Theory of Parallel and Distributed Systems (WS2016/17)

Kapitel 1
First Algorithms for PRAM

Walter Unger

Lehrstuhl für Informatik 1

8:54 Uhr, den 28. November 2016
Inhalt I

1 Motivation and History
   - Systolic Arrays and Vector Computer
   - Transputer
   - Parallel Rechner
   - PRAM

2 PRAM Introduction
   - Definition
   - Or
   - Sum
   - Matrices
   - Prefixsum
   - Maximum
   - Identify Root

3 Situation

4 Efficiency
   - Definition
   - Overview

5 Selection
   - Idea for the k-th Element
   - Examples
   - Algorithm and Running Time

6 Merging
   - Sequential Merging
   - Parallel Merging
   - Parallel Merging
Motivation

There are limits to the computing power of a single Computer

Computers become cheaper

Specialized computers are expensive

There are tasks with large data

Many problems are very complex

Weather and other Simulations

Crash tests

Military applications

Large data: (SETI, ...)

More similar problems

Thus there is the need for computers with more than one CPU

or a quantum computer?
Pipeline: (systolic array)

- There is a sequence of processors \((P_i)\) \(1 \leq i \leq n\).
- Processor \(P_1\) receives the input.
- Output of \(P_1\) will be passed as the input of \(P_2\).
- Output of \(P_i\) will be passed as the input of \(P_{i+1}\) \(1 \leq i < n\).
- Processor \(P_n\) delivers the final output.
- Processors may be different.
- Processors may run different programs.
- Intermediate outputs may be buffered.
- Pipelining is one important type of parallel system (in practice).
Systolic Arrays

- Idea: use more than one data stream.
- Data streams may interact with each other.
- Each processor is the same.
- There is a global synchronization.
- Processors may run simple programs.
- Advantage: really fast (for special applications).
Systolic Array with three data streams
Vector Computer

- Vector of processes.
- Each processor has different data.
- But each processor executes the same programm.
- Addition of two vectors:
  1. Read vector $A$
  2. Read vector $B$
  3. Add (each processor)
  4. Output the summ

- **Single Instruction Multiple Data** SIMD-Computer.
- **Aim**: **Multiple Instruction Multiple Data** MIMD-Computer.
- I.e. Fast processors with fast connections.
Example: Transputer

- Advantage: very flexible, any fixed network of degree 4 possible.
- Disadvantage: long wires may be necessary, only a fixed network possible.
Beispiel: Transputer II
Parallele Computer I

- Advantage: “normal” CPUs.
- Advantage: fast links possible.
- Advantage: no special hardware.
- Advantage: variable network, may change during execution.
- Advantage: very large networks may be possible.
- Disadvantage: still a limited degree for the network.
- Disadvantage: large network are complicated.
- Problem: cooling large systems.
- Problem: fault tolerance.
- Problem: construct such a system.
- Problem: generate good data throughput with constant degree network.
- Problem: do the program structures fit the structure of the network.
Parallel Computer II (Goodput)

- Look for good networks.
- Trees, Grids, Pyramids, ...
- $HQ(n)$, $CCC(n)$, $BF(n)$, $SE(n)$, $DB(n)$, ...
- Pancake Network and Burned Pancake Network.
- Problem: Physical placement of the processors.
- Problem: Length of wires.
- Problem: Has the network a nice structure.
- If the network becomes too large, we may use efficiency.
- Solution: choose a mixed network structure.
Parallel Computer III (Network)
Parallel Computer IV (Network)
Parallel Computer V (Network)

1. CPU and memory are one logical unit:

```
CPU   RAM   CPU   RAM   CPU   RAM   CPU   RAM
```

Network

2. CPUs and memory are connected by a network:

```
CPU   CPU   CPU   CPU   CPU
```

```
RAM   RAM   RAM   RAM   RAM
```

Network

The difference is more on the practical side.
**PRAM (theoretical model)**

- Ignore/unify the costs for each computation step.
- Ignore/unify the costs for each communication step.
Definition RAM

- RAM: Random Access Machine
- CPU may access any memory cell
- Memory is unlimited
- Complexity measurements
  - uniform: each operation cost one unit
  - logarithmic: cost are measured according to the size of the numbers
Idea of PRAM

- Many processes
- Common program
- Program may select single processors
- Common memory
Definition PRAM

- Consists of processors $P_i$ with $1 \leq i \leq p$ (processor has id $i$).
- Consists of registers $R_j$ with $1 \leq j \leq m$.
- Each processor has some local registers.
- Each processor $P_i$ may access each register $R_j$.
- Each processor executes the same program.
- The program is synchronized, thus each processor executes the same instructions.
- A selection is possible by using the processor id.
- The input of length $n$ is written to registers $R_j$ with $1 \leq j \leq n$.
- The output is placed in some known registers.
- The registers contain words (numbers) in the uniform cost measurement.
- The registers contain bits in the logarithmic cost measurement.
Definition PRAM

The following instructions are possible:

1. processor $P_i$ reads register $R_j$: $R_j \rightarrow P_i(x)$.
2. processor $P_i$ writes value of $x$ into register $R_j$: $P_i(x) \rightarrow R_j$.
3. processor may do some local computation using local registers:
   
   $x := y \times 5$.

For the access to the register we have the following variations:

- EREW _Exclusive Read Exclusive Write_
- CREW _Concurrent Read Exclusive Write_
- CRCW _Concurrent Read Concurrent Write_
- ERCW _Exclusive Read Concurrent Write_

Write conflicts may be solved using the following rules:

- Arbitrary: any processor gets access to the register.
- Common: all processors writing to the same register have to write the same value.
- Priority: the processor with the smallest id gets access to the register.
Computation of an “Or” (Idea)
Computing an “Or”

- Task: Compute \( x = \bigvee_{i=1}^{n} x_i \).
- Input: \( x_i \) is in register \( R_i \) (\( 1 \leq i \leq n \)).
- Output computed in \( R_{n+1} \).
- Program: Or
  ```
  for all \( P_i \) where \( 1 \leq i \leq n \) do in parallel
  \( R_i \rightarrow P_i(x) \)
  if \( x = \text{true} \) then \( P_i(x) \rightarrow R_{n+1} \)
  ```
- Running time: \( O(1) \) (exact 2 steps).
- Number of processors: \( n \).
- Memory: \( n + 1 \).
- Possible models: ERCW (Arbitrary, Common oder Priority).
Computing an “Or” (EREW)

- Problem:
  no writing of two processors 
  to the same register  
  at the same time. 

