Sparse Matrices for High-Performance Graph Computation

John R. Gilbert
University of California, Santa Barbara

with Aydin Buluc, LBNL; Armando Fox, UCB; Shoaib Kamil, MIT;
Adam Lugowski, UCSB; Lenny Oliker, LBNL, Sam Williams, LBNL

ENS Lyon
September 25, 2012

Support: Intel, Microsoft, DOE Office of Science, NSF
Outline

• Motivation

• Sparse matrices for graph algorithms

• CombBLAS: sparse arrays and graphs on parallel machines

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

• Specialization: getting the best of both worlds
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
An analogy?

Continuous physical modeling

Linear algebra

Computers

Discrete structure analysis

Graph theory

Computers
**Top 500 List (June 2012)**

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

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer/Year Vendor</th>
<th>Cores</th>
<th>R_max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOE/NNSA/LLNL United States</td>
<td>Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom / 2011 IBM</td>
<td>1572864</td>
<td>16324.75</td>
</tr>
<tr>
<td>2</td>
<td>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu</td>
<td>705024</td>
<td>10510.00</td>
</tr>
<tr>
<td>3</td>
<td>DOE/SC/Argonne National Laboratory United States</td>
<td>Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM</td>
<td>786432</td>
<td>8162.38</td>
</tr>
<tr>
<td>4</td>
<td>Leibniz Rechenzentrum Germany</td>
<td>SuperMUC - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR / 2012 IBM</td>
<td>147456</td>
<td>2897.00</td>
</tr>
<tr>
<td>5</td>
<td>National Supercomputing Center in Tianjin China</td>
<td>Tianhe-1A - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT</td>
<td>186368</td>
<td>2566.00</td>
</tr>
<tr>
<td>6</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td>Jaguar - Cray XK6, Opteron 6274 16C 2.20GHz, Cray Gemini interconnect, NVIDIA 2090 / 2009 Cray Inc.</td>
<td>298592</td>
<td>1941.00</td>
</tr>
<tr>
<td>7</td>
<td>CINECA Italy</td>
<td>Fermi - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM</td>
<td>163840</td>
<td>1725.49</td>
</tr>
<tr>
<td>8</td>
<td>Forschungszentrum Juelich (FZJ) Germany</td>
<td>JuQUEEN - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM</td>
<td>131072</td>
<td>1380.39</td>
</tr>
<tr>
<td>9</td>
<td>CEA/TGCC-GENCI France</td>
<td>Curie thin nodes - Bull B510, Xeon E5-2680 8C 2.70GHz, Infiniband QDR / 2012 IBM</td>
<td>77184</td>
<td>1359.00</td>
</tr>
<tr>
<td>10</td>
<td>National Supercomputing Centre in Shenzhen (NSCS) China</td>
<td>Nebulae - Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Bull</td>
<td>120640</td>
<td>1271.00</td>
</tr>
<tr>
<td>11</td>
<td>NASA/Ames Research Center/NAS United States</td>
<td>Pleiades - SGI Altix ICE X/8200EX/8400EX, Xeon 54xx 3.0/5570/5670/E5-2670 2.93/2.6 /3.00/3.0 Ghz, Infiniband QDR/FDR / 2011 SGI</td>
<td>125980</td>
<td>1243.00</td>
</tr>
</tbody>
</table>
Graph 500 List (June 2012)

