Parallel Greedy Graph Matching using an Edge Partitioning Approach

Md. Mostofa Ali Patwary*
Rob H. Bisseling** Fredrik Manne*

*Department of Informatics, University of Bergen, Norway

**Department of Mathematics, Utrecht University, the Netherlands

September 25, 2010
HLPP 2010, Baltimore, Maryland
USA
Outline

Introduction
  Problem Definitions
  Objectives

Matching Algorithms
  Sequential Karp–Sipser Algorithm
  Parallel Matching Algorithm
  Communication Requirements

Experiments
  Experimental Setup
  Experimental Results

Conclusion
Matching

- Given a graph $G = (V, E)$.
- A Matching $M$ is a pairing of adjacent vertices such that each vertex is matched with at most one other vertex.
- In other words, $M$ is the set of independent edges.

Figure: $G = (V, E)$
Matching

- Given a graph $G = (V, E)$.
- A Matching $M$ is a pairing of adjacent vertices such that each vertex is matched with at most one other vertex.
- In other words, $M$ is the set of independent edges.
- $|M| = 2$. 

![Graph G](image)
Matching

- Given a graph $G = (V, E)$.
- A Matching $M$ is a pairing of adjacent vertices such that each vertex is matched with at most one other vertex.
- In other words, $M$ is the set of independent edges.
- $|M| = 2$.
- $|M| = 3$.

Figure: $G = (V, E)$
The Matching Problem

- Find a matching $M$ such that
  - $M$ has maximum cardinality.

Figure: $G = (V, E)$
The Matching Problem

- Find a matching $M$ such that
  - $M$ has maximum cardinality.
  - Edge weight $w$ of $M$ is maximum for edge weighted graph.
  - $w(M) = 11$.

Figure: $G = (V, E)$
The Matching Problem

- Find a matching $M$ such that
  - $M$ has maximum cardinality.
  - Edge weight $w$ of $M$ is maximum for edge weighted graph.
    - $w(M) = 11$.
    - $w(M) = 17$.
- In this work we consider maximum cardinality matching.

Figure: $G = (V, E)$
Applications

- **Combinatorial optimization**, e.g. assignment problem, stable marriage problem.
- **Linear solvers**, e.g. improve pivoting.
- **Load balancing** in parallel computation, e.g. graph partitioning.
- **Bioinformatics**, e.g. alignment problems.
Maximum Cardinality Matching: \( G = (V, E) \)

A general greedy framework:

1: \( M = \emptyset \)
2: while \( E \neq \emptyset \) do
3: Pick the BEST remaining edge \((v, w)\).
4: Add \((v, w)\) to the matching \( M \).
5: Remove all edges incident on \( v \) and \( w \) from \( E \).

Figure: \( G = (V, E) \)
Maximum Cardinality Matching: Example

A general greedy framework:

1. \( M = \emptyset \)
2. while \( E \neq \emptyset \) do
3. Pick the BEST remaining edge \((v, w)\).
4. Add \((v, w)\) to the matching \(M\).
5. Remove all edges incident on \(v\) and \(w\) from \(E\).

![Diagram of a graph with labeled nodes a through f]

Figure: \( M = \emptyset \)
Maximum Cardinality Matching: Example

A general greedy framework:

1: \( M = \emptyset \)
2: while \( E \neq \emptyset \) do
3: Pick the BEST remaining edge \((v, w)\).
4: Add \((v, w)\) to the matching \( M \).
5: Remove all edges incident on \( v \) and \( w \) from \( E \).

Figure: \( M = (d, e) \)
Maximum Cardinality Matching: Example

A general greedy framework:

1. \( M = \emptyset \)
2. \textbf{while} \( E \neq \emptyset \) \textbf{do}
3. Pick the \textbf{BEST} remaining edge \((v, w)\).
4. Add \((v, w)\) to the matching \(M\).
5. Remove all edges incident on \(v\) and \(w\) from \(E\).

\[ M = (d, e) \]
A general greedy framework:

1: \( M = \emptyset \)
2: **while** \( E \neq \emptyset \) **do**
3: Pick the **BEST** remaining edge \((v, w)\).
4: Add \((v, w)\) to the matching \(M\).
5: Remove all edges incident on \(v\) and \(w\) from \(E\).