- Idea: combine pairwise the results 

- With this idea, computing the sum is also possible. 

- Thus computing the “Or” is just a special case of computing a sum.
Computing the Sum (Idea)
Computing the Sum (Idea)

\[ P_1 \quad P_2 \quad P_3 \quad P_4 \]

\[ 103 \quad 45 \quad 30 \quad 15 \]

\[ 12 \quad 6 \quad 34 \quad 5 \quad 7 \quad 23 \quad 4 \quad 11 \]
Computing the sum (EREW)

Assume w.l.o.g. $n = 2^k$ for $k \in \mathbb{N}$.

- Task: compute $x = \sum_{i=1}^{n} x_i$ with $n = 2^k$.
- Input: $x_i$ is in register $R_i$ ($1 \leq i \leq n$).
- Output: should be in $R_1$ (input may be overwritten).
- Model: EREW.

Program: Summe

for all $P_i$ where $1 \leq i \leq n/2$ do in parallel

$R_{2 \cdot i - 1} \rightarrow P_i(x)$

for $j = 1$ to $k$ do

if $(i - 1) \equiv 0 \pmod{2^{j-1}}$ then

$R_{2 \cdot i - 1 + 2^{j-1}} \rightarrow P_i(y)$

$x := x + y$

$P_i(x) \rightarrow R_{2 \cdot i - 1}$

- Running time: $O(k) = O(\log n)$ (precise $3 \cdot k + 1$ steps).
- Number of processors: $n/2$.
- Size of memory: $n$. 

Addition of Matrices

- Let $A, B$ two $(n \times n)$-Matrices.
- Sum $A + B$ is computable with $n^2$ processors in Zeit $O(1)$ on a EREW PRAM.
- $R_1$ till $R_{n^2}$ contain $A$ (one row after the other).
- $R_{1+n^2}$ bis $R_{2.n^2}$ contains $B$ (one row after the other).
- Result in $R_{1+2.n^2}$ bis $R_{3.n^2}$.
- Programm: MatSumme

\[
\text{for all } P_i \text{ where } 1 \leq i \leq n^2 \text{ do in parallel}
\]

$R_i \rightarrow P_i(x)$

$R_{i+n^2} \rightarrow P_i(y)$

$x := x + y$

$P_i(x) \rightarrow R_{i+2.n^2}$

- Running time: $O(1)$.
- Number of processors: $O(n^2)$.
- Size of memory: $O(n^2)$. 

Assume w.l.o.g $n = 2^k$ for $k \in \mathbb{N}$. 
Assume w.l.o.g $n = 2^k$ for $k \in \mathbb{N}$.

- Let $A, B$ be two $(n \times n)$-Matrices.
- $R_1$ till $R_{n^2}$ contain $A$ (one row after the other).
- $R_{1+n^2}$ bis $R_{2 \cdot n^2}$ contains $B$ (one row after the other).
- Result in $R_{1+2 \cdot n^2}$ bis $R_{3 \cdot n^2}$
- Register $A_{i,j} = R_{(i-1) \cdot n+j}$ ($1 \leq i, j \leq n$).
- Register $B_{i,j} = R_{(i-1) \cdot n+j+n^2}$ ($1 \leq i, j \leq n$).
- Register $C_{i,j} = R_{(i-1) \cdot n+j+2 \cdot n^2}$ ($1 \leq i, j \leq n$).
- processor $P_{i,j} = P_{(i-1) \cdot n+j}$ ($1 \leq i, j \leq n$).
- Use the above notation to simplify the algorithm.
- Each processor has to do some hidden local computation to implement the above expressions.
Multiplikation of Matrices

- Let $A, B$ be two $(n \times n)$-Matrices.
- Product $A \cdot B$ is computable with $n^2$ processors in time $O(n)$ on a CREW PRAM.
- **Programm: MatrProd 1**
  - for all $P_{i,j}$ where $1 \leq i, j \leq n$ do in parallel
    - $h = 0$
    - for $l = 1$ to $n$ do
      - $A_{i,l} \rightarrow P_{i,j}(a)$
      - $B_{i,j} \rightarrow P_{i,j}(b)$
      - $h = h + a \cdot b$
      - $P_{i,j}(h) \rightarrow C_{i,j}$
  - Running time: $O(n)$.
  - Number of processors: $O(n^2)$.
  - Size of memory: $O(n^2)$.

\[
A_{i,j} = R(i-1) \cdot n + j \\
B_{i,j} = R(i-1) \cdot n + j + n^2 \\
C_{i,j} = R(i-1) \cdot n + j + 2 \cdot n^2 \\
P_{i,j} = P(i-1) \cdot n + j
\]
Multiplikation of Matrices

- Let $A$, $B$ be two $(n \times n)$-Matrices
- Product $A \cdot B$ is computable with $n^2$ processors in time $O(n)$ on a EREW PRAM.
- Programm: MatrProd 2
  for all $P_{i,j}$ where $1 \leq i, j \leq n$ do in parallel
  $h = 0$
  for $l = 1$ to $n$ do
    $A_{i,l} \rightarrow P_{i,j}(a)$
    $B_{l,j} \rightarrow P_{i,j}(b)$
    $h = h + a \cdot b$
    $P_{i,j}(h) \rightarrow C_{i,j}$
- Running time: $O(n)$.
- Number of processors: $O(n^2)$.
- Size of memory: $O(n^2)$. 

$$A_{i,j} = R(i-1) \cdot n + j$$
$$B_{i,j} = R(i-1) \cdot n + j + n^2$$
$$C_{i,j} = R(i-1) \cdot n + j + 2 \cdot n^2$$
$$P_{i,j} = P(i-1) \cdot n + j$$
Compute the Prefixsum

Problem:

- Task: Compute $s_i = \sum_{j=1}^{i} x_j$ for $1 \leq i \leq n$.
- Input: $x_j$ is in register $R_j$ ($1 \leq j \leq n$).
- Output: $s_i$ should be in register $R_i$ for $1 \leq i \leq n$. 
Computing Prefixsum (Idea)
Computing the Prefixsum

- **Task:** Compute \( s_i = \sum_{j=1}^{i} x_j \) for \( 1 \leq i \leq n \).
- **Input:** \( x_j \) is in register \( R_j \) (\( 1 \leq j \leq n \)).
- **Output:** \( s_i \) should be in register \( R_i \) for \( 1 \leq i \leq n \).
- **Model:** EREW

**Programm:** Summe

```
for all \( P_i \) where \( 1 \leq i \leq n \) do in parallel
    \( R_i \rightarrow P_i(x) \)
    for \( j = 1 \) to \( k \) do
        if \( i > 2^{j-1} \) then
            \( R_{i-2^{j-1}} \rightarrow P_i(y) \)
            \( x := x + y \)
            \( P_i(x) \rightarrow R_i \)
```

- **Running time:** \( O(k) = O(\log n) \) (precisely \( 3 \cdot k + 1 \) steps).
- **Number of processors:** \( n \).
- **Size of memory:** \( n \).
Compute the Maximum

- Task: Compute \( m = \max_{j=1}^{i} x_j \) with \( n = 2^k \).
- Input: \( x_j \) is in register \( R_j \) (\( 1 \leq j \leq n \)).
- Output: \( m \) should be in register \( R_{n+1} \).
- Possible with \( n \) processors in time \( O(\log n) \) using a EREW PRAM.
- Question: could it be done faster? (i.e. on a ERCW PRAM).
- A maximum is larger or equal than all other values.
- Idea: compare all pairs of numbers.
- The maximum will always win.
### Compute the Maximum (Idea)

<table>
<thead>
<tr>
<th>22</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>59</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>57</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>52</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>61</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>67</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>56</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>12</td>
<td>14</td>
<td>56</td>
<td>23</td>
<td>67</td>
<td>49</td>
<td>27</td>
<td>61</td>
<td>52</td>
<td>57</td>
<td>59</td>
<td>26</td>
<td>41</td>
<td>33</td>
</tr>
</tbody>
</table>
Compute the Maximum (Idea)

|   | 22 | 33 | 41 | 26 | 59 | 67 | 52 | 61 | 27 | 49 | 67 | 23 | 56 | 14 | 12 | 34 | 34 | 12 | 14 | 56 | 23 | 67 | 49 | 27 | 61 | 52 | 67 | 59 | 26 | 41 | 33 | 22 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
Computing the Maximum

- Task: Compute \( m = \max_{j=1}^i x_j \) with \( n = 2^k \).
- Input: \( x_j \) is in register \( R_j \) (\( 1 \leq x_j \leq n \)).
- Output: \( m \) in register \( R_{n+1} \).
- Model: CRCW.
- Program: Maximum

\[
\text{for all } P_{i,1} \text{ where } 1 \leq i \leq n \text{ do in parallel} \\
\quad P_{i,1}(1) \rightarrow W_i \\
\text{for all } P_{i,j} \text{ where } 1 \leq i, j \leq n \text{ do in parallel} \\
\quad R_i \rightarrow P_{i,j}(a) \\
\quad R_j \rightarrow P_{i,j}(b) \\
\quad \text{if } a < b \text{ then } P_{i,j}(0) \rightarrow W_i \\
\text{for all } P_{i,1} \text{ where } 1 \leq i \leq n \text{ do in parallel} \\
\quad W_i \rightarrow P_{i,1}(h) \\
\quad \text{if } h = 1 \text{ then} \\
\quad \quad R_i \rightarrow P_{i,1}(h) \\
\quad \quad P_{i,1}(h) \rightarrow R_{n+1}
\]
Computing the Maximum

- Programm: Maximum
  
  for all $P_{i,1}$ where $1 \leq i \leq n$ do in parallel
  
  $P_{i,1}(1) \rightarrow W_i$

  for all $P_{i,j}$ where $1 \leq i, j \leq n$ do in parallel
  
  $R_i \rightarrow P_{i,j}(a)$
  $R_j \rightarrow P_{i,j}(b)$

  if $a < b$ then $P_{i,j}(0) \rightarrow W_i$

  for all $P_{i,1}$ where $1 \leq i \leq n$ do in parallel
  
  $W_i \rightarrow P_{i,1}(h)$

  if $h = 1$ then
  
  $R_i \rightarrow P_{i,1}(h)$
  $P_{i,1}(h) \rightarrow R_{n+1}$

- Running time: $O(1)$.

- Number of processors: $O(n^2)$.

- Memory: $O(n)$. 
**Identify the Roots of a Forest**

- Nodes are identified by numbers from 1 till \( n \).
- Input: Father of node \( i \) is written in register \( R_i \).
- For the roots \( i \) we have: in register \( R_i \) is written \( i \).
- **Programm: Ranking**
  
  for all \( P_i \) where \( 1 \leq i \leq n \) do in parallel
  
  for \( j = 1 \) to \( \lceil \log n \rceil \) do
  
  \( R_i \rightarrow P_i(h) \)
  
  \( R_h \rightarrow P_i(h) \)
  
  \( P_i(h) \rightarrow R_i \)

  Running time: \( O(\log n) \).

  Number of processors: \( O(n) \).

  Memory: \( O(n) \).

