Parallel Graph Libraries: Where do we go from here?

Aydın Buluç
Lawrence Berkeley National Laboratory

KDT Spring Mind Meld
March 5, 2012
**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)**

**Domain scientists**

**Graph experts**

**Discrete structure analysis**

**Graph theory**

**Computers**

- KDT is higher level (graph abstractions)
- Combinatorial BLAS is for performance
Outline

• The bottleneck: Communication
• The problem with graph partitioners
• Architectural evolution
• Data diversity: Graph characteristics
• Algorithmic evolution: Beat the worst case
• Functionality evolution: Most important kernels
• Peripherals: Database/visualization integration
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 algorithm: Sparse SUMMA

Based on dense SUMMA
General implementation that handles rectangular matrices
Multiplication with the restriction operator

\[ \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \ast \\ \ast \\ \ast \end{bmatrix} \]
The need to reduce communication

- Normalized communication/computation breakdown
- Scale 23 R-MAT times restriction operator of order 4
Comparison of SpGEMM implementations

(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.

SpSUMMA = 2-D data layout (Combinatorial BLAS)
EpetraExt = 1-D data layout (Trilinos)
Remember the 2D algorithm

\[ Bandwidth = \Theta \left( \frac{dn}{\sqrt{p}} \right) \]
Generalize SUMMA to 2.5D

[Ballard, B., Demmel, Grigori, Schwartz]

Maximum replicas:

\[ c \leq \frac{3\sqrt{p}}{d^{2/3}} \]

Bandwidth:

\[ \Theta\left(\frac{d^{4/3} n}{p^{2/3}}\right) \]

- Better scaling with p
- Worse with d
Recursive all-pairs shortest paths

\( A = A^*; \) \hspace{1cm} \% \text{recursive call}
\( B = AB; \hspace{0.5cm} C = CA; \)
\( D = D + CB; \)
\( D = D^*; \) \hspace{1cm} \% \text{recursive call}
\( B = BD; \hspace{0.5cm} C = DC; \)
\( A = A + BC; \)

+ is “min”, \( \times \) is “add”
Novel 2.5D APSP algorithm

[Solomonik, B., Demmel; 2012]

Cyclic step

Bandwidth: \( W_{bc-2.5D}(n, p) = O(n^2 / \sqrt{cp}) \)

Latency: \( S_{bc-2.5D}(p) = O\left(\sqrt{cp \log^2 (p)}\right) \)

c: number of replicas

Optimal for any memory size!
Novel 2.5D APSP algorithm

![Graph showing performance with different values of n and c]
Outline

- The bottleneck: Communication
- The problem with graph partitioners
- Architectural evolution
- Data diversity: Graph characteristics
- Algorithmic evolution: Beat the worst case
- Functionality evolution: Most important kernels
- Peripherals: Database/visualization integration
1D parallel BFS algorithm

ALGORITHM:
1. Find owners of the current frontier’s adjacency [computation]
2. Exchange adjacencies via all-to-all. [communication]
3. Update distances/parents for unvisited vertices. [computation]
2D parallel BFS algorithm
[B., Madduri, 2011]

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]
Orderings for the CoPapersCiteseer graph
[B, Madduri. Graph Partitioning for Scalable Distributed Graph Computations]
BFS All-to-all phase total communication volume normalized to # of edges (m)

<table>
<thead>
<tr>
<th>Graph name</th>
<th>Natural</th>
<th>Random</th>
<th>PaToH</th>
<th>% compared to m</th>
</tr>
</thead>
<tbody>
<tr>
<td>eu-2005</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coAuthorsDBLP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coPapersCiteseer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kron-simple-logn18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delaunay_n20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rgg_n_2_20_s0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# of partitions: 4, 64
Ratio of max. communication volume across iterations to average communication volume

Graph name

- eu-2005
- coAuthorsDBLP
- coPapersCiteeseer
- kron-simple-logn18
- delaunay_n20
- rgg_n_2_20_s0

Ratio over total volume

# of partitions

4 64
Reduction in total All-to-all communication volume with 2D partitioning

Graph name
- eu-2005
- coAuthorsDBLP
- coPapersCiteseer
- kron-simple-logn18
- delaunay_n20
- rgg_n_2_20_s0

<table>
<thead>
<tr>
<th>Graph name</th>
<th>Natural</th>
<th>Random</th>
<th>PaToH</th>
<th>Ratio compared to 1D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>64</td>
<td>4</td>
<td>64</td>
</tr>
</tbody>
</table>

# of partitions
Outline

• The bottleneck: Communication
• The problem with graph partitioners
• Architectural evolution
• Data diversity: Graph characteristics
• Algorithmic evolution: Beat the worst case
• Functionality evolution: Most important kernels
• Peripherals: Database/visualization integration
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
But they are NOT the same

Low diameter R-MAT graph vs.
Long skinny genome graph

Gene linkage map, courtesy Yan et al.
Parallel BFS strategies

1. Expand current frontier (level-synchronous approach, suited for low diameter graphs)

   - O(D) parallel steps
   - Adjacencies of all vertices in current frontier are visited in parallel

2. Stitch multiple concurrent traversals (Ullman-Yannakakis, for high-diameter graphs)

   - path-limited searches from “super vertices”
   - APSP between “super vertices”
Outline

• **The bottleneck: Communication**
• **The problem with graph partitioners**
• Architectural evolution
• **Data diversity: Graph characteristics**
• **Algorithmic evolution: Beat the worst case**
• Functionality evolution: Most important kernels
• Peripherals: Database/visualization integration
Bottom-up BFS
[Beamer, Asanović, Patterson, 2011]

![Graph Diagram]

<table>
<thead>
<tr>
<th>Step</th>
<th>Frontier Size</th>
<th>Fraction of Runtime</th>
<th>Edge Examinations</th>
<th>Failed Attempts</th>
<th>Fraction Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0.00002</td>
<td>242</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>242</td>
<td>0.01836</td>
<td>5,055,487</td>
<td>2,031,553</td>
<td>0.402</td>
</tr>
<tr>
<td>2</td>
<td>3,023,934</td>
<td>0.63358</td>
<td>2,902,729,050</td>
<td>2,847,737,876</td>
<td>0.981</td>
</tr>
<tr>
<td>3</td>
<td>54,991,174</td>
<td>0.32917</td>
<td>1,309,552,404</td>
<td>1,304,547,038</td>
<td>0.996</td>
</tr>
<tr>
<td>4</td>
<td>5,005,366</td>
<td>0.01755</td>
<td>5,870,543</td>
<td>5,855,182</td>
<td>0.997</td>
</tr>
<tr>
<td>5</td>
<td>15,361</td>
<td>0.00133</td>
<td>15,406</td>
<td>15,368</td>
<td>0.997</td>
</tr>
<tr>
<td>6</td>
<td>38</td>
<td>0.00001</td>
<td>38</td>
<td>38</td>
<td>1.0</td>
</tr>
<tr>
<td>Total</td>
<td>63,036,116</td>
<td>1.0</td>
<td>4,223,223,170</td>
<td>4,160,187,055</td>
<td>0.985</td>
</tr>
</tbody>
</table>

BFS does not have to be O(m) all the time!
Acknowledgments...

David Bader, Grey Ballard, Scott Beamer, Ceren Budak, James Demmel, Armando Fox, John Gilbert, Laura Grigori, Bruce Hendrickson, Shoaib Kamil, Jeremy Kepner, Adam Lugowski, Kamesh Madduri, Lenny Oliker, Steve Reinhardt, Viral Shah, Oded Schwartz, Edgar Solomonik, Sam Williams