High-Productivity and High-Performance Analysis of Filtered Semantic Graphs

Aydı̇n Buluç
Lawrence Berkeley National Laboratory
February 28, 2013
SIAM CSE, Boston
Acknowledgments

Key contributors, past and present:

• Erika Duriakova (University College Dublin)
• Armando Fox (UC Berkeley)
• John Gilbert (UC Santa Barbara)
• Shoaib Kamil (MIT)
• Adam Lugowski (UC Santa Barbara)
• Lenny Oliker (Berkeley Lab)
• Steve Reinhardt (Cray Inc / YarcData)
• Sam Williams (Berkeley Lab)

This work is funded by:

U.S. Department of Energy
Office of Science
A useless binary graph

Good for benchmarking though (i.e. Graph500)
A useful semantic graph

class edge_attr:
    isText
    isPhoneCall
    weight
class edge_attr:
    isText
    isPhoneCall
    weight

G.addEFilter(lambda e: e.weight > 0)
Edge filter illustration

G.addEFilter(lambda e: e.weight > 0)
G.addEFilter(lambda e: e.isPhoneCall)

class edge_attr:
    isText
    isPhoneCall
    weight
The need for filters

Graph of text & phone calls

Betweenness centrality

Betweenness centrality on text messages

Betweenness centrality on phone calls
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

Discrete structure analysis
Graph theory
Computers
KDT is higher level (graph abstractions)
- Combinatorial BLAS is for performance
Breadth-first search in the language of linear algebra
Particular semiring operations:

**Multiply:** select

**Add:** minimum

parents:

\[ \begin{array}{ccc}
1 & & 7 \\
1 & & 7 \\
\end{array} \]

\[ A^T \]

\[ X \]

\[ A^T X \]
Multiple traverses outgoing edges
Add chooses among incoming edges

parents:

from

1

1

2

4

7

4

2

7

1

2

4

A

T

X

A^T X
Select vertex with minimum label as parent

from

to

parents:
Result: Deterministic breadth-first search

from

1

7

to

1

7

$A^T$

X

$A^TX$
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
• Version 0.2 released in March 2012; Version 0.3 expected in a month

Lugowski, Alber, B., Gilbert, Reinhardt, Teng, and Waranis. A flexible open-source toolbox for scalable complex graph analysis. SIAM Conference on Data Mining (SDM), 2012
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')
```

Algorithm implementation is agnostic to the filters applied

Lugowski, B., Gilbert, Reinhardt. Scalable complex graph analysis with the knowledge discovery toolbox. In ICASSP, 2012
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-agnostic 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>
Targeting “domain experts”

- Performance
- Conceptual simplicity
- Customizability

- Combinatorial BLAS
- KDT with just in time compilation
- KDT’s semantic graphs
- KDT’s non-semantic graphs
Problems with Customizing in KDT

• Filtering on attributed semantic graphs is slow
  • In plain KDT, filters are pure Python functions.
  • Requires a per-vertex or per-edge upcall into Python
  • Can be as slow as 80X compared to pure C++

• Adding new graph algorithms to KDT is slow
  • A new graph algorithm = composing linear algebraic primitives + customizing the *semiring* operation
  • *Semirings* in Python; similar performance bottleneck
Review: Selective Embedded Just In Time Specialization (SEJITS)

Catanzaro, Kamil, Lee, Asanovic, Demmel, Keutzer, Shalf, Yelick, Fox. SEJITS: Getting productivity and performance with selective embedded JIT specialization. PMEA, 2009
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

B., Duriakova, Gilbert, Fox, Kamil, Lugowski, Oliker, Williams. High-Performance and High-Productivity Analysis of Filtered Semantic Graphs, *IPDPS*, 2013
Details about the experimental setting

- Filtered breadth-first search and maximal independent set
- Edge values are generated to guarantee a particular filter permeability by weighting the random number generator.

```python
struct TwitterEdge {
    bool follower;
    time_t latest; // set if count > 0
    short count; // number of retweets
};
```

The edge filter written in Python:
(translated to C++ on the fly by SEJITS)

```python
class MyFilter(PcbFilter):
    def __init__(self, target_date):
        self.target = strptime(target_date)

    def filter(e):
        # if it is a retweet edge
        if (e.count > 0 and
            # and it is before the target date
            e.latest < self.target):
            return True
        else:
            return False
```
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]
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>
<th>Tweet</th>
<th>Follow</th>
<th>Tweet&amp;follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>0.5</td>
<td></td>
<td>0.7</td>
<td>65.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Medium</td>
<td>4.2</td>
<td></td>
<td>14.2</td>
<td>386.5</td>
<td>4.8</td>
</tr>
<tr>
<td>Large</td>
<td>11.3</td>
<td></td>
<td>59.7</td>
<td>589.1</td>
<td>12.5</td>
</tr>
<tr>
<td>Huge</td>
<td>16.8</td>
<td></td>
<td>102.4</td>
<td>634.2</td>
<td>15.6</td>
</tr>
</tbody>
</table>

Statistics about the largest strongly connected components of the Twitter graphs:

<table>
<thead>
<tr>
<th></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>
A *roofline model* for shows how SEJITS moves KDT analytics from being Python *compute bound* to being *bandwidth bound*.
As the compute limitations are lifted, parallel scaling gets **harder** due to higher bandwidth/sec requirements of the computation.
Roofline analysis: Why SEJITS+KDT works?

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.
Main contribution

- Both Boost Graph Library (BGL) and Parallel Boost Graph Library (PBGL) implement the filtered graph abstraction.
- Why do we re-invent the wheel?

- High-productivity programming
- Targeting domain scientists

```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
```
Conclusions

• **KDT + Combinatorial BLAS**: Making parallel graph analysis accessible to domain scientists.
• Layered **software architecture** allows concurrent advances in performance and functionality.
• High-performance **filtered semantic graph processing** is possible without changes from the graph algorithm developer.
• Possible to write callbacks in high-level language while retaining low-level language performance
• Possible to define datatypes at runtime [Ongoing work]