Fast Sparse Matrix and Sparse Vector Multiplication on the GPU

May 25, 2015
Carl Yang, Yangzihao Wang, John D. Owens
University of California, Davis
Applications of Interest

- Bioinformatics
- Social network analysis
- Computer vision
- Fraud detection
- Urban planning
Challenges with Graph Analysis

- Size
  - Too big to be analyzed without help of computer
- Structure
  - Scale-free, power-law, mesh, road network
  - Memory access pattern varies
Linear Algebra

- Linear algebra is a powerful way of thinking about graph algorithms:
  - Fixed communication patterns
  - Operations on matrix blocks exploit caches
- How do we test validity of this idea?
Duality between SpMV and BFS

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0
\end{bmatrix}
\times
\begin{bmatrix}
1 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
\]
Duality between SpMV and BFS

\[
\begin{align*}
0 & 0 0 0 0 0 \\
1 & 0 0 0 0 0 \\
0 & 1 0 0 0 0 \\
0 & 1 1 0 0 0 \\
0 & 0 1 1 0 0 \\
0 & 0 0 1 1 0 \\
\end{align*}
\times
\begin{align*}
1 & 0 \\
0 & 1 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
\end{align*}
= 
\begin{align*}
0 & 0 \\
0 & 1 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
\end{align*}
\]
Duality between SpMV and BFS
Duality between SpMV and BFS

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0
\end{bmatrix} \times \begin{bmatrix}
0 \\
0 \\
1 \\
0 \\
0
\end{bmatrix} = \begin{bmatrix}
0 \\
0 \\
1 \\
0 \\
1
\end{bmatrix}
\]

GABB 2015
Related Work

- Sparse matrix sparse vector multiplication (SpMSpV) has been implemented on distributed systems
  - Gilbert, Reinhardt and Shah, 2007
  - Buluç and Madduri, 2011
SpMV

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
1 \\
1 \\
0
\end{bmatrix}
=
\begin{bmatrix}
0 \\
0 \\
1 \\
1 \\
0
\end{bmatrix}
\begin{bmatrix}
0 \\
1 \\
2 \\
2 \\
-
\end{bmatrix}
\]
SpMSpV

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
\end{array}
\]

Want: Use (2, 3) to find product
Linear Combination

Op2: Set union

\[
\begin{pmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
1 & 1 \\
\end{pmatrix}
\begin{pmatrix}
0 \\
0 \\
0 \\
1 \\
1 \\
\end{pmatrix}
= 
\begin{pmatrix}
0 \\
0 \\
0 \\
1 \\
1 \\
\end{pmatrix}
\begin{pmatrix}
3 & 4 \\
4 \\
\end{pmatrix}
\]

Op1: Select

\[
\begin{pmatrix}
0 & 0 & 1 & 1 & 0 \\
\end{pmatrix}
\begin{pmatrix}
2 & 3 \\
\end{pmatrix}
\]

GABB 2015
What is unique about GPU?

• Many simple processors (2880 cores for K40c) to perform SIMD processing
• Native support for gather and scatter operations
  • Gather: \( b[i] = a[k[i]] \)
  • Scatter: \( b[k[i]] = a[i] \)
Each BFS iteration consists of:

1) StreamCompact
2) Gather
3) Sort
4) Scatter
Visual Representation

Op2: Set union

\[
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 1 \\
1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
1 \\
\end{bmatrix}
= \begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
1 \\
\end{bmatrix}
\]

Op1: Select

\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 \\
\end{bmatrix}
\]

{3, 4} {4}

1) StreamCompact
Visual Representation

Op2: Set union

\[
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
1 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
1 \\
1 \\
\end{bmatrix}
\]

\[
= 
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
1 \\
1 \\
\end{bmatrix}
\]

\[
\{3, 4\}
\]

2) Gather

\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 \\
\end{bmatrix}
\]

\[
\{2, 3\}
\]

1) StreamCompact

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
\end{array}
\]

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
\end{array}
\]

GABB 2015
Visual Representation

Op2: Set union

4) Scatter

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
\{3, 4\} & \{4\} & \{1\} & \{3, 4\} \\
\end{array}
\]

4) Scatter

Op1: Select

1) StreamCompact

2) Gather

\[
\begin{array}{cccc}
0 & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
\{2, 3\} & \{3, 4\} & \{3, 4\} & \{3, 4\} \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 4 & 3 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

