Efficient Processing of Large Graphs via Input Reduction

Amlan Kusum, Keval Vora, Rajiv Gupta, Iulian Neamtiu

HPDC’16 – Kyoto, Japan
04 June, 2016
Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution
Graph Processing

- Iterative graph algorithms
- Vertices are processed over continuously
- Highly parallel execution
Graph Processing

- Iterative graph algorithms
- Vertices are processed over continuously
- Highly parallel execution

<table>
<thead>
<tr>
<th>t</th>
<th>v₀</th>
<th>v₁</th>
<th>v₂</th>
<th>v₃</th>
<th>v₄</th>
<th>v₅</th>
<th>v₆</th>
<th>v₇</th>
<th>v₈</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>t₁</td>
<td>0</td>
<td>1</td>
<td>∞</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
<td>10</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>t₂</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>∞</td>
<td>10</td>
<td>11</td>
<td>∞</td>
</tr>
<tr>
<td>t₃</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>t₄</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>t₅</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>t₆</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution

<table>
<thead>
<tr>
<th>$t$</th>
<th>$v_0$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
<th>$v_7$</th>
<th>$v_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>1</td>
<td>$\infty$</td>
<td>1</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>10</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>$\infty$</td>
<td>10</td>
<td>11</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>$t_5$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>$t_6$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution
- Challenging due to ever-growing graph sizes

<table>
<thead>
<tr>
<th>$t$</th>
<th>$v_0$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
<th>$v_7$</th>
<th>$v_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>1</td>
<td>$\infty$</td>
<td>1</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>10</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>$\infty$</td>
<td>10</td>
<td>11</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>$t_5$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>$t_6$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Graph Processing

- Iterative graph algorithms
  - Vertices are processed over continuously
  - Highly parallel execution
- Challenging due to ever-growing graph sizes
- Convergence speed is dependent on initializations

How to find better initializations?
Key Idea

- Compute initial values using a smaller signature of the original graph
  - Generate smaller graph using light-weight input reduction techniques
**Key Idea**

- Compute initial values using a smaller signature of the original graph
- Generate smaller graph using light-weight input reduction techniques
Key Idea

time(Input Reduction) + time(Phase 1) + time(Phase 2) < time(Original)
Outline

- Input Reduction
- Vertex Transformations
- Correctness of Results
- Evaluation
- Conclusion
Input Reduction

- Must be light-weight & general
  - Multilevel graph partitioning [SC’95, SC’01]
    - Matching based contraction [ICPP’95, JPDC’98]
    - Pruning based on edge costs affecting paths [ICDM’10]
  - Gate graph for shortest paths problem [ICDM’11]

- Develop vertex level transformations
  - Easily parallelizable using the vertex centric graph processing systems
Vertex Transformations

- Maintain structural integrity of the graph
  - Preserve the overall connectivity

- Light-weight
  - Local
  - Non-interfering
Vertex Transformations
Vertex Transformations

if (inDegree(v) = 0) then
  ▷ apply $T_1$: drop $v \rightarrow *$
  $E' \leftarrow E' \setminus \text{outEdges}(v)$

if (outDegree(v) = 0) then
  ▷ apply $T_2$: drop $* \rightarrow v$
  $E' \leftarrow E' \setminus \text{inEdges}(v)$
Vertex Transformations

Algorithm 1

Graph Reduction Algorithm.

1: 

Algorithm TRANSFORM \((G(V, E))\)

2: 

\(E_0\)

3: 

for \(v \in V\) do

4: 

5: 

\(\text{if } \text{inDegree}(v) = 0\) then

6: 

\(\triangleright \text{ apply } T_1: \text{ drop } v!\triangleright\)

7: 

\(E_0 \cap \text{outEdges}(v)\)

8: 

9: 

\(\text{if } \text{outDegree}(v) = 0\) then

10: 

\(\triangleright \text{ apply } T_2: \text{ drop } \triangleright!v\)

11: 

\(E_0 \cap \text{inEdges}(v)\)

12: 

13: 

\(\text{if } \text{inDegree}(v) = \text{outDegree}(v) = 1\) then

14: 

