Sparse Matrices for High Performance Graph Analytics

John R. Gilbert
University of California, Santa Barbara

Washington State University
February 4, 2016

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

Ariful Azad, David Bader, Lucas Bang, Jon Berry, Eric Boman, Aydin Buluc, John Conroy, Kevin Deweese, Erika Duriakova, Armando Fox, Joey Gonzalez, Shoaib Kamil, Jeremy Kepner, Tristan Konolige, Manoj Kumar, Adam Lugowski, Tim Mattson, Brad McRae, Henning Meyerhenke, Dave Mizell, Jose Moreira, Lenny Oliker, Carey Priebe, Steve Reinhardt, Lijie Ren, Eric Robinson, Viral Shah, Veronika Strnadova-Neely, Yun Teng, Joshua Vogelstein, Drew Waranis, Sam Williams
Outline

• Motivation: Graph applications
• Mathematics: Sparse matrices for graph algorithms
• Software: CombBLAS, KDT, QuadMat
• Standards: The Graph BLAS effort
“I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were ‘sparse’ in the matrix, and that typically the triangular matrices associated with the forward and back solution provided by Gaussian elimination would remain sparse if pivot elements were chosen with care”

- Harry Markowitz, describing the 1950s work on portfolio theory that won the 1990 Nobel Prize for Economics
Graphs and sparse matrices: Cholesky factorization

\[ A = LL^T \]

**Symmetric Gaussian elimination:**

for \( j = 1 \) to \( n \)

add edges between \( j \)’s higher-numbered neighbors

**Fill:** new nonzeros in factor
Computational models of the physical world

- Cortical bone
- Trabecular bone
Large graphs are everywhere...

- Internet structure
- Social interactions
- Scientific datasets: biological, chemical, cosmological, ecological, ...

WWW snapshot, courtesy Y. Hyun

Yeast protein interaction network, courtesy H. Jeong
Social network analysis (1993)

Co-author graph from 1993 Householder symposium
Social network analysis (2016)

Facebook graph:
> 1,500,000,000 vertices
The middleware challenge for graph analysis

- Continuous physical modeling
  - Linear algebra
  - Computers

- Discrete structure analysis
  - Graph theory
  - Computers
Top 500 List (November 2015)

**Top500 Benchmark:**
Solve a large system of linear equations by Gaussian elimination

\[
P A = L X U
\]
Graph500 Benchmark:
Breadth-first search in a large power-law graph

<table>
<thead>
<tr>
<th>No.</th>
<th>Rank</th>
<th>Machine</th>
<th>Installation Site</th>
<th>Number of nodes</th>
<th>Number of cores</th>
<th>Problem scale</th>
<th>GTEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>K computer (Fujitsu Custom)</td>
<td>RIKEN Advanced Institute for Computational Science (AICS)</td>
<td>82944</td>
<td>663552</td>
<td>40</td>
<td>38621.4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>DOE/NNSA/LLNL Sequoia (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>Lawrence Livermore National Laboratory</td>
<td>98304</td>
<td>1572864</td>
<td>41</td>
<td>23751</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>DOE/SC/Argonne National Laboratory Mira (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>Argonne National Laboratory</td>
<td>49152</td>
<td>786432</td>
<td>40</td>
<td>14982</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>JUQUEEN (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>Forschungszentrum Juelich (FZJ)</td>
<td>16384</td>
<td>262144</td>
<td>38</td>
<td>5848</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Fermi (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>CINECA</td>
<td>8192</td>
<td>131072</td>
<td>37</td>
<td>2567</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Tianhe-2 (MilkyWay-2) (National University of Defense Technology - MPP)</td>
<td>Changsha, China</td>
<td>8192</td>
<td>196608</td>
<td>36</td>
<td>2061.48</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>Turing (IBM - BlueGene/Q, Power BQC 16C 1.60GHz)</td>
<td>CNRS/IDRIS-GENCI</td>
<td>4096</td>
<td>65536</td>
<td>36</td>
<td>1427</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>Blue Joule (IBM - BlueGene/Q, Power BQC 16C 1.60GHz)</td>
<td>Science and Technology Facilities Council - Daresbury Laboratory</td>
<td>4096</td>
<td>65536</td>
<td>36</td>
<td>1427</td>
</tr>
</tbody>
</table>
\[ PA = LU \]

34 Petaflops

38 Terateps

34 Peta / 38 Tera is about 900.
Floating-Point vs. Graphs, November 2015

34 Petaflops

\[ P A = L x U \]

38 Terateps

Nov 2015: 34 Peta / 38 Tera \( \sim 900 \)

Nov 2010: 2.5 Peta / 6.6 Giga \( \sim 380,000 \)
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*B

y = A*x

μ = xᵀ y
Coarse-grained parallelism can be exploited by abstractions at the right level.

