Graphs as Matrices

Lecture 22

CSCI 4974/6971

28 Nov 2016

Today's Biz

1. **Reminders**
2. Review
3. Graphs as Matrices

Reminders

- Assignment 6: due date Dec 8th
- Final Project Presentation: December 8th
- Project Report: December 11th
- Office hours: Tuesday & Wednesday 14:00-16:00 Lally 317
  - Or email me for other availability

Today's Biz

1. Reminders
2. **Review**
3. Graphs as Matrices

Quick Review

Temporal Networks:

- Graphs change over time
- Defining static vs. dynamic graphs
  - All links from all timeframes
  - Discrete model - construct temporal graph using updates that occur within certain windows
  - Continuous model - process updates as edges added or deleted, utilize streaming algorithms
- Dynamic distance, centrality, etc.
  - Consider both distance in hops and distance in time

Today's Biz

1. Reminders
2. Review
3. **Graphs as Matrices**

GraphBLAS: Motivation and Mathematical Foundations

Tim Mattson, Intel

GraphBLAS: Motivation and Mathematical Foundations

http://istc-bigdata.org/GraphBlas/

Tim Mattson

Intel Labs

timothy.g.mattson@intel.com

… and the "GraphBLAS gang":

David Bader (GATech), Aydın Buluç (LBNL), John Gilbert (UCSB), Joseph Gonzalez (UCB), Jeremy Kepner (MIT Lincoln Labs)

Outline

- Introduction: Graphs and Linear Algebra
- The Draft GraphBLAS primitives
- Conclusion/Summary

Motivation: History of BLAS

- BLAS: The Basic Linear Algebra subroutines

BLAS 1	$y \leftarrow \alpha x + y$	Lawson, Hanson, Kincaid and Krogh, 1979	LINPACK
BLAS 2	$y \leftarrow \alpha Ax + \beta y$	Dongarra, Du Croz, Hammarling and Hanson, 1988	LINPACK on vector machines
BLAS 3	$C \leftarrow \alpha AB + \beta C$	Dongarra, Du Croz, Hammarling and Hanson, 1990	LAPACK on cache based machines

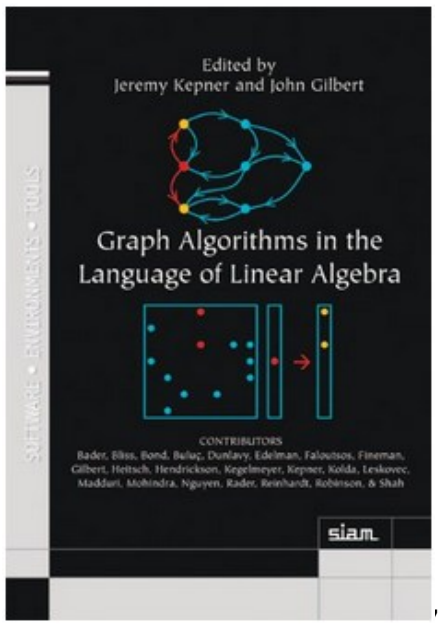
- The BLAS supported a separation of concerns:
  - HW/SW optimization experts tuned the BLAS for specific platforms.
  - Linear algebra experts built software on top of the BLAS .. high performance “for free”.

- It is difficult to overestimate the impact of the BLAS … they revolutionized the practice of computational linear algebra.

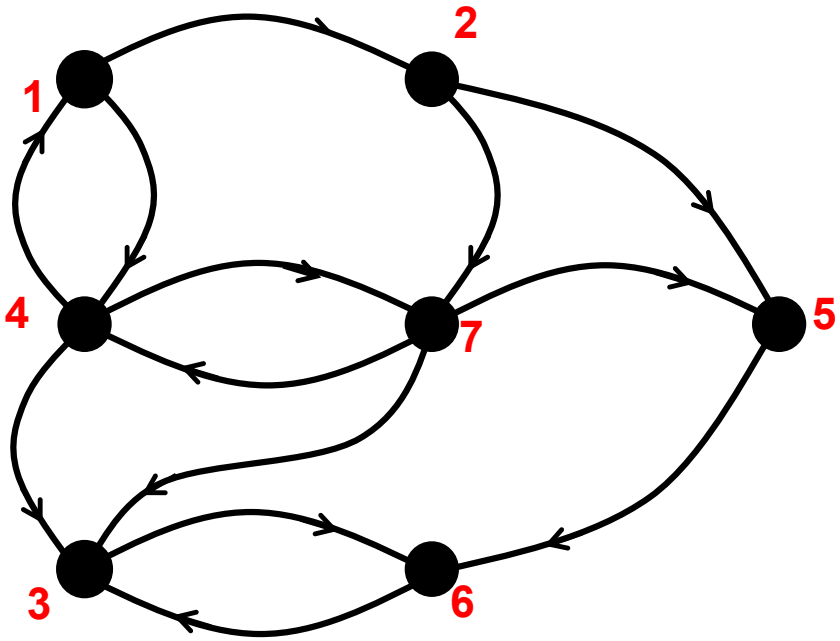
Can we standardize "the BLAS" of graph algorithms

- No, it is not reasonable to define a common set of graph algorithm building blocks:
  - Matching Algorithms to the hardware platform results in too much diversity to support a common set of "graph BLAS".
  - There is little agreement on how to represent graph algorithms and data structures.
  - Early standardization can inhibit innovation by locking in a sub-optimum status quo

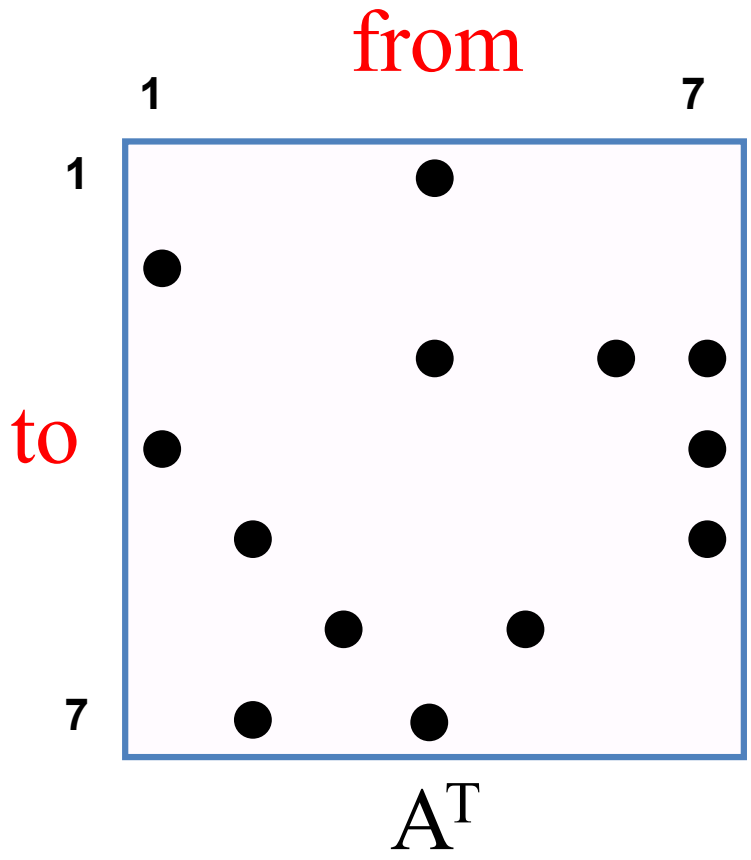
- Yes, it is reasonable to define a common set of graph algorithm building blocks … for Graphs in the language of Linear algebra.
  - Representing graphs in the language of linear algebra is a mature field … the algorithms, high level interfaces, and implementations vary, but the core primitives are well established .



Graphs in the Language of Linear Algebra

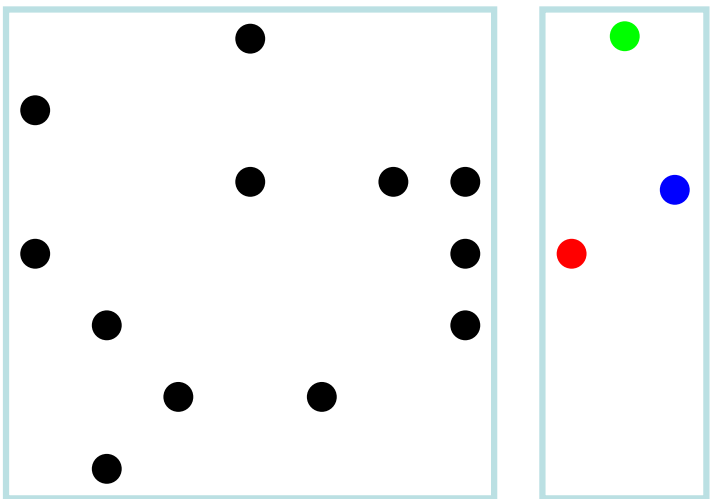


These two diagrams are equivalent representations of a graph.