**Figure:** \( M = (d, e); (a, b) \)
A general greedy framework:

1. \( M = \emptyset \)
2. while \( E \neq \emptyset \) do
3. Pick the BEST remaining edge \((v, w)\).
4. Add \((v, w)\) to the matching \(M\).
5. Remove all edges incident on \(v\) and \(w\) from \(E\).

Figure: \( M = (d, e); (a, b) \)
A general greedy framework:

1. \( M = \emptyset \)
2. \textbf{while} \( E \neq \emptyset \) \textbf{do}
3. Pick the \textbf{BEST} remaining edge \((v, w)\).
4. Add \((v, w)\) to the matching \(M\).
5. Remove all edges incident on \(v\) and \(w\) from \(E\).

\[ M = (d, e); (a, b); (c, f) \]
A general greedy framework:

1: \( M = \emptyset \)
2: \textbf{while} \( E \neq \emptyset \) \textbf{do}
3: \quad \text{Pick the BEST remaining edge} \ (v, w).
4: \quad \text{Add} \ (v, w) \ \text{to the matching} \ M.
5: \quad \text{Remove} \ all \ edges \ incident \ on \ v \ and \ w \ from \ E.

Figure: \( M = (d, e); (a, b); (c, f) \)
▶ **HOW** to choose the **BEST** edge of the remaining edges?
  ▶ What should the criteria be?
▶ Although **exact algorithms** are polynomial, they could be **expensive** in practice.
▶ Therefore, the common choice is **heuristics**, which -
  ▶ gives **high-quality** matchings in many cases.
  ▶ is much **faster** for large problem sizes.
  ▶ is **easier** to implement.
Heuristics - Best Edge

- **Simple greedy** [Möhring and Müller-Hannemann, 1995, Magun, 1998].
  - Picks an edge \((v, w)\) where \(v\) and \(w\) are unmatched vertices.

- **Static Mindegree**
  - Picks the minimum degree unmatched vertex \(v\) and find a lower degree unmatched neighbour \(w\).

- **Dynamic Mindegree** - Updates degree after deletion of edges.

- **Karp–Sipser algorithm** - Keeps track of degree 1 vertices only + Simple greedy [Aronson et al., 1998].
  - This is the method of choice in many cases [Langguth et al., 2010].
Objectives

- Investigate the parallelization of Maximum Cardinality Matching for distributed memory computers.
- The Karp–Sipser algorithm has been picked.
  - High quality matching quickly.
Sequential **Karp–Sipser** Algorithm: **Idea**

- A vertex $v$ is singleton if $d(v) = 1$.
- **Idea:** Match singleton vertices. If there is no singleton vertex, run simple greedy algorithm, that is, pick edges randomly.
Sequential \textbf{Karp–Sipser Algorithm: Details}

1: \( M \leftarrow \emptyset \)
2: \( \textbf{while } E \neq \emptyset \textbf{ do} \)
3: \( \textbf{if } E \text{ has singleton vertices then} \)
4: \( \text{Pick a singleton vertex } v \text{ uniformly at random.} \)
5: \( \text{Let } (v, w) \text{ be the only edge adjacent to } v. \)
6: \( \textbf{else} \)
7: \( \text{Pick an edge } (v, w) \text{ uniformly at random.} \)
8: \( \textbf{Add } (v, w) \text{ to the matching } M. \)
9: \( \textbf{Remove all edges incident on } v \text{ and } w \text{ from } E. \)
10: \( \textbf{return } M \)
Sequential Karp–Sipser Algorithm: Example

Figure: $G = (V, E)$
Sequential Karp–Sipser Algorithm: Example

Figure: $M = \emptyset$
Sequential \textit{Karp–Sipser} Algorithm: Example

Figure: $M = \emptyset$
Sequential Karp–Sipser Algorithm: Example

Figure: $M = (a, b)$
Sequential **Karp–Sipser** Algorithm: Example

**Figure:** $M = (a, b)$
Sequential Karp–Sipser Algorithm: Example

Figure: \( M = (a, b); (i, h) \)
Sequential \textbf{Karp–Sipser} Algorithm: Example

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{example_graph}
\caption{\( M = (a, b); (i, h) \)}
\end{figure}
Sequential **Karp–Sipser Algorithm** Algorithm: Example

Figure: $M = (a, b); (i, h); (d, f)$
Sequential **Karp–Sipser** Algorithm: Example

Figure: $M = (a, b); (i, h); (d, f)$
Sequential **Karp–Sipser** Algorithm: Example

Figure: $M = (a, b); (i, h); (d, f); (c, e)$
Our Parallel Matching Algorithm

- Assume that the graph is **distributed** among the processors.
  - Vertex based distribution (in matrix term, **1D**).
  - Edge based distribution (in matrix term, **2D**).
Our Parallel Matching Algorithm: Idea

- **Idea**: Each processor operates in *synchronized rounds* (BSP).
  - Performs a *local version* of the *sequential* algorithm.
  - Communicates.
  - Processes incoming messages.