GABB 2015
1) StreamCompact

Frontier: 1 0 1 1 0 0 0 0

Want: compact array (0 2 3)
1) StreamCompact (cont'd)

<table>
<thead>
<tr>
<th>Num</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontier</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1) Scan | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |
2) Compact | 0 | 2 | 3 |

For all the frontier values that are 1, store the corresponding Num in location given by Scan.
1) StreamCompact (cont'd)

<table>
<thead>
<tr>
<th>Num</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontier</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1) Scan

| 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 |

2) Compact

| 0 | 2 | 3 |

For all the frontier values that are 1, store the corresponding Num in location given by Scan.
1) StreamCompact (cont'd)

<table>
<thead>
<tr>
<th>Num</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontier</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1) Scan

| Num  | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |

2) Compact

| Num  | 0 | 2 | 3 |

For all the frontier values that are 1, store the corresponding Num in location given by Scan.
<table>
<thead>
<tr>
<th>Num</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontier</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1) Scan

| 1) Scan | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |

2) Compact

| 2) Compact | 0 | 2 | 3 |

For all the frontier values that are 1, store the corresponding Num in location given by Scan.
1) StreamCompact (cont'd)

<table>
<thead>
<tr>
<th>Num</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontier</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1) Scan

| 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 |

2) Compact

| 0 | 2 | 3 |

For all the frontier values that are 1, store the corresponding Num in location given by Scan.
2) Gather

Given:

Frontier (0 2 3)
RowPtr (0 3 6 9 11 13)
ColInd (1 2 3 0 2 4 4
5 6 5 6 2 5 6)

Want: (1 2 3 4 5 6 5 6)
Given:

Frontier (0 2 3)
RowPtr (0 3 6 9 11 13)
ColInd (1 2 3 0 2 4 4 5 6 5 6 2 5 6)

Want: (1 2 3 4 5 6 5 6)
Given:

Frontier: (0 2 3)
RowPtr: (0 3 6 9 11 13)
ColInd: (1 2 3 0 2 4 4 5 6 5 6 2 5 6)

Want:

(1 2 3 4 5 6 5 6)
2) Gather (cont'd)

Given:
- Frontier: (0 2 3)
- RowPtr: (0 3 6 9 11 13)
- ColInd: (1 2 3 0 2 4 4 5 6 5 6)

Want: (1 2 3 4 5 6 5 6)
Given:

<table>
<thead>
<tr>
<th>Frontier</th>
<th>0</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowPtr</td>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>ColInd</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Want:

| 1  | 2  | 3  | 4  | 5  | 6  | 5  | 6  |
3) Sort

Given: unsorted list
(1 2 3 4 5 6 5 6)

Want: sorted list
(1 2 3 4 5 5 6 6)
4) Scatter

Given sorted or unsorted list:

(1 2 3 4 5 6 5 6)

Want: dense bit vector (which can be used to update BFS result)

(0 1 1 1 1 1 1 1 0)
Generalize to Matrix Multiplication

\[
\begin{align*}
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0
\end{align*}
\]

\[
\begin{align*}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0
\end{align*}
\]

\[x = 0 \]

\[
\begin{align*}
0 & 1 & 2 & 2 & -
\end{align*}
\]
Generalize to Matrix Multiplication

Op2: Set union

\[
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
1 & 1 \\
\end{bmatrix}
\]

Op1: Select

\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 \\
\end{bmatrix}
\]

= \[
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 1 \\
1 & 1 \\
\end{bmatrix}
\]

- We need to change Op1 to 'x' and Op2 to '+'
Generalize to Matrix Multiplication

Op2: '+'

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

{3, 4}  2 x {5, 4}

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>?</td>
</tr>
</tbody>
</table>

Op1: 'x'

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

3 x {7}

We need to change Op1 to 'x' and Op2 to '+'
Generalize to Matrix Multiplication

Op1: 'x'

Op2: '+'

\[
\begin{bmatrix}
0 & 0 & 2 & 3 & 0 \\
0 & 0 & 5 & 0 & 4 \\
4 & 7 & \{3, 4\} & \{4\} & \{3, 4\} \\
\{3, 4\} & \{4\} & \{10, 8\} + \{21\} = \{10, 29\}
\end{bmatrix}
\]

- We need to change Op1 to 'x' and Op2 to '+'
Implementation

1) StreamCompact
2) Gather
3) Sort  (no longer optional)  2b) Scalar Multiply
4) Scatter  3b) Segmented Reduce
Implementation

