The middleware of scientific computing

- Continuous physical modeling
  - Linear algebra
  - Computers

- Discrete structure analysis
  - Graph theory
  - Computers
The challenge of the software stack

• By analogy to numerical scientific computing...

• What should the combinatorial BLAS look like?
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

\[ A^T \]

\[ X \]
Multiple-source breadth-first search

\[ A^T \]

\[ X \]

\[ A^TX \]

\[ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 5 \]

\[ 6 \]
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
Subgraph extraction via sparse triple product

Extract

\[
\begin{array}{c|c|c|c|c|c}
1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 1 & 1 & & & \\
2 & & & & & \\
3 & & & & & \\
\end{array}
\times
\begin{array}{c|c|c|c|c|c}
1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\end{array}
\times
\begin{array}{c|c|c|}
1 & 1 & \\
\hline
1 & 1 & \\
\end{array}
= \\
\begin{array}{c|c|c|c|c|c}
1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\end{array}
\times
\begin{array}{c|c|c|c|c|c}
1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\end{array}
\times
\begin{array}{c|c|c|}
1 & 1 & \\
\hline
1 & 1 & \\
\end{array}
\]
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:**
- \( \text{Pr (wedge i-j-k makes a triangle with edge i-k)} \)
- \( 3 \times \frac{\# \text{ triangles}}{\# \text{ wedges}} \)
- \( 3 \times \frac{4}{19} = 0.63 \) in example
- may want to compute for each vertex \( j \)

**Inefficient way to count triangles with matrices:**
- \( A = \) adjacency matrix
- \# 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)
- $3 \times \frac{\# \text{ triangles}}{\# \text{ wedges}}$
- $3 \times \frac{4}{19} = 0.63$ 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)

\[ A = L + U \quad \text{(hi->lo + lo->hi)} \]

\[ L \times U = B \quad \text{(wedge, low hinge)} \]

\[ A \land B = C \quad \text{(closed wedge)} \]

\[ \text{sum}(C)/2 = 4 \text{ triangles} \]
Betweenness centrality \([\text{Robinson 2008}]\)

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

\begin{verbatim}
1 b = 0
2 for 1 \leq r \leq N_v 
3   do 
4       d = 0
5       S = 0
6       p = 0, p_r = 1
7       f = a_{r,:}
8       while f \neq 0
9          do 
10             d = d + 1
11             p = p + f
12             s_{d,:} = f
13             f = fA \times \neg p
14          while d \geq 2
15             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
21             b = b + u
\end{verbatim}
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
Sparse array-based primitives

Sparse matrix-matrix multiplication (SpGEMM)

Element-wise operations

Sparse matrix-dense vector multiplication

Sparse matrix indexing

Matrices over various semirings: \((+ \cdot x), (\text{min} \cdot +), (\text{or} \cdot \text{and}), \ldots\)
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: $\times$, $\oplus$; and, or; $\oplus$, min; . . .

- 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>$(\mathbb{R}, +, \times)$ Real Field</td>
<td>Standard numerical linear algebra</td>
</tr>
<tr>
<td>$({0,1},</td>
<td>, &amp;)$ Boolean Semiring</td>
</tr>
<tr>
<td>$(\mathbb{R} \cup {\infty}, \min, +)$ Tropical Semiring</td>
<td>Shortest paths</td>
</tr>
<tr>
<td>$(\mathbb{R} \cup {\infty}, \min, \times)$</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>
Jon Berry challenge problems for GALA (all final project possibilities)

- Clustering coefficient (triangle counting)
- Connected components (bully algorithm)
- Maximum independent set (NP-hard)
- Maximal independent set (Luby algorithm)
- Single-source shortest paths
- Special betweenness (for subgraph isomorphism)
Fast Approximate Neighborhood Function (S. Vigna) (final project possibility)

• Distribution of distances between vertices in a graph
  – For each vertex v, how many vertices at distance k?
  – What’s the average distance (or closeness) between vertices?

