Graphs in the Language of Linear Algebra: Applications, Software, and Challenges

John R. Gilbert
University of California, Santa Barbara

Graph Algorithm Building Blocks
May 19, 2014

Support: Intel, Microsoft, DOE Office of Science, NSF
Thanks …

Lucas Bang (UCSB), Jon Berry (Sandia), Eric Boman (Sandia), Aydin Buluc (LBL), John Conroy (CCS), Kevin Deweese (UCSB), Erika Duriakova (Dublin), Armando Fox (UCB), Shoaib Kamil (MIT), Jeremy Kepner (MIT), Tristan Konolige (UCSB), Adam Lugowski (UCSB), Tim Mattson (Intel), Brad McRae (TNC), Dave Mizell (YarcData), Lenny Oliker (LBL), Carey Priebe (JHU), Steve Reinhardt (YarcData), Lijie Ren (Google), Eric Robinson (Lincoln), Viral Shah (UIDAI), Veronika Strnadova (UCSB), Yun Teng (UCSB), Joshua Vogelstein (Duke), Drew Waranas (UCSB), Sam Williams (LBL)
Outline

• A few sample applications
• Sparse matrices for graph algorithms
• Software: CombBLAS, KDT, QuadMat
• Challenges, issues, and questions
Problem: scale to millions of markers times thousands of individuals, with “unknown” rates > 50%

Tools used or desired: spanning trees, approximate TSP, incremental connected components, spectral and custom clustering, k-nearest neighbors

Results: using more data gives better genomic maps
Alignment and matching of brain scans
[Conroy, G, Kratzer, Lyzinski, Priebe, Vogelstein 2014]

- Problem: match functional regions across individuals
- Tools: Laplacian eigenvectors, geometric spectral partitioning, clustering, and more. . .
Landscape connectivity modeling

[Habitat quality, gene flow, corridor identification, conservation planning]

- Targeting larger problems: Yellowstone-to-Yukon corridor

- Tools: Graph contraction, connected components, Laplacian linear systems
Combinatorial acceleration of Laplacian solvers
[Boman, Deweese, G 2014]

\[(B^{-1/2} A B^{-1/2}) (B^{1/2} x) = B^{-1/2} b\]

- Problem: approximate target graph by sparse subgraph
- \(Ax = b\) in nearly linear time in theory [ST08, KMP10, KOSZ13]
- Tools: spanning trees, subgraph extraction and contraction, breadth-first search, shortest paths, . . .
The middleware challenge for graph analysis

Continuous physical modeling

- Linear algebra
- Computers

Discrete structure analysis

- Graph theory
- Computers
The middleware challenge for graph analysis

- By analogy to numerical scientific computing...

- What should the combinatorial BLAS look like?

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

$C = A \times B$

$y = A \times x$

$\mu = x^T y$
Sparse array primitives for graph manipulation

Sparse matrix-matrix multiplication (SpGEMM)

Element-wise operations

Sparse matrix-dense vector multiplication

Sparse matrix indexing

Matrices over various semirings: (+ . x), (min . +), (or . and), …
Examples of semirings in graph algorithms

<table>
<thead>
<tr>
<th>Real field: ((\mathbb{R}, +, \times))</th>
<th>Classical numerical linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean algebra: (({0, 1},</td>
<td>, &amp;))</td>
</tr>
<tr>
<td>Tropical semiring: ((\mathbb{R} \cup {\infty}, \min, +))</td>
<td>Shortest paths</td>
</tr>
<tr>
<td>((S, \text{select, select}))</td>
<td>Select subgraph, or contract nodes to form quotient graph</td>
</tr>
<tr>
<td>(edge/vertex attributes, vertex data aggregation, edge data processing)</td>
<td>Schema for user-specified computation at vertices and edges</td>
</tr>
</tbody>
</table>
Multiple-source breadth-first search
Multiple-source breadth-first search

- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
Graph contraction via sparse triple product

Contract

1 5 2 3 4 6

A1

A2

A3

1 2 3 4 5 6

x

1 2 3 4 5 6

x

1 1
1 1
1 1
1
1
1

=
Subgraph extraction via sparse triple product

\[ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} \]
Counting triangles (clustering coefficient)

Clustering coefficient:
- $\Pr$ (wedge $i$-$j$-$k$ makes a triangle with edge $i$-$k$)
- $3 \times \frac{\# \text{ triangles}}{\# \text{ wedges}}$
- $3 \times \frac{4}{19} = 0.63$ in example
- may want to compute for each vertex $j$
Counting triangles (clustering coefficient)

Clustering coefficient:
• $Pr \text{ (wedge i-j-k makes a triangle with edge i-k)}$
• $3 * \frac{\# \text{ triangles}}{\# \text{ wedges}}$
• $3 * \frac{4}{19} = 0.63$ in example
• may want to compute for each vertex $j$