- **The reason of using BSP is**:
  - Enhances load balancing by detecting at an earlier stage that a processor has run out of work.
  - Takes some of the *tediousness away of message-passing*.
  - Many *communication optimizations* can be left to the system.
The Parallel Matching Algorithm: Processor $P_i$

1: while $E \neq \emptyset$ do
2:   for Pre-specified number of vertices and $E_i \neq \emptyset$ do
3:     if $E_i$ has singleton vertices then
4:       Pick a singleton vertex $v$.
5:       Let $(v, w)$ be the only edge adjacent to $v$.
6:     else
7:       Pick an edge $(v, w)$ randomly.
8:       Try to match $v$ with $w$.
9:   BSP-Sync()
10:  PROCESS-MESSAGES()
The Parallel Matching Algorithm: Messages

- Original vertex (owned) and ghost vertex.
- Matching requests: *Local* and *Non-Local*.
- Confirmation back and removal of edges.

\[ G = (V, E) \]
The Parallel Matching Algorithm: Messages

- Original vertex (owned) and ghost vertex.
- Matching requests: Local and Non-Local.
- Confirmation back and removal of edges.

![Graph Diagram]

Figure: $G = (V, E)$
The Parallel Matching Algorithm: Messages

**Table:** Summary of message types used.

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match request</td>
<td>Matches a vertex $v$ with $w$</td>
</tr>
<tr>
<td>Confirmation</td>
<td>Confirms success of matching $v$</td>
</tr>
<tr>
<td>Removal</td>
<td>Removes all edges adjacent to $v$</td>
</tr>
<tr>
<td>Handover</td>
<td>Hands over vertex $v$ to a nonowner</td>
</tr>
<tr>
<td>Give-up</td>
<td>Removes a processor from $nonOwners(v)$</td>
</tr>
<tr>
<td>Criticality</td>
<td>Local count of vertex $v$ became 1</td>
</tr>
</tbody>
</table>
Communication Volume: Upper and Lower Bounds

Parallel Matching compared to Parallel Sparse Matrix Vector Multiplication ($SpMV$):

\[ \frac{1}{2} \cdot Vol(SpMV) \leq Vol(Matching) \leq \frac{3}{2} \cdot Vol(SpMV) + R, \]

$R$ represents the number of random match requests that failed during the algorithm.
Test Sets and Experimental Setup

- **Huygens**, an IBM pSeries 575 supercomputer, 104 nodes, each with 16 processors (IBM Power6 dual-core 4.7 GHz) and 128 GByte of memory.
- **Linux, C++** using the BSPonMPI [Suijlen, 2010], IBM XL C/C++ compiler, -O3 optimization level.
- **Mondriaan package** [Vastenhouw and Bisseling, 2005] to distribute the graphs among the processors.
We use 4 different type test sets.

- **Set 1 (rw1-rw10):** 10 real-world graphs.
- **Set 2 (rw11-rw14):** 4 real-world graphs.
- **Set 3 (sw1-sw3):** 3 synthetic small-world graphs.
- **Set 4 (er1-er3):** 3 Erdös-Rényi random graphs [Bader and Madduri, 2006].
Test Sets and Experimental Setup...

### Table: Structural properties of the input graphs.

|     | \(|V|\) | \(|E|\)       | Degree \(\text{avg}\) \(\text{max}\) |     | \(|V|\) | \(|E|\)       | Degree \(\text{avg}\) \(\text{max}\) |
|-----|--------|--------------|--------------------------------------|-----|--------|--------------|--------------------------------------|
| rw1 | 999,999| 3,995,992    | 3 4                                  | rw11| 281,903| 3,985,272    | 14 38,625                             |
| rw2 | 1,585,478| 6,075,348     | 3 5                                  | rw12| 16,783 | 9,306,644    | 554 14,671                            |
| rw3 | 52,804  | 10,561,406    | 200 2,702                            | rw13| 683,446| 13,269,352   | 19 83,470                             |
| rw4 | 2,063,494| 12,964,640    | 6 95                                 | rw14| 343,791| 26,493,322   | 77 434                                |
| rw5 | 63,838  | 14,085,020    | 220 3,422                            |     | sw1    | 50,000       | 14,112,206                            |
|     |         |              |                                      |     | sw2    | 75,000       | 24,466,808                            |
|     |         |              |                                      |     | sw3    | 100,000      | 33,727,170                            |
|     |         |              |                                      |     | er1    | 100,000      | 3,319,658                             |
|     |         |              |                                      |     | er2    | 150,000      | 6,753,302                             |
|     |         |              |                                      |     | er3    | 200,000      | 12,008,022                            |
Experimental Results: Communication Volume

