Inhalt I

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

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

3 Efficiency
   - Situation
   - Definition
   - Overview

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

5 Merging
   - Sequential Merging
   - Parallel Merging
   - Parallel Merging
Motivation

1. There are limits to the computing power of a single computer
2. Computers become cheaper
3. Specialized computers are expensive
4. There are tasks with large data
5. Many problems are very complex
   - 1. Weather and other simulations
   - 2. Crash tests
   - 3. Military applications
   - 4. Large data: (SETI, ...)
   - 5. More similar problems

6. Thus there is a need for computers with more than one CPU
7. Or a quantum computer?
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 program.
- Addition of two vectors:
  1. Read vector A
  2. Read vector B
  3. Add (each processor)
  4. Output the summation

- **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.
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:
   - CPUs and memory are connected by a network:
   - The difference is more on the practical side.
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 $1 \leq i \leq p$ (prozessor 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 programm.
- The programm 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)

\[
\begin{array}{cccccccc}
 x = 0 & x = 1 & x = 0 & x = 0 & x = 1 & x = 0 & x = 0 & x = 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
\longrightarrow & \lor & \lor & \lor & \lor & \lor & \lor & \rightarrow 1
\end{array}
\]
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 $R_{n+1}$.

Programm: 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)

103  45  30  15

$P_1$  $P_2$  $P_3$  $P_4$

12  6  34  5  7  23  4  11
Computing the sum (EREW)

- 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: $n$.

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

---

Motivation and History

PRAM Introduction

Efficiency

Selection

Merging

1:23 Sum

Walter Unger 8.11.2016 21:42

WS2016/17
Addition of Matrices

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

- 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

for all $P_i$ where $1 \leq i \leq n^2$ 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: $O(n^2)$. 
Multiplication of Matrices

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.
- Program: 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: $O(n^2)$.  

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

```plaintext
for all $P_i$ where $1 \leq i \leq n$ do in parallel
    $R_i \rightarrow P_i(x)$
    for $j = 1$ to $k$
        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></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>22</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>1</td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td>1</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>0</td>
<td>1</td>
<td>0</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>1</td>
<td>0</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>0</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>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>52</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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
</tr>
<tr>
<td>27</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>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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>23</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>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</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>
</tr>
<tr>
<td>34</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
### Compute the Maximum (Idea)

<table>
<thead>
<tr>
<th></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>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td></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>1</td>
</tr>
<tr>
<td>33</td>
<td></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>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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>1</td>
</tr>
<tr>
<td>26</td>
<td></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>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>59</td>
<td></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>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>67</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>52</td>
<td></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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>61</td>
<td></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>27</td>
<td></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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td></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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>67</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td></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>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>56</td>
<td></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></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></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></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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>
</tr>
</tbody>
</table>

| 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.

Programm: Maximum

