Theory of Parallel and Distributed Systems
(WS2016/17)

Kapitel 3
More Algorithms

Walter Unger
Lehrstuhl für Informatik 1

13:30 Uhr, den 22. November 2016
Inhalt I

1 Colourings I
   - Cycle
   - Trees

2 Eulerian cycle
   - Introduction

3 Matchings
   - Introduction
   - Algorithm
   - Running times
Colouring Problem

- Given undirected graph \( G = (V, E) \) and \( k \in \mathbb{N} \).
- Compute [exists?] Function \( c : V \mapsto \{1, \cdots, k\} \) with:
  - \( \forall \{a, b\} \in E : c(a) \neq c(b) \).
- Colouring number (chromatic index) of \( G \):
  \[
  \chi(G) := \min\{k \mid \exists c : V \mapsto \{1, \cdots, k\} \mid \forall \{a, b\} \in E : c(a) \neq c(b)\}.
  \]

- Colouring problem is NP-complete.
- Let \( G = C_n \), i.e. \( G = (\{v_0, \cdots, v_{n-1}\}, \{\{v_i, v_{(i+1) \mod n}\} \mid 0 \leq i < n\}) \).
- Then we have \( \chi(C_n) \leq 3 \) and \( \chi(C_{2 \cdot n}) \leq 2 \) (\( \chi(C_{2 \cdot n+1}) = 3 \)).
- We do not have a nice order on the nodes:
- let \( \pi(i) \) be a permutation
- Let \( G = C_n \), i.e.
  \[
  G = (\{v_0, \cdots, v_{n-1}\}, \{\{v_{\pi(i)}, v_{\pi((i+1) \mod n)}\} \mid 0 \leq i < n\}).
  \]
Parallel Colouring Algorithm of (on) a cycle (Idea)

- A processor $P_i$ works on $v_{\pi(i-1)}$ for some permutation $\pi$.
- Register $R_i$ holds $\pi(i - 1)$.
- Register $N_i$ holds $\pi(i)$.
- In register $C_i$ will be the colour of $v_{R_i}$.
- Initialize $C_i$ with $i$.
- Reduce step by step the number of colours.
- We will use the colours $\{0, 1, \cdots, n\}$.
Parallel Colouring Algorithm of (on) a cycle (Idea)

Programm: colour-cycle

for all $P_{i+1}$ where $0 \leq i < n$ do in parallel

1. $\pi(i-1) \rightarrow R_i$
2. $\pi(i) \rightarrow N_i$
3. $c = i$
4. $c \rightarrow C_i$

repeat $\lceil \log^*(n) \rceil + 2$ times

1. $C_{N_i} \rightarrow c'$
2. minimal $k$ with: $((c \gg k) \% 2) \neq ((c' \gg k) \% 2)$.
3. $c = 2 \cdot k + ((c \gg k) \% 2)$.
4. $c \rightarrow C_i$
Parallel Colouring Algorithm of (on) a cycle (Idea)

- At the start we are using $n$ colours.
- Within each colour-reduction will the colouring stay correct.
- Within each colour reduction will the colouring number be reduced from $x$ to $\log(x) + O(1)$.
- After $[\log^*(n)]$ reductions steps will be the colouring numbers $\leq 5$.
- A second reduction of colours will follow now:
Last Steps

- The rows hold $c$ and the columns hold $c'$.
- The entries in the table hold the new $c$.

<table>
<thead>
<tr>
<th></th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

- We only have the colours 000, 001, 010, 011, 100, 101 ($\leq 5$).
Parallel Colouring Algorithm of (on) a cycle (Idea)

Programm: colour-cycle

for all $P_{i+1}$ where $0 \leq i < n$ do in parallel

$\pi(i - 1) \rightarrow R_i$
$\pi(i) \rightarrow N_i$
$c = i$
$c \rightarrow C_i$

repeat $\lceil \log^*(n) \rceil + 2$ times

$C_{N_i} \rightarrow c'$

minimal $k$ with: $((c \gg k) \mod 2) \neq ((c' \gg k) \mod 2)$.
$c = 2 \cdot k + ((c \gg k) \mod 2)$.
$c \rightarrow C_i$

for $r := 5$ downto 3 do:

if $c = r$ then

$C_{N_i} \rightarrow c'$
$c' \rightarrow C_i$
$C_{N_i} \rightarrow c''$
$c := \min(\{0, 1, 2\} \setminus \{c', c''\})$
$c \rightarrow C_i$
Colouring a Cycle

**Theorem:**

A cycle with $n$ nodes could be coloured with $n$ processors in time $O(\log^* n)$ with at most 3 colours.

