Sparse Matrices for High-Performance Graph Analytics

John R. Gilbert
University of California, Santa Barbara

Chesapeake Large-Scale Analytics Conference
October 17, 2013

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

Aydin Buluc (LBL), Kevin Deweese (UCSB),
Erika Duriakova (Dublin), Armando Fox (UCB),
Shoaib Kamil (MIT), Jeremy Kepner (MIT),
Adam Lugowski (UCSB), Tim Mattson (Intel),
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 Waranis (UCSB),
Sam Williams (LBL)
Outline

• Motivation
• Sparse matrices for graph algorithms
• CombBLAS: sparse arrays and graphs on parallel machines
• KDT: attributed semantic graphs in a high-level language
• Standards for graph algorithm primitives
A few biological graph analysis problems

- **Connective abnormalities in schizophrenia** [van den Heuvel et al.]
  - Problem: understand disease from anatomical brain imaging
  - Tools: betweenness centrality, shortest path length
  - Results: global statistics on connection graph correlate w/ diagnosis

- **Genomics for biofuels** [Strnadova et al.]
  - Problem: scale to millions of markers times thousands of individuals
  - Tools: min spanning tree, customized clustering
  - Results: using much more data leads to much better genomic maps

- **Alignment and matching of brain scans** [Vogelstein et al.]
  - Problem: match corresponding functional regions across individuals
  - Tools: graph partitioning, clustering, and more...
  - Results: in progress
The middleware of scientific computing

Continuous physical modeling

- Linear algebra

Computers

Discrete structure analysis

- Graph theory

Computers
Top 500 List (June 2013)

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

\[
P A = L X U
\]
Graph 500 List (June 2013)

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>DOE/NNSA/LLNL Sequoia (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>Lawrence Livermore National Laboratory</td>
<td>65536</td>
<td>1048576</td>
<td>40</td>
<td>15363</td>
</tr>
<tr>
<td>2</td>
<td>2</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>785432</td>
<td>40</td>
<td>14328</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>JUQUEEN (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>Forschungszentrum Juelich (FZI)</td>
<td>16384</td>
<td>262144</td>
<td>38</td>
<td>5848</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>K computer (Fujitsu - Custom supercomputer)</td>
<td>RIKEN Advanced Institute for Computational Science (AICS)</td>
<td>65536</td>
<td>524288</td>
<td>40</td>
<td>5524.12</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>196508</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 1.60 GHz)</td>
<td>Science and Technology Facilities Council - Daresbury Laboratory</td>
<td>4096</td>
<td>65536</td>
<td>36</td>
<td>1427</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>DIRAC (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)</td>
<td>University of Edinburgh</td>
<td>4096</td>
<td>65536</td>
<td>36</td>
<td>1427</td>
</tr>
</tbody>
</table>
33.8 Petaflops

\[ \text{PA} = \text{LU} \]

15.3 Terateps

33.8 Peta / 15.3 Tera is about 2200.
33.8 Petaflops

P A = L x U

15.3 Terateps

Jun 2013: 33.8 Peta / 15.3 Tera ~ 2,200
Nov 2010: 2.5 Peta / 6.6 Giga ~ 380,000
The challenge of the software stack

- By analogy to numerical scientific computing...

- What should the combinatorial BLAS look like?

**Basic Linear Algebra Subroutines (BLAS):**

```
C = A*B
y = A*x
μ = x^T y
```

<table>
<thead>
<tr>
<th>Speed in Megalops</th>
<th>Order of vectors/matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>600</td>
</tr>
<tr>
<td>250</td>
<td>500</td>
</tr>
<tr>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>150</td>
<td>300</td>
</tr>
<tr>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

UCSB
Outline

• Motivation
• Sparse matrices for graph algorithms
• CombBLAS: sparse arrays and graphs on parallel machines
• KDT: attributed semantic graphs in a high-level language
• Standards for graph algorithm primitives
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

A1

A2

A3

\[ \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & & & & 1 & \\
2 & & & & 1 & \\
3 & & & & 1 & \\
\end{array} \]

\[ \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & \bullet & \bullet & \bullet & 1 & \\
2 & \bullet & \bullet & \bullet & \bullet & \\
3 & \bullet & \bullet & \bullet & \bullet & \\
4 & \bullet & \bullet & \bullet & \bullet & \\
5 & \bullet & \bullet & \bullet & \bullet & \\
6 & \bullet & \bullet & \bullet & \bullet & \\
\end{array} \]

\[ \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 1 & 1 & \\
2 & 1 & 1 & 1 & \\
3 & 1 & 1 & 1 & \\
\end{array} \]