  Model: CREW.
Short Summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>processors</th>
<th>memory</th>
<th>time</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>$O(n/t)$</td>
<td>$O(n)$</td>
<td>$O(t)$</td>
<td>ERCW</td>
</tr>
<tr>
<td>Or</td>
<td>$O(n/\log n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Maximum</td>
<td>$O(n^2/t)$</td>
<td>$O(n)$</td>
<td>$O(t)$</td>
<td>CRCW</td>
</tr>
<tr>
<td>Sum</td>
<td>$O(n/\log n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Ranking</td>
<td>$O(n/\log n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Prefixsum</td>
<td>$O(n/\log n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.sum</td>
<td>$O(n^2/\log n)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^2/\log n)$</td>
<td>$O(n^2)$</td>
<td>$O(n \cdot \log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3/\log n)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3/\log n)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

Question: May we save some processors?
May we do this saving in any situation?
How do we estimate the efficiency of a parallel algorithm?
Cost Measurement

Let $A$ be any parallel algorithm, we denote:

- $T_A(n)$ the running time of $A$.
- $P_A(n)$ the number of processors used by $A$.
- $R_A(n)$ the number of registers used by $A$.
- $W_A(n)$ the number of accesses to registers done by $A$.
- $ST(n)$ the running time of the best [known] sequential algorithm.
- $\text{Eff}_A(n) := \frac{ST(n)}{P_A(n) \cdot T_A(n)}$ the efficiency of $A$.
- $\text{AEff}_A(n) := \frac{W_A(n)}{P_A(n) \cdot T_A(n)}$ the usage efficiency of $A$. 
### Efficiency

<table>
<thead>
<tr>
<th>Problem</th>
<th>processors</th>
<th>time</th>
<th>$W(n)$</th>
<th>$AEff$</th>
<th>Modell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>$O(n/t)$</td>
<td>$O(t)$</td>
<td>$O(n)$</td>
<td>1</td>
<td>ERCW</td>
</tr>
<tr>
<td>Or</td>
<td>$O(n/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>1</td>
<td>EREW</td>
</tr>
<tr>
<td>Maximum</td>
<td>$O(n^2/t)$</td>
<td>$O(t)$</td>
<td>$O(n^2)$</td>
<td>1</td>
<td>CRCW</td>
</tr>
<tr>
<td>Sum</td>
<td>$O(n/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>1</td>
<td>EREW</td>
</tr>
<tr>
<td>Ranking</td>
<td>$O(n/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>1</td>
<td>CREW</td>
</tr>
<tr>
<td>Prefixsum</td>
<td>$O(n/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>1</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.sum</td>
<td>$O(n^2/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n^2)$</td>
<td>1</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^2/\log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n^3)$</td>
<td>1</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n^3)$</td>
<td>1</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n^3)$</td>
<td>1</td>
<td>EREW</td>
</tr>
</tbody>
</table>
## Efficiency

<table>
<thead>
<tr>
<th>Problem</th>
<th>processors</th>
<th>time $t$</th>
<th>$ST(n)$</th>
<th>$Eff$</th>
<th>Modell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>$O(n/t)$</td>
<td>$O(t)$</td>
<td>$O(n)$</td>
<td>1</td>
<td>ERCW</td>
</tr>
<tr>
<td>Or</td>
<td>$O(n/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>1</td>
<td>EREW</td>
</tr>
<tr>
<td>Maximum</td>
<td>$O(n^2/t)$</td>
<td>$O(t)$</td>
<td>$O(n)$</td>
<td>$O(1/n)$</td>
<td>CRCW</td>
</tr>
<tr>
<td>Sum</td>
<td>$O(n/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>1</td>
<td>CREW</td>
</tr>
<tr>
<td>Ranking</td>
<td>$O(n/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>1</td>
<td>EREW</td>
</tr>
<tr>
<td>Prefixsum</td>
<td>$O(n/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n^2)$</td>
<td>1</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.sum</td>
<td>$O(n^2/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n^{2.276})$</td>
<td>$O(n^{-0.734})$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^2/\log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n^{2.276})$</td>
<td>$O(n^{-0.734})$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n^{2.276})$</td>
<td>$O(n^{-0.734})$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3/\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(n^{2.276})$</td>
<td>$O(n^{-0.734})$</td>
<td>EREW</td>
</tr>
</tbody>
</table>
k-th Element

- Task: Compute the $k$-th ($k$-smallest) element in a unsorted sequence $S = \{s_1, \ldots, s_n\}$.

- Lower bound: $n - 1$ comparisons

- Start with a nice sequential algorithm

- Programm: Select($k, S$)
  
  if $|S| \leq 50$ then return $k$-th number in $S$
  
  Split $S$ in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$
  
  Sort each $H_i$
  
  Let $M$ be the sequence of the middle elements in $H_i$
  
  $m := \text{Select}(\lceil|M|/2\rceil, M)$
  
  $S_1 := \{s \in S \mid s < m\}$
  
  $S_2 := \{s \in S \mid s = m\}$
  
  $S_3 := \{s \in S \mid s > m\}$
  
  if $|S_1| \geq k$ then return $\text{Select}(k, S_1)$
  
  if $|S_1| + |S_2| \geq k$ then return $m$
  
  return $\text{Select}(k - |S_1| - |S_2|, S_3)$
Example for the k-th Element (Slow Motion)

**Input/Data:**

<table>
<thead>
<tr>
<th></th>
<th>61</th>
<th>76</th>
<th>58</th>
<th>37</th>
<th>96</th>
<th>31</th>
<th>49</th>
<th>50</th>
<th>9</th>
<th>70</th>
<th>18</th>
<th>29</th>
<th>18</th>
<th>30</th>
<th>68</th>
<th>4</th>
<th>48</th>
<th>82</th>
<th>11</th>
<th>46</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>78</td>
<td>40</td>
<td>38</td>
<td>24</td>
<td>33</td>
<td>58</td>
<td>82</td>
<td>35</td>
<td>33</td>
<td>90</td>
<td>54</td>
<td>10</td>
<td>44</td>
<td>46</td>
<td>49</td>
<td>38</td>
<td>30</td>
<td>14</td>
<td>29</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>7</td>
<td>40</td>
<td>91</td>
<td>45</td>
<td>64</td>
<td>21</td>
<td>30</td>
<td>94</td>
<td>30</td>
<td>41</td>
<td>67</td>
<td>2</td>
<td>79</td>
<td>42</td>
<td>28</td>
<td>15</td>
<td>9</td>
<td>44</td>
<td>88</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>94</td>
<td>8</td>
<td>77</td>
<td>47</td>
<td>32</td>
<td>79</td>
<td>65</td>
<td>24</td>
<td>4</td>
<td>3</td>
<td>15</td>
<td>74</td>
<td>23</td>
<td>41</td>
<td>66</td>
<td>82</td>
<td>59</td>
<td>63</td>
<td>52</td>
<td>64</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2</td>
<td>44</td>
<td>48</td>
<td>72</td>
<td>82</td>
<td>12</td>
<td>16</td>
<td>51</td>
<td>2</td>
<td>22</td>
<td>42</td>
<td>41</td>
<td>56</td>
<td>55</td>
<td>72</td>
<td>42</td>
<td>67</td>
<td>64</td>
<td>61</td>
<td>90</td>
</tr>
</tbody>
</table>