Inefficient way to count triangles with matrices:
• $A = \text{adjacency matrix}$
• $\# \text{ triangles} = \text{trace}(A^3) / 6$
• but $A^3$ is likely to be pretty dense
Counting triangles (clustering coefficient)

Clustering coefficient:
- \( \Pr \) (wedge \( i-j-k \) makes a triangle with edge \( i-k \))
- \( 3 \times \frac{\# \text{ triangles}}{\# \text{ wedges}} \)
- \( 3 \times 4 / 19 = 0.63 \) in example
- may want to compute for each vertex \( j \)

Cohen’s algorithm to count triangles:
- Count triangles by lowest-degree vertex.
- Enumerate “low-hinged” wedges.
- Keep wedges that close.
Counting triangles (clustering coefficient)

A = L + U  (hi→lo + lo→hi)
L × U = B  (wedge, low hinge)
A ∨ B = C  (closed wedge)

\[ \text{sum}(C)/2 = 4 \text{ triangles} \]
A few other graph algorithms we’ve implemented in linear algebraic style

- Maximal independent set (KDT/SEJITS) [BDFGKLOW 2013]
- Peer-pressure clustering (SPARQL) [DGLMR 2013]
- Time-dependent shortest paths (CombBLAS) [Ren 2012]
- Gaussian belief propagation (KDT) [LABGRTW 2011]
- Markoff clustering (CombBLAS, KDT) [BG 2011, LABGRTW 2011]
- Betweenness centrality (CombBLAS) [BG 2011]
- Hybrid BFS/bully connected components (CombBLAS) [Konolige, in progress]
- Geometric mesh partitioning (Matlab 😊) [GMT 1998]
Graph algorithms in the language of linear algebra

- Kepner et al. study [2006]: fundamental graph algorithms including min spanning tree, shortest paths, independent set, max flow, clustering, …

- SSCA#2 / centrality [2008]

- Basic breadth-first search / Graph500 [2010]

- Beamer et al. [2013] direction-optimizing breadth-first search, implemented in CombBLAS
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.
- Scalable performance from laptop to 100,000-processor HPC.
- Open source software.
- Version 1.4.0 released January 16, 2014.
### Some Combinatorial BLAS functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Math Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpGEMM</td>
<td>- sparse matrices $A$ and $B$</td>
<td>sparse matrix</td>
<td>$C = \text{op}(A) \times \text{op}(B)$</td>
</tr>
<tr>
<td></td>
<td>- unary functors (op)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpM{Sp}V</td>
<td>- sparse matrix $A$</td>
<td>sparse/dense vector</td>
<td>$y = A \times x$</td>
</tr>
<tr>
<td>(Sp: sparse)</td>
<td>- sparse/dense vector $x$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpEWiseX</td>
<td>- sparse matrices or vectors</td>
<td>in place or sparse</td>
<td>$C = A \times B$</td>
</tr>
<tr>
<td></td>
<td>- binary functor and predicate</td>
<td>matrix/vector</td>
<td></td>
</tr>
<tr>
<td>Reduce</td>
<td>- sparse matrix $A$ and functors</td>
<td>dense vector</td>
<td>$y = \text{sum}(A, \text{op})$</td>
</tr>
<tr>
<td>SpRef</td>
<td>- sparse matrix $A$</td>
<td>sparse matrix</td>
<td>$B = A(p,q)$</td>
</tr>
<tr>
<td></td>
<td>- index vectors $p$ and $q$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpAsgn</td>
<td>- sparse matrices $A$ and $B$</td>
<td>none</td>
<td>$A(p,q) = B$</td>
</tr>
<tr>
<td></td>
<td>- index vectors $p$ and $q$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>- sparse matrix $A$</td>
<td>none</td>
<td>check manual</td>
</tr>
<tr>
<td></td>
<td>- dense matrix or vector $X$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apply</td>
<td>- any matrix or vector $X$</td>
<td>none</td>
<td>$\text{op}(X)$</td>
</tr>
<tr>
<td></td>
<td>- unary functor (op)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Combinatorial BLAS: Distributed-memory reference implementation

Combinatorial BLAS functions and operators

DistMat ➔ CommGrid ➔ FullyDistVec

... HAS A

DenseDistMat ➔ SpDistMat ➔ SpMat ➔ SpDistVec ➔ DenseDistVec

Polymorphism

Enforces interface only

DCSC  CSC  Triples  CSB
Matrix/vector distributions, interleaved on each other.

Default distribution in **Combinatorial BLAS**.