= 

\[ \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\bullet & \bullet & \bullet & \bullet & \\
\bullet & \bullet & \bullet & \bullet & \\
\bullet & \bullet & \bullet & \bullet & \\
\bullet & \bullet & \bullet & \bullet & \\
\bullet & \bullet & \bullet & \bullet & \\
\bullet & \bullet & \bullet & \bullet & \\
\end{array} \]
Subgraph extraction via sparse triple product
Betweenness centrality [Robinson 2008]

\[ b = \text{BetweennessCentrality}(G = A : B^{N_V \times N_V}) \]

1. \( b = 0 \)
2. for \( 1 \leq r \leq N_V \)
   do
   4. \( d = 0 \)
   5. \( S = 0 \)
   6. \( p = 0, p_r = 1 \)
   7. \( f = a_{r,:} \)
   8. while \( f \neq 0 \)
      do
      10. \( d = d + 1 \)
      11. \( p = p + f \)
      12. \( s_{d,:} = f \)
      13. \( f = fA \times \neg p \)
      while \( d \geq 2 \)
         do
         16. \( w = s_{d,:} \times (1 + u) \div p \)
         17. \( w = Aw \)
         18. \( w = w \times s_{d-1,:} \times p \)
         19. \( u = u + w \)
         20. \( d = d - 1 \)
      end
      21. \( b = b + u \)

Variables:
- \( A \) : sparse adjacency matrix
- \( f \) : sparse fringe vector
- \( p \) : shortest path vector
- \( S \) : sparse depth matrix
- \( u \) : centrality update vector

Storage:
- \( B^{S(NxN)} \) : \( O(M+N) \)
- \( Z^{S(N)} \) : \( O(N) \)
- \( Z^{N} \) : \( O(N) \)
- \( B^{S(NxN)} \) : \( O(N) \)
- \( R^{N} \) : \( O(N) \)

Storage: \( O(M+N) \)
Time: \( O(MN + N^2) \)
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, using CombBLAS
Sparse array-based primitives

Sparse matrix-matrix multiplication (SpGEMM)

Element-wise operations

Sparse matrix-dense vector multiplication

Sparse matrix indexing

Matrices over various semirings: $(+ \cdot x)$, $(\text{min} \cdot +)$, $(\text{or} \cdot \text{and})$, ...
The case for sparse matrix graph primitives

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

<table>
<thead>
<tr>
<th>Traditional graph computations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data driven, unpredictable communication.</td>
</tr>
<tr>
<td>Irregular and unstructured, poor locality of reference</td>
</tr>
<tr>
<td>Fine grained data accesses, dominated by latency</td>
</tr>
</tbody>
</table>
Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

<table>
<thead>
<tr>
<th>Traditional graph computations</th>
<th>Graphs in the language of linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data driven, unpredictable communication.</td>
<td>Fixed communication patterns</td>
</tr>
<tr>
<td>Irregular and unstructured, poor locality of reference</td>
<td>Operations on matrix blocks exploit memory hierarchy</td>
</tr>
<tr>
<td>Fine grained data accesses, dominated by latency</td>
<td>Coarse grained parallelism, bandwidth limited</td>
</tr>
</tbody>
</table>
Matrices over semirings

- E.g. matrix multiplication \( C = AB \) (or matrix/vector):
  \[
  C_{i,j} = A_{i,1} \times B_{1,j} + A_{i,2} \times B_{2,j} + \cdots + A_{i,n} \times B_{n,j}
  \]

- Replace scalar operations \( \times \) and \( + \) by
  \( \otimes \) : associative, distributes over \( \oplus \)
  \( \oplus \) : associative, commutative

- Then \( C_{i,j} = A_{i,1} \otimes B_{1,j} \oplus A_{i,2} \otimes B_{2,j} \oplus \cdots \oplus A_{i,n} \otimes B_{n,j} \)

- Examples: \( \times, +; \text{and.or}; +, \text{min}; \ldots \)

- Same data reference pattern and control flow
### Examples of semirings in graph algorithms

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>((R, +, x)) Real Field</td>
<td>Standard numerical linear algebra</td>
</tr>
<tr>
<td>(({0,1},</td>
<td>, &amp;)) Boolean Semiring</td>
</tr>
<tr>
<td>((R U {\infty}, \min, +)) Tropical Semiring</td>
<td>Shortest paths</td>
</tr>
<tr>
<td>((R U {\infty}, \min, x))</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>
Outline