\(\triangleright \text{ apply } T_3: \text{ bypass } v\)

15: 

\(E_0 \cap \{u \rightarrow v, v \rightarrow w\}\) \(\bigcup \{u \rightarrow w\}\)

16: 

where \(\{u \rightarrow v, v \rightarrow w\} \subseteq E'\)

17: 

18: 

19: 

\(\text{if } \text{all inNeighbors}(v) \text{ are unchanged}\) then

20: 

\(\triangleright \text{ apply } T_4: \text{ coalesce } v \text{ and inNeighbors}(v)\)

21: 

\(E_0\) coalesce \((G, E_0, v)\)

22: 

end for

23: 

if \(G\) requires further reduction then

24: 

25: 

26: 

27: 

\(\triangleright \text{ apply } T_5: \text{ drop } v\)

28: 

\(E_0 \cap \{v \rightarrow w\}\)

29: 

30: 

31: 

\(\text{if } \text{inDegree}(v) > \text{threshold}\) then

32: 

\(\triangleright \text{ apply } T_6: \text{ drop some } \triangleright!v\)

33: 

\(E_0 \cap \{\text{inEdges}(v)\}\)

34: 

35: 

36: 

end if

37: 

end if

38: 

return \(E_0\) of \(G_0\)

39: 

end algorithm

40: 

Algorithm COALESCE \((G(V, E), E_0, v)\)

41: 

42: 

43: 

44: 

45: 

\(E_0 \cap \{w \rightarrow v\}\)

46: 

47: 

48: 

49: 

\(E_0 \cap \{u \rightarrow w\}\)

50: 

\(E_0 \cap \{v \rightarrow u\}\)

51: 

end for

52: 

end for

53: 

return \(E_0\) of \(G_0\)

54: 

end algorithm
Vertex Transformations

**Algorithm 1**

```plaintext
Algorithm TRANSFORM(G(V, E))

E_0 = E

for v \in V do
  if (inDegree(v) = 0) then
    \(\triangleright\) apply \(T_1\): drop \(v!\)
    \(E_0 = E_0 \cap \text{outEdges}(v)\)
  if (outDegree(v) = 0) then
    \(\triangleright\) apply \(T_2\): drop \(\triangleright\)
    \(E_0 = E_0 \cap \text{inEdges}(v)\)
  if (inDegree(v) = outDegree(v) = 1) then
    \(\triangleright\) apply \(T_3\): bypass \(v\)
    \(E_0 = E_0 \cap \{u!v, v!w\} \setminus \{(u!w)\}\)
  if (all inNeighbors(v) are unchanged) then
    \(\triangleright\) apply \(T_4\): coalesce \(v\) and inNeighbors(v)
    \(E_0 = \text{coalesce}(G, E_0, v)\)
  end if
end for

if (G requires further reduction) then
  for v \in V s.t. v is unchanged do
    if (w \in \text{outNeighbors}(v) s.t. w is unchanged and \(\text{outNeighbors}(v) \cap \text{inNeighbors}(w) \neq \emptyset\)) then
      \(\triangleright\) apply \(T_5\): drop \(v!w\)
      \(E' \leftarrow E' \setminus \{(v \rightarrow w)\}\)
    end if
  end for
end if

return \(E_0\) of \(G_0\)
```

**Algorithm 2**

```plaintext
Algorithm COALESCE(G(V, E), E_0, v)

for (w \in \text{inEdges}(v)) do
  \(E_0 = E_0 \cap \{w!v\}\)
  for (u \in \text{inEdges}(w)) do
    \(E_0 = E_0 \cap \{u!w\}\)
    \(E_0 = E_0 \setminus \{u!v\}\)
  end for
  for (w!u) \in \text{outEdges}(w) do
    \(E_0 = E_0 \setminus \{w!u\}\)
    \(E_0 = E_0 \setminus \{(v \rightarrow u)\}\)
  end for
end for

return \(E_0\) of \(G_0\)
```
Vertex Transformations
Other Details

- More vertex transformations
  - Some relax structural integrity
- Order of transformations
  - Unified graph reduction algorithm
Processing workflow

Original Graph

Input Reduction

Reduced Graph

Process (Phase 1)