M:

|   | 61 | 8  | 44 | 47 | 45 | 64 | 49 | 30 | 33 | 30 | 22 | 42 | 23 | 46 | 55 | 38 | 42 | 63 | 44 | 61 | 50 |

sorted M:

|   | 8  | 22 | 23 | 30 | 30 | 33 | 38 | 42 | 42 | 44 | 44 | 45 | 46 | 47 | 49 | 50 | 55 | 61 | 61 | 63 | 64 |
### Example for the k-th Element

**Input/Data:**

```
4  60  16  28  89  56  97  8  11  94  7  0  86  51  40  16  78  67  55  49  39
90  56  21  3  31  35  47  18  52  50  69  93  94  26  32  11  37  33  44  82  10
97  56  72  49  87  70  95  83  74  70  28  94  13  92  80  26  80  94  53  11  21
41  1  89  2  2  89  14  55  85  65  32  11  60  43  10  48  18  18  60  55  57
31  33  75  68  56  22  54  22  1  11  8  16  89  60  70  38  16  57  70  56  62
```

**M:**

```
41  56  72  28  56  56  54  22  52  65  28  16  86  51  40  26  37  57  55  55  39
```

**sorted M:**

```
16  22  26  28  28  37  39  40  41  51  52  54  55  55  56  56  56  57  65  72  86
```
Example for the k-th Element (Worst Case)

Input/Data:

<table>
<thead>
<tr>
<th>60</th>
<th>76</th>
<th>56</th>
<th>75</th>
<th>51</th>
<th>94</th>
<th>86</th>
<th>78</th>
<th>78</th>
<th>84</th>
<th>63</th>
<th>51</th>
<th>93</th>
<th>63</th>
<th>88</th>
<th>54</th>
<th>76</th>
<th>82</th>
<th>87</th>
<th>63</th>
<th>93</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>69</td>
<td>56</td>
<td>62</td>
<td>50</td>
<td>85</td>
<td>94</td>
<td>90</td>
<td>60</td>
<td>73</td>
<td>84</td>
<td>94</td>
<td>90</td>
<td>93</td>
<td>85</td>
<td>67</td>
<td>81</td>
<td>66</td>
<td>60</td>
<td>85</td>
<td>88</td>
</tr>
<tr>
<td>44</td>
<td>31</td>
<td>0</td>
<td>17</td>
<td>13</td>
<td>0</td>
<td>18</td>
<td>33</td>
<td>39</td>
<td>34</td>
<td>11</td>
<td>75</td>
<td>65</td>
<td>84</td>
<td>68</td>
<td>62</td>
<td>78</td>
<td>85</td>
<td>51</td>
<td>75</td>
<td>88</td>
</tr>
<tr>
<td>38</td>
<td>17</td>
<td>22</td>
<td>2</td>
<td>21</td>
<td>16</td>
<td>25</td>
<td>24</td>
<td>21</td>
<td>0</td>
<td>16</td>
<td>87</td>
<td>84</td>
<td>94</td>
<td>51</td>
<td>73</td>
<td>78</td>
<td>67</td>
<td>66</td>
<td>58</td>
<td>66</td>
</tr>
<tr>
<td>33</td>
<td>8</td>
<td>21</td>
<td>4</td>
<td>6</td>
<td>20</td>
<td>12</td>
<td>11</td>
<td>14</td>
<td>36</td>
<td>38</td>
<td>66</td>
<td>64</td>
<td>92</td>
<td>50</td>
<td>57</td>
<td>57</td>
<td>67</td>
<td>64</td>
<td>63</td>
<td>66</td>
</tr>
</tbody>
</table>

M:

| 44 | 31 | 22 | 17 | 21 | 20 | 25 | 33 | 39 | 36 | 38 | 75 | 84 | 92 | 68 | 62 | 78 | 67 | 64 | 63 | 88 |

sorted M:

| 17 | 20 | 21 | 22 | 25 | 31 | 33 | 36 | 38 | 39 | 44 | 62 | 63 | 64 | 67 | 68 | 75 | 78 | 84 | 88 | 92 |
Running Time

- For some constants $c, d$ we get:
  - $T(n) \leq d \cdot n$ for $n \leq 50$
  - $T(n) \leq c \cdot n + T(n/5) + T(3n/4)$

if $|S| \leq 50$ then return $k$-th number in $S$
Split $S$ in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$
Sort each $H_i$
Let $M$ be the sequence of the middle elements in $H_i$
\[ m := \text{Select}(\lceil |M|/2 \rceil, M) \]
\[ S_1 := \{ s \in S \mid s < m \} \]
\[ S_2 := \{ s \in S \mid s = m \} \]
\[ S_3 := \{ s \in S \mid s > m \} \]
if $|S_1| \geq k$ then return $\text{Select}(k, S_1)$
if $|S_1| + |S_2| \geq k$ then return $m$
return $\text{Select}(k - |S_1| - |S_2|, S_3)$
Claim: $T(n) \leq 20 \cdot r \cdot n$ with $r = \max(d, c)$.

Proof:

$n = 50$: 

$$T(n) \leq c \cdot n + \frac{d \cdot n}{5} + \frac{3 \cdot d \cdot n}{4}$$

$n > 50$:

$$T(n) \leq c \cdot n + T\left(\frac{d \cdot n}{5}\right) + T\left(\frac{3 \cdot d \cdot n}{4}\right)$$

$$T(n) \leq c \cdot n + 4 \cdot r \cdot n + 15 \cdot r \cdot n$$

Running time $T(n)$ is in $O(n)$. 
Parallel k-Select

- Input $S = \{s_1, \cdots, s_n\}$.
- Processors $P_1, P_2, \cdots P_{\lceil n^{1-x} \rceil}$, thus $P(n) = \lceil n^{1-x} \rceil$.
- Each $P_i$ knows $n, P(n)$.
- Each $P_i$ works on $\lceil n^x \rceil$ elements.
- We will now create a parallel version of the program Select(k,S).
- We will get a parallel recursive program.