Table: Communication volume in 1000 words for $p = 32$.

<table>
<thead>
<tr>
<th>Name</th>
<th>SpMV 1D</th>
<th>SpMV 2D</th>
<th>Matching 1D</th>
<th>Matching 2D</th>
<th>Name</th>
<th>SpMV 1D</th>
<th>SpMV 2D</th>
<th>Matching 1D</th>
<th>Matching 2D</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw1 (ecology2)</td>
<td>53</td>
<td>51</td>
<td>60</td>
<td>55</td>
<td>rw11 (Stanford)</td>
<td>340</td>
<td>141</td>
<td>479</td>
<td>234</td>
</tr>
<tr>
<td>rw2 (G3_circuit)</td>
<td>81</td>
<td>65</td>
<td>92</td>
<td>73</td>
<td>rw12 (gupta3)</td>
<td>710</td>
<td>44</td>
<td>1,305</td>
<td>61</td>
</tr>
<tr>
<td>rw3 (crankseg_1)</td>
<td>78</td>
<td>78</td>
<td>155</td>
<td>152</td>
<td>rw13 (St_Berk.)</td>
<td>716</td>
<td>448</td>
<td>1,152</td>
<td>812</td>
</tr>
<tr>
<td>rw4 (kkt_power)</td>
<td>118</td>
<td>120</td>
<td>106</td>
<td>107</td>
<td>rw14 (F1)</td>
<td>139</td>
<td>130</td>
<td>148</td>
<td>139</td>
</tr>
<tr>
<td>rw5 (crankseg_2)</td>
<td>92</td>
<td>90</td>
<td>181</td>
<td>171</td>
<td>sw1</td>
<td>1,007</td>
<td>417</td>
<td>2,111</td>
<td>303</td>
</tr>
<tr>
<td>rw6 (af_shell8)</td>
<td>51</td>
<td>47</td>
<td>85</td>
<td>65</td>
<td>sw2</td>
<td>1,957</td>
<td>829</td>
<td>3,999</td>
<td>563</td>
</tr>
<tr>
<td>rw7 (inline_1)</td>
<td>104</td>
<td>105</td>
<td>115</td>
<td>118</td>
<td>sw3</td>
<td>2,017</td>
<td>802</td>
<td>4,255</td>
<td>528</td>
</tr>
<tr>
<td>rw8 (ldoor)</td>
<td>131</td>
<td>128</td>
<td>140</td>
<td>148</td>
<td>er1</td>
<td>1,856</td>
<td>1,133</td>
<td>1,788</td>
<td>1,157</td>
</tr>
<tr>
<td>rw9 (af_shell10)</td>
<td>113</td>
<td>105</td>
<td>169</td>
<td>150</td>
<td>er2</td>
<td>3,451</td>
<td>1,841</td>
<td>3,721</td>
<td>1,635</td>
</tr>
<tr>
<td>rw10 (boneS10)</td>
<td>150</td>
<td>145</td>
<td>228</td>
<td>189</td>
<td>er3</td>
<td>5,476</td>
<td>2,569</td>
<td>6,350</td>
<td>1,990</td>
</tr>
</tbody>
</table>

- 2D takes less communication and moving from 1D to 2D gives a savings of a factor of 2 for Set 3 and 4, even larger savings for Set 2, and a modest gain in Set 1.
Experimental Results: Speedup

How many vertices, $VpR$ to process per round?

Table: Speedup as a function of $VpR$ for $p = 32$.