```plaintext
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
        $R_i \rightarrow P_{i,1}(h)$
        $P_{i,1}(h) \rightarrow R_{n+1}$
```
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

\[
\text{for all } P_i \text{ where } 1 \leq i \leq n \text{ do in parallel} \\
\text{for } j = 1 \text{ to } \lceil \log n \rceil \text{ do} \\
\quad R_i \rightarrow P_i(h) \\
\quad R_h \rightarrow P_i(h) \\
\quad P_i(h) \rightarrow R_i \\
\]

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?
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.
- $Eff_A(n) := \frac{ST(n)}{P_A(n) \cdot T_A(n)}$ the efficiency of $A$.
- $AEff_A(n) := \frac{W_A(n)}{P_A(n) \cdot T_A(n)}$ 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>timet</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>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>$O(n^2.276)$</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>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>
</tbody>
</table>
Task: Compute the \( k \)-th \((k\text{-smallest})\) element in a unsorted sequence \( S = \{s_1, \ldots, s_n\} \).

Lower bound: \( n - 1 \) comparisons

Start with a nice sequential algorithm

Programm: Select\( (k, S) \)

\[
\text{if } |S| \leq 50 \text{ then return } k\text{-th number in } S
\]

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 \( H_i \)

\[
m := \text{Select}(\lceil |M|/2 \rceil, M)
\]

\[
S_1 := \{s \in S \mid s < m\}
\]

\[
S_2 := \{s \in S \mid s = m\}
\]

\[
S_3 := \{s \in S \mid s > m\}
\]

\[
\text{if } |S_1| \geq k \text{ then return } \text{Select}(k, S_1)
\]

\[
\text{if } |S_1| + |S_2| \geq k \text{ then return } m
\]

\[
\text{return } \text{Select}(k - |S_1| - |S_2|, S_3)
\]
Example for the k-th Element (Slow Motion)

**Input/Data:**

| 41 | 11 | 2 | 45 | 32 | 44 | 86 | 18 | 38 | 8 | 87 | 77 | 86 | 75 | 9 | 52 | 43 | 77 | 83 | 67 | 71 |
| 70 | 4  | 11| 22 | 87 | 62 | 26 | 92 | 51 | 62 | 27 | 81 | 36 | 46 | 37 | 88 | 33 | 42 | 77 | 62 | 60 |
| 26 | 26 | 34 | 69 | 20 | 55 | 0  | 67 | 64 | 3  | 46 | 76 | 65 | 53 | 71 | 49 | 50 | 82 | 49 | 52 | 52 |
| 36 | 10 | 59 | 43 | 33 | 55 | 21 | 38 | 49 | 29 | 44 | 18 | 23 | 31 | 14 | 75 | 72 | 18 |
| 16 | 14 | 82 | 71 | 86 | 40 | 80 | 73 | 52 | 5  | 62 | 8  | 62 | 71 | 65 | 64 | 65 | 14 | 96 | 69 | 28 |

**M:**

| 36 | 11 | 34 | 45 | 33 | 55 | 26 | 67 | 52 | 8  | 46 | 76 | 62 | 53 | 37 | 52 | 43 | 42 | 77 | 67 | 52 |

**sorted M:**

| 8  | 11 | 26 | 33 | 34 | 36 | 37 | 42 | 43 | 45 | 46 | 52 | 52 | 52 | 53 | 55 | 62 | 67 | 67 | 76 | 77 |
Example for the k-th Element

Input/Data:

<table>
<thead>
<tr>
<th>42</th>
<th>11</th>
<th>32</th>
<th>58</th>
<th>28</th>
<th>93</th>
<th>70</th>
<th>54</th>
<th>45</th>
<th>73</th>
<th>47</th>
<th>88</th>
<th>22</th>
<th>60</th>
<th>62</th>
<th>3</th>
<th>88</th>
<th>53</th>
<th>83</th>
<th>56</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>3</td>
<td>58</td>
<td>2</td>
<td>0</td>
<td>59</td>
<td>60</td>