Proof: see above.

**Theorem:**

A cycle of $n$ processors may colour itself in time $O(\log^* n)$ with at most 3 colours.

Proof: see above.

**Theorem:**

A cycle of $n$ processors needs at least $(\log^* n)$ time to colour itself with at most 3 colours.

Proof: see V4.
Colouring a Tree

- A processor $P_i$ works on $v_{\pi(i-1)}$ for some permutation $\pi$.
- Register $R_i$ holds $\pi(i - 1)$.
- Register $N_i$ holds $\pi(j - 1)$ where $j$ is the father of $i$.
- The father of the root $r$ is $r$.
- In register $C_i$ will be the colour of $v_{R_i}$.
- Initialize $C_i$ with $i$.
- Reduce step by step the number of colours.
- We will use the colours $\{0, 1, \ldots, n\}$. 
Parallel Colouring Algorithm of (on) a tree (Idea)

Programm: colour-cycle

for all $P_{i+1}$ where $0 \leq i < n$ do in parallel

$\pi(i - 1) \rightarrow R_i$

$\pi(i) \rightarrow N_i$

$c = i$

$c \rightarrow C_i$

repeat $\lceil \log^*(n) \rceil + 2$ times

$C_{N_i} \rightarrow c'$

minimal $k$ with: $((c \gg k) \% 2) \neq ((c' \gg k) \% 2)$.

$c = 2 \cdot k + ((c \gg k) \% 2)$.

$c \rightarrow C_i$
Parallel Colouring Algorithm of (on) a tree (Idea)

- At the start we are using $n$ colours.
- Within each colour-reduction will the colouring stay correct.
- Within each colour reduction will the colouring number be reduced from $x$ to $\log(x) + O(1)$.
- After $\lceil \log^*(n) \rceil$ reductions steps will be the colouring numbers $\leq 5$.
- A second reduction of colours will follow now:
Parallel Colouring Algorithm of (on) a tree (Idea)

Programm: colour-tree

\textbf{for all} $P_{i+1}$ where $0 \leq i < n$ \textbf{do in parallel}
\begin{itemize}
  \item $\pi(i - 1) \to R_i$
  \item $\pi(i) \to N_i$
  \item $c = i$ and $c \to C_i$
  \item \textbf{repeat $\lceil \log^*(n) \rceil + 2$ times, if $R_i \neq N_i$}
    \begin{itemize}
      \item $C_{N_i} \to c'$
      \item minimal $k$ with: $((c \gg k) \% 2) \neq ((c' \gg k) \% 2)$.
      \item $c = 2 \cdot k + ((c \gg k) \% 2)$.
      \item $c \to C_i$
    \end{itemize}
  \item \textbf{for $r := 5$ downto 3 do:}
    \begin{itemize}
      \item \textbf{if} $c = r$ \textbf{then}
        \begin{itemize}
          \item $C_{N_i} \to c'$
          \item $c' \to C_i$
          \item $C_{N_i} \to c''$
          \item $c := \min(\{0, 1, 2\} \setminus \{c', c''\})$
          \item $c \to C_i$
        \end{itemize}
    \end{itemize}
\end{itemize}
Parallel Colouring Algorithm of (on) a tree (Idea)

Programm: colour-cycle

for all $P_{i+1}$ where $0 \leq i < n$ do in parallel

\[
\begin{align*}
\pi(i - 1) &\rightarrow R_i \\
\pi(j - 1) &\rightarrow N_i \text{ with } j \text{ is father of } i \\
c &= i \text{ and } c \rightarrow C_i
\end{align*}
\]

repeat $\lceil \log^*(n) \rceil + 2$ times

\[
\begin{align*}
C_{N_i} &\rightarrow c' \\
\text{minimal } k \text{ with: } ((c \gg k) \% 2) \neq ((c' \gg k) \% 2) \\
c &= 2 \cdot k + ((c \gg k) \% 2) \\
c &\rightarrow C_i \\
\text{if } R_i = N_i \text{ then } c = \min(\{0, 1\} \setminus R_i) \text{ else } c = C_{N_i}
\end{align*}
\]