Graph500 Benchmark:
Breadth-first search in a large power-law graph

<table>
<thead>
<tr>
<th>Rank</th>
<th>Installation Site</th>
<th>Machine</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>DOE/SC/Argonne National Laboratory</td>
<td>Mira/BlueGene/Q</td>
<td>32768</td>
<td>524288</td>
<td>38</td>
<td>3541</td>
</tr>
<tr>
<td>1</td>
<td>LLNL</td>
<td>Sequola/Blue Gene/Q</td>
<td>32768</td>
<td>524288</td>
<td>38</td>
<td>3541</td>
</tr>
<tr>
<td>2</td>
<td>DARPA Trial Subset, IBM Development Engineering</td>
<td>Power 775, POWER7 8C 3.836 GHz</td>
<td>1024</td>
<td>32768</td>
<td>35</td>
<td>508.05</td>
</tr>
<tr>
<td>3</td>
<td>Information Technology Center, The University of Tokyo</td>
<td>Oakleaf-FX (Fujitsu PRIMEHPC FX 10)</td>
<td>4800</td>
<td>76800</td>
<td>38</td>
<td>358.1</td>
</tr>
<tr>
<td>4</td>
<td>GSIC Center, Tokyo Institute of Technology</td>
<td>HP Cluster Platform SL390s G7 (three Tesla cards per node)</td>
<td>1356</td>
<td>16392</td>
<td>35</td>
<td>317.09</td>
</tr>
<tr>
<td>5</td>
<td>Brookhaven National Laboratory</td>
<td>BLUE GENE/Q</td>
<td>1024</td>
<td>16384</td>
<td>34</td>
<td>294.293</td>
</tr>
<tr>
<td>6</td>
<td>DOE/SC/Argonne National Laboratory</td>
<td>Vesta/BlueGene/Q</td>
<td>1024</td>
<td>16384</td>
<td>34</td>
<td>292.363</td>
</tr>
<tr>
<td>7</td>
<td>NASA-Ames / Parallel Computing Lab, Intel Labs</td>
<td>Pleiades - SGI ICE-X, dual plane hypercube FDR infiniband, E5-2670 &quot;sandybridge&quot;</td>
<td>1024</td>
<td>16384</td>
<td>34</td>
<td>270.33</td>
</tr>
<tr>
<td>8</td>
<td>NERSC/LBNL</td>
<td>XE6</td>
<td>4817</td>
<td>115600</td>
<td>35</td>
<td>254.074</td>
</tr>
<tr>
<td>9</td>
<td>NNSA and IBM Research, T.J. Watson</td>
<td>NNSA/SC Blue Gene/Q</td>
<td>4096</td>
<td>65536</td>
<td>32</td>
<td>236</td>
</tr>
<tr>
<td>10</td>
<td>GSIC Center, Tokyo Institute of Technology</td>
<td>TSUBAME 2.0 (CPU only)</td>
<td>1356</td>
<td>16392</td>
<td>36</td>
<td>202.68</td>
</tr>
</tbody>
</table>
16.3 Petaflops

\[ P \begin{bmatrix} A \end{bmatrix} = \begin{bmatrix} L \end{bmatrix} \times \begin{bmatrix} U \end{bmatrix} \]

3.54 Terateps

16.3 Peta / 3.54 Tera is about 4600
The challenge of the software stack

- By analogy to numerical scientific computing... 

- What should the combinatorial BLAS look like?

Basic Linear Algebra Subroutines (BLAS): Speed (MFlops) vs. Matrix Size (n)

\[ C = A \times B \]

\[ y = A \times x \]

\[ \mu = x^T y \]
Outline

• Motivation

• Sparse matrices for graph algorithms

• CombBLAS: sparse arrays and graphs on parallel machines

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

• Specialization: getting the best of both worlds
Sparse array-based primitives

Sparse matrix-matrix multiplication (SpGEMM)

Element-wise operations

Sparse matrix-dense vector multiplication

Sparse matrix indexing

Matrices on various semirings: \((x, +)\), \((\text{and, or})\), \((+, \text{min})\), ...
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
Indexing sparse arrays in parallel (extract subgraphs, coarsen grids, etc.)

\[ \text{SpRef: } B = A(I, J) \]
\[ \text{SpAsgn: } B(I, J) = A \]
\[ \text{SpExpAdd: } B(I, J) += A \]

\[ A, B: \text{ sparse matrices} \]
\[ I, J: \text{ vectors of indices} \]

\[ \text{SpRef using mixed-mode sparse matrix-matrix multiplication (SpGEMM). Ex: } B = A([2, 4], [1, 2, 3]) \]
Graph contraction via sparse triple product

A1

A2

A3

Contract

=
Subgraph extraction via sparse triple product
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>
Outline