• Motivation

• Sparse matrices for graph algorithms

• CombBLAS: sparse arrays and graphs on parallel machines

• KDT: attributed semantic graphs in a high-level language

• Standards for graph algorithm primitives
### Combinatorial BLAS: Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Applies to</th>
<th>Parameters</th>
<th>Returns</th>
<th>Matlab Phrasing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SpGEMM</strong></td>
<td>Sparse Matrix (as friend)</td>
<td>( \mathbf{A}, \mathbf{B} ): sparse matrices&lt;br&gt;( \text{trA} ): transpose ( \mathbf{A} ) if true&lt;br&gt;( \text{trB} ): transpose ( \mathbf{B} ) if true</td>
<td>Sparse Matrix</td>
<td>( \mathbf{C} = \mathbf{A} \times \mathbf{B} )</td>
</tr>
<tr>
<td><strong>SpMV</strong></td>
<td>Sparse Matrix (as friend)</td>
<td>( \mathbf{A} ): sparse matrices&lt;br&gt;( \mathbf{x} ): sparse or dense vector(s)&lt;br&gt;( \text{trA} ): transpose ( \mathbf{A} ) if true</td>
<td>Sparse or Dense Vector(s)</td>
<td>( \mathbf{y} = \mathbf{A} \times \mathbf{x} )</td>
</tr>
<tr>
<td><strong>SpewiseX</strong></td>
<td>Sparse Matrices (as friend)</td>
<td>( \mathbf{A}, \mathbf{B} ): sparse matrices&lt;br&gt;( \text{notA} ): negate ( \mathbf{A} ) if true&lt;br&gt;( \text{notB} ): negate ( \mathbf{B} ) if true</td>
<td>Sparse Matrix</td>
<td>( \mathbf{C} = \mathbf{A} \times \mathbf{B} )</td>
</tr>
<tr>
<td><strong>Reduce</strong></td>
<td>Any Matrix (as method)</td>
<td>( \text{dim} ): dimension to reduce&lt;br&gt;( \text{binop} ): reduction operator</td>
<td>Dense Vector</td>
<td>\text{sum}(\mathbf{A})</td>
</tr>
<tr>
<td><strong>SpRef</strong></td>
<td>Sparse Matrix (as method)</td>
<td>( \mathbf{p} ): row indices vector&lt;br&gt;( \mathbf{q} ): column indices vector</td>
<td>Sparse Matrix</td>
<td>( \mathbf{B} = \mathbf{A}(\mathbf{p}, \mathbf{q}) )</td>
</tr>
<tr>
<td><strong>SpAsgn</strong></td>
<td>Sparse Matrix (as method)</td>
<td>( \mathbf{p} ): row indices vector&lt;br&gt;( \mathbf{q} ): column indices vector&lt;br&gt;( \mathbf{B} ): matrix to assign</td>
<td>None</td>
<td>( \mathbf{A}(\mathbf{p}, \mathbf{q}) = \mathbf{B} )</td>
</tr>
<tr>
<td><strong>Scale</strong></td>
<td>Any Matrix (as method)</td>
<td>( \text{rhs} ): any object (except a sparse matrix)</td>
<td>None</td>
<td>Check guiding principles 3 and 4</td>
</tr>
<tr>
<td><strong>Scale</strong></td>
<td>Any Vector (as method)</td>
<td>( \text{rhs} ): any vector</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td><strong>Apply</strong></td>
<td>Any Object (as method)</td>
<td>( \text{unop} ): unary operator (applied to non-zeros)</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
Combinatorial BLAS: Distributed-memory reference implementation

Combinatorial BLAS functions and operators

DistMat

CommGrid

FullyDistVec

DenseDistMat

SpDistMat

SpMat

SpDistVec

DenseDistVec

DCSC

CSC

Triples

CSB

Enforces interface only

Polymorphism
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
Parallel sparse matrix-matrix multiplication algorithm

\[ C_{ij} += \text{HyperSparseGEMM}(A^{\text{recv}}, B^{\text{recv}}) \]

2D algorithm: Sparse SUMMA (based on dense SUMMA)
General implementation that handles rectangular matrices
1D vs. 2D scaling for sparse matrix-matrix multiplication

(a) R-MAT $\times$ R-MAT product (scale 21).
(b) Multiplication of an R-MAT matrix of scale 23 with the restriction operator of order 8.

