Distributed VS Parallel implementations of graph algorithms

Alexis SIRETA, Lazar PETROV
Outline
About graph computing
What is a graph?

A graph is a set of nodes connected to each other by edges.
What kind of graphs?

Edges can be:

- Directed
- Undirected

Unweighted

Directed

Undirected

Weighted

5
A connected graph is a graph in which there is a path between every pair of nodes.
How to represent a graph?

Adjacency matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node1</td>
<td>0</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Node2</td>
<td>7</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>Node3</td>
<td>9</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>
How to represent a graph?

Edge list

<table>
<thead>
<tr>
<th>Nodea</th>
<th>Nodeb</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node1</td>
<td>Node2</td>
<td>7</td>
</tr>
<tr>
<td>Node1</td>
<td>Node3</td>
<td>9</td>
</tr>
<tr>
<td>Node2</td>
<td>Node3</td>
<td>8</td>
</tr>
<tr>
<td>Node2</td>
<td>Node1</td>
<td>7</td>
</tr>
<tr>
<td>Node3</td>
<td>Node1</td>
<td>9</td>
</tr>
<tr>
<td>Node3</td>
<td>Node2</td>
<td>8</td>
</tr>
</tbody>
</table>
What are graphs used for?

Data representation of a wide range of problems:

- Finding shortest path from A to B
- Representing database
- Find related topics

...and plenty more!
Problem!

Graphs are getting **VERY** big:

**Example:**

*Directed network of hyper links between the articles of the Chinese online encyclopedia Baidu.*

17,643,697 edges

source: http://konect.uni-koblenz.de/networks/zhishi-baidu-internallink
Solution!

Use Parallel or Distributed systems
Distributed and Parallel systems
Our Research Project
Goal and Questions

Compare the performances of parallel and distributed implementations of a graph algorithm

Questions:

Can we really compare algorithms running on different architectures?

How do the algorithms scale?

How do they adapt to other architectures?
Hypothesis: Distributed will run slower than parallel for small graphs because of communication latency but will run faster for big graphs because of memory access time
Procedure

- Choose two implementations of one graph algorithm
- Build a theoretical model of the execution time
- Run the algorithms on the Uva cluster
- Explain the results and adapt the theoretical model if needed
Minimum Spanning Tree
What is it?

Is relevant for connected undirected graphs
Which algorithm choose?

Several classical algorithms: Prim, Kruskal, **Boruvka**

**Boruvka**: This is the most used for parallel and distributed implementations, therefore this is the one we chose.

**Parallel implementation**: Bor-el, described in the paper “Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs” by David A. Bader and Guojing Cong.

**Distributed implementation**: GHS, described in “A distributed algorithm for minimum weight spanning trees” by R. G. Gallager, P. A. Humblet and P. M. Spira.
Sequential algorithm
Initialize components
Finding MWOE
Creating new components
Finding MWOE
Creating new component
Here is the Minimum spanning tree

A -- 7 -- B
|       |
D -- 4 -- E
|       |
F -- 6 -- G
|       |
B -- 10 -- C
|       |
E -- 8 -- G
Bor-el algorithm (Parallel)
Example Graph
## Edge list representation

<table>
<thead>
<tr>
<th>MST</th>
<th>A B 7</th>
<th>D B 9</th>
<th>F E 12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A D 4</td>
<td>D E 15</td>
<td>F G 13</td>
</tr>
<tr>
<td></td>
<td>B A 7</td>
<td>D F 6</td>
<td>G E 8</td>
</tr>
<tr>
<td></td>
<td>B C 11</td>
<td>E B 10</td>
<td>G F 13</td>
</tr>
<tr>
<td></td>
<td>B D 9</td>
<td>E C 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B E 10</td>
<td>E D 15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C B 11</td>
<td>E F 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C E 5</td>
<td>E G 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D A 4</td>
<td>F D 6</td>
<td></td>
</tr>
</tbody>
</table>
## Select MWOE

<table>
<thead>
<tr>
<th>MST</th>
<th>A D 4</th>
<th>B A 7</th>
<th>C E 5</th>
<th>D A 4</th>
<th>E C 5</th>
<th>F D 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B 7</td>
<td>D B 9</td>
<td>E C 5</td>
<td>E G 8</td>
<td>F D 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A D 4</td>
<td>D E 15</td>
<td>E D 15</td>
<td>E F 12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B A 7</td>
<td>D F 6</td>
<td>E F 12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B C 11</td>
<td>E B 10</td>
<td>E G 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B D 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B E 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C B 11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C E 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
These are the edges we selected.
These are the edges we selected
Pointer jumping example
Pointer jumping