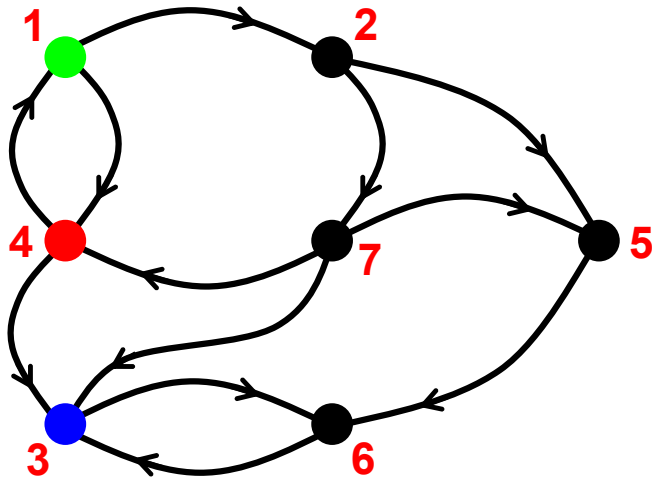


A = the adjacency matrix … Elements nonzero when vertices are adjacent

Multiple-source breadth-first search

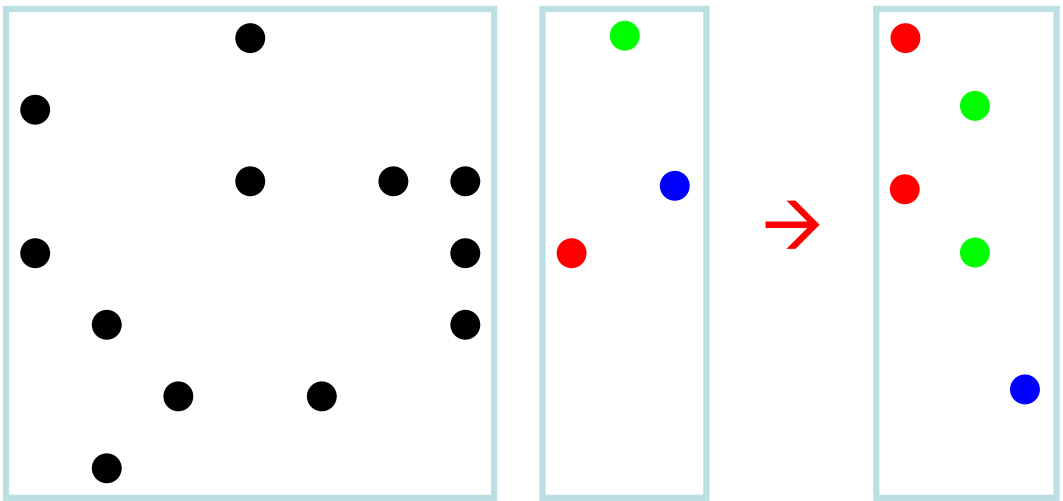


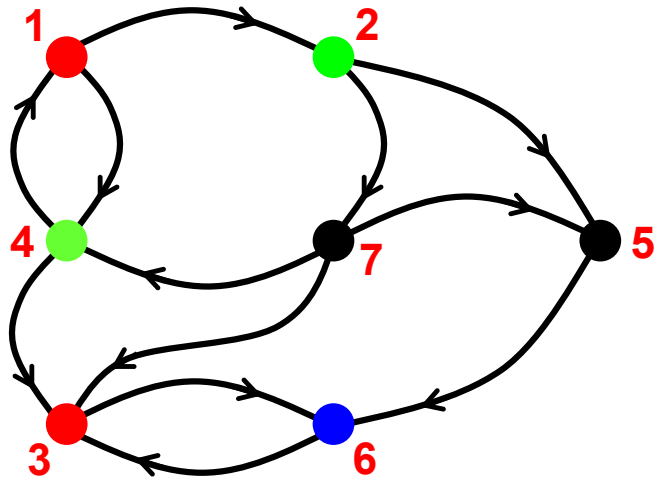
A^T X



UCSB

Multiple-source breadth-first search





- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

Multiplication of sparse matrices captures Breadth first search and serves as the foundation of all algorithms based on BFS

UCSB

Moving beyond BFS with Algebraic Semirings

- A semiring generalizes the operations of traditional linear algebra by replacing (+,*) with binary operations (Op1, Op2)
  - Op1 and Op2 have identity elements sometimes called 0 and 1
  - Op1 and Op2 are associative.
  - Op1 is commutative, Op2 distributes over op1 from both left and right
  - The Op1 identify is an Op2 annihilator.

Moving beyond BFS with Algebraic Semirings

- A semiring generalizes the operations of traditional linear algebra by replacing (+,*) with binary operations (Op1, Op2)
  - Op1 and Op2 have identity elements sometimes called 0 and 1
  - Op1 and Op2 are associative.
  - Op1 is commutative, Op2 distributes over op1 from both left and right
  - The Op1 identify is an Op2 annihilator.

(R, +, *, 0, 1) Real Field	Standard operations in linear algebra

Notation: (R, +, *, 0, 1)

Scalar type, Op1, Op2, Identity Op1, Identity Op2

Moving beyond BFS with Algebraic Semirings

- A semiring generalizes the operations of traditional linear algebra by replacing (+,*) with binary operations (Op1, Op2)
  - Op1 and Op2 have identity elements sometimes called 0 and 1
  - Op1 and Op2 are associative.
  - Op1 is commutative, Op2 distributes over op1 from both left and right
  - The Op1 identify is an Op2 annihilator.

(R, +, *, 0, 1) Real Field	Standard operations in linear algebra
({0,1}, |, &, 0, 1) Boolean Semiring	Graph traversal algorithms
(R U {∞},min, +, ∞, 0) Tropical semiring	Shortest path algorithms
(R U {∞}, min, *, ∞, 1)	Selecting a subgraph or contracting nodes to form a quotient graph.

The case for graph primitives based on sparse matrices

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

Traditional graph computations
Data driven, unpredictable communication.
Irregular and unstructured, poor locality of reference
Fine grained data accesses, dominated by latency

The case for graph primitives based on sparse matrices

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

Traditional graph computations	Graphs in the language of linear algebra
Data driven, unpredictable communication.	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse grained parallelism, bandwidth limited

GraphBLAS launched at HPEC'13 … co-authors of the GraphBLAS position paper

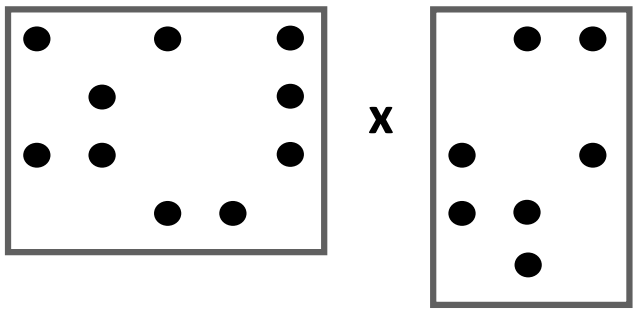
Tim Mattson	Intel Corporation	David Bader	Georgia Tech
Jon Berry	Sandia National Laboratory	Aydın Buluç	Lawrence Berkeley National Laboratory
Jack Dongarra	University of Tennessee	Christos Faloutsos	(Carnegie Melon University
John Feo	Pacific Northwest National Laboratory	John Gilbert	UC Santa Barbara
Joseph Gonzalez	UC Berkeley	Bruce Hendrickson	(Sandia National Laboratory
Jeremy Kepner	MIT	Charles Leiserson	MIT
Andrew Lumsdaine	Indiana University	David Padua	(University of Illinois at Urbana-Champaign
Stephen Poole	Oak Ridge	Steve Reinhardt	Cray Corporation
Mike Stonebraker	MIT	Steve Wallach	Convey Corp.
Andrew Yoo	Lawrence Livermore National Laboratory		

Outline

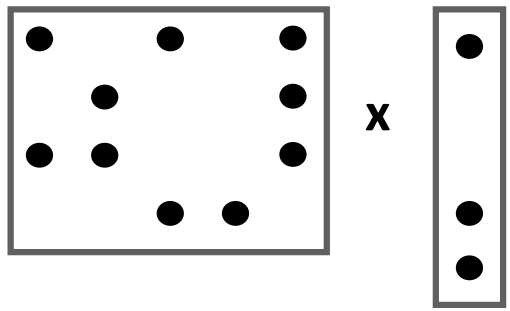
- Introduction: Graphs and Linear Algebra
- The Draft GraphBLAS primitives
- Conclusion/Summary

Linear-algebraic primitives

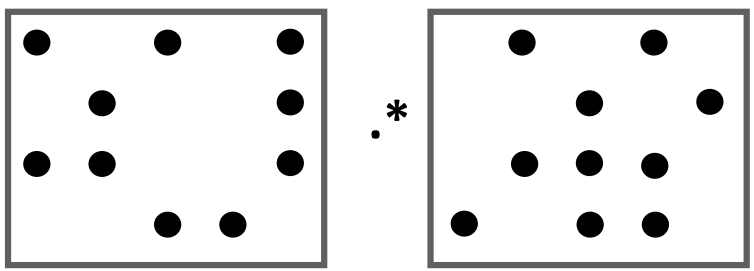
Sparse matrix-sparse matrix multiplication



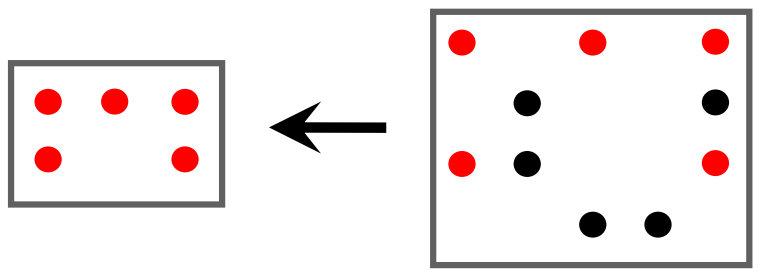
Sparse matrix-sparse vector multiplication



Element-wise operations



Sparse matrix indexing



The Combinatorial BLAS implements these, and more, on arbitrary semirings, e.g. (·, +), (and, or), (+, min)

UCSB

Draft GraphBLAS functions*

Function	Parameters	Returns	Math Notation
SpGEMM	- sparse matrices **A, B** and **C** - unary functors (op)	sparse matrix	**C** += op(**A**) * op(**B**)
SpM{Sp}V **(Sp: sparse)**	- sparse matrix **A** - sparse/dense vector **x**	sparse/dense vector	**y** = **A** * **x**
SpEWiseX	- sparse matrices or vectors - binary functor and predicate	in place or sparse matrix/vector	**C** = **A** .* **B**
Reduce	- sparse matrix **A** and functors	dense vector	**y** = sum(**A**, op)
SpRef	- sparse matrix **A** - index vectors **p** and **q**	sparse matrix	**B** = **A**(**p**,**q**)
SpAsgn	- sparse matrices **A** and **B** - index vectors **p** and **q**	none	**A**(**p**,**q**) = **B**
Scale	- sparse matrix **A** - dense matrix **B** or vector **X**	none	∀ **A**(i,j)!=0: **A**(i,j) *=**B**(i,j) + related forms for **X**
Apply	- any matrix or vector **X** - unary functor (op)	none	op(**X**)

*based on the Combinatorail BLAS from Buluç and Gilbert

Matrix times Matrix over semiring

Inputs
- matrix **A**: $\mathbb{S}^{MxN}$ (sparse or dense)
- matrix **B**: $\mathbb{S}^{NxL}$ (sparse or dense)

Optional Inputs
- matrix **C**: $\mathbb{S}^{MxL}$ (sparse or dense)
- scalar "add" function $\oplus$
- scalar "multiply" function $\otimes$
- transpose flags for **A**, **B**, **C**

Outputs
- matrix **C**: $\mathbb{S}^{MxL}$ (sparse or dense)

Implements $\mathbf{C} \oplus= \mathbf{A} \oplus.\otimes \mathbf{B}$

for j = 1 : N

$\mathbf{C}(i,k) = \mathbf{C}(i,k) \oplus (\mathbf{A}(i,j) \otimes \mathbf{B}(j,k))$

If input **C** is omitted, implements

$\mathbf{C} = \mathbf{A} \oplus.\otimes \mathbf{B}$

Transpose flags specify operation on $\mathbf{A}^T$, $\mathbf{B}^T$, and/or $\mathbf{C}^T$ instead

Notes
- $\mathbb{S}$ is the set of scalars, user-specified
- $\mathbb{S}$ defaults to IEEE double float
- $\oplus$ defaults to floating-point +
- $\otimes$ defaults to floating-point *