<table>
<thead>
<tr>
<th>Vertex/edge graph computations</th>
<th>Graphs in the language of linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpredictable, data-driven communication patterns</td>
<td>Fixed communication patterns</td>
</tr>
<tr>
<td>Irregular data accesses, with poor locality</td>
<td>Matrix block operations exploit memory hierarchy</td>
</tr>
<tr>
<td>Fine grained data accesses, dominated by latency</td>
<td>Coarse grained parallelism, limited by bandwidth not latency</td>
</tr>
</tbody>
</table>
Sparse array primitives for graphs

Sparse matrix-sparse matrix multiplication

Sparse matrix-sparse vector multiplication

Element-wise operations

Sparse matrix indexing

Matrices over various semirings: $(+, \times), (\text{and, or}), (\text{min, +}), ...$
Multiple-source breadth-first search

AT

B
Multiple-source breadth-first search

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

<table>
<thead>
<tr>
<th>( “values”: edge/vertex attributes, “add”: vertex data aggregation, “multiply”: edge data processing )</th>
<th>General schema for user-specified computation at vertices and edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real field: (R, +, *)</td>
<td>Numerical linear algebra</td>
</tr>
<tr>
<td>Boolean algebra: ({0 1},</td>
<td>, &amp;)</td>
</tr>
<tr>
<td>Tropical semiring: (R U {∞}, min, +)</td>
<td>Shortest paths</td>
</tr>
<tr>
<td>(S, select, select)</td>
<td>Select subgraph, or contract nodes to form quotient graph</td>
</tr>
</tbody>
</table>
Graph contraction via sparse triple product

A1

A2

A3

Contract

A1

A2

A3

1 1 1
1 1 1
1 1 1

1 1 1
1 1 1
1 1 1

X

1 2 3 4 5 6

1 2 3 4 5 6

X

1 1
1 1
1 1

=
Subgraph extraction via sparse triple product
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)

Cluster coefficient:
- $\text{Pr (wedge } i-j-k \text{ makes a triangle with edge } i-k)$
- $3 \times \# \text{ triangles} / \# \text{ wedges}$
- $3 \times \frac{4}{19} = 0.63$ 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)

\[ A = L + U \quad \text{(hi->lo + lo->hi)} \]
\[ L \times U = B \quad \text{(wedge, low hinge)} \]
\[ A \land B = C \quad \text{(closed wedge)} \]
\[ \text{sum}(C)/2 = 4 \text{ triangles} \]
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 1.5.0 released February 2016.
Combinatorial BLAS in distributed memory

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
Benchmarks graph analytics frameworks

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

Satish et al. "Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets", in SIGMOD’14
Combinatorial BLAS “users” (Jan 2016)

- IBM (T.J.Watson, Zurich, & Tokyo)
- Intel
- Cray
- Microsoft
- Stanford
- MIT
- UC Berkeley
- Carnegie-Mellon
- Georgia Tech
- Purdue
- Ohio State
- U Texas Austin
- NC State
- UC San Diego
- UC Merced
- UC Santa Barbara
- Berkeley Lab
- Sandia Labs
- Columbia
- U Minnesota
- Duke
- Indiana U
- Mississippi State
- SEI
- Paradigm4
- Mellanox
- IHPC (Singapore)
- Tokyo Inst of Technology
- Chinese Academy of Sciences
- U Canterbury (New Zealand)
- King Fahd U (Saudi Arabia)
- Bilkent U (Turkey)
- U Ghent (Belgium)
Knowledge Discovery Toolbox
http://kdt.sourceforge.net/

A general graph library with operations based on linear algebraic primitives

• 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)
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.
A few other graph algorithms we’ve implemented in linear algebraic style

- Bipartite maximal & max cardinality matching (CombBLAS) [ABP 2015]
- 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]
- 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
The Basic Linear Algebra Subroutines had a revolutionary impact on computational linear algebra.

<table>
<thead>
<tr>
<th>BLAS 1</th>
<th>vector ops</th>
<th>Lawson, Hanson, Kincaid, Krogh, 1979</th>
<th>LINPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS 2</td>
<td>matrix-vector ops</td>
<td>Dongarra, Du Croz, Hammarling, Hanson, 1988</td>
<td>LINPACK on vector machines</td>
</tr>
<tr>
<td>BLAS 3</td>
<td>matrix-matrix ops</td>
<td>Dongarra, Du Croz, Duff, Hammarling, 1990</td>
<td>LAPACK on cache based machines</td>
</tr>
</tbody>
</table>

- Experts in mapping algorithms to hardware tune BLAS for specific platforms.
- Experts in numerical linear algebra build software on top of the BLAS to get high performance “for free.”

