An Overview of the Combinatorial BLAS
and Knowledge Discovery Toolbox

Aydın Buluç
Lawrence Berkeley National Laboratory
November 10, 2011

In collaboration with:
John Gilbert and Adam Lugowski (UCSB)
Kamesh Madduri (PSU)
Steve Reinhardt (CRAY)
Large graphs are everywhere

Internet structure
Social interactions

WWW snapshot, courtesy Y. Hyun

Scientific datasets: biological, chemical, cosmological, …

Yeast protein interaction network, courtesy H. Jeong
“...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do”
“...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do”
“...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do”

Dealing with software is hard!

High performance computing (HPC) software is harder!
“...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do.”

Dealing with software is hard!

High performance computing (HPC) software is harder!

Deal with parallel HPC software?
Software for Linear Algebra

- Once state-of-the-art
- Now we have PLASMA (for multicores) and MAGMA (for GPUs)
(Proposed) Software for Graph Analysis

Knowledge Discovery Toolbox (KDT)

PyCombBLAS

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
Linear-algebraic primitives

Sparse matrix-matrix Multiplication (SpGEMM)

Element-wise operations

Sparse matrix-sparse vector multiplication

Matrices on semirings, e.g. ($\times$, $+$), (and, or), ($+$, min)

Sparse Matrix Indexing
Breadth-first Search in Combinatorial BLAS

from 1 to 7
parents:

from

\[ \begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & \\
\end{array} \]

\[ A^T \]

\[ X \]

\[ \rightarrow \]

\[ A^T X \]
Select vertex with minimum label as parent

Parents:

from

A^TX

to

A^TX
parents:

from

to

\[ A^T \]

\[ X \]

\[ A^{TX} \]
from $A^T$ to $AX$
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 class hierarchy

- DistMat
  - DenseDistMat
  - SpDistMat
- CommGrid
- FullyDistVec
- SpMat
  - DCSC
  - CSC
  - Triples
  - CSB
- SpDistVec
- DenseDistVec

Combinatorial BLAS functions and operators

Enforces interface only

Polymorphism

HAS A

... HAS A
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><strong>SpGEMM</strong></td>
<td>Sparse Matrix (as friend)</td>
<td>( \mathbf{A}, \mathbf{B} ): sparse matrices</td>
<td>Sparse Matrix</td>
<td>( \mathbf{C} = \mathbf{A} \ast \mathbf{B} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{tr}A ): transpose ( \mathbf{A} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{tr}B ): transpose ( \mathbf{B} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SpMV</strong></td>
<td>Sparse Matrix (as friend)</td>
<td>( \mathbf{A} ): sparse matrices</td>
<td>Sparse or Dense Vector(s)</td>
<td>( \mathbf{y} = \mathbf{A} \ast \mathbf{x} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mathbf{x} ): sparse or dense vector(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{tr}A ): transpose ( \mathbf{A} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SpEWiseX</strong></td>
<td>Sparse Matrices (as friend)</td>
<td>( \mathbf{A}, \mathbf{B} ): sparse matrices</td>
<td>Sparse Matrix</td>
<td>( \mathbf{C} = \mathbf{A} \ast \mathbf{B} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{not}A ): negate ( \mathbf{A} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{not}B ): negate ( \mathbf{B} ) if true</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Reduce</strong></td>
<td>Any Matrix (as method)</td>
<td>( \text{dim} ): dimension to reduce</td>
<td>Dense Vector</td>
<td>\text{sum}(\mathbf{A})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{binop} ): reduction operator</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SpRef</strong></td>
<td>Sparse Matrix (as method)</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></td>
<td>( \mathbf{q} ): column indices vector</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SpAsgn</strong></td>
<td>Sparse Matrix (as method)</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></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><strong>Scale</strong></td>
<td>Any Matrix (as method)</td>
<td>( \text{rhs} ): any object</td>
<td>none</td>
<td>Check guiding</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>principles 3 and 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(except a sparse matrix)</td>
<td></td>
<td></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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Apply</strong></td>
<td>Any Object (as method)</td>
<td>( \text{unop} ): unary operator</td>
<td>None</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(applied to non-zeros)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2D layout for sparse matrices & vectors

Matrix/vector distributions, interleaved on each other.

Default distribution in Combinatorial BLAS.