$c \rightarrow C_i$

for $r := 5$ downto 3 do:

\[
\begin{align*}
\text{if } c = r \text{ then} \\
C_{N_i} &\rightarrow c' \\
c' &\rightarrow C_i \\
C_{N_i} &\rightarrow c'' \\
c &:= \min(\{0, 1, 2\} \setminus \{c', c''\}) \\
c &\rightarrow C_i
\end{align*}
\]
Theorem:
A tree with \( n \) nodes could be coloured with \( n \) processors in time \( O(\log^* n) \) with at most 3 colours.

Proof: see above.

Theorem:
A tree of \( n \) processors may colour itself in time \( O(\log^* n) \) with at most 3 colours.

Proof: see above.
Eulerian cycle

**Definition:**
A graph $G = (V, E)$ is called Eulerian, iff there exists a cycle which visits each edge precisely once.

**Theorem**
A non-directed graph $G = (V, E)$ is Eulerian
- $G$ is connected and
- each node of $G$ has even degree.

**Theorem**
A directed graph $G = (V, E)$ is Eulerian
- $G$ is strong connected and
- each node as as many incoming edges as outgoing ones.

Problem: Compute Eulerian cycle on Eulerian graphs.
Idea

- Non Parallel:
  - Start with a free edge and follow free/unused edges till a cycle is closed.
  - Repeat till all edges are in some cycle.
  - Join pairs of cycles into a single one.
  - Repeat till just one cycle remains.

- If $G$ is non-directed, then make a directed version of $G$.
- Compute a cover of cycles.
- Compute an additional cycle which meets each cycle precisely once.
- Uses these to compute a cycle for $G$.
- Delete some edges to get a Eulerian cycle for $G$. 
Change a non-directed Graph into a directed one

- $G$ contain $m$ non-directed edges.
- Substitute each non-directed edge with two directed ones: 
  $\{i,j\}$ becomes $(i,j)$ and $(j,i)$.
- Define a successor for each edge:
  - The neighbors of $v$ are: $v_0, v_1, \ldots, v_{d-1}$.
  - Then define for all $i$:
    \[
    \text{Succ}((v_i, v)) := (v, v_{(i+1) \mod d}) \quad \text{und} \quad \text{Succ}((v_{(i+1) \mod d}, v)) := (v, v_i).
    \]
- Each directed edge is in precisely one cycle (defined by $\text{Succ}$).
- For each cycle $C$ exists one cycle $C'$, which consists the reverse edges.
- We will now delete one of the two cycles $C$ or $C'$.
Generating a directed Graph

- Identify the generated cycles:
  - Let \( \min(((i,j),(k,l))) := \begin{cases} (i,j) & \text{if } i \leq k \lor i = k \land j < l \\ (k,l) & \text{otherwise} \end{cases} \).
  - For each edge \( e \) define \( \text{Edge}'(e) = e \);
  - For all edges \( e \) repeat \( \log m \) times:
    - \( \text{Edge}'(e) = \min(\text{Edge}'(e), \text{Edge}'(\text{Succ}(e))) \)
    - \( \text{Succ}(e) = \text{Succ}(\text{Succ}(e)) \).

- For each edge \( (i,j) \): if \( \min(((i,j),(j,i))) \neq (i,j) \) then let \( \text{Edge}'(e) = 0 \).

- Thus we have selected for each non-directed edge a directed one (resp. a direction).

- Possible with \( m \) in time \( O(\log m) \).