1. Easy solution for small $S$.
2. Split $S$ into small sub-sequences for the processors.
3. Compute parallel the median of the sub-sequences.
4. Compute parallel and recursive the median of medians.
5. Compute the splitting into the three sub-sequences.
6. Do the final recursion.
### Example for the k-th Element

**Input/Data:**

| 43 | 22 | 69 | 12 | 21 | 9 | 83 | 26 | 45 | 66 | 42 | 78 | 27 | 96 | 85 | 46 | 53 | 36 | 41 | 48 | 93 | 61 | 52 | 59 | 2 | 72 | 24 | 33 | 28 | 77 | 87 | 89 | 36 | 43 | 42 |
| 79 | 95 | 97 | 25 | 1 | 13 | 93 | 61 | 3 | 28 | 95 | 32 | 78 | 55 | 88 | 92 | 50 | 71 | 97 | 85 | 1 | 29 | 76 | 47 | 96 | 81 | 34 | 90 | 20 | 45 | 26 | 40 | 18 | 17 | 32 |
| 9 | 94 | 29 | 27 | 81 | 60 | 81 | 8 | 34 | 89 | 65 | 91 | 97 | 82 | 48 | 64 | 39 | 82 | 63 | 94 | 23 | 33 | 51 | 38 | 10 | 41 | 14 | 61 | 84 | 37 | 66 | 58 | 34 | 95 | 48 |
| 41 | 92 | 19 | 71 | 69 | 2 | 88 | 67 | 39 | 5 | 68 | 62 | 7 | 19 | 40 | 27 | 45 | 54 | 68 | 66 | 10 | 35 | 54 | 31 | 92 | 17 | 24 | 29 | 8 | 0 | 12 | 91 | 22 | 64 | 32 |
| 31 | 14 | 28 | 34 | 89 | 88 | 45 | 57 | 77 | 51 | 31 | 52 | 85 | 42 | 29 | 33 | 21 | 7 | 32 | 66 | 50 | 24 | 66 | 21 | 39 | 73 | 57 | 93 | 6 | 26 | 2 | 91 | 30 | 81 | 45 |
| 51 | 69 | 83 | 74 | 12 | 31 | 89 | 11 | 36 | 42 | 2 | 92 | 69 | 1 | 5 | 83 | 43 | 68 | 69 | 84 | 90 | 96 | 96 | 46 | 78 | 59 | 69 | 11 | 79 | 18 | 7 | 85 | 83 | 72 | 66 |
| 76 | 94 | 19 | 12 | 60 | 60 | 87 | 93 | 7 | 81 | 86 | 16 | 87 | 58 | 57 | 34 | 83 | 89 | 87 | 5 | 82 | 9 | 20 | 66 | 81 | 14 | 13 | 5 | 18 | 45 | 15 | 48 | 60 | 69 | 56 |
| 7 | 60 | 89 | 36 | 59 | 18 | 58 | 55 | 20 | 61 | 71 | 36 | 88 | 26 | 80 | 86 | 17 | 84 | 21 | 13 | 52 | 66 | 97 | 33 | 9 | 46 | 13 | 87 | 31 | 12 | 69 | 67 | 55 | 44 | 92 |
| 50 | 49 | 67 | 35 | 32 | 70 | 77 | 17 | 24 | 70 | 64 | 84 | 40 | 37 | 5 | 26 | 62 | 74 | 84 | 46 | 28 | 20 | 92 | 66 | 64 | 76 | 64 | 23 | 28 | 3 | 30 | 95 | 37 | 51 | 72 |
| 76 | 13 | 62 | 73 | 67 | 25 | 41 | 16 | 81 | 53 | 63 | 32 | 41 | 39 | 46 | 49 | 31 | 90 | 86 | 79 | 20 | 29 | 8 | 13 | 47 | 18 | 59 | 30 | 54 | 30 | 20 | 38 | 84 | 18 | 42 |
| 88 | 42 | 92 | 78 | 44 | 89 | 91 | 50 | 83 | 39 | 78 | 91 | 52 | 63 | 68 | 34 | 67 | 43 | 91 | 59 | 24 | 63 | 23 | 69 | 13 | 79 | 58 | 56 | 86 | 69 | 88 | 78 | 69 | 7 | 69 | 23 | 20 | 88 | 63 |
| 44 | 10 | 64 | 13 | 69 | 80 | 9 | 12 | 87 | 37 | 74 | 49 | 20 | 28 | 51 | 64 | 1 | 22 | 76 | 23 | 31 | 79 | 58 | 56 | 86 | 69 | 88 | 78 | 69 | 7 | 69 | 23 | 20 | 88 | 63 |
| 87 | 12 | 56 | 72 | 24 | 54 | 10 | 20 | 39 | 68 | 77 | 77 | 87 | 44 | 25 | 70 | 34 | 3 | 23 | 75 | 97 | 53 | 23 | 20 | 19 | 58 | 18 | 86 | 97 | 60 | 3 | 49 | 44 | 47 | 18 |
| 76 | 54 | 73 | 35 | 60 | 1 | 94 | 38 | 97 | 7 | 55 | 69 | 12 | 37 | 68 | 87 | 89 | 24 | 75 | 21 | 89 | 35 | 51 | 16 | 1 | 93 | 92 | 54 | 63 | 72 | 6 | 69 | 57 | 2 | 39 |
| 23 | 74 | 72 | 74 | 28 | 33 | 51 | 0 | 56 | 77 | 54 | 0 | 90 | 33 | 31 | 11 | 70 | 1 | 78 | 31 | 17 | 22 | 39 | 16 | 24 | 95 | 63 | 74 | 23 | 38 | 40 | 59 | 2 | 79 | 64 |

\[ P_1 \quad P_2 \quad P_3 \quad P_4 \quad P_5 \quad P_6 \quad P_7 \quad P_8 \quad P_9 \quad P_{10} \quad P_{11} \quad P_{12} \quad P_{13} \quad P_{14} \quad P_{15} \quad P_{16} \quad P_{17} \quad P_{18} \quad P_{19} \quad P_{20} \quad P_{21} \quad P_{22} \quad P_{23} \quad P_{24} \quad P_{25} \quad P_{26} \quad P_{27} \quad P_{28} \quad P_{29} \quad P_{30} \quad P_{31} \quad P_{32} \quad P_{33} \quad P_{34} \quad P_{35} \]