<table>
<thead>
<tr>
<th>$VpR =</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
<th>1600</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
<th>1600</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw1</td>
<td>0.67</td>
<td>0.74</td>
<td>0.62</td>
<td>0.40</td>
<td>0.24</td>
<td>4.25</td>
<td>5.32</td>
<td>6.15</td>
<td>6.17</td>
<td>6.45</td>
</tr>
<tr>
<td>rw2</td>
<td>0.66</td>
<td>0.72</td>
<td>0.59</td>
<td>0.38</td>
<td>0.20</td>
<td>25.36</td>
<td>18.99</td>
<td>30.55</td>
<td>29.55</td>
<td>30.35</td>
</tr>
<tr>
<td>rw3</td>
<td>12.65</td>
<td>13.07</td>
<td>15.13</td>
<td>14.53</td>
<td>14.42</td>
<td>1.18</td>
<td>1.59</td>
<td>1.83</td>
<td>1.85</td>
<td>1.73</td>
</tr>
<tr>
<td>rw4</td>
<td>1.55</td>
<td>1.30</td>
<td>0.72</td>
<td>0.31</td>
<td>0.17</td>
<td>13.15</td>
<td>16.67</td>
<td>19.54</td>
<td>21.63</td>
<td>24.23</td>
</tr>
<tr>
<td>rw6</td>
<td>6.26</td>
<td>9.29</td>
<td>12.92</td>
<td>14.03</td>
<td>13.82</td>
<td>27.87</td>
<td>31.16</td>
<td>33.85</td>
<td>33.91</td>
<td>33.75</td>
</tr>
<tr>
<td>rw7</td>
<td>9.19</td>
<td>11.17</td>
<td>12.09</td>
<td>12.85</td>
<td>12.88</td>
<td>33.35</td>
<td>40.83</td>
<td>42.18</td>
<td>44.64</td>
<td>42.43</td>
</tr>
<tr>
<td>rw8</td>
<td>6.93</td>
<td>8.45</td>
<td>9.22</td>
<td>9.25</td>
<td>8.83</td>
<td>5.20</td>
<td>6.02</td>
<td>7.64</td>
<td>8.60</td>
<td>9.51</td>
</tr>
<tr>
<td>rw9</td>
<td>6.44</td>
<td>9.66</td>
<td>12.19</td>
<td>13.08</td>
<td>11.50</td>
<td>7.15</td>
<td>9.60</td>
<td>11.00</td>
<td>12.71</td>
<td>13.63</td>
</tr>
<tr>
<td>rw10</td>
<td>7.07</td>
<td>8.41</td>
<td>8.82</td>
<td>7.97</td>
<td>6.60</td>
<td>14.31</td>
<td>15.97</td>
<td>18.14</td>
<td>19.72</td>
<td>21.55</td>
</tr>
</tbody>
</table>

Speedup increases with $VpR$. 
Experimental Results: Matching Quality

How many vertices, $VpR$ to process per round?

Table: Matching quality (in %) as a function of $VpR$ for $p = 32$

<table>
<thead>
<tr>
<th>$VpR$</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
<th>1600</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
<th>1600</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw1</td>
<td>98.15</td>
<td>98.14</td>
<td>98.13</td>
<td>98.08</td>
<td>98.12</td>
<td>rw11</td>
<td>71.75</td>
<td>71.61</td>
<td>71.48</td>
<td>71.32</td>
</tr>
<tr>
<td>rw2</td>
<td>96.71</td>
<td>96.69</td>
<td>96.61</td>
<td>96.52</td>
<td>96.45</td>
<td>rw12</td>
<td>98.31</td>
<td>98.00</td>
<td>97.35</td>
<td>97.35</td>
</tr>
<tr>
<td>rw4</td>
<td>88.55</td>
<td>88.58</td>
<td>88.58</td>
<td>88.57</td>
<td>88.57</td>
<td>rw14</td>
<td>99.54</td>
<td>99.52</td>
<td>99.53</td>
<td>99.51</td>
</tr>
<tr>
<td>rw5</td>
<td>99.26</td>
<td>99.24</td>
<td>99.24</td>
<td>99.20</td>
<td>99.18</td>
<td>sw1</td>
<td>79.81</td>
<td>78.07</td>
<td>77.06</td>
<td>75.66</td>
</tr>
<tr>
<td>rw6</td>
<td>99.93</td>
<td>99.93</td>
<td>99.92</td>
<td>99.93</td>
<td>99.93</td>
<td>sw2</td>
<td>90.74</td>
<td>88.87</td>
<td>86.25</td>
<td>84.09</td>
</tr>
<tr>
<td>rw7</td>
<td>99.56</td>
<td>99.55</td>
<td>99.55</td>
<td>99.54</td>
<td>99.53</td>
<td>sw3</td>
<td>81.87</td>
<td>80.13</td>
<td>78.47</td>
<td>77.29</td>
</tr>
<tr>
<td>rw8</td>
<td>98.58</td>
<td>98.58</td>
<td>98.58</td>
<td>98.58</td>
<td>98.57</td>
<td>er1</td>
<td>97.50</td>
<td>93.45</td>
<td>85.67</td>
<td>78.69</td>
</tr>
<tr>
<td>rw9</td>
<td>99.94</td>
<td>99.94</td>
<td>99.94</td>
<td>99.94</td>
<td>99.94</td>
<td>er2</td>
<td>98.43</td>
<td>95.63</td>
<td>89.12</td>
<td>82.54</td>
</tr>
<tr>
<td>rw10</td>
<td>99.58</td>
<td>99.56</td>
<td>99.55</td>
<td>99.55</td>
<td>99.55</td>
<td>er3</td>
<td>95.98</td>
<td>93.14</td>
<td>88.94</td>
<td>83.42</td>
</tr>
</tbody>
</table>

