A Toolbox for High-Performance Graph Computation

John R. Gilbert
University of California, Santa Barbara

UC Berkeley ParLab
September 15, 2011

Support: Intel, Microsoft, DOE Office of Science, NSF
Team (2010-2011)

- PI: JRG
- UCSB PhD students: Adam Lugowski, Lijie Ren
- UCSB MS students: Yun Teng
- UCSB undergrads: Chris Lock, Drew Waranis
- Berkeley Lab: Aydın Buluç
- Microsoft Corporation: Steve Reinhardt, David Alber
- Research support: Intel Corporation, Microsoft Corporation, DOE Office of Science, NSF
Outline

• Motivation
• Libraries: CombBLAS and KDT
• Algorithms
• Plans
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 2011)

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

\[
P A = L x U
\]

<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>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu</td>
<td>548352</td>
<td>8162.00</td>
</tr>
<tr>
<td>2</td>
<td>National Supercomputing Center in Tianjin China</td>
<td>Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT</td>
<td>186368</td>
<td>2566.00</td>
</tr>
<tr>
<td>3</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td>Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.</td>
<td>224162</td>
<td>1759.00</td>
</tr>
<tr>
<td>4</td>
<td>National Supercomputing Centre in Shenzhen (NSCS) China</td>
<td>Nebulae - Dawning TC3600 Blade, Intel X5650, Nvidia Tesla C2050 GPU / 2010 Dawning</td>
<td>120640</td>
<td>1271.00</td>
</tr>
<tr>
<td>5</td>
<td>GSIC Center, Tokyo Institute of Technology Japan</td>
<td>TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP</td>
<td>73278</td>
<td>1192.00</td>
</tr>
<tr>
<td>6</td>
<td>DOE/NNSA/LANL/SNL United States</td>
<td>Cielo - Cray XE6 8-core 2.4 GHz / 2011 Cray Inc.</td>
<td>142272</td>
<td>1110.00</td>
</tr>
<tr>
<td>7</td>
<td>NASA Ames Research Center/NAS United States</td>
<td>Pleiades - SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband / 2011 SGI</td>
<td>111104</td>
<td>1088.00</td>
</tr>
<tr>
<td>8</td>
<td>DOE/SC/LBNL/NERSC United States</td>
<td>Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.</td>
<td>153408</td>
<td>1054.00</td>
</tr>
</tbody>
</table>
Graph 500 List (June 2011)

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

<table>
<thead>
<tr>
<th>Rank</th>
<th>Machine</th>
<th>Owner</th>
<th>Problem Size</th>
<th>TEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Intrepid (IBM Blue Gene/P, 32,768 nodes / 131,072 cores)</td>
<td>ANL</td>
<td>38</td>
<td>18,508,000,000</td>
</tr>
<tr>
<td>2</td>
<td>Jugene (IBM Blue Gene/P, 32,768 nodes / 131,072 cores)</td>
<td>Forschungszentrum Jülich</td>
<td>38</td>
<td>18,416,700,000</td>
</tr>
<tr>
<td>3</td>
<td>Lomonosov (MPP, 4096 nodes / 8192 cores)</td>
<td>Moscow State University</td>
<td>37</td>
<td>43,471,500,000</td>
</tr>
<tr>
<td>4</td>
<td>Hopper (Cray XE6, 1800 nodes / 43,200 cores)</td>
<td>LBL</td>
<td>37</td>
<td>25,075,200,000</td>
</tr>
<tr>
<td>5</td>
<td>Franklin (Cray XT4, 4000 nodes / 16,000 cores)</td>
<td>LBL</td>
<td>36</td>
<td>19,955,100,000</td>
</tr>
<tr>
<td>6</td>
<td>Lonestar (Dell PowerEdge M610, 512 nodes / 6144 cores)</td>
<td>TACC</td>
<td>34</td>
<td>8,080,000,000</td>
</tr>
<tr>
<td>7</td>
<td>Kraken (Appro, 1 node / 32 cores / Fusion I/O)</td>
<td>LLNL</td>
<td>34</td>
<td>55,948,453</td>
</tr>
<tr>
<td>8</td>
<td>Red Sky (Sun, 512 nodes / 4096 cores)</td>
<td>SNL</td>
<td>33</td>
<td>9,470,000,000</td>
</tr>
<tr>
<td>9</td>
<td>Endeavor (Westmere X5670, 256 processors / 3072 cores)</td>
<td>Intel</td>
<td>33 (Toy MPI Simple)</td>
<td>6,860,000,000</td>
</tr>
<tr>
<td>10</td>
<td>SGI Altix UV 1000 (2048 cores)</td>
<td>SGI</td>
<td>32</td>
<td>10,161,300,000</td>
</tr>
<tr>
<td>11</td>
<td>IBM BlueGene/P, 2048 nodes / 8192 cores</td>
<td>Moscow State University</td>
<td>32</td>
<td>6,930,560,000</td>
</tr>
<tr>
<td>12</td>
<td>Blacklight (SGI Altix UV 1000, 512 processors)</td>
<td>PSC</td>
<td>32 (Small)</td>
<td>4,452,270,000</td>
</tr>
</tbody>
</table>
8.1 Petaflops