Specific cases and function names:
- SpGEMM: sparse matrix times sparse matrix
- SpMSpV: sparse matrix times sparse vector
- SpMV: Sparse matrix times dense vector
- SpMM: Sparse matrix times dense matrix

Sparse Matrix Indexing & Assignment

Inputs

matrix **A**: $\mathbb{S}^{MxN}$ (sparse)

matrix **B**: $\mathbb{S}^{|p|x|q|}$ (sparse)

vector $p \subseteq \{1, \ldots, M\}$

vector $q \subseteq \{1, \ldots, N\}$

Optional Inputs

none

Outputs

matrix **A**: $\mathbb{S}^{MxN}$ (sparse)

matrix **B**: $\mathbb{S}^{|p|x|q|}$ (sparse)

SpRef Implements **B** = **A**(p,q)

```
for i = 1 : |p|
  for j = 1 : |q|
    B(i,j) = A(p(i),q(j))
```

SpAsgn Implements **A**(p,q) = **B**

```
for i = 1 : |p|
  for j = 1 : |q|
    A(p(i),q(j)) = B(i,j)
```

Notes

$\mathbb{S}$ is the set of scalars, user-specified

$\mathbb{S}$ defaults to IEEE double float

$|p|$ = length of vector p

$|q|$ = length of vector q

Specific cases and function names

SpRef: get sub-matrix

SpAsgn: assign to sub-matrix

Element-Wise Operations

Inputs

matrix **A**: $\mathbb{S}^{MxN}$ (sparse or dense)

matrix **B**: $\mathbb{S}^{MxN}$ (sparse or dense)

Optional Inputs

matrix **C**: $\mathbb{S}^{MxN}$ (sparse or dense)

scalar "add" function $\oplus$

scalar "multiply" function $\otimes$

Outputs

matrix **C**: $\mathbb{S}^{MxN}$ (sparse or dense)

Implements $\mathbf{C} \oplus= \mathbf{A} \otimes \mathbf{B}$

for i = 1 : M

  for j = 1 : N

    $\mathbf{C}(i,j) = \mathbf{C}(i,j) \oplus (\mathbf{A}(i,j) \otimes \mathbf{B}(i,j))$

If input **C** is omitted, implements

$\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$

Notes

$\mathbb{S}$ is the set of scalars, user-specified

$\mathbb{S}$ defaults to IEEE double float

$\oplus$ defaults to floating-point +

$\otimes$ defaults to floating-point *

Specific cases and function names:

SpEWiseX: matrix elementwise

M=1 or N=1: vector elementwise

Scale: when **A** or **B** is a scalar

Apply/Update

Inputs

matrix **A**: $\mathbb{S}^{MxN}$ (sparse or dense)

Optional Inputs

matrix **C**: $\mathbb{S}^{MxN}$ (sparse or dense)

scalar "add" function $\oplus$

unary function f()

Outputs

matrix **C**: $\mathbb{S}^{MxN}$ (sparse or dense)

Implements $\mathbf{C} \oplus= f(\mathbf{A})$

```
for i = 1 : M
  for j = 1 : N
    if A(i,j) ≠ 0
      C(i,j) = C(i,j) ⊕ f(A(i,j))
```

If input **C** is omitted, implements $\mathbf{C} = f(\mathbf{A})$

Notes

$\mathbb{S}$ is the set of scalars, user-specified

$\mathbb{S}$ defaults to IEEE double float

$\oplus$ defaults to floating-point +

Specific cases and function names:

Apply: matrix apply

M=1 or N=1: vector apply

Matrix/Vector Reductions

Inputs
matrix **A**: $\mathbb{S}^{MxN}$ (sparse or dense)

Optional Inputs
vector **c**: $\mathbb{S}^{M}$ or $\mathbb{S}^{N}$ (sparse or dense)
scalar "add" function $\oplus$
dimension d: 1 or 2

Outputs
matrix **c**: $\mathbb{S}^{MxN}$ (sparse or dense)

Implements $\mathbf{c}(i) \oplus= \oplus_j \mathbf{A}(i,j)$

```
for i = 1 : M
  for j = 1 : N
    c(i) = c(i) ⊕ A(i,j)
```

If input **C** is omitted, implements

$\mathbf{c}(i) = \oplus_j \mathbf{A}(i,j)$

Notes
$\mathbb{S}$ is the set of scalars, user-specified
$\mathbb{S}$ defaults to IEEE double float
$\oplus$ defaults to floating-point +
d defaults to 2

Specific cases and function names:
Reduce (d = 1): reduce matrix to row vector
Reduce (d = 2): reduce matrix to col vector

Outline

- Introduction: Graphs and Linear Algebra
- The Draft GraphBLAS primitives
- Conclusion/Summary

Conclusion/Summary

- The time is right to define a Standard to support "Graphs in the Language of Linear Algebra".

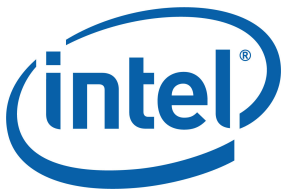
- Agreeing on a standard could have a transformative impact on Graph Algorithms research … much as the original BLAS did on computational Linear Algebra.

- Starting from the CombBLAS (Buluç and Gilbert), we have produced an initial Draft set of Primitives.

- Join with us to turn this into a final spec
  - Follow our work at: http://istc-bigdata.org/GraphBlas/
  - Send email to timothy.g.mattson@intel.com if you want to be added to the GraphBLAS Google Group and join the effort

The Graph BLAS effort and its implications for Exascale

Aydin Buluc, LBNL

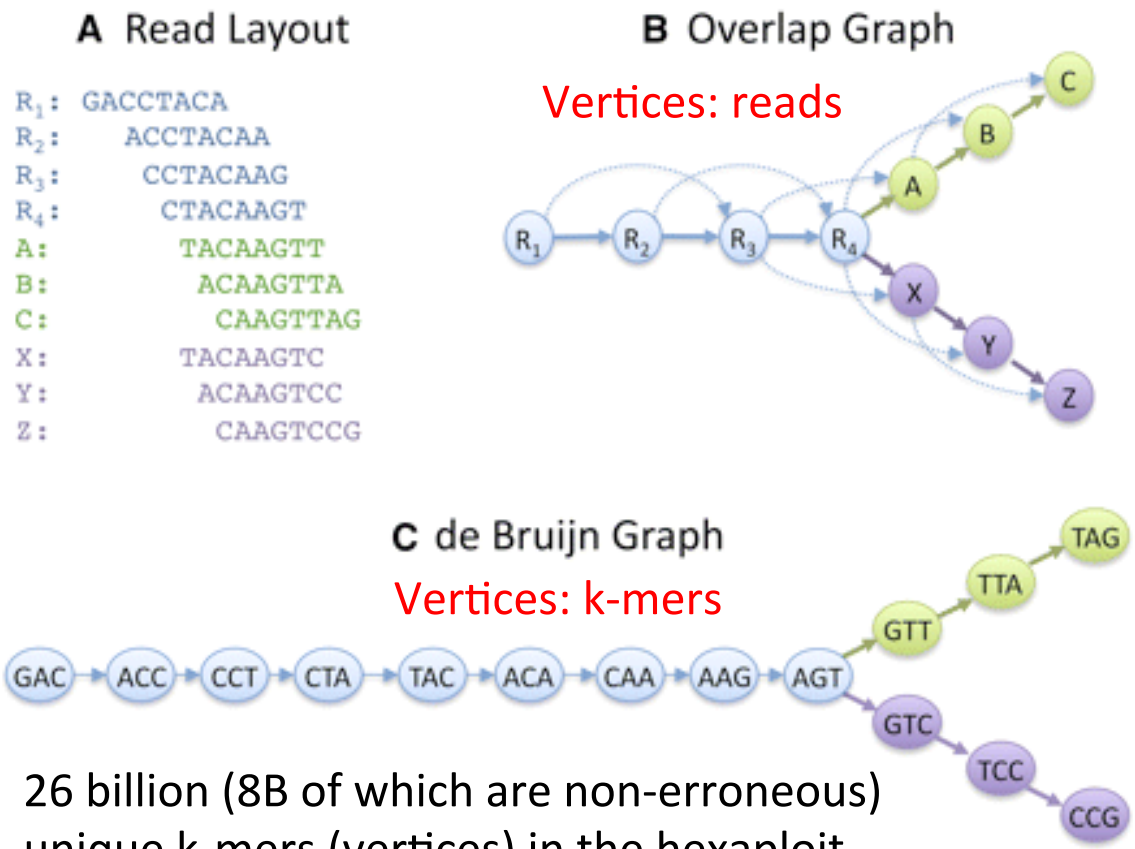

The Graph BLAS effort and its implications for Exascale

David Bader (GA Tech), **Aydın Buluç (LBNL)**, John Gilbert (UCSB), Joseph Gonzalez (UCB), Jeremy Kepner (MIT LL), Tim Mattson (Intel)

SIAM Workshop on Exascale Applied Mathematics Challenges and Opportunities (EX14)
July 6, 2014

Graphs matter in Applied Math

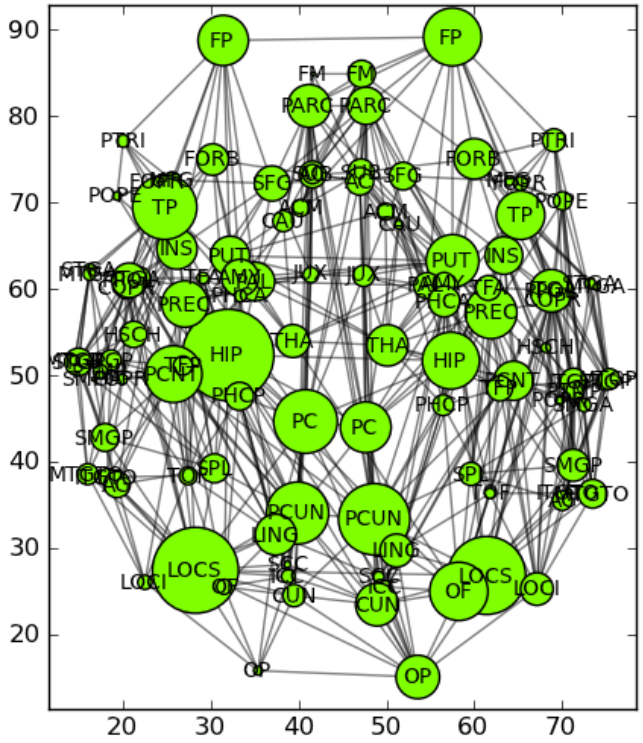
Whole genome assembly



26 billion (8B of which are non-erroneous) unique k-mers (vertices) in the hexaploit wheat genome W7984 for k=51