A

B

C

D

E

F

G
Pointer jumping
Pointer jumping
Create supervertex
## In the edge list

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MST</td>
<td>MST</td>
<td>MST</td>
</tr>
<tr>
<td>A D 4</td>
<td>B A 7</td>
<td>C E 5</td>
</tr>
<tr>
<td>A D 4</td>
<td>B A 7</td>
<td>D A 4</td>
</tr>
<tr>
<td>A D 4</td>
<td>B A 7</td>
<td>E C 5</td>
</tr>
<tr>
<td>A D 4</td>
<td>B A 7</td>
<td>F D 6</td>
</tr>
<tr>
<td>A B 7</td>
<td>D B 9</td>
<td>F E 12</td>
</tr>
<tr>
<td>A B 7</td>
<td>D B 9</td>
<td>F G 13</td>
</tr>
<tr>
<td>A B 7</td>
<td>D B 9</td>
<td>G E 8</td>
</tr>
<tr>
<td>A B 7</td>
<td>D B 9</td>
<td>G F 13</td>
</tr>
<tr>
<td>B C 11</td>
<td>D F 6</td>
<td>E B 10</td>
</tr>
<tr>
<td>B C 11</td>
<td>D F 6</td>
<td>E B 10</td>
</tr>
<tr>
<td>B C 11</td>
<td>D F 6</td>
<td>E B 10</td>
</tr>
<tr>
<td>B C 11</td>
<td>D F 6</td>
<td>E B 10</td>
</tr>
<tr>
<td>B D 9</td>
<td>E C 5</td>
<td>E D 15</td>
</tr>
<tr>
<td>B D 9</td>
<td>E C 5</td>
<td>E D 15</td>
</tr>
<tr>
<td>B D 9</td>
<td>E C 5</td>
<td>E D 15</td>
</tr>
<tr>
<td>B D 9</td>
<td>E C 5</td>
<td>E D 15</td>
</tr>
<tr>
<td>B E 10</td>
<td>E F 12</td>
<td>E G 8</td>
</tr>
<tr>
<td>B E 10</td>
<td>E F 12</td>
<td>E G 8</td>
</tr>
<tr>
<td>B E 10</td>
<td>E F 12</td>
<td>E G 8</td>
</tr>
<tr>
<td>B E 10</td>
<td>E F 12</td>
<td>E G 8</td>
</tr>
<tr>
<td>C B 11</td>
<td>F D 6</td>
<td></td>
</tr>
<tr>
<td>C B 11</td>
<td>F D 6</td>
<td></td>
</tr>
<tr>
<td>C B 11</td>
<td>F D 6</td>
<td></td>
</tr>
<tr>
<td>C B 11</td>
<td>F D 6</td>
<td></td>
</tr>
</tbody>
</table>
In the edge list

<table>
<thead>
<tr>
<th>A A 7</th>
<th>A A 9</th>
<th>A C 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>A A 4</td>
<td>A C 15</td>
<td>A C 13</td>
</tr>
<tr>
<td>A A 7</td>
<td>A A 6</td>
<td>C C 8</td>
</tr>
<tr>
<td>A C 11</td>
<td>C A 10</td>
<td>C A 13</td>
</tr>
<tr>
<td>A A 9</td>
<td>C C 5</td>
<td></td>
</tr>
<tr>
<td>A C 10</td>
<td>C A 15</td>
<td></td>
</tr>
<tr>
<td>C A 11</td>
<td>C A 12</td>
<td></td>
</tr>
<tr>
<td>C C 5</td>
<td>C C 8</td>
<td></td>
</tr>
<tr>
<td>A A 4</td>
<td>A A 6</td>
<td></td>
</tr>
</tbody>
</table>
Find Mwoe
Found Spanning tree

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>D</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>C</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>E</td>
<td>10</td>
</tr>
</tbody>
</table>
Theoretical analysis of Bor-el
Size of graph in memory

Number of edges

$4E \log(N) / p + 2E \log(E) / p$

Number of processors

N: number of nodes

2 times each edge

log(N) size of one node in memory

2 nodes id per edge

Size of weights in memory

N : number of nodes

log(N) size of one node in memory
Average number of edges

$E = k \cdot N - \frac{N}{4} \cdot (2 \cdot k + 1)$

$E$ decreases of at least $N/2$ each iteration. Let's say $E = kN$
Memory access time

Main memory

- cache1
  - cache2
- cache1
  - cache2
- cache1
  - cache2

1 CC

10 CC

100 CC
Memory access time

Size of cache 1

\[ \min\left(1, \frac{s_1}{S}\right) \cdot 1 + \left(1 - \min\left(1, \frac{s_1}{S}\right)\right) \cdot \left(\min\left(1, \frac{s_2}{S}\right) \cdot 10 + \left(1 - \min\left(1, \frac{s_2}{S}\right)\right) \cdot 100 \cdot p\right) \]

Size of cache 2

Size of graph in memory
Memory access time

\[ \text{k=N} \]
\[ s_1 = 16 \text{ kb} \]
\[ s_2 = 4 \text{ Mb} \]
\[ p=2 \]
Number of memory accesses