Today every computer, phone, etc. comes with /usr/lib/libblas
Can we standardize a “Graph BLAS”?
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 universal 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.
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.
### Some Graph BLAS basic functions
(names not final)

<table>
<thead>
<tr>
<th>Function (CombBLAS equiv)</th>
<th>Parameters</th>
<th>Returns</th>
<th>Matlab notation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>matmul</code> (SpGEMM)</td>
<td>- sparse matrices A and B</td>
<td>sparse matrix</td>
<td>C = A * B</td>
</tr>
<tr>
<td></td>
<td>- optional unary functs</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>matvec</code> (SpM{Sp}V)</td>
<td>- sparse matrix A</td>
<td>sparse/dense vector</td>
<td>y = A * x</td>
</tr>
<tr>
<td></td>
<td>- sparse/dense vector x</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ewisemult, add, ...</code></td>
<td>- sparse matrices or vectors</td>
<td>in place or sparse matrix/vector</td>
<td>C = A .* B</td>
</tr>
<tr>
<td>(SpE WiseX)</td>
<td>- binary funct, optional unarys</td>
<td></td>
<td>C = A + B</td>
</tr>
<tr>
<td><code>reduce</code> (Reduce)</td>
<td>- sparse matrix A and funct</td>
<td>dense vector</td>
<td>y = sum(A, op)</td>
</tr>
<tr>
<td><code>extract</code> (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><code>assign</code> (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><code>buildMatrix</code> (Sparse)</td>
<td>- list of edges/triples (i, j, v)</td>
<td>sparse matrix</td>
<td>A = sparse(i, j, v, m, n)</td>
</tr>
<tr>
<td><code>getTuples</code> (Find)</td>
<td>- sparse matrix A</td>
<td>edge list</td>
<td>[i, j, v] = find(A)</td>
</tr>
</tbody>
</table>
Matrix times matrix over semiring

Inputs
matrix \( A \): \( S^{M \times N} \) (sparse or dense)
matrix \( B \): \( S^{N \times L} \) (sparse or dense)

Optional Inputs
matrix \( C \): \( S^{M \times L} \) (sparse or dense)
scalar “add” function \( \oplus \)
scalar “multiply” function \( \otimes \)
transpose flags for \( A, B, C \)

Outputs
matrix \( C \): \( S^{M \times L} \) (sparse or dense)

Implements
\[
C \oplus= A \oplus \odot B
\]
for \( j = 1 : N \)
\[
C(i,k) = C(i,k) \oplus (A(i,j) \otimes B(j,k))
\]
If input \( C \) is omitted, implements
\[
C = A \oplus \odot B
\]

Transpose flags specify operation on \( A^T, B^T \), and/or \( C^T \) instead

Notes
\( S \) is the set of scalars, user-specified
\( S \) 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
Breadth-first search with (draft) Graph BLAS

```c
7 GrB_info BFS(GrB_Vector *v, GrB_Matrix A, GrB_index s)
8 /*
9 * Given a boolean n x n adjacency matrix A and a source vertex s, performs a BFS traversal
10 * of the graph and sets v[i] to the level in which vertex i is visited (v[s] == 1).
11 * If i is not reachable from s, then v[i] = 0. (Vector v should be empty on input.)
12 */
13 {
14    GrB_index n;
15    GrB.Matrix.nrows(&n, A); // n = # of rows of A
16
17    GrB.Vector_new(v, GrB.INT32, n); // Vector<int32_t> v(n)
18    GrB.assign(v, 0); // v = 0
19
20    GrB.Vector q; // vertices visited in each level
21    GrB.Vector_new(&q, GrB.BOOL, n); // Vector<bool> q(n)
22    GrB.assign(&q, false);
23    GrB.assign(&q, true, s); // q[s] = true, false everywhere else
24
25    GrB.Space Boolean;
26    GrB.Space_new(&Boolean, GrB.BOOL, GrB.BOOL, GrB.BOOL, GrB.BOOL, GrB.LOR, GrB.LAND, false, true);
27
28    GrB.Descriptor desc; // Descriptor for vxm
29    GrB.Descriptor_new(&desc);
30    GrB.Descriptor_add(desc, GrB.ARG1, GrB.NOP); // no operation on the vector
31    GrB.Descriptor_add(desc, GrB.ARG2, GrB.NOP); // no operation on the matrix
32    GrB.Descriptor_add(desc, GrB.MASK, GrB.LNOT); // invert the mask
33
34    /* BFS traversal and label the vertices. */
35    int32_t d = 1;
36    bool succ = false;
37    do {
38        GrB.assign(v, d, q);
39        GrB.vxm(&q, Boolean,q, A,*v, desc);
40        GrB_reduce(&succ, q, GrB.LOR);
41        d++;
42    } while (succ);
43
44    GrB.free(q);
45    GrB.free(Boolean);
46    GrB.free(desc);
47
48    return GrB_SUCCESS;
49 }
```
Conclusion

• Matrix computation is beginning to repay a 50-year debt to graph algorithms.

• Graphs in the language of linear algebra are sufficiently mature to support a standard set of BLAS.

• It helps to look at things from two directions.
Thank You!

Continuous physical modeling

Discrete structure analysis

Linear algebra & graph theory

Computers

http://graphblas.org