<td>1</td>
<td>31</td>
<td>42</td>
<td>1</td>
<td>81</td>
<td>14</td>
<td>4</td>
<td>18</td>
<td>32</td>
<td>79</td>
<td>76</td>
<td>39</td>
<td>7</td>
<td>53</td>
</tr>
<tr>
<td>65</td>
<td>94</td>
<td>59</td>
<td>94</td>
<td>83</td>
<td>80</td>
<td>31</td>
<td>74</td>
<td>83</td>
<td>18</td>
<td>63</td>
<td>68</td>
<td>47</td>
<td>73</td>
<td>65</td>
<td>60</td>
<td>94</td>
<td>52</td>
<td>88</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>91</td>
<td>38</td>
<td>90</td>
<td>53</td>
<td>70</td>
<td>23</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>90</td>
<td>29</td>
<td>76</td>
<td>81</td>
<td>77</td>
<td>75</td>
<td>19</td>
<td>63</td>
<td>0</td>
<td>91</td>
<td>43</td>
</tr>
<tr>
<td>8</td>
<td>91</td>
<td>89</td>
<td>6</td>
<td>20</td>
<td>69</td>
<td>52</td>
<td>35</td>
<td>81</td>
<td>49</td>
<td>0</td>
<td>6</td>
<td>55</td>
<td>43</td>
<td>41</td>
<td>87</td>
<td>89</td>
<td>86</td>
<td>55</td>
<td>44</td>
<td>51</td>
</tr>
</tbody>
</table>

M:

| 42 | 91 | 58 | 58 | 28 | 70 | 52 | 35 | 45 | 42 | 47 | 68 | 47 | 60 | 62 | 60 | 88 | 63 | 55 | 44 | 43 |

sorted M:

| 28 | 35 | 42 | 42 | 43 | 44 | 45 | 47 | 52 | 55 | 58 | 58 | 60 | 60 | 62 | 62 | 63 | 68 | 70 | 88 | 91 |
### Example for the k-th Element (Worst Case)

**Input/Data:**

<table>
<thead>
<tr>
<th>77</th>
<th>94</th>
<th>85</th>
<th>52</th>
<th>85</th>
<th>68</th>
<th>56</th>
<th>58</th>
<th>85</th>
<th>76</th>
<th>73</th>
<th>64</th>
<th>57</th>
<th>50</th>
<th>56</th>
<th>94</th>
<th>62</th>
<th>74</th>
<th>64</th>
<th>93</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>93</td>
<td>59</td>
<td>87</td>
<td>85</td>
<td>60</td>
<td>51</td>
<td>91</td>
<td>57</td>
<td>86</td>
<td>76</td>
<td>91</td>
<td>62</td>
<td>91</td>
<td>51</td>
<td>71</td>
<td>55</td>
<td>92</td>
<td>68</td>
<td>52</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>22</td>
<td>6</td>
<td>27</td>
<td>32</td>
<td>17</td>
<td>34</td>
<td>8</td>
<td>41</td>
<td>24</td>
<td>87</td>
<td>64</td>
<td>73</td>
<td>76</td>
<td>76</td>
<td>80</td>
<td>52</td>
<td>59</td>
<td>56</td>
<td>88</td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>9</td>
<td>5</td>
<td>2</td>
<td>27</td>
<td>26</td>
<td>28</td>
<td>13</td>
<td>29</td>
<td>7</td>
<td>55</td>
<td>74</td>
<td>67</td>
<td>70</td>
<td>83</td>
<td>94</td>
<td>79</td>
<td>51</td>
<td>65</td>
<td>74</td>
</tr>
<tr>
<td>36</td>
<td>23</td>
<td>38</td>
<td>34</td>
<td>1</td>
<td>43</td>
<td>38</td>
<td>9</td>
<td>23</td>
<td>35</td>
<td>5</td>
<td>53</td>
<td>76</td>
<td>90</td>
<td>66</td>
<td>90</td>
<td>94</td>
<td>84</td>
<td>80</td>
<td>86</td>
<td>62</td>
</tr>
</tbody>
</table>

**M:**

| 36 | 23 | 38 | 34 | 27 | 43 | 38 | 34 | 23 | 41 | 24 | 64 | 64 | 73 | 66 | 83 | 80 | 79 | 64 | 65 | 74 |

**sorted M:**

| 23 | 23 | 24 | 27 | 34 | 34 | 36 | 38 | 38 | 41 | 43 | 64 | 64 | 64 | 65 | 66 | 73 | 74 | 79 | 80 | 83 |
Running Time

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