• Expensive to compute exactly

• Nifty but simple data structure gives good approximation fast

• Could be implemented as sparse matrix times sparse vector, with the nifty data structure in the semiring

• E.g., using Combinatorial BLAS library

• Link to paper on course web site
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

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.
## 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>
</table>
| SpGEMM   | Sparse Matrix (as friend) | **A, B**: sparse matrices  
**trA**: transpose **A** if true  
**trB**: transpose **B** if true | Sparse Matrix | **C** = **A** * **B** |
| SpMV     | Sparse Matrix (as friend) | **A**: sparse matrices  
**x**: sparse or dense vector(s)  
**trA**: transpose **A** if true | Sparse or Dense Vector(s) | **y** = **A** * **x** |
| SpEWISEX | Sparse Matrices (as friend) | **A, B**: sparse matrices  
**notA**: negate **A** if true  
**notB**: negate **B** if true | Sparse Matrix | **C** = **A** * **B** |
| REDUCE   | Any Matrix (as method) | dim: dimension to reduce  
binop: reduction operator | Dense Vector | sum(**A**) |
| SpREF    | Sparse Matrix (as method) | **p**: row indices vector  
**q**: column indices vector | Sparse Matrix | **B** = **A**(p, **q**) |
| SpASGN   | Sparse Matrix (as method) | **p**: row indices vector  
**q**: column indices vector  
**B**: matrix to assign | none | **A**(p, **q**) = **B** |
| SCALE    | Any Matrix (as method) | **rhs**: any object  
(except a sparse matrix) | none | Check guiding principles 3 and 4 |
| SCALE    | Any Vector (as method) | **rhs**: any vector | none | none |
| APPLY    | Any Object (as method) | **unop**: unary operator  
(applied to non-zeros) | None | none |
Combinatorial BLAS: Distributed-memory reference implementation

Combinatorial BLAS functions and operators

- DistMat
- CommGrid
- FullyDistVec

- DenseDistMat
- SpDistMat
- SpMat
- SpDistVec
- DenseDistVec

... HAS A

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
Parallel sparse matrix-matrix multiplication algorithm

2D algorithm: Sparse SUMMA (based on dense SUMMA)

General implementation that handles rectangular matrices

\[ C_{ij} += \text{HyperSparseGEMM}(A_{recv}, B_{recv}) \]
1D vs. 2D scaling for sparse matrix-matrix multiplication

**Restriction on Freescale matrix**

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Seconds</th>
<th>SpSUMMA</th>
<th>EpetraExt</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>25X</td>
<td>3.1X</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>2.4X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>2.0X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>1.6X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>4.0X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>8.0X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Restriction on GHS psdef/ldoor matrix**

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Seconds</th>
<th>SpSUMMA</th>
<th>EpetraExt</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>27X</td>
<td>1.6X</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>2.5X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>6.7X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>14X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>18X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>27X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SpSUMMA = 2-D data layout (Combinatorial BLAS)  
EpetraExt = 1-D data layout
Almost linear scaling until bandwidth costs start to dominate.

Scaling proportional to $\sqrt{p}$ afterwards.

$T_{comp} = O(n)$

$T_{comm} = O(n\sqrt{p})$
Work In Progress:
QuadMat Shared-memory data structure

Blocks store a fixed number of elements. More dense parts of the matrix have more blocks, but each block has enough data for meaningful computation.

subdivide by dimension on power of 2 indices

m rows

n columns
Example

Scale 10 RMAT (887x887, 21304 non-nulls) up to 1024 non-nulls per block

Sorted by degree

Blue blocks: uint16_t indices
Green blocks: uint8_t indices
Each (tiny) dot is a non-null
Implementation in templated C++

• Parallelized with TBB’s task scheduler.
  – *Continuation passing style.*

• Leaf operations are instantiated templates with statically-known data types.
  – No virtual functions or other barriers to compiler optimization. Note: slow compilation times due to number of permutations of block types.

• Only pay for what you use.
  – Sorts are optional.
  – Overhead for variable-length data only if used.
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)

- Domain scientists
- Graph algorithm developers
- HPC scientists and engineers

• 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., degree())
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank

- Hypergraphs and sparse matrices
- Graph primitives (e.g., 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
```
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
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)
- V3 release April 2013 (V4 expected spring 2014)
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.

Betweenness Centrality

\[ C_B(v) = \frac{\sum_{s \neq v \neq t} \sigma_{st}(v)}{\sigma_{st}} \]

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.

PageRank

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".

Belief Propagation

\[ \sum_{\text{between neighbors}} P_{ij} \mu_j = \mu_i \]

\[ \mu_i = \frac{1}{\sum_{j \neq i} A_{ij}} \]

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 \( \mu \) of each variable; the converged mean-value vector approximates the actual solution.
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
- KDT Algorithm
- CombBLAS Primitive
- Filter (Py)
- Semiring (Py)
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.
SEJITS+KDT multicore performance

- MIS = Maximal Independent Set
- 36 cores of Mirasol (Intel Xeon E7-8870)
- Erdős-Rényi (Scale 22, edgefactor=4)

Synthetic data with weighted randomness to match filter permeability
Notation: [semiring impl] / [filter impl]
A *roofline model* for shows how SEJITS moves KDT analytics from being Python *compute bound* to being *bandwidth bound*.

- Breadth-first search
- 576 cores of Hopper (Cray XE6 at NERSC with AMD Opterons)
- R-MAT (Scale 25, edgefactor=16, symmetric)
SEJITS+KDT real graph performance

- Breadth-first search
- 16 cores of Mirasol (Intel Xeon E7-8870)

Sizes (vertex and edge counts) of different combined Twitter graphs.

<table>
<thead>
<tr>
<th>Label</th>
<th>Vertices (millions)</th>
<th>Edges (millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tweet</td>
<td>Follow</td>
</tr>
<tr>
<td>Small</td>
<td>0.5</td>
<td>0.7</td>
</tr>
<tr>
<td>Medium</td>
<td>4.2</td>
<td>14.2</td>
</tr>
<tr>
<td>Large</td>
<td>11.3</td>
<td>59.7</td>
</tr>
<tr>
<td>Huge</td>
<td>16.8</td>
<td>102.4</td>
</tr>
</tbody>
</table>

Statistics about the largest strongly connected components of the Twitter graphs

<table>
<thead>
<tr>
<th>Label</th>
<th>Vertices</th>
<th>Edges traversed</th>
<th>Edges processed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>78,397</td>
<td>147,873</td>
<td>29.4 million</td>
</tr>
<tr>
<td>Medium</td>
<td>55,872</td>
<td>93,601</td>
<td>54.1 million</td>
</tr>
<tr>
<td>Large</td>
<td>45,291</td>
<td>73,031</td>
<td>59.7 million</td>
</tr>
<tr>
<td>Huge</td>
<td>43,027</td>
<td>68,751</td>
<td>60.2 million</td>
</tr>
</tbody>
</table>
Roofline analysis: Why does SEJITS+KDT work?

Even with SEJITS, there are run-time overheads with function calls via pointers.

How is it so close to the Combinatorial BLAS performance?

Because once we are bandwidth bound, additional complexity does not hurt.
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 (original) BLAS

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>

- 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 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 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.

Graph Algorithm Building Blocks workshop: GABB @ IPDPS May 2014
Sparse array attribute survey

<table>
<thead>
<tr>
<th>Function</th>
<th>Graph BLAS</th>
<th>Comb BLAS</th>
<th>Sparse BLAS</th>
<th>STINGER</th>
<th>D4M</th>
<th>SciDB</th>
<th>Tensor Toolbox</th>
<th>Julia</th>
<th>GraphLab</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>1.3.0</td>
<td>2006</td>
<td>r633</td>
<td>2.5</td>
<td>13.9</td>
<td>2.5</td>
<td>0.2.0</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>any</td>
<td>C++</td>
<td>F,C,C++</td>
<td>C</td>
<td>Matlab</td>
<td>C++</td>
<td>Matlab, C++</td>
<td>Julia</td>
<td>C++</td>
</tr>
<tr>
<td>Dimension</td>
<td>2</td>
<td>1, 2</td>
<td>2</td>
<td>1, 2, 3</td>
<td>2</td>
<td>1 to 100</td>
<td>2, 3</td>
<td>1,2</td>
<td>2</td>
</tr>
<tr>
<td>Index Base</td>
<td>0 or 1</td>
<td>0</td>