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

43 Gigateps

8.1 Peta / 43 Giga is about 190,000!
Outline

- Motivation
- Libraries: CombBLAS and KDT
- Algorithms
- Plans
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)

\[ C = A \times B \]

\[ y = A \times x \]

\[ \mu = x^T y \]
Sparse array-based primitives

Sparse matrix-matrix multiplication (SpGEMM)

Element-wise operations

Sparse matrix-dense vector multiplication

Sparse matrix indexing

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

\[ A^T \quad X \]
Multiple-source breadth-first search

- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
The case for sparse matrices

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>
Combinatorial BLAS: A matrix-based graph library

- Reference implementation in MPI
- Matrix operations over user-defined (and some built-in) semirings
- Highly templated C++
- Reference implementation in MPI

Architecture of matrix classes

- Also sparse & dense vectors, distributed and local
- Matrix operations over user-defined (and some built-in) semirings
### 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 (as friend)</td>
<td>A, B: sparse matrices, trA: transpose A if true, trB: transpose B if true</td>
<td>Sparse Matrix</td>
<td>( C = A \times B )</td>
</tr>
<tr>
<td>SpMV</td>
<td>Sparse Matrix (as friend)</td>
<td>A: sparse matrices, x: sparse or dense vector(s), trA: transpose A if true</td>
<td>Sparse or Dense Vector(s)</td>
<td>( y = A \times x )</td>
</tr>
<tr>
<td>SpEWiseX</td>
<td>Sparse Matrices (as friend)</td>
<td>A, B: sparse matrices, notA: negate A if true, notB: negate B if true</td>
<td>Sparse Matrix</td>
<td>( C = A \times B )</td>
</tr>
<tr>
<td>Reduce</td>
<td>Any Matrix (as method)</td>
<td>dim: dimension to reduce, binop: reduction operator</td>
<td>Dense Vector</td>
<td>( \text{sum}(A) )</td>
</tr>
<tr>
<td>SpRef</td>
<td>Sparse Matrix (as method)</td>
<td>p: row indices vector, q: column indices vector</td>
<td>Sparse Matrix</td>
<td>( B = A(p, q) )</td>
</tr>
<tr>
<td>SpAsgn</td>
<td>Sparse Matrix (as method)</td>
<td>p: row indices vector, q: column indices vector, B: matrix to assign</td>
<td>none</td>
<td>( A(p, q) = B )</td>
</tr>
<tr>
<td>Scale</td>
<td>Any Matrix (as method)</td>
<td>rhs: any object (except a sparse matrix)</td>
<td>none</td>
<td>Check guiding principles 3 and 4</td>
</tr>
<tr>
<td>Scale</td>
<td>Any Vector (as method)</td>
<td>rhs: any vector</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Apply</td>
<td>Any Object (as method)</td>
<td>unop: unary operator (applied to non-zeros)</td>
<td>None</td>
<td>none</td>
</tr>
</tbody>
</table>
BFS in “vanilla” MPI Combinatorial BLAS

- Graph500 benchmark at scale 29, C++ (or KDT) calling CombBLAS
- NERSC “Hopper” machine (Cray XE6)
- [Buluç & Madduri]: New hybrid CombBLAS MPI + OpenMP gets 17.8 GTEPS at scale 32 on 40,000 cores of Hopper
Outline

• Motivation
• Libraries: CombBLAS and KDT
• Algorithms
• Plans
Knowledge Discovery Toolbox
http://kdt.sourceforge.net/

A general graph library with operations based on linear algebraic primitives
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

A general graph library with operations based on linear algebraic primitives
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
- A collaboration among UCSB, Lawrence Berkeley National Lab, and Microsoft Technical Computing
- Open source software, released under New BSD license
- v0.1 released March 2011; v0.2 expected October 2011
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.,* \texttt{degree()}*\texttt{)}
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank

```python
# bigG contains the input graph
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)
clus = G.cluster('Markov')
clusNedge = G.nedge(clus)
smallG = G.contract(clus)
# visualize
```

