optimization techniques in native code and recommendations for graph frameworks

*"our goal is **not** to come up with a new graph processing benchmark or propose a new graph framework, but to analyze existing approaches better to **find out where they fall short**"*

Iterative Graph Computation

- **PageRank**
  - Iteratively computes rank (web page popularity) for each vertex (web page) in a directed graph (reference web)

Probability of a random jump

Pagerank of vertex j at iteration t

$$PR^{t+1}(i) = r + (1-r) * \sum_{j|(j,i)\in E} \frac{PR^t(j)}{\text{degree}(j)}$$

- **Breadth Frist Search (BFS)**
  - Traverses an undirected, unweighted graph from one vertex and compute the minimal distance
  - In each iteration:

$$Distance(i) = \min_{j|(j,i)\in E} Distance(j) + 1$$

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"

51

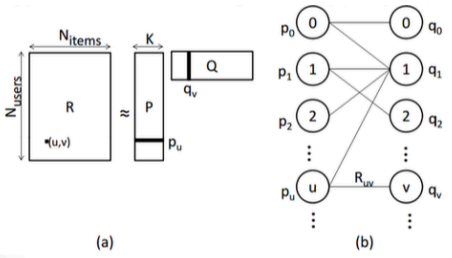
Iterative Graph Computation Algorithms

- **Triangle Counting**
  - Each pair of vertices in an edge compare their neighbourhood lists and count the number of shared neighbours

$$N_{triangles} = \sum_{i,j,k,i<j<k} E_{ij} \wedge E_{jk} \wedge E_{ik}$$

Existence of edge between i and k

- **Collaborative Filtering**
  - Estimates the rating of an item by a given user



$$\min_{\mathbf{p},\mathbf{q}} \sum_{(u,v)\in R} (\mathbf{R}_{uv} - \mathbf{p}_u^T \mathbf{q}_v)^2 + \lambda_p ||\mathbf{p}_u||^2 + \lambda_q ||\mathbf{q}_v||^2$$

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"

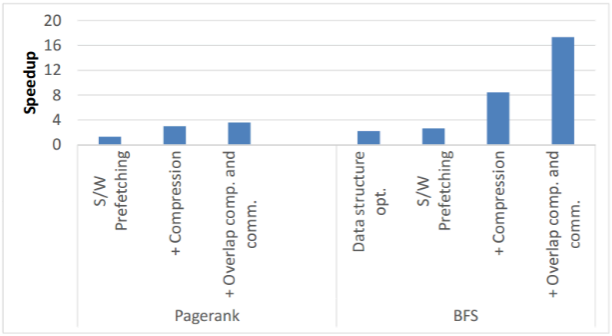
52

Experimental Setup

Dataset	# Vertices	# Edges
Facebook [1] Wikipedia [2] LiveJournal [2] Netflix [3]	2,937,612 3,566,908 4,847,571 480,189 users 17,770 movies	41,919,708 84,751,827 85,702,475 99,072,112 ratings
Twitter [4] Yahoo Music [5]	61,578,415 1,000,990 users 624,961 items	1,468,365,182 252,800,275 ratings
Synthetic Graph500	536,870,912	8,589,926,431
Synthetic Collaborative Filtering	63,367,472 users 1,342,176 items	16,742,847,256 ratings

Optimizations

- Key optimizations in native implementation
  - Data structures
  - Data compression
  - Overlap of Computation and Communication
  - Message passing mechanisms
  - Partitioning schemes



Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"

Experiment Results: Native Code

- Native hand-optimized implementation efficiency

Algorithm	Single Node		4 Nodes	
	H/W limitation	Efficiency	H/W limitation	Efficiency
PageRank	Memory BW	78 GBps (92%)	Network BW	2.3 GBps (42%)
BFS	Memory BW	64 GBps (74%)	Memory BW	54 GBps (63%)
Coll. Filtering	Memory BW	47 GBps (54%)	Memory BW	35 GBps (41%)
Triangle Count.	Memory BW	45 GBps (52%)	Network BW	2.2 GBps (40%)

55

Experimental Comparison (1)

- Slowdown factors of framework performance against native code on a *single node*