• Motivation
• Sparse matrices for graph algorithms
• CombBLAS: sparse arrays and graphs on parallel machines
• KDT: attributed semantic graphs in a high-level language
• Specialization: getting the best of both worlds
Some 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>SpGEMM</td>
<td>Sparse Matrix</td>
<td>( \mathbf{A}, \mathbf{B} ): sparse matrices</td>
<td>Sparse Matrix</td>
<td>( \mathbf{C} = \mathbf{A} \times \mathbf{B} )</td>
</tr>
<tr>
<td></td>
<td>(as friend)</td>
<td>( \text{trA}: ) transpose ( \mathbf{A} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{trB}: ) transpose ( \mathbf{B} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpMV</td>
<td>Sparse Matrix</td>
<td>( \mathbf{A}: ) sparse matrices</td>
<td>Sparse or Dense Vector(s)</td>
<td>( \mathbf{y} = \mathbf{A} \times \mathbf{x} )</td>
</tr>
<tr>
<td></td>
<td>(as friend)</td>
<td>( \mathbf{x}: ) sparse or dense vector(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{trA}: ) transpose ( \mathbf{A} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpEWiseX</td>
<td>Sparse Matrices</td>
<td>( \mathbf{A}, \mathbf{B}: ) sparse matrices</td>
<td>Sparse Matrix</td>
<td>( \mathbf{C} = \mathbf{A} \times \mathbf{B} )</td>
</tr>
<tr>
<td></td>
<td>(as friend)</td>
<td>( \text{notA}: ) negate ( \mathbf{A} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{notB}: ) negate ( \mathbf{B} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduce</td>
<td>Any Matrix</td>
<td>( \text{dim}: ) dimension to reduce</td>
<td>Dense Vector</td>
<td>\text{sum}(\mathbf{A})</td>
</tr>
<tr>
<td></td>
<td>(as method)</td>
<td>( \text{binop}: ) reduction operator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpRef</td>
<td>Sparse Matrix</td>
<td>( \mathbf{p}: ) row indices vector</td>
<td>Sparse Matrix</td>
<td>( \mathbf{B} = \mathbf{A}(\mathbf{p}, \mathbf{q}) )</td>
</tr>
<tr>
<td></td>
<td>(as method)</td>
<td>( \mathbf{q}: ) column indices vector</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpAsgn</td>
<td>Sparse Matrix</td>
<td>( \mathbf{p}: ) row indices vector</td>
<td>none</td>
<td>( \mathbf{A}(\mathbf{p}, \mathbf{q}) = \mathbf{B} )</td>
</tr>
<tr>
<td></td>
<td>(as method)</td>
<td>( \mathbf{q}: ) column indices vector</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mathbf{B}: ) matrix to assign</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>Any Matrix</td>
<td>( \text{rhs}: ) any object</td>
<td>none</td>
<td>Check guiding</td>
</tr>
<tr>
<td></td>
<td>(as method)</td>
<td>(except a sparse matrix)</td>
<td></td>
<td>principles 3 and 4</td>
</tr>
<tr>
<td>Scale</td>
<td>Any Vector</td>
<td>( \text{rhs}: ) any vector</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>(as method)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apply</td>
<td>Any Object</td>
<td>( \text{unop}: ) unary operator</td>
<td>None</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>(as method)</td>
<td>(applied to non-zeros)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

2D layout for sparse matrices & vectors
BFS in “vanilla” MPI Combinatorial BLAS

- Graph500 benchmark at scale 29, C++ (or KDT) calling CombBLAS
- NERSC “Hopper” machine (Cray XE6)
Strong scaling of vertex relabeling

symmetric permutation $\Leftrightarrow$ relabeling graph vertices

- RMAT Scale 22; edge factor=8; $a=0.6$, $b=c=d=0.4/3$
- Franklin/NERSC, each node is a quad-core AMD Budapest
1D vs. 2D scaling for sparse matrix-matrix multiplication

In practice, 2D algorithms have the potential to scale, but not linearly

\[ T_{\text{comm}}(2D) = \alpha p \sqrt{p} + \beta cn \sqrt{p} \]

\[ T_{\text{comp}}(\text{optimal}) = c^2 n \]
Parallel sparse matrix-matrix multiplication algorithms

\[ C_{ij} += \text{HyperSparseGEMM}(A_{\text{recv}}, B_{\text{recv}}) \]