```latex
\begin{align*}
\text{if } |S| &\leq 50 \text{ then return } k\text{-th number in } S \\
\text{Split } S \text{ in } \lceil n/5 \rceil \text{ sub-sequences } H_i \text{ of size } \leq 5 \\
\text{Sort each } H_i \\
\text{Let } M \text{ 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\} \\
\text{if } |S_1| &\geq k \text{ then return } \text{Select}(k, S_1) \\
\text{if } |S_1| + |S_2| &\geq k \text{ then return } m \\
\text{return } \text{Select}(k - |S_1| - |S_2|, S_3)
\end{align*}
```
Running Time

- Claim: $T(n) \leq 20 \cdot r \cdot n$ with $r = \max(d, c)$.

- Proof:
  - $n = 50$:
    
    
    \[
    T(n) \leq c \cdot n + \frac{d \cdot n}{5} + \frac{3 \cdot d \cdot n}{4}
    \]
  
  - $n > 50$:
    
    \[
    T(n) \leq c \cdot n + T\left(\frac{d \cdot n}{5}\right) + T\left(\frac{3 \cdot d \cdot n}{4}\right)
    \]

- 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:**

```
29 12 85 65 13 71 73 68 88 72 78 53 48 77 55 71 78 24 66 86 58 22 41 97 21 58 84 58 54 0 41 84 93 8 47
57 11 80 33 82 40 82 46 24 35 5 59 69 40 83 89 57 92 31 48 21 29 75 32 60 46 55 35 45 52 8 51 16 3 75
3 91 81 84 49 65 9 67 7 35 89 76 19 61 91 18 83 92 94 26 69 95 56 14 60 13 28 7 91 72 17 69 94 42 65
3 95 87 23 89 86 23 61 43 1 17 30 0 86 54 51 56 51 84 76 32 31 43 88 49 28 21 23 63 2 86 11 15 4 29
29 2 73 65 88 83 73 5 10 93 72 89 5 45 20 74 25 33 12 22 86 54 74 21 94 27 83 70 30 10 73 1 26 64 50
19 27 2 25 11 31 79 33 89 79 2 6 57 57 59 78 11 35 90 86 96 57 80 43 6 61 54 87 85 93 22 70 13 29 44
68 71 38 46 35 39 69 86 60 34 60 86 82 13 74 44 79 37 45 91 58 23 36 48 74 11 19 20 97 16 63 30 17 60 19
82 8 13 76 35 92 64 3 62 55 43 44 29 25 31 5 19 70 42 12 67 58 68 2 27 19 38 28 18 88 0 92 29 8 97
33 72 33 21 5 85 96 0 4 85 42 24 58 28 25 4 16 66 19 16 39 16 96 82 44 35 82 16 79 64 87 95 54 80 82
56 81 33 90 29 64 68 43 45 28 30 20 84 63 53 51 27 34 24 16 55 90 31 81 96 27 45 97 73 39 52 66 91 89 39
48 19 16 31 65 19 32 47 85 56 59 47 0 50 31 75 60 9 69 62 67 85 61 29 94 90 7 35 91 46 12 0 79 57 96
18 91 48 68 55 80 38 66 86 5 68 19 46 92 6 95 94 78 39 40 93 76 93 60 51 64 67 17 14 42 79 91 72 33 64
6 76 49 22 76 53 34 63 83 16 50 11 92 60 95 32 89 18 60 46 82 3 64 37 55 3 16 17 95 28 55 0 21 16 86
92 7 23 57 24 13 36 42 83 28 87 19 37 96 89 40 32 11 46 90 50 80 92 35 33 67 8 21 89 42 81 69 17 33 6
73 24 44 21 35 26 65 40 18 36 44 75 97 61 26 12 32 11 74 0 63 28 61 48 66 49 16 74 33 2 49 82 13 55 32
```

**M:**

```
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_10 P_11 P_12 P_13 P_14 P_15 P_16 P_17 P_18 P_19 P_20 P_21 P_22 P_23 P_24 P_25 P_26 P_27 P_28 P_29 P_30 P_31 P_32 P_33 P_34 P_35
```

**sorted M:**

```
26 27 28 33 33 35 35 35 35 38 42 43 44 44 46 46 46 46 48 50 50 51 52 54 54 55 55 38 28 73 42 52 69 26 33 50
26 27 28 33 33 35 35 35 35 38 42 43 44 44 46 46 46 46 48 50 50 51 52 54 54 55 55 38 28 73 42 52 69 26 33 50
```
Parallel k-Select

Programm: ParSelect(k,S)

1: \[ \text{if } |S| \leq k_1 \text{ then } P_1 \text{ returns } Select(k, S). \]

2: \[ S \text{ is split into } \lceil |S|^{1-x} \rceil \text{ sub-sequences } S_i \text{ with } |S_i| \leq \lceil n^x \rceil \]
   \[ P_i \text{ stores the start-address of } S_i. \]

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

4: \[ m := ParSelect(\lceil |M|/2 \rceil, M). \]

5: \[ \text{More to come!} \]
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:
$l_i := \sum_{j=1}^{i} |L_j|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.
$e_i := \sum_{j=1}^{i} |E_j|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.
$g_i := \sum_{j=1}^{i} |G_j|$ 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 \( \text{ParSelect}(k,L) \)
if \( |L| + |E| \geq k \) then return \( m \)
return \( \text{Select}(k - |L| - |E|, G) \)
Parallel k-Select (Running Time)

Programm: ParSelect(k,S)

1: \(O(1)\)
   \[\text{if } |S| \leq k_1 \text{ then } P_1 \text{ returns } \text{Select}(k,S)\].

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

4: \(T_{\text{ParSelect}}(n^{1-x})\)
   \[m := \text{ParSelect}(\lceil |M|/2 \rceil, M)\].
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:

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

- **Input:**
  \[ A = (a_1, a_2, \cdots, a_r) \] 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
  
  \[
  \begin{align*}
  &i := 1; j := 1; n := r + s \\
  &\text{for } k := 1 \text{ to } n \text{ do} \\
  &\quad \text{if } a_i < b_j \\
  &\quad \quad \text{then } c_k := a_i; i := i + 1; \\
  &\quad \quad \text{else } c_k := b_j; j := j + 1;
  \end{align*}
  \]

- 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. \( \Omega(n) \).
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 ($\leq |A| - |B| \leq 1$).
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 $B$.
4. The median of $A$ and $B$ is in one block-pair of the four blocks.
5. Search recursiv 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