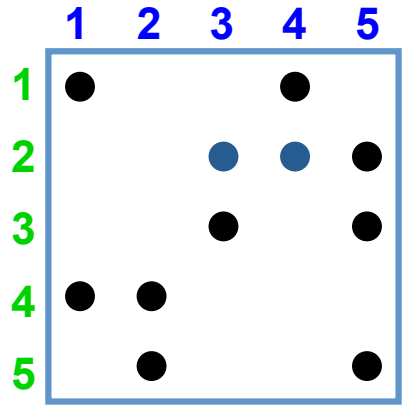
Schatz et al. (2010) Perspective: Assembly of Large Genomes w/2nd-Gen Seq. Genome Res. (figure reference)

Graph Theoretical analysis of Brain Connectivity

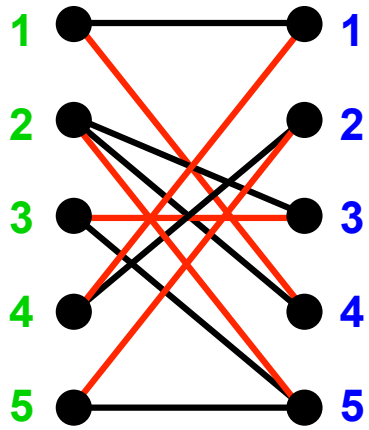


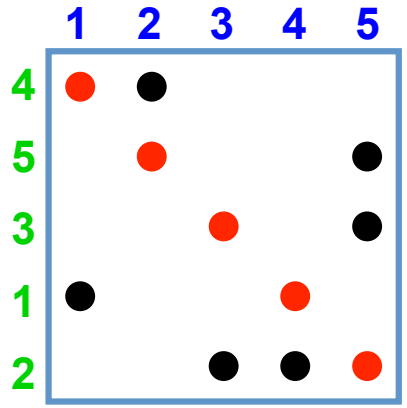
Potentially millions of neurons and billions of edges with developing technologies

Graphs matter in Applied Math



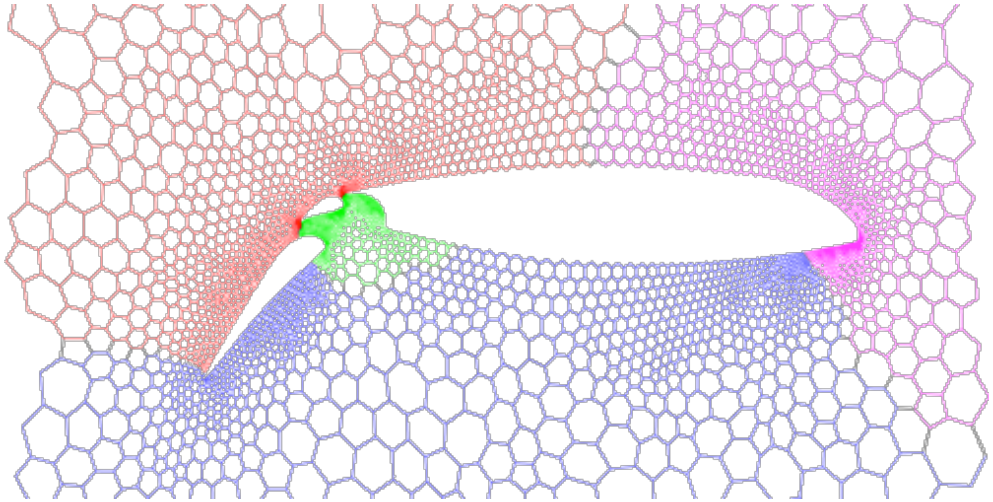
A





PA

Matching in bipartite graphs: Permuting to heavy diagonal or block triangular form

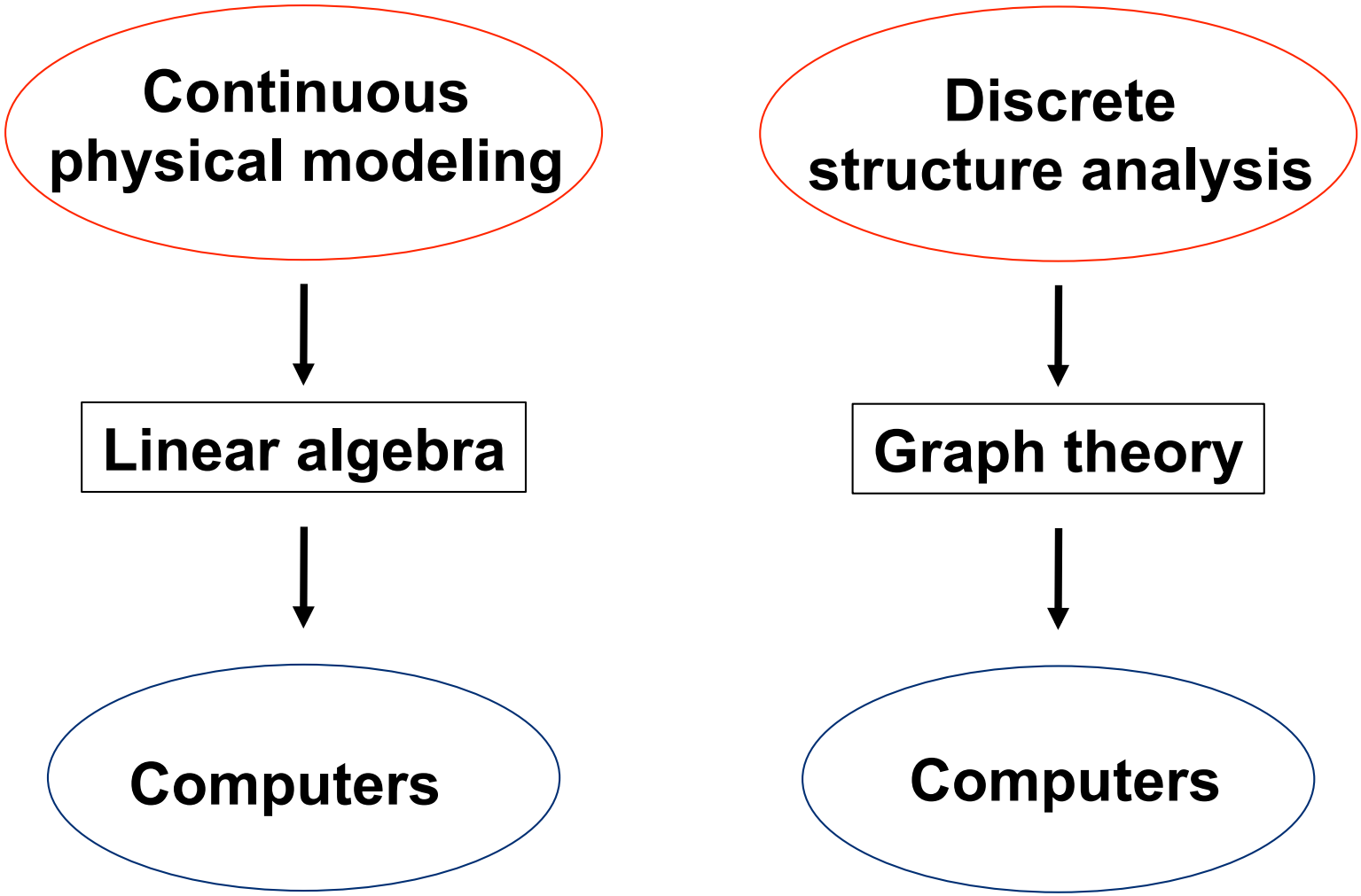


Graph partitioning: *Dynamic load balancing* in parallel simulations

Picture (left) credit: Sanders and Schulz

Problem size: as big as the sparse linear system to be solved or the simulation to be performed

Graphs as middleware

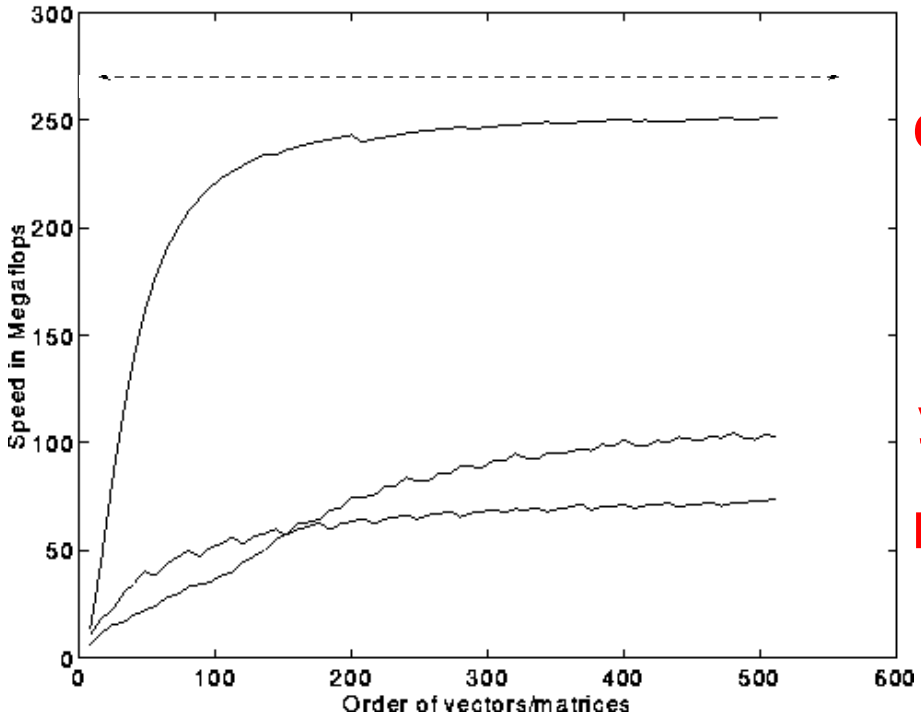


Graphs as middleware

By analogy to numerical scientific computing. . .

What should the Combinatorial BLAS look like?