- 1-D data layout
- 2-D data layout (Combinatorial BLAS)
Almost linear scaling until bandwidth costs starts to dominate

Scaling proportional to $\sqrt{p}$ afterwards

$T_{comp} = O(n)$

$T_{comm} = O(n\sqrt{p})$
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
Outline

• Motivation
• Sparse matrices for graph algorithms
• CombBLAS: sparse arrays and graphs on parallel machines
• KDT: attributed semantic graphs in a high-level language
• Standards for graph algorithm primitives
Parallel graph analysis software

Discrete structure analysis

Graph theory

Computers
Parallel graph analysis software

Knowledge Discovery Toolbox (KDT)

Distributed Combinatorial BLAS

Shared-address space Combinatorial BLAS

Communication Support (MPI, GASNet, etc)

Threading Support (OpenMP, Cilk, etc)

Discrete structure analysis

Graph theory

Computers

• KDT is higher level (graph abstractions)
• Combinatorial BLAS is for performance
Domain expert vs. graph expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (e.g., \texttt{degree()} )
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank

- Hypergraphs and sparse matrices
- Graph primitives (e.g., \texttt{bfsTree()} )
- SpMV / SpGEMM on semirings
Domain expert vs. graph expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (e.g., degree, vectors)
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank
- Hypergraphs and sparse matrices
- Graph primitives (e.g., bfsTree())
- SpMV / SpGEMM on semirings

```python
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)
clusters = G.cluster('Markov')
clusNedge = G.nedge(clusters)
smallG = G.contract(clusters)
# visualize
```
(Semantic) directed graphs
- constructors, I/O
- basic graph metrics (e.g., degree)
- vectors

Clustering / components

Centrality / authority:
betweenness centrality, PageRank

Hypergraphs and sparse matrices

Graph primitives (e.g., bfsTree)

SpMV / SpGEMM on semirings

```python
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)
clusters = G.cluster('Markov')
clusNedge = G.nedge(clusters)
smallG = G.contract(clusters)

# visualize
```
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)
• V0.3 release April 2013
A few KDT applications

Markov Clustering (MCL) finds clusters by postulating that a random walk that visits a dense cluster will probably visit many of its vertices before leaving.

We use a Markov chain for the random walk. This process is reinforced by adding an inflation step that uses the Hadamard product and rescaling.

PageRank says a vertex is important if other important vertices link to it.

Each vertex (webpage) votes by splitting its PageRank score evenly among its out edges (links). This broadcast (an SpMV) is followed by a normalization step (ColWise). Repeat until convergence.

PageRank is the stationary distribution of a Markov Chain that simulates a "random surfer".

Betweenness Centrality says that a vertex is important if it appears on many shortest paths between other vertices. An exact computation requires a BFS for every vertex. A good approximation can be achieved by sampling starting vertices.

Gaussian belief propagation (GaBP) is an iterative algorithm for solving the linear system of equations Ax = b, where A is symmetric positive definite.

GaBP assumes each variable follows a normal distribution. It iteratively calculates the precision P and mean value μ of each variable; the converged mean-value vector approximates the actual solution.
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

Standard KDT

Python

KDT Algorithm

Filter (Py)

Semiring (Py)

C++

CombBLAS Primitive
SEJITS for filter/semiring acceleration

- Standard KDT
  - KDT Algorithm
  - Filter (Py)
  - Semiring (Py)
- KDT+SEJITS
  - KDT Algorithm
  - Filter (C++)
  - Semiring (C++)

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.
Outline

• Motivation
• Sparse matrices for graph algorithms
• CombBLAS: sparse arrays and graphs on parallel machines
• KDT: attributed semantic graphs in a high-level language
• Standards for graph algorithm primitives
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, Hammarling, Hanson, 1990</td>
<td>LAPACK on cache based machines</td>
</tr>
</tbody>
</table>

- Separation of concerns:
  - Experts in mapping algorithms onto hardware tuned BLAS to specific platforms.
  - Experts in linear algebra built software on top of the BLAS to obtain high performance “for free”.
- Today every computer, phone, etc. comes with /usr/lib/libblas
Can we define and standardize the “Graph BLAS”? 

- **No**, it is not reasonable to define a universal set of graph algorithm building blocks:
  - Huge diversity in matching algorithms to hardware platforms.
  - No consensus on data structures and 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 graph algorithm 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.
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 Mellon 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.
Conclusion

• It helps to look at things from two directions.

• Sparse arrays and matrices yield useful primitives and algorithms for high-performance graph computation.

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