- We consider in the following on directed graphs.
Step 1

- Let $G = (V, E)$ be a directed graph.
- Sort the edges into an array $Edge$.
  using the order: $(i, j) < (k, l) \iff j < l \lor (j = l \land i < k)$.
- Sort the edges into an array $Succ$.
  using the order: $(i, j) < (k, l) \iff i < k \lor (i = k \land j < l)$.
- We have already defined the cycles:
  Successor of edge $e = Edge(i)$ is the edge $Succ(i)$.
- We also store in $P(i)$ the position of $Succ(i)$ in $Edge$.
  I.e. $Edge(P(i)) = Succ(i)$.
- This information could be updated during the sorting of $Succ$.
- This could be done in time $O(\log m)$ using $O(m)$ processors.
Step 2

- Situation: We have a directed graph covered by cycles.
- Problem: Compute for each edge \( e \) the cycles where \( e \) belongs to.
- Solution: compute for each cycle the minimal edge
  \(((i,j) < (k,l) ⇔ i < k \lor (i = k \land j < l))\).
- Algorithm:

  Programm:

  \[
  \text{for all } P_i \text{ where } 1 \leq i \leq m \text{ do in parallel} \\
  \text{CycleRep}(i) := \text{Succ}(i) \\
  \text{for } i := 1 \text{ to } \lceil \log m \rceil \text{ do:} \\
  \quad \text{CycleRep}(i) := \min(\text{CycleRep}(i), \text{CycleRep}(P(i))) \\
  \quad P(i) := P(P(i))
  \]

- We use again the doubling technique.
- Possible in time \( O(\log m) \) using \( O(m) \) Processors.
Step 2 (Continued)

- Situation: the cycles of the coverage are identified by $\text{CycleRep}$.
- Problem: join the cycle into a single one.
- Solution: Identify the nodes of the cycle.
  
  $C = \{\text{CycleRep}(i) \mid 1 \leq i \leq m\}$. (Note $C$ is a edge set)

  $G' = V \cup C$

  $E' = \{(u, v) \mid u \in V, v \in C : v \text{ is identified in the cycle by } u\}$

Computing of $E'$:

Programm:

for all $P_i$ where $1 \leq i \leq m$ do in parallel

$(u, v) = \text{Edge}(i)$

$\text{Edge}'(2 \cdot i) = (u, \text{CycleRep}(i))$

$\text{Edge}'(2 \cdot i + 1) = (v, \text{CycleRep}(i))$
Step 2 (Continued)

- Situation: Cover of cycles and graph $G'$ defined.
- Problem: there are multiple edges.
- Solution: sort them out.
- Sort $Edge'$.
- Programm:
  
  ```
  for all $P_i$ where $1 \leq i \leq m$ do in parallel
  if $Edge'(i) = Edge'(i + 1)$ then $Edge(i) = \infty$
  ```
  
  Sort $Edge'$.
- Consider only the first $|E'|$ elements of $Edge'$.
- Problem: node $u$ could appear several times in a cycle $v$.
- As before we may compute a single representative.
- Let these edge be $(i, u) = Cert(u, v)$.
- May be done in time $O(\log m)$ using $O(m)$ processors.
Step 3

- Situation: Covering of the cycles and graph $G'$ computed.
- Problem: Compute cycle in $G'$.
- Solution: compute spanning tree $T$ for the bipartite Graph $G'$.
- To compute spanning tree we need $O(\log^2 m)$ time with $O(m/\log^2 m)$ Processors.
- Then we substitute each edge in $T$ with two directed edges.
- The new graph $T'$ is Eulerian.
- The Eulerian cycle is easy to find:
- To do so, compute for each node of the tree the order of edges.
- Could be done in time $O(\log m)$ using $O(m)$ processors.
Step 4

- **Situation**: We have a cover of cycles for $G$ and $T'$.
- **Problem**: Find cycle $L$ in $G'$.
- **Solution**: Combine the cycles using $Cert(u, v)$.
- $L$ will also contain the Eulerian cycle in $G$.
- For each cycle $v$ in $G$ $Cert(u, v)$ gives us an edge, at which we may exchange between $v$ and the cycle in $T'$.
- These points of change will be used to construct a single cycle $L$.
- **Time $O(1)$ using $O(m)$ Processors.**
Step 5

- Situation: we have a cycle for $G$ and $T'$.
- Problem: find cycle in $G$.
- Solution: delete edges from $T'$.
- Programm:
  
  for all $P_i$ where $1 \leq i \leq m$ do in parallel
  
  if $\text{Succ}(i) \in T'$ then $\text{Succ}(i) := \text{Succ}(\text{Succ}(i))$
  
  if $\text{Succ}(i) \in T'$ then $\text{Succ}(i) := \text{Succ}(\text{Succ}(i))$

- Uses time $O(1)$ with $O(m)$ processors.
- Total time is: $O(\log^2 m)$ using $O(m)$ processors.
- Also possible: $O(\log^2 m)$ time using $O(m/\log^2 m)$ processors.
Definition

Let $G = (V, E)$ be a non-directed graph.