The matching quality decreases with $VpR$. 
Parallel Matching Algorithm: Maximum Speedup

Figure: Maximum speedup obtained using 1D and 2D.

- Speedup in almost all cases (1D and 2D).
- Test Set 1 and 2 - Same speedup for 1D and 2D.
- Test Set 3 and 4 - 2D is better than 1D.
Parallel Matching Algorithm: Corresponding Quality

Figure: Matching quality in % - Sequential, 1D and 2D.

- Test Set 1 and 2 - Sequential, 1D, and 2D - same quality.
- Test Set 3 - 1D and 2D perform better than Sequential.
- Test Set 4, 2D gives better quality compared to 1D.
We have parallelize a Greedy Graph Matching Algorithm for distributed memory computers.

We have obtained good speedups for many graphs without compromising the quality of the matching.

Edge-based partitioning (2D) gives larger scalability and better matching quality compared to vertex-based partitioning (1D).

\[ \frac{1}{2} \cdot Vol(SpMV) \leq Vol(Matching) \leq \frac{3}{2} \cdot Vol(SpMV) + R. \]

In practice, the range is between 0.63 to 1.95 times \( Vol(SpMV) \) for 2, 4, 8, 16, 32, and 64 processors.
Future Works

- Extend this work for Parallel Maximum Weighted Matching.
- We intend to generalize this approach for the whole class where an edge-based approach will be suitable.
Thank you.
1D and 2D

- In both 1D and 2D cases, we consider only the lower triangle and the edges are unique among the processors.
- The difference between 1D and 2D:
  - For 2D we try to divide the edges equally among the processors.
  - For 1D, we try to divide the vertices equally among the processors.
- But still for 1D case, all the edges of a vertex may not be in the same processor always.
- This way, we can view vertex partitioning as a special case of edge partitioning.
- To keep the parallel matching algorithm unchanged irrespective of partitioning, we did this modification from the conventional 1D.
Why bulk-synchronous parallel (BSP)

- BSP is characterized by alternating between computation phases and communication phases, each ended by a global barrier synchronization.
  - Enhances load balancing by detecting at an earlier stage that a processor has run out of work.
  - BSPlib communication library [Hill et al., 1998] takes some of the tediousness away of message-passing for irregular computations.
  - Many communication optimizations can be left to the system.
The Sequential **KARP–SIPSER** Algorithm: Analysis

- There are two phases of in the execution of the **KARP–SIPSER** algorithm.
  - **Phase 1**: Starts at the begining of the while loop and ends when the current graph has no singleton vertex.
  - **Phase 2**: The remainder of the algorithm.

- If $M_1$ is the set of vertices chosen in Phase 1, then there exists some maximum cardinality matching that contains $M_1$, [Aronson et al., 1998, Fact 1].

- Almost all the remaining vertices are matched by the **KARP–SIPSER** algorithm in the special case where $G$ is a random graph [Aronson et al., 1998, Chebolu et al., 2008].


Bibliography II

University of Florida sparse matrix collection. 
NA Digest, 92.

BSPlib: The BSP programming library. 

Parasol matrices. 
http://www.parallab.uib.no/projects/.
Bibliography III


Suijlen, W. J. (2010). BSPonMPI: An implementation of the BSPlib standard on top of MPI, Version 0.3.