1) StreamCompact
2) Gather
3) Sort (no longer optional)
4) Scatter

2b) Scalar Multiply
3b) Segmented Reduce
Implementation

1) StreamCompact
2) Gather
   2b) Scalar Multiply
3) Sort
   3b) Segmented Reduce
4) Scatter
Experimental Setup

- **CPU**
  - 2× 3.50 GHz Intel 4-core E5-2637 v2 Xeon CPUs
  - 528 GB of main memory

- **GPU**
  - NVIDIA K40c GPU
  - 15 SMs, 192 ALUs per SM, 745 MHz, 12 GB on-board memory
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Graph (Gunrock)</th>
<th>SpMSpV</th>
<th>SpMV</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>ak2010</td>
<td>0.932</td>
<td>1.686</td>
<td>0.427</td>
<td>45K</td>
<td>25K</td>
</tr>
<tr>
<td>belgium_osm</td>
<td>13.053</td>
<td>63.937</td>
<td>97.280</td>
<td>1.4M</td>
<td>1.5M</td>
</tr>
<tr>
<td>coAuthorsDBLP</td>
<td>2.829</td>
<td>4.530</td>
<td>6.213</td>
<td>0.30M</td>
<td>0.98M</td>
</tr>
<tr>
<td>delaunay_n13</td>
<td>0.820</td>
<td>1.085</td>
<td>0.568</td>
<td>8.2K</td>
<td>25K</td>
</tr>
<tr>
<td>delaunay_n21</td>
<td>2.207</td>
<td>11.511</td>
<td>22.241</td>
<td>2.1M</td>
<td>6.3M</td>
</tr>
<tr>
<td>soc-LiveJournal1</td>
<td>33.953</td>
<td>73.722</td>
<td>214.357</td>
<td>4.8M</td>
<td>68.9M</td>
</tr>
<tr>
<td>kron_g500-log21</td>
<td>15.194</td>
<td>70.935</td>
<td>230.609</td>
<td>2.1M</td>
<td>90M</td>
</tr>
</tbody>
</table>
Results for kron scale 16-21

- SpMSpV
- SpMV
- CPU

Runtime (ms) vs. Edges Traversed (millions)
Scatter and Sort End Up Taking Up Runtime

Normalized Runtime

Dataset

GABB 2015
Scatter Does Not Scale as well as Sort
Conclusions

- A primitive is needed to handle sparse matrix sparse vector multiplication
- For memory-bound problems, additional computations (sorting) can be undertaken to improve memory locality
As fast as Gunrock without two algorithm-specific optimizations

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Runtime (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Graph (Gunrock)</td>
</tr>
<tr>
<td>ak2010</td>
<td>0.932</td>
</tr>
<tr>
<td>belgium_osm</td>
<td>13.053</td>
</tr>
<tr>
<td>coAuthorsDBLP</td>
<td>2.829</td>
</tr>
<tr>
<td>delaunay_n13</td>
<td>0.820</td>
</tr>
<tr>
<td>delaunay_n21</td>
<td>2.207</td>
</tr>
<tr>
<td>soc-LiveJournal1</td>
<td>33.953</td>
</tr>
<tr>
<td>kron_g500-log21</td>
<td>15.194</td>
</tr>
</tbody>
</table>

GABB 2015
Gunrock with two algorithm-specific optimizations off
Spectrum

Specific
- Idempotent discovery
- Native Graph Library (Gunrock)

Faster

Push-pull

BFS (SpMSpV, Gunrock optm. off)

General
- SpMSpV
- SpMV

Slower
Key Questions

- Is there no way to do SpMSpV without sorting?
- Where on the spectrum should GraphBLAS be?
  - Is SpMSpV worth including?
Where does SpMSpV appear?

- Breadth-first search (BFS)
- Maximal independent set (MIS)
- Bipartite graph matching
- Graph coloring
Future Work

- Reduced bit sort instead of full sort
- Try optimizations that make native graph frameworks fast
  - Idempotent discovery (comes with cost of losing work-efficiency, but avoids sorting)
  - Push-pull technique
- Delta-stepping to reduce size of frontier and tune ratio of redundant work:parallelism for specific hardware
Acknowledgements

• Co-authors Yangzihao Wang and John D. Owens
• Thanks to Yuduo Wu, Yuechao Pan and Leyuan Wang
• This work was funded by the DARPA XDATA program and the NSF


Why parallel?

Internet
Social networks

Scientific simulations