- Hypergraphs and sparse matrices
- Graph primitives *(e.g.,* \texttt{bfsTree()}*\texttt{)}
- 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: between 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
# bigG contains the input graph
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)
clus = G.cluster('Markov')
clusNedge = G.nedge(clus)
smallG = G.contract(clus)

# visualize
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)
```
Graph API (v0.2)

Real applications

Community Detection

Network Vulnerability Analysis

centrality('exactBC')
centrality('approxBC')

pageRank

cluster('Markov')
cluster('spectral')

Graph500

DiGraph
bfsTree, isBfsTree
plus utility (e.g., DiGraph,nvert, toParVec,degree,load,UFget,+,*,
sum,subgraph,reverseEdges)

HyGraph
bfsTree, isBfsTree
plus utility (e.g., HyGraph,nvert, toParVec,degree,load,UFget)

(Sp)ParVec
(e.g., +,*,,|,&,>,==,[], abs,max,sum,range,
norm,hist,randPerm, scale, topK)

SpParMat
(e.g., +,* , SpMM, SpMV, SpRef, SpAsgn)

Applets

Real applications

Community Detection

Network Vulnerability Analysis

centrality('exactBC')
centrality('approxBC')

pageRank

cluster('Markov')
cluster('spectral')

Graph500

DiGraph
bfsTree, isBfsTree
plus utility (e.g., DiGraph,nvert, toParVec,degree,load,UFget,+,*,
sum,subgraph,reverseEdges)

HyGraph
bfsTree, isBfsTree
plus utility (e.g., HyGraph,nvert, toParVec,degree,load,UFget)

(Sp)ParVec
(e.g., +,*,,|,&,>,==,[], abs,max,sum,range,
norm,hist,randPerm, scale, topK)

SpParMat
(e.g., +,* , SpMM, SpMV, SpRef, SpAsgn)

Building blocks
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 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 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.
1. Cull relevant data
2. Build input graph
3. Analyze input graph
4. Visualize result graph

- Gene data
- Email
- Twitter
- Facebook
- Video
- Sensor
- Web
- ...

KNOWLEDGE DISCOVERY WORKFLOW
1. Cull relevant data
2. Build input graph
3. Analyze input graph
4. Visualize result graph

- Gene data
- Email
- Twitter
- Facebook
- Video
- Sensor
- Web
- ...
1. Cull relevant data
2. Build input graph
3. Analyze input graph
4. Visualize result graph

- Gene data
- Email
- Twitter
- Facebook
- Video
- Sensor
- Web
- ...

Disk-based technologies
KDT
Viz tools

Knowledge Discovery Workflow
Outline

• Motivation
• Libraries: CombBLAS and KDT
• Algorithms
• Plans
Two versions of sparse GEMM

1D block-column distribution

$$C_i = C_i + A B_i$$

2D block checkerboard distribution

$$C_{ij} += A_{ik} B_{kj}$$
2D layout for sparse matrices & vectors

- 2D matrix layout wins over 1D with large core counts and with limited bandwidth/compute
- 2D vector layout sometimes important for load balance
- Scalable with increasing number of processes

Matrix/vector distributions, interleaved on each other.

Default distribution in Combinatorial BLAS.
Submatrices are “hypersparse” \((i.e. \text{nnz} \ll n)\)

\[
nnz' = \frac{c}{\sqrt{p}} \rightarrow 0
\]

- A data structure or algorithm that depends on matrix dimension \(n\) (e.g. CSR or CSC) is asymptotically too wasteful for submatrices
- Use doubly-compressed (DCSC) or compressed sparse block (CSB) data structures instead.

Average of \(c\) nonzeros per column

Total Storage:

\[
O(n + nnz) \Rightarrow O(n\sqrt{p} + nnz)
\]
Comparison of SpGEMM implementations

- **SpSUMMA** = 2-D data layout (Combinatorial BLAS)
- **EpetraExt** = 1-D data layout (Trilinos)

(a) R-MAT × R-MAT product (scale 21).
(b) Multiplication of an R-MAT matrix of scale 23 with the restriction operator of order 8.
Indexing sparse arrays in parallel (extract subgraphs, coarsen grids, etc.)

**SpRef:** \( B = A(I, J) \)

**SpAsgn:** \( B(I, J) = A \)

**SpExpAdd:** \( B(I, J) += A \)

\( A, B: \) sparse matrices

\( I, J: \) vectors of indices

**SpRef** using mixed-mode sparse matrix-matrix multiplication (**SpGEMM**). Ex: \( B = A([2,4], [1,2,3]) \)
Sequential \texttt{SpRef} and \texttt{SpAsgn}

\begin{verbatim}
function B = spref(A,I,J)
    R = sparse(1:length(I),I,1,length(I),size(A,1));
    Q = sparse(J,1:length(J),1,size(A,2),length(J));
    B = R*A*Q;
\end{verbatim}

\begin{verbatim}
function C = spasgn(A,I,J,B)
    [ma,na] = size(A);
    [mb,nb] = size(B);
    R = sparse(I,1:mb,1,ma,mb);
    Q = sparse(1:nb,J,1,nb,na);
    S = sparse(I,I,1,ma,ma);
    T = sparse(J,J,1,na,na);
    C = A + R*B*Q - S*A*T;
\end{verbatim}

\[
A + \begin{pmatrix}
    0 & 0 & 0 \\
    0 & B & 0 \\
    0 & 0 & 0
\end{pmatrix} - \begin{pmatrix}
    0 & 0 & 0 \\
    0 & A(I,J) & 0 \\
    0 & 0 & 0
\end{pmatrix}
\]
1. Form R from I in parallel, on a 3x3 processor grid
Parallel algorithm for \textit{SpRef}

2. \textbf{SpGEMM} using memory-efficient Sparse SUMMA.

\[ C_{ij} += P_{ik} A_{kj} \]

Minimize temporaries by:

- Splitting local matrix, and broadcasting multiple times
- Deleting P (and A if in-place) after forming C=P*A
Strong scaling of SpRef

random symmetric permutation $\Leftrightarrow$ relabeling graph vertices
- RMAT Scale 22; edge factor=8; $a=.6$, $b=c=d=.4/3$
- Franklin/NERSC, each node is a quad-core AMD Budapest
Strong scaling of \textit{SpRef}

Extracts 10 random (induced) subgraphs, each with $|V|/10$ vert. Higher span $\rightarrow$ Decreased parallelism $\rightarrow$ Lower speedup
Outline

• Motivation
• Libraries: CombBLAS and KDT
• Algorithms
• Plans
Coming in v0.2: Attributed Semantic Graphs and Filters

**Example:**
- Vertex types: Person, Phone, Camera
- Edge types: PhoneCall, TextMessage, CoLocation
- Edge attributes: StartTime, EndTime
- Calculate centrality just for PhoneCalls and TextMessages between times sTime and eTime