2D algorithm: Sparse SUMMA (based on dense SUMMA)
General implementation that handles rectangular matrices
Sequential “hypersparse” kernel

Operates on the strictly $O(\text{nnz})$ DCSC data structure

Sparse outer-product formulation with multi-way merging

Efficient in parallel, i.e. $T(1) \approx p \cdot T(p)$

**Time complexity:**

$$O(\text{flops} \cdot \lg n_i + \text{nnz}(A) + \text{nnz}(B))$$

- independent of dimension

**Space complexity:**

$$O(\text{nnz}(A) + \text{nnz}(B) + \text{nnz}(C))$$

- independent of flops
Square sparse matrix multiplication

Almost linear scaling until bandwidth costs starts to dominate

Scaling proportional to $\sqrt{p}$ afterwards

$$T_{\text{com}} = \alpha p \sqrt{p} + \beta cn \sqrt{p}$$

$$T_{\text{comp}}(\text{optimal}) = c^2 n$$
Outline

- Motivation
- Sparse matrices for graph algorithms
- CombBLAS: sparse arrays and graphs on parallel machines
- KDT: attributed semantic graphs in a high-level language
- Specialization: getting the best of both worlds
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

Domain scientists

Graph algorithm developers

HPC scientists and engineers
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

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.*, `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

```python
G = bigG.subgraph(comp==giantComp)
G = G.cluster('Markov')
```
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
```

```python
L = G.toSpParMat()
d = L.sum(kdt.SpParMat.Column)
L = -L
L.setDiag(d)
M = kdt.SpParMat.eye(G.nvert()) - mu*L
pos = kdt.ParVec.rand(G.nvert())
for i in range(nsteps):
pos = M.SpMV(pos)
```
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)
- V0.2 released March 2012

A general graph library with operations based on linear algebraic primitives
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.

Markov Clustering

Betweenness Centrality

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.

Belief Propagation

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

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.
The need for filters

Graph of text & phone calls

Betweenness centrality

Betweenness centrality on text messages

Betweenness centrality on phone calls
Attributed semantic graphs and filters

Example:

- Vertex types: Person, Phone, Camera, Gene, Pathway
- Edge types: PhoneCall, TextMessage, CoLocation, Sequence Similarity
- Edge attributes: StartTime, EndTime
- Calculate centrality just for emails among engineers sent between times sTime and eTime

```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))

start = dt.now() - dt.timedelta(days=30)
end = dt.now()

# G denotes the graph
G.addVFilter(onlyEngineers)
G.addEFilter(timedEmail(start, end))

# rank via centrality based on recent email transactions among engineers
bc = G.rank('approxBC')
```
Filter options and implementation

- Filter defined as unary predicates, checked in order they were added
- Each KDT object maintains a stack of filter predicates
- All operations respect filters, enabling filter-ignorant algorithm design

<table>
<thead>
<tr>
<th>On-the-fly filters</th>
<th>Materialized filters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edges are retained</td>
<td>Edges are pruned on copy</td>
</tr>
<tr>
<td>Check predicate on each edge/vertex traversal</td>
<td>Check predicate once on materialization</td>
</tr>
<tr>
<td>Cheap but done on each run</td>
<td>Expensive but done once</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
- Specialization: getting the best of both worlds
On-the-fly filter performance issues

- Write filters as semiring ops in C, wrap in Python
  - Good performance but hard to write new filters
- Write filters in Python and call back from CombBLAS
  - Flexible and easy but runs slow
- The issue is local computation, not parallelism or comms
- All user-written semirings face the same issue.
Solution: Just-in-time specialization (SEJITS)

- On first call, translate Python filter or semiring op to C++
- Compile with GCC
- Call the compiled code thereafter
- (Lots of details omitted ….)
Filtered BFS with SEJITS

Time (in seconds) for a single BFS iteration on Scale 23 RMAT (8M vertices, 130M edges) with 10% of elements passing filter. Machine is Mirasol.
Conclusion

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

• A CSC moral: Things are always clearer when you look at them from two directions.

• Just-in-time specialization is key to performance of flexible user-programmable graph analytics.

kdt.sourceforge.net/
gauss.cs.ucsb.edu/~aydin/CombBLAS/