Process (Phase 2)

Graphs with node labels and edge labels.
Processing workflow

1. Input Reduction

2. Reduced Graph

3. Process (Phase 1)

4. Process (Phase 2)

5. Reduced Graph

6. Original Graph
Input Reduction
Input Reduction
Input Reduction
Input Reduction
Input Reduction
Input Reduction
Workflow

1. Input Reduction
2. Reduced Graph
3. Process (Phase 1)
4. Process (Phase 2)
5. Reduced Graph
6. Original Graph
Workflow
Processing Reduced Graph

- Use the original iterative algorithm
Workflow

Original Graph → Input Reduction

1. Input Reduction

2. Reduced Graph

3. Process (Phase 1)

4. Process (Phase 2)

5. Process (Phase 2) → Reduced Graph

6. Reduced Graph → Original Graph
Workflow

1. Input Reduction
2. Reduced Graph
3. Process (Phase 1)
4. Process (Phase 2)
5. Original Graph
6. Reduced Graph
Mapping Results

- Use default values for missing vertices
Processing Original Graph
Workflow

1. Input Reduction
2. Reduced Graph
3. Process (Phase 1)
4. Process (Phase 2)
5. Original Graph
6. Reduced Graph
Workflow

Original Graph → 1. Input Reduction → 2. Reduced Graph → 3. Process (Phase 1) → 4. Process (Phase 2) → 5. Reduced Graph → 6. Process (Phase 2) → Original Graph
Correctness: SSSP Example
Correctness: SSSP Example
Correctness: SSSP Example
Correctness: SSSP Example
Correctness: SSSP Example
Correctness

- Transformation properties
  - Level of vertices, edges and components
  - Allow developing & reasoning for new transformations
- Algorithm behavior can be reasoned
  - Phase 2 initializations
  - Properties of aggregation function

- Correctness argued for algorithms used
  - 5 accurate and 1 approximate
Evaluation

- Techniques independent of frameworks & processing environment
  - Incorporated in Galois [PLDI’11]
  - Single machine: 24-core, 32GB RAM

- 6 benchmarks
  - PR, SSSP, SSWP, CC, GC, CD

- 4 input graphs
  - Friendster (|E| = 2.6B), Twitter (|E| = 1.5B),
    UKDomain (|E| = 936M), RMAT-24 (|E| = 268M)
Reduction

\[
ERP = \frac{|E_{REDUCED}|}{|E_{ORIGINAL}|} \times 100
\]
Reduction

\[ ERP = \frac{|E_{\text{reduced}}|}{|E_{\text{original}}|} \times 100 \]
Reduction

\[ ERP = \frac{|E_{REDUCED}|}{|E_{ORIGINAL}|} \times 100 \]
Reduction

\[ ERP = \frac{|E_{\text{REduced}}|}{|E_{\text{OriGinal}}|} \times 100 \]
Execution Time

- Speedups over parallel versions
- Speedups increase as ERP decreases up to an extent
- 1.3x - 1.7x for 75% - 50%
- Structural dissimilarity for very low ERP

\[ ERP = \frac{|E_{\text{reduced}}|}{|E_{\text{original}}|} \times 100 \]
Input Reduction

- Transformations are local, i.e., parallelizable
- Higher reduction requires more work
Memory Overhead

- Tracking dissimilar elements
  - Newly added vertices & edges

Diagram: Memory Overhead for Friendster ($|E| = 2.6\text{B}$)

- 75% ERP
- 70% ERP
- 60% ERP
- 50% ERP
- 40% ERP
- 30% ERP

Memory Overhead vs. ERP
Community Detection

Friendster ($|E| = 2.6B$)

- Reduced Graph
- Original Graph

Baseline
- ERP-40
More Results

- Contribution of individual transformations
  - Some transformations more useful than others
  - Different graphs benefit from different transformations
- Improvement in scalability
- Results for all inputs
Conclusion

- Input reduction using transformations that are
  - Light-weight
  - Parallelizable
  - General
- Correctness reasoned using fine-grained transformation properties
- Achieve 1.25-2.14x speedups
Thanks

- GRASP
  - http://grasp.cs.ucr.edu/