- $M \subset E$ is called a matching, iff $\forall e, e' \in M : e \cap e' = \emptyset$.
- $M$ is called maximal matching, iff $\nexists e \in E : M \cup \{e\}$ is a matching.
- $M$ is called maximum matching, iff for all matchings $M'$ we have $|M'| \leq |M|$.

Sequential: $O(m \log m)$ for maximal matching.

Idea: Choose any free edge and delete all incident edges.

Sequential: $O(m^3)$ for maximum matching.

Idea: enlarging alternating pathes.
Idea

- Let $\Delta(G)$ be the maximal degree of $G$.
- Enlarge the matching step by step by several edges.
- There will be $O(\log_{3/2} n)$ phases.
- $i$-te phase $F_i$ has $G_i$ as input and will output $M_i$.
- $G_1 = G$ and final result: $\cup M_i$.
- Within each phase $F_i$ we will call the procedure $DegreeSplit$ $(1 + \log(\Delta(G)))$-times.
- Within each step within a phase we will half the node degree.
- We denote with $G(i, j)$ the graph considered in the $j$-th Step of the $i$-th phase.
- We will describe the procedure $DegreeSplit$.
- Let $k$ be the smallest number with $2^k \leq \Delta(G) \leq 2^{k+1}$.
- We will call all nodes $v$ with $\delta(v) \geq 2^k$ active.
Step 1

- Compute all active nodes of $G(i, j)$
  - Determine the degree in time $O(\log \Delta(G(i, j)))$ with $O(m)$ processors.
  - Determine the maximum degree in time $O(\log n)$ with $O(n)$ processors.
  - Then the active nodes are known in time $O(1)$ using $O(n)$ processors.

- Total running time: $O(\log n)$ using $O(m)$ processors.
Step 2

- Compute the graph $G^*(i,j)$ as follows:
  - Compute all nodes that are incident to active nodes.
  - Determine the new node degree.
  - If there are nodes with odd degree connect them to a new node $v$.

- Total running time: $O(\log n)$ using $O(m)$ processors.
- $G^*(i,j)$ might not be connected.
- Each component of $G^*(i,j)$ contains an Eulerian cycle.
- Note that each node $v$ has even degree.
Step 3

- Compute an Eulerian cycle on each component of $G^*(i,j)$.
- This needs time $O(\log^2 n)$ with $O(m + n)$ processors.
- Note that the additional $n$ processors result from the additional edges.
- Label the edges from the Eulerian cycle alternating with 0 and 1.
- For the component with the additional node $v$ start with $v$ using label 0.
- For all other components start at an arbitrary node with label 1.
- Running time: $O(\log n)$ with $O(m + n)$ processors.
- Use Parallel Prefix to compute the labels.
Step 4

- Delete all edges with label 0.

- If the remaining graph $G^{**}(i,j)$ is not a matching then $G(i,j + 1) = G^{**}(i,j) \setminus \{v\}$.

- If the remaining graph $G^{**}(i,j)$ is a matching then $M_i = E(G^{**}(i,j))$.

- Running time: $O(1)$ with $O(m + n)$ processors.

- Running time of the procedure $DegreeSplit$: $O(\log^2 n)$ with $O(m + n)$ processors.

- It remains to show: After at most $1 + \log(\Delta(G(i,j)))$ steps $DegreeSplit$ computes a matching.

- It remains to show: After at most $O(\log_{3/2} n)$ phases the matching is optimal.
Lemma:

Let \( G \) be the input of \textit{DegreeSplit}, then \textit{DegreeSplit} will compute a matching after \( 1 + \log(\Delta(G)) \) iterations.