Basic Linear Algebra Subroutines (BLAS): Ops/Sec vs. Matrix Size



$C = A*B$

$y = A*x$

$\mu = x^T y$

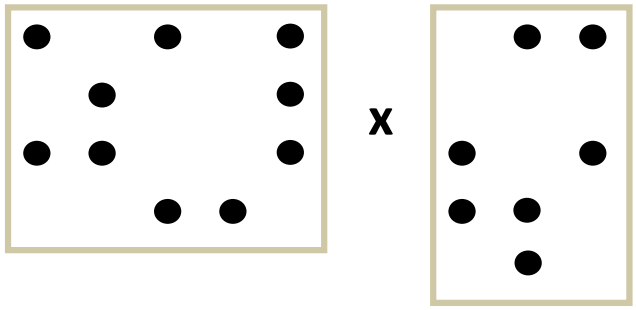
The case for sparse matrices

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

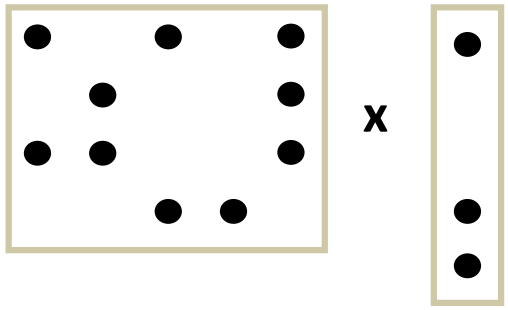
Traditional graph computations	Graphs in the language of linear algebra
Data driven, unpredictable communication.	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse grained parallelism, bandwidth limited

Linear-algebraic primitives for graphs

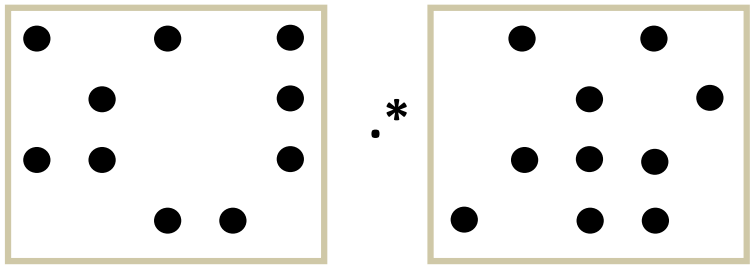
Sparse matrix-sparse matrix multiplication



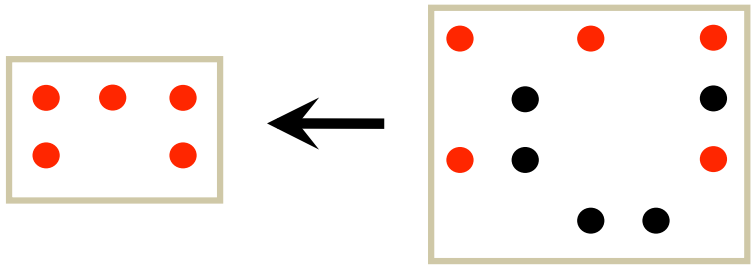
Sparse matrix-sparse vector multiplication



Element-wise operations



Sparse matrix indexing



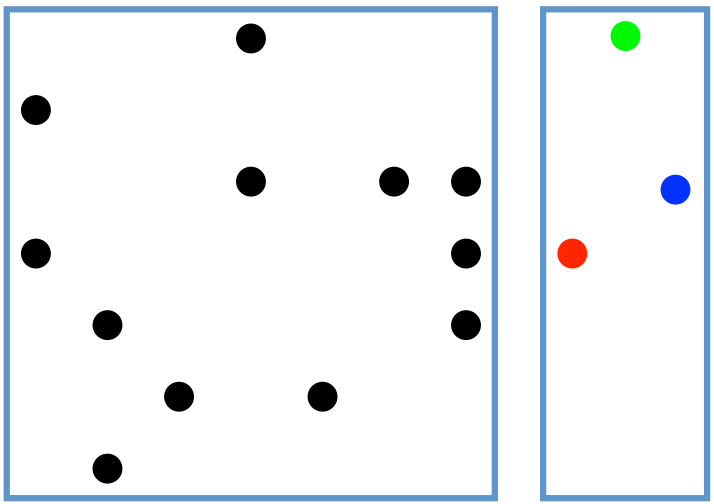
The Combinatorial BLAS implements these, and more, on arbitrary semirings, e.g. (×, +), (and, or), (+, min)

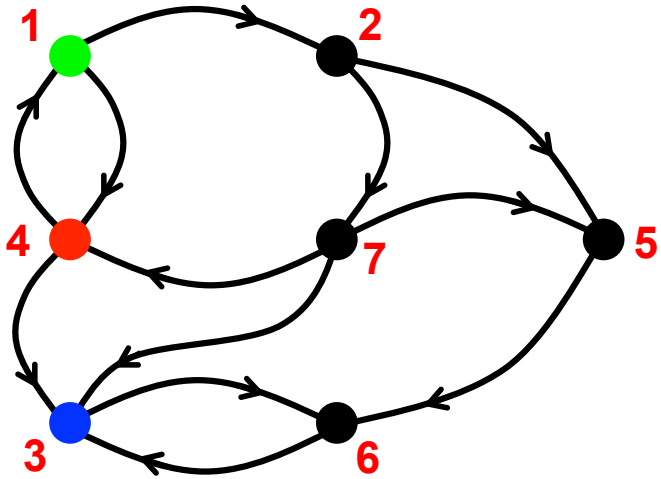
Examples of semirings in graph algorithms

Real field: **(R, +, x)**	Classical numerical linear algebra
Boolean algebra: **({0 1}, |, &)**	Graph traversal
Tropical semiring: **(R U {∞}, min, +)**	Shortest paths
(S, select, select)	Select subgraph, or contract nodes to form quotient graph
(edge/vertex attributes, vertex data aggregation, edge data processing)	Schema for user-specified computation at vertices and edges
(R, max, +)	Graph matching &network alignment
(R, min, times)	Maximal independent set

- **Shortened semiring notation: (Set, Add, Multiply)**. Both identities omitted.
- **Add:** Traverses edges, **Multiply:** Combines edges/paths at a vertex
- Neither add nor multiply needs to have an inverse.
- Both **add** and **multiply** are **associative**, **multiply distributes** over **add**

Multiple-source breadth-first search

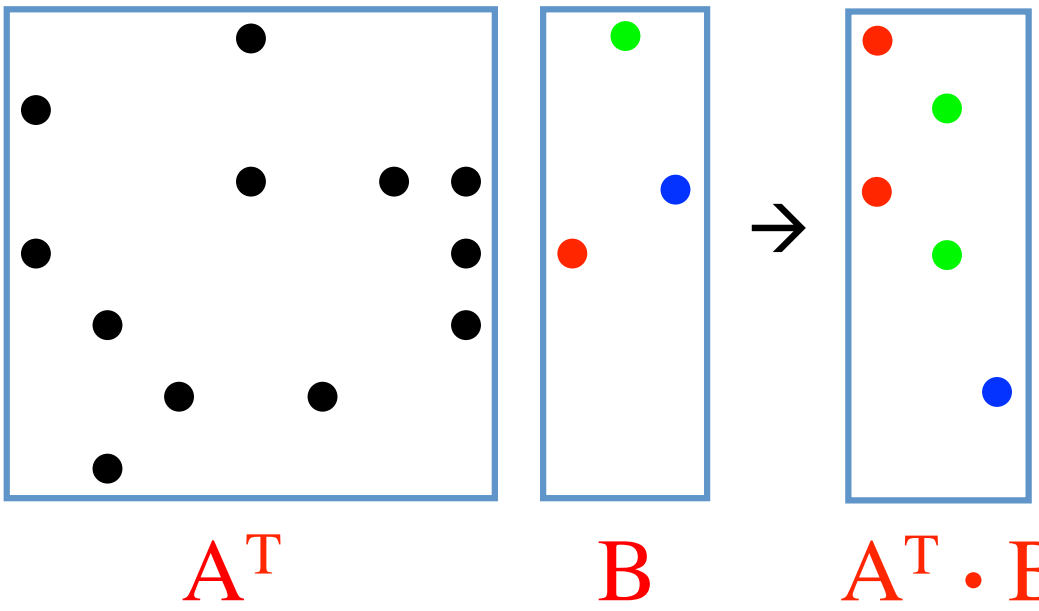


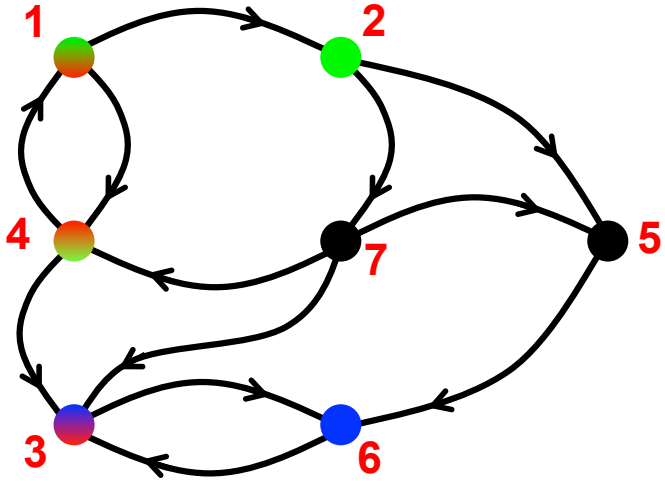


- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
- Highly-parallel implementation for Betweenness Centrality*

*: A measure of influence in graphs, based on shortest paths

Multiple-source breadth-first search





- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
- Highly-parallel implementation for Betweenness Centrality*

*: A measure of influence in graphs, based on shortest paths

Graph algorithm comparison (LA: linear algebra)

Slide inspiration: Jeremy Kepner (MIT LL)

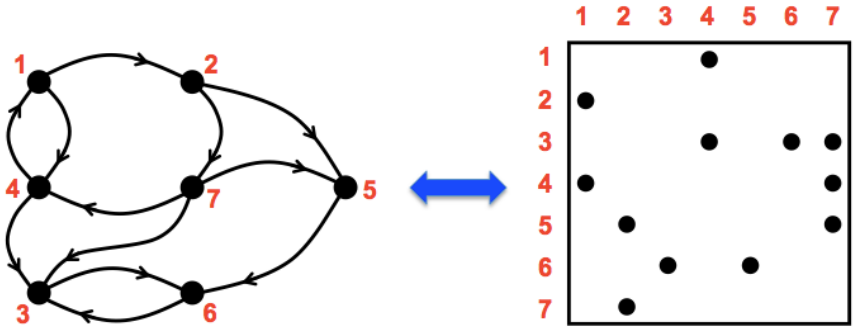
Algorithm (Problem)	Canonical Complexity	LA-Based Complexity	Critical Path (for LA)
Breadth-first search	$\Theta(m)$	$\Theta(m)$	$\Theta(diameter)$
Betweenness Centrality (unweighted)	$\Theta(mn)$	$\Theta(mn)$	$\Theta(diameter)$
All-pairs shortest-paths (dense)	$\Theta(n^3)$	$\Theta(n^3)$	$\Theta(n)$
Prim (MST)	$\Theta(m+n \log n)$	$\Theta(n^2)$	$\Theta(n)$
Borůvka (MST)	$\Theta(m \log n)$	$\Theta(m \log n)$	$\Theta(\log^2 n)$
Edmonds-Karp (Max Flow)	$\Theta(m^2n)$	$\Theta(m^2n)$	$\Theta(mn)$
Greedy MIS (MIS)	$\Theta(m+n \log n)$	$\Theta(mn+n^2)$	$\Theta(n)$
Luby (MIS)	$\Theta(m+n \log n)$	$\Theta(m \log n)$	$\Theta(\log n)$

Majority of selected algorithms can be represented with array-based constructs with equivalent complexity.