Formula given by the paper on bor-el

$$\left(\frac{8E+N+N\log(N)}{p} + \frac{4EC\log\left(\frac{2E}{P}\right)}{p}\right)\log(N)$$

C is an unknown constant: using their experimental results we found it is around 3.21
Computation complexity

Formula given by the paper on bor-el

\[ \left( \frac{E}{p} \right) \log(E) \log(N) \]
Plot execution time

\[ k = N \]
\[ s_1 = 16 \text{ kb} \]
\[ s_2 = 4 \text{ Mb} \]
\[ p = 2^{-10} \]
Plot execution time

- $p=2$
- $p=10$
Analysis

Plot does not vary with p because time highly dominated by memory access for very big graphs.
GHS algorithm (Distributed)
Example graph
State of each edge

- **Branch** edges are those that have already been determined to be part of the MST.
- **Rejected** edges are those that have already been determined not to be part of the MST.
- **Basic** edges are neither branch edges nor rejected edges.
State of each edge

Each processor stores:

- The state of any of its incident edges, which can be either of \{basic, branch, reject\}
- Identity of its fragment (the weight of a core edge - for single-node fragments, the proc. id)
- Local MWOE
- MWOE for each branching-out edge
- Parent channel (route towards the root)
- MWOE channel (route towards the MWOE of its appended subfragment)
Type of messages

- **New fragment(identity):** coordination message sent by the root at the end of a phase
- **Test(identity):** for checking the status of a basic edge
- **Reject, Accept:** response to Test
- **Report(weight):** for reporting to the parent node the MWOE of the appended subfragment
- **Merge:** sent by the root to the node incident to the MWOE to activate union of fragments
- **Connect(My Id):** sent by the node incident to the MWOE to perform the union
Phase 0: Every node is a fragment

...And every node is the root of its fragment
Phase 1: Find MWOE
Phase 1: select new root
Phase 1: root broadcast new identity
Phase 1: Find MWOE

Diagram:
- Node 4: Test 7, reject
- Node 5: Accept 10, reject

Connections:
- 4 to 4: 9
- 4 to 5: 15
- 5 to 5: 8
- 4 to 5: 13
Phase 1: Find MWOE
Phase 1: Report to root
Phase 1: Send connect
Phase 1 : New root
Phase 1: Broadcast ID
Phase 1: MST!
Theoretical analysis of GHS
Theoretical execution time

Number of messages sent per node:

Max size of messages sent:

Speed of connection:

\[ \frac{(2E + 5N\log(N) - 1) + 3N}{N} \]

\[ \log(E) + \log(8N) \]

1 Gb/s
Analysis

Theoretically the distributed algorithm is **ALWAYS** way faster than the parallel one.

This is true with our hypothesis of a network **without latencies** and **one host per node**.
Experiments
The Uva cluster

18 nodes with 16 cores each

Max graph size = 82656 edges
Initially chose a **Python** implementation: Did not run properly on the cluster

Ran $N$ times (in parallel) the whole algorithm
Ghs implementation : C with MPI

Then chose a C implementation using MPI (Message Passing Interface) to communicate between processes

Did not run the algorithm until the end
Making it work

The C algorithm *worked* for a specific type of graphs

```
0 1 2 3
1 0 4 5
2 4 0 6
3 5 6 0
```
Results

GHS execution time function of number of nodes
Reasons for such different results

Very badly written algorithm

Message queues

Communication latency
Check if algorithm does not send too many messages

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Theoretical value (msg sent)</th>
<th>Experimental value (msg sent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>224</td>
<td>110410</td>
<td>216712</td>
</tr>
<tr>
<td>128</td>
<td>37100</td>
<td>56717</td>
</tr>
<tr>
<td>64</td>
<td>10250</td>
<td>8200</td>
</tr>
<tr>
<td>32</td>
<td>2710</td>
<td>1573</td>
</tr>
</tbody>
</table>
Check if not a queuing problem

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Number of cores</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16</td>
<td>3.153</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>3.583</td>
</tr>
</tbody>
</table>
Communication latency

Add a latency every time a process sends a message

Theoretical latency needed : 0.1 s
Empirical latency found : 0.025 s

Don't forget that the implementation sends twice the theoretical amount of messages !
Communication latency

Add a latency every time a process sends a message

Theoretical latency needed : 0.1 s

Empirical latency found (between two nodes) : 0.025 s

Don't forget that the implementation sends twice the theoretical amount of messages !
Communication latency

There is no latency if we run the algorithm on one node.

Possibly if we run the algorithm on a $N$ core node we match the theoretical speed.
Further work

Investigate the other factors that caused the bad performance

Investigate the best architectures to run the distributed algorithm
Conclusion

Parallel algorithm way faster than the distributed one

Causes of bad performances of GHS is communication latency caused by MPI and bad implementation of the algorithm

Uva cluster is not optimized for algorithms that require a lot of communication

Nevertheless it is possible to find implementations and architectures that will make GHS outperform bor-el and this should be investigated