Overcoming MPI Communication Overhead for Distributed Community Detection

NAW SAFRIN SATTAR
SHAIKH ARIFUZZAMAN

Big Data and Scalable Computing Research Lab

New Orleans, LA 70148 USA
Introduction

• Louvain algorithm
  – A well-known and efficient method for detecting communities

• Community
  – a subset of nodes having more inside connections than outside
Motivation

• Community Detection Challenges
  – Large networks emerging from online social media
    • Facebook
    • Twitter
  – Other scientific disciplines
    • Sociology
    • Biology
    • Information & technology
• Load balancing
  – Minimize communication overhead
  – Reduce idle times of processors leading to increased speedup
Parallelization Challenges

Shared Memory

• Merits
  – Conventional multi-core processors

• Demerits
  – Scalability limited by moderate no. of available cores
  – Physical cores limited for the scalable chip size restriction
  – Shared global address space size limited for memory constraint

Distributed Memory

• Merits
  – utilize a large number of processing nodes
  – freedom of communication among processing nodes through passing messages

• Demerits
  – An efficient communication scheme required
Louvain Algorithm

- Detects community based on modularity optimization
- Better than other community detection algorithms in terms of
  - Computation time and
  - Quality of the detected communities

Modularity Calculation

\[ Q = \frac{1}{2m} \sum_{ij} \left[ A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \]

Here,

\( Q \) = Modularity

\( A_{ij} \) = Link weight between nodes i and j

\( m \) = Total link weight in the network

\( k_i \) = Sum of the link weights attached to node i

\( c_i \) = Community to which node i is assigned

\( \delta(c_i, c_j) \) = Kronecker delta. Value is 1 when nodes i and j are assigned to the same community. Otherwise, the value is 0

Big Data and Scalable Computing Research Lab
Louvain Algorithm

- 2 Phases
  - Modularity Optimization - looking for "small" communities by local optimization of modularity
  - Community Aggregation - aggregating nodes of the same community a new network is built with the communities as nodes
Shared Memory Parallel Algorithm

• Parallelize computational task-wise
  – iterate over the full network
  – the neighbors of a node

• Work done by multiple threads
  – minimize the workload
  – do the computation faster
Distributed Memory Parallel Algorithm

Algorithm 1: Our Parallel Louvain using MPI

Data: Input Graph $G(V,E)$
Result: (Vertex, Community) Pair

1. while increase in modularity do
2.     $G(V,E)$ is divided into $p$ processes;
3.     Each graph$_i$.bin contains $\left\lfloor \frac{n}{p} \right\rfloor$ vertices and corresponding edges in adjacency list format;
4.     for Each processor $P_i$ (executing in parallel) do
5.         Gather_Neighbour_Info();
6.         Compute_Community();
7.         Exchange_Updated_Community();
8.         Resolve_Community_Duality();
9.         Exchange_Duality_Resolved_Community();
10.        Find_Unique_Communities();
11.        Compute_Modularity();
12.        Generate_NextLevel_Graph();
13.        if number_of_communities $< i$ then
14.            $i \leftarrow \frac{\text{number_of_communities}}{2}$;
15.        end
16.     end
17. end
Hybrid Parallel Algorithm

• Both MPI and OpenMP together
• Flexibility to balance between both shared and distributed memory system

☑ Challenge
  ➢ Demerits of Distributed Memory Overweigh the performance
DPLAL- Distributed Parallel Louvain Algorithm with Load-balancing

• Similar approach as Distributed Memory Parallel Algorithm
• Load balancing of Input Graph using Graph-partitioner METIS
• Re-computation required for each function being calculated from Input Graph
Experimental Setup

• Language
  – C++

• Libraries
  – Open Multi-Processing (OpenMP)
  – Message Passing Interface (MPI)
  – METIS

• Environment
  – Louisiana Optical Network Infrastructure (LONI) QB2 compute cluster
    • 1.5 Petaflop peak performance
    • 504 compute nodes
    • over 10,000 Intel Xeon processing cores of 2.8 GHz
## Dataset