($n = |V|$ and $m = |E|$)

Combinatorial BLAS

http://gauss.cs.ucsb.edu/~aydin/CombBLAS



An extensible distributed-memory library offering a small but powerful set of linear algebraic operations specifically targeting graph analytics.

- Aimed at graph algorithm designers/programmers who are not expert in mapping algorithms to parallel hardware.
- Flexible templated C++ interface; 2D data decomposition
- Scalable performance from laptop to 100,000-processor HPC.
- Open source software (v1.4.0 released January, 2014)

Matrix times Matrix over semiring

Inputs
- matrix **A**: $\mathbb{S}^{MxN}$ (sparse or dense)
- matrix **B**: $\mathbb{S}^{NxL}$ (sparse or dense)

Optional Inputs
- matrix **C**: $\mathbb{S}^{MxL}$ (sparse or dense)
- scalar "add" function $\oplus$
- scalar "multiply" function $\otimes$
- transpose flags for **A**, **B**, **C**

Outputs
- matrix **C**: $\mathbb{S}^{MxL}$ (sparse or dense)

Implements $\mathbf{C} \oplus= \mathbf{A} \oplus.\otimes \mathbf{B}$

for j = 1 : N

$\mathbf{C}(i,k) = \mathbf{C}(i,k) \oplus (\mathbf{A}(i,j) \otimes \mathbf{B}(j,k))$

If input **C** is omitted, implements

$\mathbf{C} = \mathbf{A} \oplus.\otimes \mathbf{B}$

Transpose flags specify operation on $\mathbf{A}^T$, $\mathbf{B}^T$, and/or $\mathbf{C}^T$ instead

Notes
- $\mathbb{S}$ is the set of scalars, user-specified
- $\mathbb{S}$ defaults to IEEE double float
- $\oplus$ defaults to floating-point +
- $\otimes$ defaults to floating-point *

Specific cases and function names:
- SpGEMM: sparse matrix times sparse matrix
- SpMSpV: sparse matrix times sparse vector
- SpMV: Sparse matrix times dense vector
- SpMM: Sparse matrix times dense matrix

Can we standardize a "Graph BLAS"?

No, it's not reasonable to define a universal set of building blocks.

Huge diversity in matching graph algorithms to hardware platforms.
No consensus on data structures or linguistic primitives.
Lots of graph algorithms remain to be discovered.
Early standardization can inhibit innovation.

Yes, it *is* reasonable to define a common set of building blocks…
… for graphs as linear algebra.

Representing graphs in the language of linear algebra is a mature field.
Algorithms, high level interfaces, and implementations vary.
But the core primitives are well established.

The Graph BLAS effort

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

Abstract-- It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitive building blocks. This paper is a position paper defining the problem and announcing our intention to launch an open effort to define this standard.

- The Graph BLAS Forum: http://istc-bigdata.org/GraphBlas/
- Graph Algorithms Building Blocks (GABB workshop at IPDPS'14): http://www.graphanalysis.org/workshop2014.html

Challenges at Exascale

"New algorithms need to be developed that identify and leverage more concurrency and that reduce synchronization and communication" - ASCR Applied Mathematics Research for Exascale Computing Report

High-performance *requirement is the invariant for {any}scale*

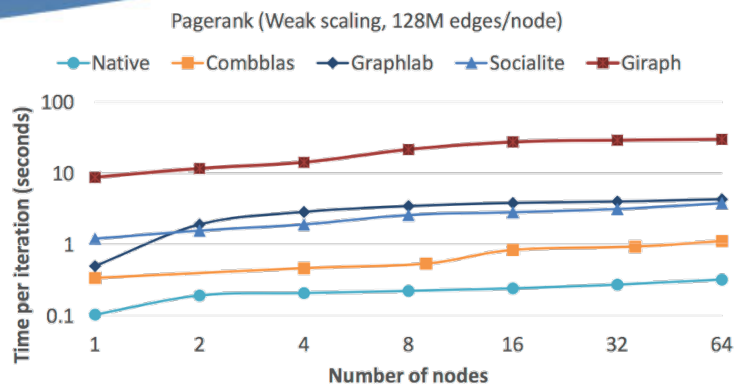
Challenges specific to Exascale and beyond:

- Power/Energy
- Data Locality
- Extreme Concurrency/Parallelism
- Resilience/ Fault tolerance

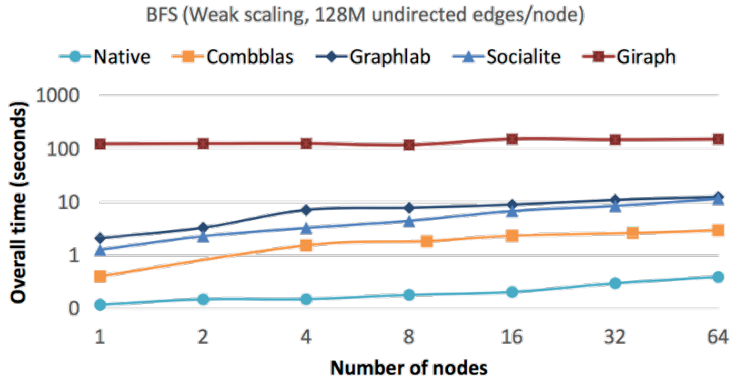
Performance of Linear Algebraic Graph Algorithms

Combinatorial BLAS fastest among all tested graph processing frameworks on 3 out of 4 benchmarks in an independent study by Intel.

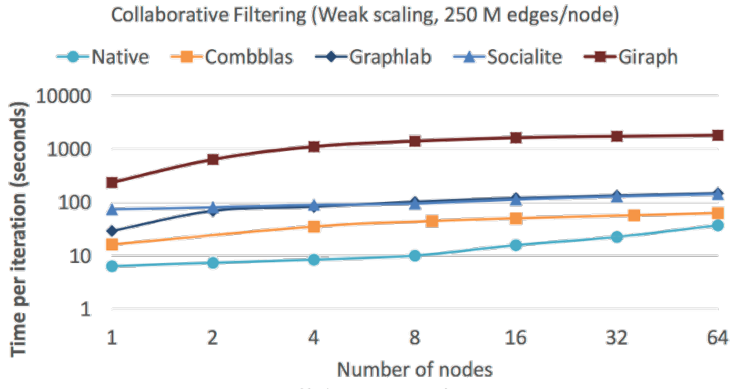
The linear algebra abstraction enables high performance, within 4X of native performance for PageRank and Collaborative filtering.



(a) PageRank



(b) Breadth-First Search

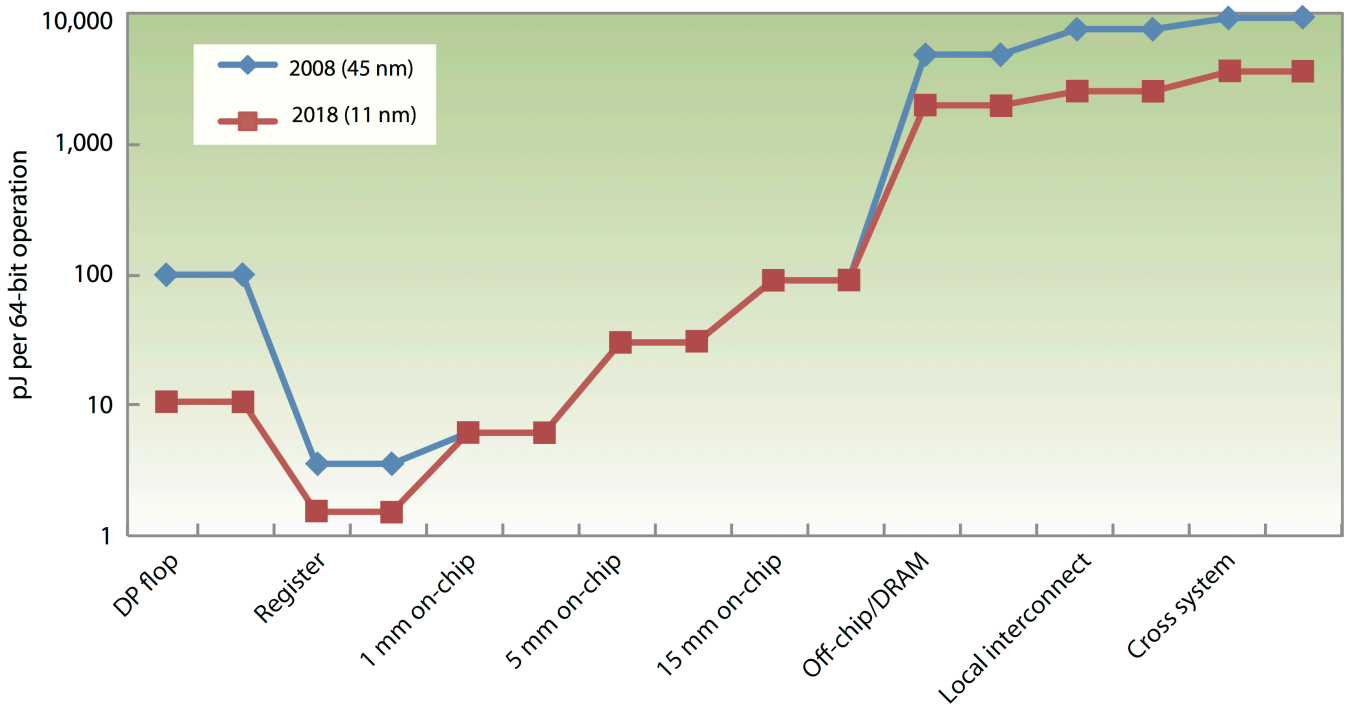


(c) Collaborative Filtering

Satish, Nadathur, et al. "Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets", in SIGMOD'14

Energy and data locality challenges

"Data movement is overtaking computation as the most dominant cost of a system both in terms of dollars and in terms of energy consumption. Consequently, we should be more explicit about reasoning about data movement."



