Distributed K-core decomposition using MPI

Penghang Liu
CSE633 presentation
K-Core

- \( k \) core \( G(V, E) \) is the maximal subgraph where each vertex \( v \in V \) is connected to at least \( k \) other vertices.

- \( k \) core is a more reliable approach in dense subgraph discovery than vertex degree.
Given a **undirected unweighted** graph \( G(V, E) \), find the core value \( k_{\text{max}} \) for every vertex \( v \in V \).

The core value \( k_{\text{max}} \) for a vertex \( v \), is the maximum \( k \) core that \( v \) belongs to.

\[
\begin{align*}
k_{\text{max}} &= 0 \\
k_{\text{max}} &= 1 \\
k_{\text{max}} &= 2 \\
k_{\text{max}} &= 3
\end{align*}
\]
Calculating K value

Real K value of its neighbors

Vertex $v_i$:

\[
\begin{array}{c}
3 \\
2 \\
2
\end{array}
\]

\[
Count(k \geq 3) = 1 < 3
\]

\[
Count(k \geq 2) = 3 \geq 2
\]

So $K = 2$
Calculating K value from degree

First, let all $K = \text{degree}$

<table>
<thead>
<tr>
<th>K</th>
<th>K'</th>
<th>K value of neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>5 3 2</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
<td>5 3 2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5 2 2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3 2 2</td>
</tr>
</tbody>
</table>
Solution (single processor)
Input: Adjacent list

Vertex:

$$\begin{array}{cccccccc}
  v_1 & v_2 & v_3 & v_4 & \ldots & v_i & \ldots & v_j & \ldots & v_n \\
\end{array}$$

$$deg_1 = 2$$

$$\begin{array}{c}
v_i \\
v_j \\
\end{array}$$

Initialize $$k_i = deg_i$$:

$$K:$$

$$\begin{array}{cccccccc}
  k_1 & k_2 & k_3 & k_4 & \ldots & k_5 & \ldots & k_6 & \ldots & k_n \\
\end{array}$$

Update K until convergence:
Distribute $n$ vertices to $m$ processors:

Vertex: $v_1$ $v_2$ $v_3$ $v_4$ … $v_5$ … $v_6$ … $v_n$

Vertex $v_i$ is assigned to processor $(i \mod m)$

Initialize each processor:

Partial adjacent list: ...

$\begin{array}{c}
\vdots
\end{array}$

$\begin{array}{c}
v_i \\
v_j \\
v_k
\end{array}$

$\begin{array}{c}
\vdots
\end{array}$

$\begin{array}{c}
\vdots
\end{array}$

$\begin{array}{c}
k_i \\
k_j \\
k_n
\end{array}$

Initialize: $k_i = deg_i$ if $v_i \in P$  
$k_i = \infty$ if $v_i \not\in P$
Sending messages:

Send local k value

\[ \ldots \quad k_i \quad \ldots \quad k_j \quad \ldots \quad k_n \]

Send \( k_i \) to \( v_j \)

Local vertex \( v_j \)

Send \( k_i \) to \( v_k \)

Send to \( v_k \) in processor \((k \mod m)\)

Receiving messages:

On initialization:

\[ k_i \quad k_j = \text{deg}_j \quad k_k = \infty \]

Receive new \( k_k \) from processor \((k \mod m)\)

If \( k_i < k'_i \): Then \( k_i \leftarrow k'_i \), send message of \( k_i \)
Pseudocode for the processor

**Initialization**

```
on initialization do
    changed ← false;
    core ← d(u);
    foreach v ∈ neighbor_V(u) do est[v] ← ∞;
    send ⟨u, core⟩ to neighbor_V(u);
```

**Receive message**

```
on receive ⟨v, k⟩ do
    if k < est[v] then
        est[v] ← k;
        t ← computeIndex(est, u, core);
        if t < core then
            core ← t;
            changed ← true;
```

**Update and send new message**

```
repeat
    if changed then
        send ⟨u, core⟩ to neighbor_V(u);
        changed ← false;
```
Pseudocode for updating K value

Algorithm 2: int computeIndex( int[] est, int u, k)

for i = 1 to k do  count[i] ← 0;
foreach v ∈ neighbor_V(u) do
  j ← min(k, est[v]);
  count[j] = count[j] + 1;
for i = k downto 2 do
  count[i - 1] ← count[i - 1] + count[i];
i ← k;
while i > 1 and count[i] < i do
  i ← i - 1;
return i;

Update $k_i$ based on the current K value of the neighbors of $v_i$. 
Experiment I: same input size, increase number of processors

<table>
<thead>
<tr>
<th>Nodes</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input size</td>
<td>1,200,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Expected result

- Graph depicting the relationship between the number of nodes and runtime.
## Experiment I: same input size, increase number of processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5.037009</td>
</tr>
<tr>
<td>4</td>
<td>2.515878</td>
</tr>
<tr>
<td>8</td>
<td>1.323840</td>
</tr>
<tr>
<td>16</td>
<td>0.916787</td>
</tr>
<tr>
<td>32</td>
<td>0.718203</td>
</tr>
<tr>
<td>64</td>
<td>0.746985</td>
</tr>
<tr>
<td>128</td>
<td>1.018293</td>
</tr>
</tbody>
</table>
Experiment II: Real-world graph vs random graph

<table>
<thead>
<tr>
<th>Nodes</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input 1</td>
<td>YouTube friendships (1,200,000 nodes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input 2</td>
<td>ER model random graph (1,200,000 nodes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Expected result

![Graph showing runtime vs number of nodes for real-world and random graphs](image)
# Experiment II: Real-world graph vs random graph

<table>
<thead>
<tr>
<th>Processor</th>
<th>Time</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5.037009</td>
<td>2.056716</td>
</tr>
<tr>
<td>4</td>
<td>2.515878</td>
<td>0.970089</td>
</tr>
<tr>
<td>8</td>
<td>1.323840</td>
<td>0.499263</td>
</tr>
<tr>
<td>16</td>
<td>0.916787</td>
<td>0.227319</td>
</tr>
<tr>
<td>32</td>
<td>0.718203</td>
<td>0.107626</td>
</tr>
<tr>
<td>64</td>
<td>0.746985</td>
<td>0.077976</td>
</tr>
<tr>
<td>128</td>
<td>1.018293</td>
<td>0.003473</td>
</tr>
</tbody>
</table>

![Graph showing comparison of real-world graph and random graph](attachment:image.png)
Accuracy Validation

```
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 2.txt youtube_true.txt | awk '{print $1-$11}' | lsort -n -lu | wc -l
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 4.txt youtube_true.txt | awk '{print $1-$11}' | lsort -n -lu | wc -l
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 8.txt youtube_true.txt | awk '{print $1-$11}' | lsort -n -lu | wc -l
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 16.txt youtube_true.txt | awk '{print $1-$11}' | lsort -n -lu | wc -l
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 32.txt youtube_true.txt | awk '{print $1-$11}' | lsort -n -lu | wc -l
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 64.txt youtube_true.txt | awk '{print $1-$11}' | lsort -n -lu | wc -l
1134890 0
[penghang@vortex2:/projects/academic/erdem/penghang/PCD/result]$ paste 128.txt youtube_true.txt | awk '{print $1-$11}' | lsort -n -lu | wc -l
1134890 0
```
Challenges

• Proposed experiment: Double the input size as well as the number of processors.

• Real-world data: Different real-world data doesn’t work the same. Can not control input size

• Random graphs: Randomly generated data is so uniform that the communication is finished in a few rounds.

• Data are not exactly distributed equally.
Future Work

Distribute the data by edges instead of nodes?
Reference


Thanks!