- 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
2D parallel BFS algorithm

ALGORITHM:
1. Gather vertices in *processor column* [communication]
2. Find owners of the current frontier’s adjacency [computation]
3. Exchange adjacencies in *processor row* [communication]
4. Update distances/parents for unvisited vertices. [computation]
Submatrix storage

Submatrices are “hypersparse” (i.e. $nnz << n$)

- Average of $c$ nonzeros per column

- $\sqrt{p}$ blocks

- $\sqrt{p}$ blocks

- $nnz' = \frac{c}{\sqrt{p}} \rightarrow 0$

- Total Storage: $O(n + nnz) \rightarrow O(n\sqrt{p} + nnz)$

- 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) data structures or compressed sparse blocks (CSB) instead.
2D hybrid parallelism

- Explicitly split submatrices to $t$ (#threads) pieces along the rows.
- Local working set is smaller by a factor of $\sqrt{t}$
  (not a factor of $t$, because $p_r$ is now a factor of $\sqrt{t}$ smaller as well)
NERSC Hopper (Cray XE6, Gemini interconnect AMD Magny-Cours)

Hybrid: In-node 6-way OpenMP multithreading

Graph500: Scale 32, R-MAT with edgefactor=16
Indexing sparse arrays in parallel

Used for extracting subgraphs, coarsening grids, relabeling vertices, etc.

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

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

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

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

<table>
<thead>
<tr>
<th>A, B: sparse matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>I, J: vectors of indices</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c}
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{c}
\text{len(I)}
\end{array}
\]

\[
\begin{array}{c}
m
\end{array}
\]

\[
\begin{array}{c|c|c}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c}
\text{len(J)}
\end{array}
\]

\[
X
\]

A, B: sparse matrices

I, J: vectors of indices

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

\[
\begin{array}{c|c|c}
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{array}
\]

\[
X
\]

A

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c}
\text{len(J)}
\end{array}
\]

\[
X
\]

Q

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c}
\text{len(J)}
\end{array}
\]
Step 1: Form R from I in parallel, on a 3x3 processor grid

Forming Q:
Similar row-wise communication, followed by Q.Transpose()
Parallel algorithm for SpRef

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

Minimize temporaries by:
- Splitting local matrix, and broadcasting multiple times
- Deleting R (and A if in-place) after forming $C = RA$
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 SpRef

Extracts 10 random (induced) subgraphs, each with $|V|/10$ vert. Higher span $\rightarrow$ Decreased parallelism $\rightarrow$ Lower speedup
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 (and now Cray)
- Open source software, released under New BSD license
- v0.1 released March 2011; v0.2 expected November 2011

A general graph library with operations based on linear algebraic primitives
Domain Expert vs. Graph Expert

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

• Hypergraphs and sparse matrices
• Graph primitives \textit{(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
```
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.*, \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
# 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)

Community Detection

Network Vulnerability Analysis

Real applications

Applets

centrality('exactBC')
centrality('approxBC')
pageRank
cluster('Markov')
cluster('spectral')
Graph500

Building blocks

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)

CombBLAS
SpMV, SpMM, etc.

New for v0.2
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))
```
## State of Graph Libraries

<table>
<thead>
<tr>
<th>Package</th>
<th>Target users</th>
<th>Interface</th>
<th>Supported memory*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Graph-alg devs</td>
<td>Domain experts</td>
<td></td>
</tr>
<tr>
<td>Pegasus</td>
<td>X</td>
<td>Hadoop</td>
<td>Distributed on-disk</td>
</tr>
<tr>
<td>Pregel</td>
<td>X</td>
<td>C++</td>
<td>Distributed on-disk</td>
</tr>
<tr>
<td>PBGL</td>
<td>X</td>
<td>C++</td>
<td>Distributed in-memory</td>
</tr>
<tr>
<td>MTGL</td>
<td>X</td>
<td>C++</td>
<td>Shared</td>
</tr>
<tr>
<td>SNAP (GA Tech)</td>
<td>X</td>
<td>C</td>
<td>Shared</td>
</tr>
<tr>
<td>SNAP (Stanford)</td>
<td>X</td>
<td>X</td>
<td>C++ / NodeXL</td>
</tr>
<tr>
<td>GraphLab</td>
<td>X</td>
<td>C++</td>
<td>Shared</td>
</tr>
<tr>
<td>CombBLAS</td>
<td>X</td>
<td>C++</td>
<td>Shared or distributed, in-memory</td>
</tr>
<tr>
<td>KDT</td>
<td>X</td>
<td>X</td>
<td>Python</td>
</tr>
</tbody>
</table>
Some future work

**Functionality axis:**
KDT Release V0.2 soon (semantic graphs & attribute filters)

**Performance axis:**
Optimal processor grid dimensions ($p = p_r \times p_c$) depend on:
- Graph size and density
- Desired concurrency
- Target architecture

*Great opportunity for autotuning.*

Tuning collectives performance
- Non-torus partitions -> unpredictable performance
- Topology aware collectives (Edgar Solomonik’s work)

*Great opportunity for BlueGene*
References


**Breadth-first search:** Aydın Buluç and Kamesh Madduri. Parallel breadth-first search on distributed memory systems. To appear, Supercomputing (SC'11).