Data movement (communication) costs **energy**

Image courtesy of Kogge & Shalf

Exascale Computing Trends: Adjusting to the "New Normal" for Computer Architecture
Kogge, Peter and Shalf, John, Computing in Science & Engineering, 15, 16-26 (2013)

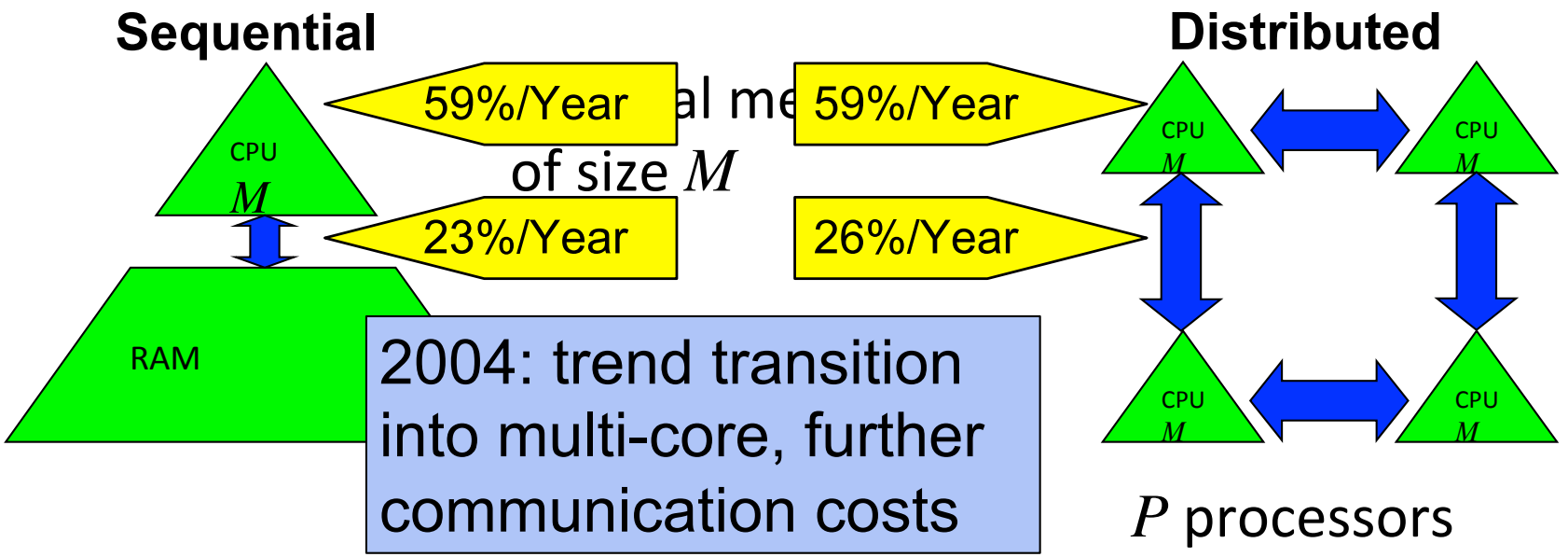
Communication-avoidance motivation

Two kinds of costs:

- Arithmetic (FLOPs)
- Communication: moving data

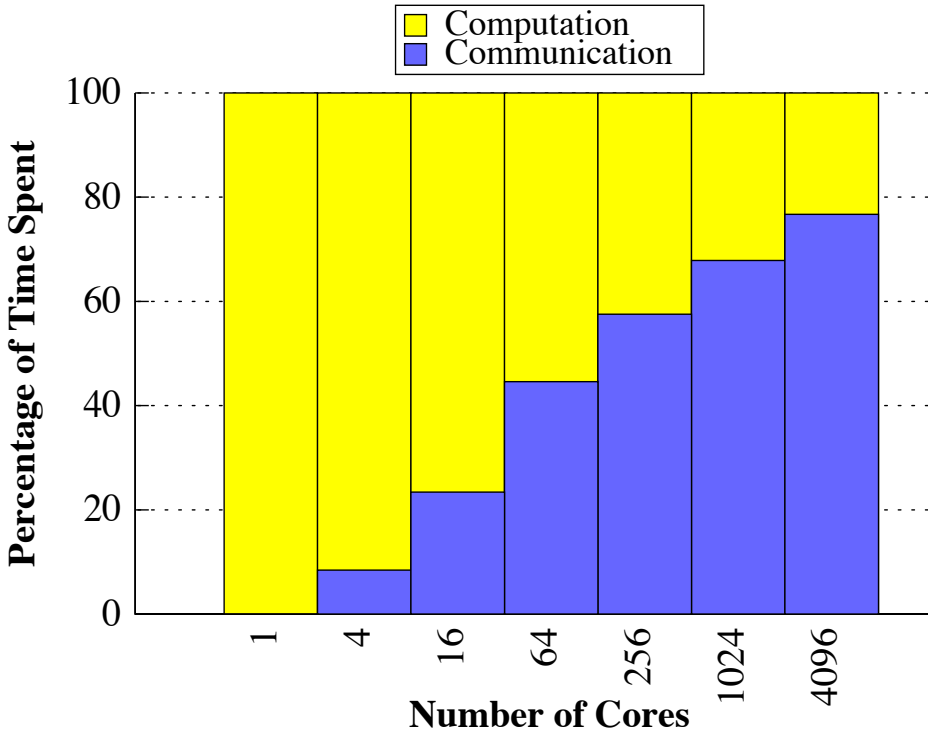
Develop faster algorithms: minimize communication (to lower bound if possible)

$$Running\ time = \gamma \cdot \#FLOPs + \beta \cdot \#Words + (\alpha \cdot \#Messages)$$



Communication crucial for graphs

- Often no surface to volume ratio.
- Very little data reuse in existing algorithmic formulations *
- Already heavily communication bound



2D sparse matrix-matrix multiply emulating:

- Graph contraction
- AMG restriction operations

Scale 23 R-MAT (scale-free graph) **times** order 4 restriction operator

Cray XT4, Franklin, NERSC

B., Gilbert. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal of Scientific Computing*, 2012

Reduced Communication Graph Algorithms

Communication avoiding approaches in linear algebra:

[A] Exploiting extra available memory (2.5D algorithms)

- typically applies to matrix-matrix operations

[B] Communicating once every k steps (k-step Krylov methods)

- typically applies to iterative sparse methods

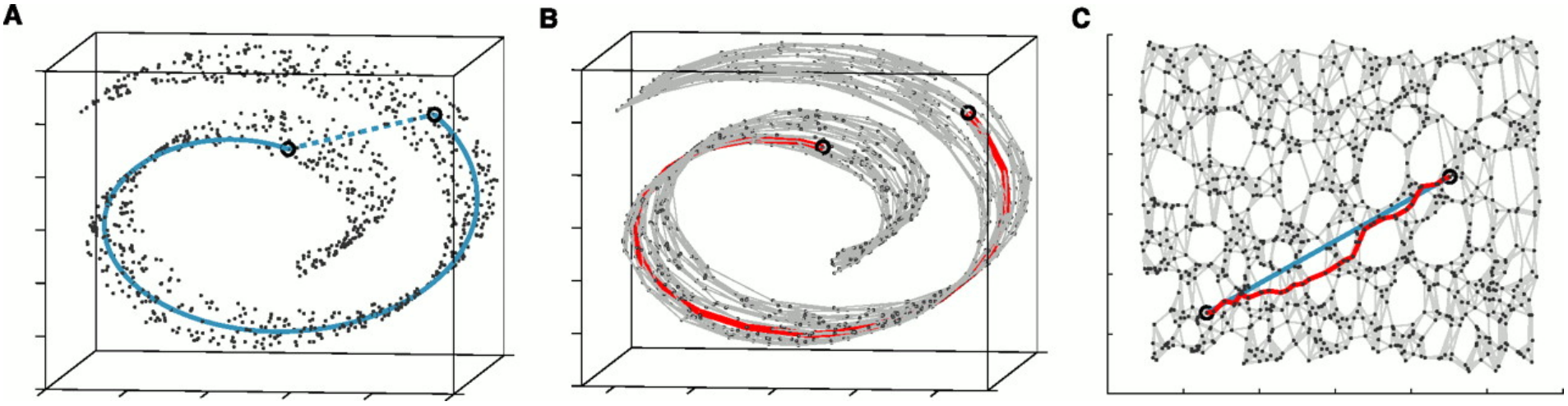
Good news: We successfully generalized A to sparse matrix-matrix multiplication (graph contraction, multi-source BFS, clustering, etc.) and all pairs shortest paths (Isomap).

Unknown: if B can be applied to iterative graph algorithms.

Manifold Learning

Isomap (Nonlinear dimensionality reduction): Preserves the intrinsic geometry of the data by using the geodesic distances on manifold between all pairs of points

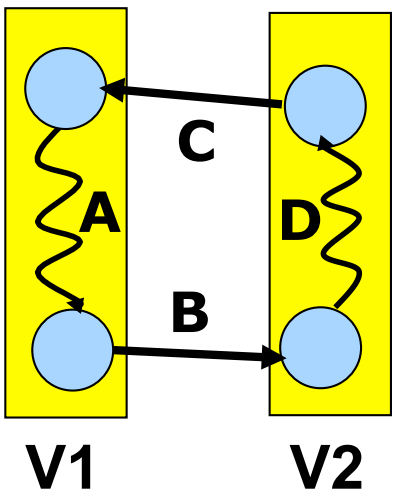
Tools used or desired:
- K-nearest neighbors
- ***All pairs shortest paths (APSP)***
- Top-k eigenvalues



Tenenbaum, Joshua B., Vin De Silva, and John C. Langford. "A global geometric framework for nonlinear dimensionality reduction." Science 290.5500 (2000): 2319-2323.

All pairs shortest paths at Scale

R-Kleene: A recursive APSP algorithm that is rich in semiring matrix-matrix multiplications



A	B
C	D

+ is "min", **×** is "add"