<table>
<thead>
<tr>
<th>Network</th>
<th>Vertices</th>
<th>Edges</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>email-Eu-core</td>
<td>1,005</td>
<td>25,571</td>
<td>Email network from a large European research institution</td>
</tr>
<tr>
<td>ego-Facebook</td>
<td>4,039</td>
<td>88,234</td>
<td>Social circles (‘friends lists’) from Facebook</td>
</tr>
<tr>
<td>wiki-Vote</td>
<td>7,115</td>
<td>1,03,689</td>
<td>Wikipedia who-votes-on-whom network</td>
</tr>
<tr>
<td>p2p-Gnutella08, 09, 04, 25, 30, 31</td>
<td>6,301 - 62,586</td>
<td>20,777 - 1,47,892</td>
<td>A sequence of snapshots of the Gnutella peer-to-peer file sharing network for different dates of August 2002</td>
</tr>
<tr>
<td>soc-Slashdot0922</td>
<td>82,168</td>
<td>9,48,464</td>
<td>Slashdot social network from February 2009</td>
</tr>
<tr>
<td>com-DBLP</td>
<td>3,17,080</td>
<td>10,49,866</td>
<td>DBLP collaboration(co-authorship) network</td>
</tr>
<tr>
<td>roadNet-PA</td>
<td>1,088,092</td>
<td>1,541,898</td>
<td>Pennsylvania road network</td>
</tr>
</tbody>
</table>
Speedup Factors of Parallel Louvain Algorithms

(a) Shared Memory Algorithm

(b) Distributed Memory Algorithm

(c) Hybrid Algorithm

Big Data and Scalable Computing Research Lab
Speedup Factor of DPLAL-Distributed Parallel Louvain Algorithm with Load Balancing

(a) Speedup results for large graphs

(b) Speedup results for relatively small graphs
Runtime Analysis of RoadNet-PA Graph with DPLAL algorithm
Runtime of DPLAL Algorithm with Increasing Network Sizes

![Graph showing runtime of DPLAL algorithm with increasing network sizes. The x-axis represents network size, and the y-axis represents time (ms). Different lines represent different processes: gathering neighbour info, exchanging updated community, exchanging duality resolved community, and gathering updated communities. The total duration is also shown.]
Comparison of METIS Partitioning Approaches

![Graph showing comparison of METIS Partitioning Approaches]

Big Data and Scalable Computing Research Lab
Performance Analysis
Sequential Algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>com-DBLP</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Comm. No.</td>
<td>Dev. (%)</td>
</tr>
<tr>
<td>Sequential</td>
<td>109,104</td>
<td>-</td>
</tr>
<tr>
<td>Shared</td>
<td>109,102</td>
<td>.0006</td>
</tr>
<tr>
<td>Distributed</td>
<td>109,441</td>
<td>0.106</td>
</tr>
<tr>
<td>Hybrid</td>
<td>104,668</td>
<td>1.39</td>
</tr>
<tr>
<td>DPLAL</td>
<td>109,063</td>
<td>0.0129</td>
</tr>
</tbody>
</table>

Another MPI based Parallel Algorithm

<table>
<thead>
<tr>
<th></th>
<th>DPLAL</th>
<th>Charith et.al</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network (node) size – Speedup</td>
<td>317,080 – 12, almost double</td>
<td>500,000 - 6</td>
</tr>
<tr>
<td>Speedup for the largest network</td>
<td>4 (1M nodes), same</td>
<td>4 (8M nodes)</td>
</tr>
<tr>
<td>Scalability for Processors</td>
<td>Upto 1000</td>
<td>Upto 16</td>
</tr>
</tbody>
</table>
Conclusion

• Our parallel algorithms for Louvain method demonstrating good speedup on several types of real-world graphs
• Implementation of Hybrid Parallel Algorithm to tune between shared and distributed memory depending on available resources
• Identification of the problems for the parallel implementations
• An optimized implementation DPLAL
  – DBLP network 12-fold speedup.
  – Our largest network, roadNetwork-PA 4-fold speedup for same number of processors
Future Works

• Improve the scalability of our algorithm for large scale graphs with billions of vertices and edges
  – other load balancing schemes to find an efficient load balancing
• Eliminate the effect of small communities hindering the detection of meaningful medium sized communities
• Investigate the effect of node ordering on the performance
  – degree based ordering
  – kcores
  – clustering coefficients