Scalable with increasing number of processes

- 2D matrix layout wins over 1D with large core counts and with limited bandwidth/compute
- 2D vector layout sometimes important for load balance
Combinatorial BLAS “users” (Sep 2013)

- IBM (T.J. Watson, Zurich, & Tokyo)
- Microsoft
- Intel
- Cray
- Stanford
- UC Berkeley
- Carnegie-Mellon
- Georgia Tech
- Ohio State
- Columbia
- U Minnesota

- King Fahd U
- Tokyo Inst of Technology
- Chinese Academy of Sciences
- U Ghent (Belgium)
- Bilkent U (Turkey)
- U Canterbury (New Zealand)
- Purdue
- Indiana U
- Mississippi State
- UC Merced
QuadMat shared-memory data structure

[Lugowski, G]

subdivide by dimension on power of 2 indices

Blocks store enough matrix elements for meaningful computation; denser parts of matrix have more blocks.
QuadMat example: Scale-10 RMAT

Scale 10 RMAT
(887x887, 21304 non-nulls)
up to 1024 non-nulls per block
In order of increasing degree

Blue blocks: uint16_t indices
Green blocks: uint8_t indices
- **Problem**: Natural recursive matrix multiplication is inefficient due to deep tree of sparse matrix additions.

- **Solution**: Rearrange into block inner product *pair lists*.

- A single matrix element can participate in pair lists with different block sizes.

- Symbolic phase followed by computational phase

- Multithreaded implementation in Intel TBB
QuadMat compared to Csparse & CombBLAS

![Graph showing speedup compared to Csparse for different datasets. The x-axis represents different datasets, including ER_18_sq, ER_20_sq, rmat_16_sq, rmat_16RP_sq, rmat_18_sq, rmat_18RP_sq, torus3D_150_sq, torus3D_160RP_sq, torus3D_200_sq, and torus3D_200RP_sq. The y-axis represents the speedup compared to Csparse on a logarithmic scale. The graph compares QuadMat and CombBLAS against Csparse.]

- **CSparse**: Dashed line
- **QuadMat**: Green line
- **CombBLAS**: Orange line
Knowledge Discovery Toolbox
http://kdt.sourceforge.net/

- Aimed at domain experts who know their problem well but don’t know how to program a supercomputer
- Easy-to-use Python interface
- Runs on a laptop as well as a cluster with 10,000 processors
- Open source software (New BSD license)
- V3 release April 2013 (V4 soon)

A general graph library with operations based on linear algebraic primitives
Attributed semantic graphs and filters

Example:

- Vertex types: Person, Phone, Camera, Gene, Pathway
- Edge types: PhoneCall, TextMessage, CoLocation, SequenceSimilarity
- Edge attributes: Time, Duration

- Calculate centrality just for emails among engineers sent between given start and end times

```python
def onlyEngineers (self):
    return self.position == Engineer

def timedEmail (self, sTime, eTime):
    return ((self.type == email) and
            (self.Time > sTime) and
            (self.Time < eTime))

G.addVFilter(onlyEngineers)
G.addEFilter(timedEmail(start, end))

# rank via centrality based on recent email transactions among engineers
bc = G.rank(‘approxBC’)```
SEJITS for filter/semiring acceleration

Embedded DSL: Python for the whole application
• Introspect, translate Python to equivalent C++ code
• Call compiled/optimized C++ instead of Python
Filtered BFS with SEJITS

Time (in seconds) for a single BFS iteration on scale 25 RMAT (33M vertices, 500M edges) with 10% of elements passing filter. Machine is NERSC’s Hopper.
What do we wish we had?

• Laplacian linear solvers and eigensolvers
  – Many applications: spectral clustering, ranking, partitioning, multicommodity flow, PDE’s, control theory, ....

• Fusing sequences of operations instead of materializing intermediate results
  – Working on some of this, e.g. matrix triple products in QuadMat

• Priority-queue algorithms: depth-first search, Dijkstra’s shortest paths, strongly connected components
  – These are hard to do in parallel at all
  – But sometimes you want to do them sequentially
A few questions for the Graph BLAS Forum

- How (or when) does the API let the user specify the “semiring scalar” objects and operations?
  - How general can the objects be?
  - What guarantees do the operations have to make?
  - Maybe there are different levels of compliance for an implementation, starting with just (double, +, *)
A few questions for the Graph BLAS Forum

• How does the API let the user “break out of the BLAS” when they need to?
  – In dense numeric BLAS and in sparse Matlab (but not in Sparse BLAS), the user can access the matrix directly, element-by-element, with a performance penalty.
  – Graph BLAS needs something like this too, or else it’s only useful to programmers who commit to it 100%.
  – “for each edge e incident on vertex v do …”
  – “for each endpoint v of edge e do …”
  – Add or delete vertex v or edge e.
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.