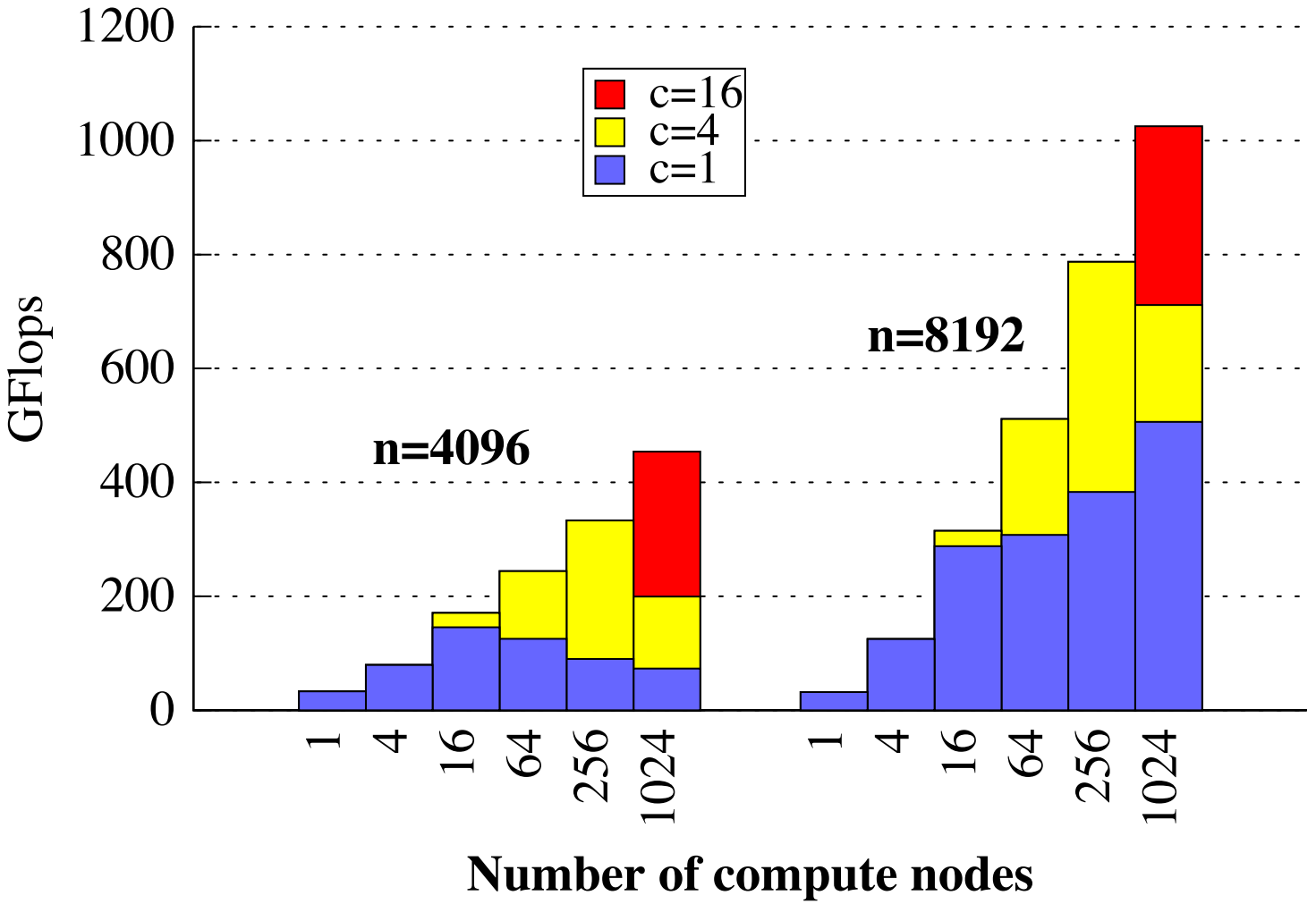
```
A = A*;      % recursive call
B = AB;  C = CA;
D = D + CB;
D = D*;      % recursive call
B = BD;  C = DC;
A = A + BC;
```

Using the "right" recursion and 2.5D communication avoiding matrix multiplication (with c replicas):

$$\text{Bandwidth}=O(n^2/\sqrt{cp}) \qquad \text{Latency}=O\left(\sqrt{cp}\log^2(p)\right)$$

Communication-avoiding APSP on distributed memory

65K vertex dense problem solved in about two minutes



c: replication in 2.5D semiring matrix multiply

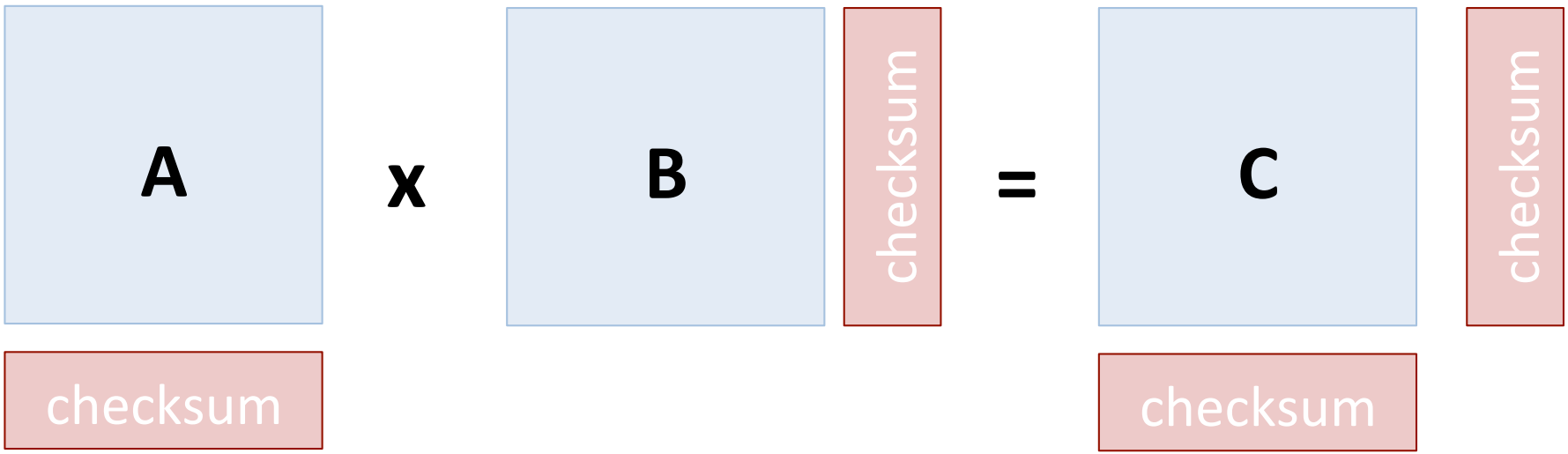
Jaguar (Titan without GPUs) at ORNL.

Nodes have 24 cores each

Solomonik, **B.**, and J. Demmel. "Minimizing communication in all-pairs shortest paths", IPDPS. 2013.

Fault Tolerance of Linear Algebraic Graph Algorithms

Literature exists on fault tolerant linear algebra operations.
Overhead: O(dN) to detect/correct O(d) errors on N-by-N matrices

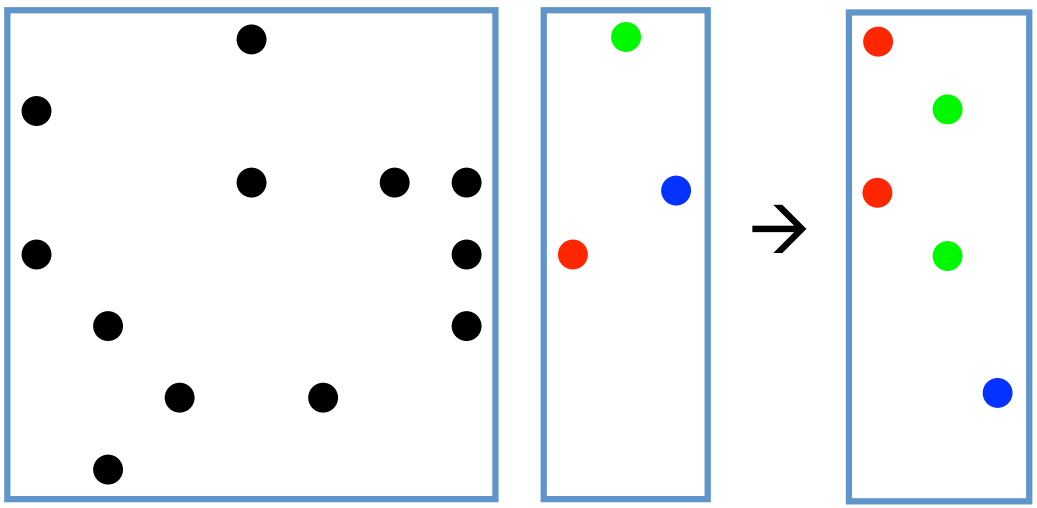


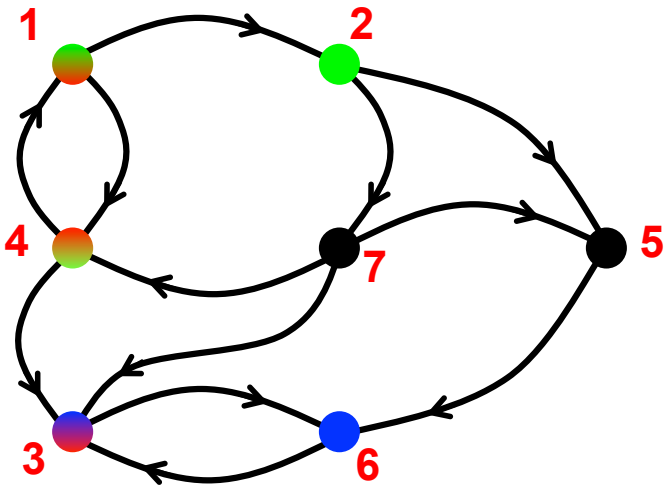
Good news: Overhead can often be tolerated in sparse matrix-matrix operations for graph algorithms.

Unknown: Techniques are for fields/rings, how do they apply to semiring algebra?

Extreme Concurrency/Parallelism

Linear algebra is the right abstraction for exploiting multiple levels of parallelism available in many graph algorithms





Encapsulates <u>three level of parallelism:</u>

1. columns(B): multiple BFS searches in parallel
2. columns(A^T)+rows(B): parallel over frontier vertices in each BFS
3. rows(A^T): parallel over incident edges of each frontier vertex

B. and J.R. Gilbert. The Combinatorial BLAS: Design, implementation, and applications. IJHPCA, 2011.

Conclusions

- We believe the state of the art for “Graphs in the language of Linear algebra” is mature enough to define a common set of building blocks
- Linear algebra did not solve its exascale challenges yet, but it is not clueless either.
- All other graph abstractions (think like a vertex, gather-apply-scatter, visitors) are clueless/ignorant about addressing exascale challenges.
- If graph algorithms ever scales to exascale, it will most likely be in the language of linear algebra.
- Come join our next event at HPEC’14

Acknowledgments

- Ariful Azad (LBL)
- Grey Ballard (UCB)
- Jarrod Chapman (JGI)
- Jim Demmel (UC Berkeley)
- John Gilbert (UCSB)
- Evangelos Georganas (UCB)
- Laura Grigori (INRIA)
- Ben Lipshitz (UCB)
- Adam Lugowski (UCSB)
- Sang-Yuh Oh (LBL)
- Lenny Oliker (Berkeley Lab)
- Steve Reinhardt (Cray)
- Dan Rokhsar (JGI/UCB)
- Oded Schwartz (UCB)
- Harsha Simhadri (LBL)
- Edgar Solomonik (UCB)
- Veronika Strnadova (UCSB)
- Sivan Toledo (Tel Aviv Univ)
- Dani Ushizima (Berkeley Lab)
- Kathy Yelick (Berkeley Lab/UCB)

My work is funded by:

Today: In class work

- Construct matrix model from graph
- Use matrix model to compute parallel breath-first search as $x = A^T x$
- Compare to native graph implementations

Blank code and data available on website (Lecture 22)

`www.cs.rpi.edu/~slotag/classes/FA16/index.html`