```python
def vfilter(self, vTypes):
    return self.type in vTypes

def efilter(self, eTypes, sTime, eTime):
    return ((self.type in eTypes) and
             (self.sTime > sTime) and
             (self.eTime < eTime))

wantedVTypes = (People)
wantedETypes = (PhoneCall, TextMessage)
start = dt.now() - dt.timedelta(hours=1)
end = dt.now()
bc = G.centrality('approxBC', filter=
    ((vfilter, wantedVTypes),
     (efilter, wantedETypes,
      start, end)))```
Options and issues in implementing filters

- Prefilter to extract the relevant subgraph
  - Simplest solution
  - Too much memory or time for some applications
Options and issues in implementing filters

• Prefilter to extract the relevant subgraph
  – Simplest solution
  – Too much memory or time for some applications

• Write filters as semiring ops in C++, wrap in Python
  – Can get good performance at CombBLAS level
  – Inflexible, hard to write new filters

SEJITS?
Options and issues in implementing filters

- Prefilter to extract the relevant subgraph
  - Simplest solution
  - Too much memory or time for some applications
- Write filters as semiring ops in C++, wrap in Python
  - Can get good performance at CombBLAS level
  - Inflexible, hard to write new filters
- Write filters in Python, call back from CombBLAS
  - Very flexible
  - But slow
Options and issues in implementing filters

- Prefilter to extract the relevant subgraph
  - Simplest solution
  - Too much memory or time for some applications

- Write filters as semiring ops in C++, wrap in Python
  - Can get good performance at CombBLAS level
  - Inflexible, hard to write new filters

- Write filters in Python, call back from CombBLAS
  - Very flexible
  - But slow

- Need a better way! SEJITS?
Plans

- **KDT:**
  - Release V0.2 soon (semantic graphs & attribute filters)
  - Evolve front end to include other parallel graph libraries
  - Selective, embedded JIT specialization to accelerate KDT/CombBLAS: Fox, Kamil et al.
  - Collectives and autotuning for discrete primitives: Williams, Oliker et al.

- More algorithms work (multicore, hybrid)
- More applications (time-dependent path planning, ….)