**M:**

| 50 | 54 | 67 | 35 | 59 | 33 | 81 | 26 | 39 | 53 | 65 | 52 | 69 | 39 | 46 | 64 | 45 | 54 | 69 | 48 | 31 | 35 | 54 | 38 | 39 | 59 | 57 | 59 | 59 | 64 | 65 | 66 | 67 | 69 | 69 | 81 |

**sorted M:**

| 21 | 26 | 28 | 31 | 33 | 35 | 35 | 36 | 37 | 38 | 39 | 39 | 45 | 45 | 46 | 48 | 50 | 51 | 52 | 53 | 54 | 54 | 54 | 54 | 57 | 59 | 59 | 64 | 65 | 66 | 67 | 69 | 69 | 81 |
Parallel k-Select

Programm: ParSelect(k,S)
1: \[
\text{if } |S| \leq k_1 \text{ then } P_1 \text{ returns } Select(k, S).
\]
2: \[
S \text{ is split into } \left\lceil |S|^{1-x} \right \rceil \text{ sub-sequences } S_i \text{ with } |S_i| \leq \left\lceil n^x \right \rceil \text{.}
\]
P_i \text{ stores the start-address of } S_i.
3: \[
\text{for all } P_i \text{ where } 1 \leq i \leq \left\lceil n^{1-x} \right \rceil \text{ do in parallel}
\]
\[
m_i := Select(\left\lceil |S_i|/2 \right \rceil, S_i)
\]
\[
P_i(m_1) \rightarrow R_i.
\]
Assume in the following that M is the sequence of these values.
4: \[
m := ParSelect(\left\lceil |M|/2 \right \rceil, M).
\]
5: More to come!
Motivation and History

PRAM Introduction

Efficiency

Selection

Merging

1:50 Algorithm and Running Time

Walter Unger 28.11.2016 8:54  WS2016/17

WSH

Parallel k-Select

Programm: ParSelect(k,S) Steps 5

5.1:

Distribute $m$ via broadcast to all $P_i$.

for all $P_i$ where $1 \leq i \leq \lceil n^{1-x} \rceil$ do in parallel

$L_i := \{ s \in S_i \mid s < m \}$

$E_i := \{ s \in S_i \mid s = m \}$

$G_i := \{ s \in S_i \mid s > m \}$

5.2:

Compute with Parallel Prefix:

$l_i := \sum_{j=1}^{i} |L_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.

$e_i := \sum_{j=1}^{i} |E_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.

$g_i := \sum_{j=1}^{i} |G_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.

Let: $l_0 = e_0 = g_0 = 0$

5.3:

Even more to come!
Parallel $k$-Select

Programm: ParSelect($k$, $S$) Steps 5+6

5.3:
Compute $L = \{ s \in S \mid s < m \}$, $E = \{ s \in S \mid s = m \}$ and $G = \{ s \in S \mid s > m \}$ as follows:
for all $P_i$ where $1 \leq i \leq \lceil n^{1-x} \rceil$ do in parallel
- $P_i$ writes $L_i$ in $R_{l_{i-1}+1}, \ldots, R_{l_i}$.
- $P_i$ writes $E_i$ in $R_{e_{i-1}+1}, \ldots, R_{e_i}$.
- $P_i$ writes $G_i$ in $R_{g_{i-1}+1}, \ldots, R_{g_i}$.

6:
if $|L| \geq k$ then return ParSelect($k$, $L$)
if $|L| + |E| \geq k$ then return $m$
return Select($k - |L| - |E|, G$)
Parallel k-Select (Running Time)

Programm: ParSelect(k,S)

1: $O(1)$
   if $|S| \leq k_1$ then $P_1$ returns $Select(k, S)$.

2: $O(\log_2(|S|^{1-x}))$ thus we have $O(\log n)$
   $S$ is split into $\lceil |S|^{1-x} \rceil$ sub-sequences $S_i$ with $|S_i| \leq \lceil n^x \rceil$
   $P_i$ stores the start-address of $S_i$.

3: $O(n^x)$
   for all $P_i$ where $1 \leq i \leq \lceil n^{1-x} \rceil$ do in parallel
   $m_i := Select(\lceil |S_i|/2 \rceil, S_i)$
   $P_i(m_1) \rightarrow R_i$.
   Assume in the following that $M$ is the sequence of these values

4: $T_{ParSelect}(n^{1-x})$
   $m := ParSelect(\lceil |M|/2 \rceil, M)$. 
Parallel k-Select (Running Time)

Programm: ParSelect(k,S) Steps 5

5.1a: $O(\log_2(n^{1-x}))$

Distribute $m$ via broadcast to all $P_i$.

5.1b: $O(|S_i|)$ thus we have $O(n^x)$

for all $P_i$ where $1 \leq i \leq \lceil n^{1-x} \rceil$ do in parallel

$L_i := \{ s \in S_i \mid s < m \}$

$E_i := \{ s \in S_i \mid s = m \}$

$G_i := \{ s \in S_i \mid s > m \}$

5.2: $O(\log_2(n^{1-x}))$

Compute with Parallel Prefix:

$l_i := \sum_{j=1}^i |L_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.

$e_i := \sum_{j=1}^i |E_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.

$g_i := \sum_{j=1}^i |G_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.