Algorithm	CombBLAS	GraphLab	SociaLite	Giraph	Galois
PageRank	1.9	3.6	2.0	39.0	1.2
BFS	2.5	9.3	7.3	567.8	1.1
Coll. Filtering	3.5	5.1	5.8	54.4	1.1
Triangle Count.	33.9	3.2	4.7	484.3	2.5

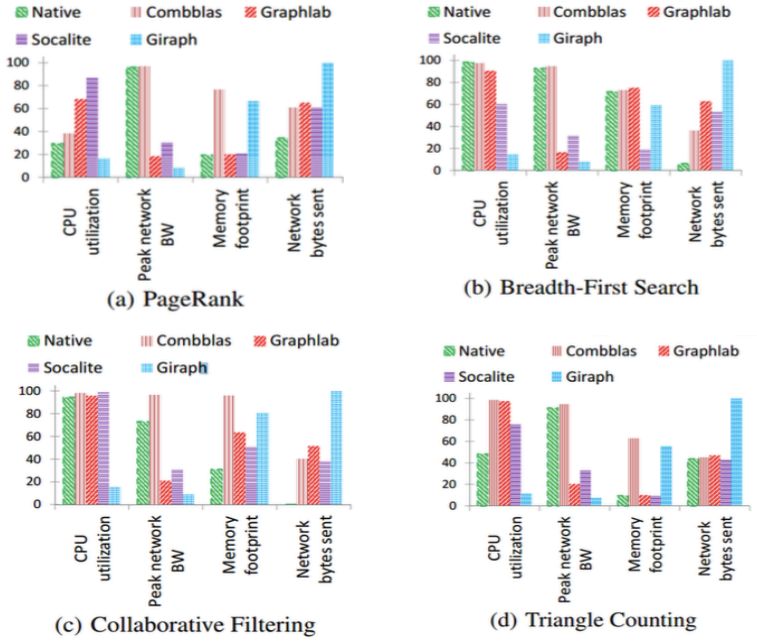
- Slowdown factors of framework performance against native code on *multiple nodes*

Algorithm	CombBLAS	GraphLab	SociaLite	Giraph
PageRank	2.5	12.1	7.9	74.4
BFS	7.1	29.5	18.9	494.3
Coll. Filtering	3.5	7.1	7.0	87.9
Triangle Count.	13.1	3.6	1.5	54.4

Figures from "Navigating the maze of graph analytics frameworks using massive graph datasets"

56

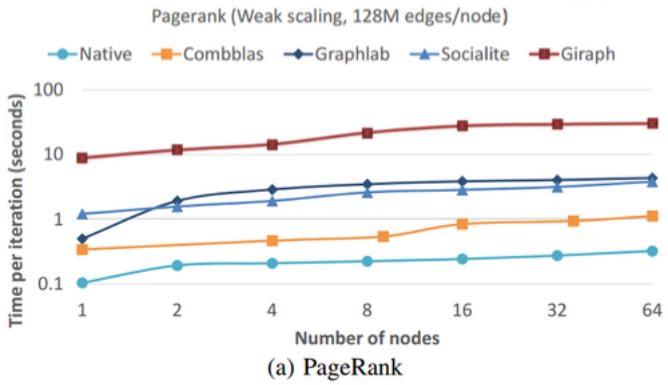
Experimental Comparison



57

Experiment Results: Multiple Nodes

- Performance on multiple nodes using large synthetic graphs



58

Reference

1. C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In EuroSys, pages 205–218, 2009.
2. T. Davis. The University of Florida Sparse Matrix Collection. http:// www.cise.ufl.edu/research/sparse/matrices.
3. J. Bennett and S. Lanning. The Netflix Prize. In KDD Cup and Workshop at ACM SIGKDD, 2007.
4. H. Kwak, C. Lee, H. Park, and S. B. Moon. What is twitter, a social network or a news media? In WWW, pages 591–600, 2010.
5. Yahoo! - Movie, Music, and Images Ratings Data Sets. http:// webscope.sandbox.yahoo.com/catalog.php?datatype =r.
6. R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang. Introducing the graph 500. Cray User's Group (CUG), 2010.

Questions