Proof:

- Let \( k \) be the smallest number with \( 2^k \leq \Delta(G) \leq 2^{k+1} + 1 \).
- Let \( G_1 \) be the result of an iteration.
- Let \( v \) be active in \( G \). It holds:
  - \( 2^k \leq \delta_G(v) \).
  - \( \lfloor \delta_G(v)/2 \rfloor \leq \delta_{G_1}(v) \leq \lfloor \delta_G(v)/2 \rfloor + 1 \).
  - \( 2^{k-1} \leq \delta_{G_1}(v) \leq 2^k + 1 \).
- Then \( v \) stays active in \( G_1 \).

- Hence the degree is halved in every step.
- There exists a \( k' \leq k \) such that \( G_{k'} \) has a degree of 3.
- After two more iterations the degree is at most one.
- So a matching is found.
Lemma:
A logarithmic number of phases is enough to compute a maximum matching.

Proof:
- Let $A_i$ be the nodes that are active in phase $F_i$.
- Then $A_i$ is a vertex cover of $G_i$.
- ($C \subseteq V$ is a vertex cover if $\forall e \in E : e \cup C \neq \emptyset$)
- We show the following;
  - Half of the nodes in a vertex cover $A_i$ can be made incident to edges from $M_i$.
  - This means it holds: $|A_i/G_{i+1}| \leq |A_i|/2$.
    with $A_i/G_{i+1}$ the nodes of $A_i$ in $G_{i+1}$
  - There are vertex covers $C_i$: $|C_{i+1}| \leq 2 \cdot |C_i|/3$. 
Outer loop (Proof)

- Let $G_k = (V, E_k)$ be the graph in the third to last loop of $DegreeSplit$.
- W.l.o.g. $G_k$ is connected with degree $\leq 3$.
- $DegreeSplit$ can w.l.o.g. remove the smallest set of edges.
- Hence it holds $|M_i| \geq |E_k|/4$. 
Outer loop (Proof)

- If $|E_k| \geq |A_i|$ then $M_i$ contains at least $|A_i|/4$ edges.
  - Both end points of an edge from $A_i$ belong to $A_i$ and
  - at least half of them are incident to $M_i$.

- If $|E_k| < |A_i|$ then $G_k$ is a tree.
  - We remove edges from $G_k$ that have a leaf as one of its end points.
  - Furthermore the incident edges are removed.
Outer loop (Proof)

- Because $\Delta(G_k) \leq 3$ at most 2 trees $T_1$ and $T_2$ remain (with $n_1 + n_2$ nodes).
- Then $((n_1 - 1) + (n_2 - 1))/4$ edges are added to $M_i$.
- Then $M_i$ contains $|A_i|/2$ nodes.
- Then it holds: $|A_i/G_{i+1}| \leq |A_i|/2$. 

$|M_i| \geq |E_k|/4$
Outer loop (Proof)

- We show using induction that $G_i$ contains a vertex cover $C_i$ with $|C_i| \leq (2/3)^{i-1}|V|$.
- We will show that $|C_{i+1}| \leq 2|C_i|/3$.
- Basis: $i = 1$: Choose $C_1 = V$.
- Case 1: $|A_i| \leq 4|C_i|/3$.
  - In phase $i$ half of the nodes are removed from $A_i$.
  - $A_i/G_{i+1}$ is a vertex cover from $G_{i+1}$.
  - $|A_i|/2 \leq (4|C_i|/3)/2 = 2|C_i|/3$.
- Case 2: $|A_i| > 4|C_i|/3$.
  - Half of the nodes from $A_i$ are removed.
  - These have end points in $M_i$.
  - $C_i$ is a vertex cover of $G_i$.
  - Then every edge has at least one end point in $C_i$.
  - At least $1/4$ of the edges in $A_i$ are contained in $C_i$.
  - $C_i/G_{i+1}$ is a vertex cover of $G_{i+1}$.
  - $|C_i/G_{i+1}| \leq |C_i| - |A_i|/4 \leq |C_i| - (4|C_i|/3)/4 = 2|C_i|/3$. 

\[ |M_i| \geq |E_k|/4 \]
\[ |A_i/G_{i+1}| \leq |A_i|/2 \]
Summary

Theorem:
A maximal vertex cover can be computed in time $O(\log^4 n)$ using $O(n + m)$ processors.

Proof:
- Outer loop: $O(\log n)$
- Inner loop: $O(\log n)$
- Running time of $DegreeSplit$: $O(\log^2 n)$. 