<td>0 or 1</td>
<td>0</td>
<td>1</td>
<td>±N</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Index Type</td>
<td>uint64</td>
<td>uint64</td>
<td>int</td>
<td>int64</td>
<td>double, string</td>
<td>int64</td>
<td>double</td>
<td>any int</td>
<td>uint64</td>
</tr>
<tr>
<td>Value Type</td>
<td>?</td>
<td>user</td>
<td>single, double, complex</td>
<td>int64</td>
<td>logical, double, complex, string</td>
<td>user</td>
<td>logical, double, complex</td>
<td>user</td>
<td>user</td>
</tr>
<tr>
<td>Null</td>
<td>0</td>
<td>user</td>
<td>0</td>
<td>0</td>
<td>≤0</td>
<td>null</td>
<td>0</td>
<td>0</td>
<td>int64(-1)</td>
</tr>
<tr>
<td>Sparse Format</td>
<td>?</td>
<td>tuple</td>
<td>undef</td>
<td>linked list</td>
<td>dense, csc, tuple</td>
<td>RLE</td>
<td>dense, csc</td>
<td>csc</td>
<td>csr/csc</td>
</tr>
<tr>
<td>Parallel</td>
<td>?</td>
<td>2D block</td>
<td>none</td>
<td>block</td>
<td>arbitrary</td>
<td>N-D block, cyclic w/ overlap</td>
<td>none</td>
<td>N-D block, cyclic w/ overlap</td>
<td>Edge based w/ vertex split</td>
</tr>
<tr>
<td>+ operations</td>
<td>user?</td>
<td>user</td>
<td>+</td>
<td>user</td>
<td>+,* ,max,min, ∩,∪</td>
<td>user</td>
<td>+</td>
<td>user</td>
<td>user</td>
</tr>
<tr>
<td>* operations</td>
<td>user?</td>
<td>user</td>
<td>*</td>
<td>user</td>
<td>user</td>
<td>user</td>
<td>user</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Matrix times matrix over semiring

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

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

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

Implements
$C \oplus= A \oplus.\otimes B$

for $j = 1 : N$
$C(i,k) = C(i,k) \oplus (A(i,j) \otimes B(j,k))$

If input C is omitted, implements
$C = A \oplus.\otimes B$

Transpose flags specify operation on $A^T$, $B^T$, and/or $C^T$ 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 $A$: $\mathbb{S}^{M \times N}$ (sparse)
- matrix $B$: $\mathbb{S}^{|p| \times |q|}$ (sparse)
- vector $p \subseteq \{1, \ldots, M\}$
- vector $q \subseteq \{1, \ldots, N\}$

**Optional Inputs**

- none

**Outputs**

- matrix $A$: $\mathbb{S}^{M \times N}$ (sparse)
- matrix $B$: $\mathbb{S}^{|p| \times |q|}$ (sparse)

**SpRef Implements**

\[ B = A(p,q) \]

\[
\text{for } i = 1 : |p| \\
\quad \text{for } j = 1 : |q| \\
\quad \quad B(i,j) = A(p(i),q(j))
\]

**SpAsgn Implements**

\[ A(p,q) = B \]

\[
\text{for } i = 1 : |p| \\
\quad \text{for } j = 1 : |q| \\
\quad \quad A(p(i),q(j)) = B(i,j)
\]

**Notes**

- $\mathbb{S}$ is the set of scalars, user-specified defaults to IEEE double float
- $|p| = \text{length of vector } p$
- $|q| = \text{length of vector } q$

**Specific cases and function names**

- SpRef: get sub-matrix
- SpAsgn: assign to sub-matrix
Element-wise operations

**Inputs**
- matrix $A$: $\mathbb{S}^{M \times N}$ (sparse or dense)
- matrix $B$: $\mathbb{S}^{M \times N}$ (sparse or dense)

**Optional Inputs**
- matrix $C$: $\mathbb{S}^{M \times N}$ (sparse or dense)
- scalar “add” function $\oplus$
- scalar “multiply” function $\otimes$

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

**Implements**
$$C \oplus= A \otimes B$$

```plaintext
for i = 1 : M
  for j = 1 : N
    C(i,j) = C(i,j) $\oplus$ (A(i,j) $\otimes$ B(i,j))
  
If input $C$ is omitted, implements $C = A \otimes B$
```

**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:**
- SpE WiseX: matrix elementwise
- $M=1$ or $N=1$: vector elementwise
- Scale: when $A$ or $B$ is a scalar
Apply/Update

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

Optional Inputs
matrix C: $\mathbb{S}^{M \times N}$ (sparse or dense)
scalar “add” function $\oplus$
unary function $f()$

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

Implements
\[
C \oplus= f(A)
\]

for $i = 1 : M$
for $j = 1 : N$
  if $A(i,j) \neq 0$
    \[
    C(i,j) = C(i,j) \oplus f(A(i,j))
    \]

If input C is omitted, implements
\[
C = f(A)
\]

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
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.