Let: $l_0 = e_0 = g_0 = 0$
Parallel k-Select (Running Time)

Programm: ParSelect(k,S) Steps 5+6

5.3: $O(n^x)$

Compute $L = \{s \in S \mid s < m\}$, $E = \{s \in S \mid s = m\}$ and $G = \{s \in S \mid s > m\}$ as follows:

for all $P_i$ where $1 \leq i \leq \lceil n^{1-x} \rceil$ do in parallel

$P_i$ writes $L_i$ in $R_{l_{i-1}+1}, \ldots, R_{l_i}$.
$P_i$ writes $E_i$ in $R_{e_{i-1}+1}, \ldots, R_{e_i}$.
$P_i$ writes $G_i$ in $R_{g_{i-1}+1}, \ldots, R_{g_i}$.

6: $T_{ParSelect}(3 \cdot n/4)$

if $|L| \geq k$ then return $ParSelect(k, L)$
if $|L| + |E| \geq k$ then return $m$
return $Select(k - |L| - |E|, G)$
Parallel k-Select (Running Time)

Adding all up we get:

- \( T_{ParSelect}(n) = c_1 \log n + c_2 \cdot n^x + T_{ParSelect}(n^{1-x}) + T_{ParSelect}(3/4 \cdot n). \)
- \( T_{ParSelect}(n) = O(n^x) \) with \( P_{ParSelect}(n) = O(n^{1-x}). \)
- \( Eff_{ParSelect}(n) = \frac{O(n)}{O(n^x) \cdot O(n^{1-x})} = O(1) \)
Sequential Merging

- **Input:**
  \[ A = (a_1, a_2, \cdots, a_r) \text{ and } B = (b_1, b_2, \cdots, b_s) \] two sorted sequences

- **Output:**
  \[ C = (c_1, c_2, \cdots, c_n) \] sorted sequence of \( A \) and \( B \) with \( n = r + s \).

- **Program:** Merge
  
  \[
  i := 1; j := 1; n := r + s \\
  \text{for } k := 1 \text{ to } n \text{ do} \\
  \quad \text{if } a_i < b_j \\
  \quad \quad \text{then } c_k := a_i; i := i + 1; \\
  \quad \quad \text{else } c_k := b_j; j := j + 1;
  \]

- Algorithm does not care about special cases.

- **Running time:** at most \( r + s \) comparisons, i.e. \( O(n) \).

- **Lower bound on the number of comparisons is** \( r + s \), i.e. \( \Omega(n) \).
Idea for Parallel Merging (CREW)

- The border lines may not intersect each other.
- Thus we may separate the two sequences into disjoint blocks.
- Let $A_i$ the $i$ block of size $\lceil r/p \rceil$.
- Let $\hat{B}_i$ block in $B$ which should be merged with $A_i$.
- Thus we may uses a PRAM easily (in this case).
Idea for Parallel Merging (CREW)

- Let $A_i$ [resp. $B_i$] the $i$ block of size $\lceil r/p \rceil$ [resp. $\lceil s/p \rceil$].
- Let $\hat{B}_i$ [resp. $A_i$] block in $B$ [resp. $A$] which should be merged with $A_i$ [resp. $B_i$].
- $P_i$ cares about $A_i$ and $\hat{B}_i$ if $|\hat{B}_i| \leq \lceil r/p \rceil$.
- Let $C$ be those where one $P_j$ takes already care of.
- $P_i$ cares about $A_i \setminus C$ and $\hat{B}_i \setminus C$. 
Parallel Merging (CREW)

1. Use $P(n)$ processors.
2. Each processor $P_i$ computes for $A[B]$ its part of size $r/P(n) [s/P(n)]$.
3. Each processor $P_i$ computes the part from $B[A]$ which should be merged with its $A$-block [$B$-block].
4. Each processor computes its $A$ or $B$ block, where only he is responsible for.
5. This block has size $O(n/P(n))$.
6. Each processor merges its block into the resulting sequence.
7. Time: $O(\log n + n/P(n))$.
8. Efficiency
   \[
   \frac{n}{O(P(n)) \cdot O(\log n + n/P(n))}.
   \]
9. Efficiency is 1 for $P(n) \leq n/\log n$. 
Do some splitting into pairs of blocks of the same size.
Rekursive splitting into pairs of blocks of the same size.
Thus we may avoid read conflicts.
Merging (EREW)

1. Use $P(n)$ processors.
2. Compute the median $m$ of the sequences $A$ and $B$.
3. Split the sequences $A$ and $B$ in two sub-sequences each of the “same” size ($-1 \leq |A| - |B| \leq 1$).
4. Continue recursively, till all sub-sequences are smaller than $n/P(n)$.
5. Do the merging in the same way as before.

Remaining problem: Find the median of two sequences.
Example for the Median for two Sorted Sequences

- Sequences $A$ and $B$ are sorted.
- Compute median $a$ of $A$ and median $b$ of $B$. 
Median for two Sorted Sequences

1. Sequences $A$ and $B$ are sorted.
2. Compute median $a$ of $A$ and median $b$ of $B$.
3. Median $a \ [b]$ splits $A \ [B]$ into half.
4. The median of $A$ and $B$ is in one block-pair of the four blocks.
5. Search recursively for the median.

Running time: $O(\log n)$
Running Time for Merging (EREW)

1. Use $P(n)$ processors.
2. Compute the median $m$ of the sequences $A$ and $B$. $O(\log n)$
3. Split the sequences $A$ and $B$ in two sub-sequences each.
4. Continue recursively, till all sub-sequences are smaller than $n/P(n)$. $O(\log n \cdot \log(P(n)))$
5. Merge in the same way as before. $O(n/P(n))$

Running time: $O(n/P(n) + \log(n)^2)$. 

Efficiency

$$\frac{O(n)}{O(P(n)) \cdot O(n/P(n) + \log(n)^2)} = \frac{O(n)}{O(n + P(n) \cdot \log(n)^2)}.$$ 

Efficiency is 1 for $P(n) < \frac{n}{(\log n)^2}$. 
Questions

- Explain the motivation behind parallel systems.
- Describe the different models of a PRAM.
- Describe idea of the k-select algorithm.
- For which problems do the running time of CWCR and EWCR algorithms differ?
Legende

- : Nicht relevant
- : Grundlagen, die implizit genutzt werden
- : Idee des Beweises oder des Vorgehens
- : Struktur des Beweises oder des Vorgehens
- : Vollständiges Wissen