Contents 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

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.
   - Weather and other simulations.
   - Crash tests.
   - Military applications.
   - Large data: (SETI, ...).
   - More similar problems.
6. Thus there is the need for computers with more than one CPU.
7. 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 intersect each other.
- Each processor is the same.
- There is a global synchronisation.
- 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 sum

- 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, ...

\(HQ(n), CCC(n), BF(n), SE(n), DB(n), \ldots\)

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)
CPU and memory are one logical unit:

1. The components are connected by a single network.

CPUs and memory are connected by a network:

2. The components are connected by multiple networks.

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 $1 \leq i \leq p$ (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)

\[\begin{align*}
x &= 0 & x &= 1 & x &= 0 & x &= 0 & x &= 1 & x &= 0 & x &= 0 & x &= 1 \\
0 & \lor 1 & 0 & \lor 0 & 0 & \lor 1 & 0 & \lor 0 & 0 & \lor 1 & \rightarrow 1
\end{align*}\]
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 $x = true$ 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)

- 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).
- Modell: 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 $k \in \mathbb{N}$. 
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 $R_{2\cdot n^2}$ contains $B$ (one row after the other).

Result in $R_{1+2\cdot n^2}$ bis $R_{3\cdot n^2}$.

Programm: MatSumme

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

$R_i \rightarrow P_i(x)$

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

$x := x + y$

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

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 $k \in \mathbb{N}$. 

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 $R_{2\cdot n^2}$ contains $B$ (one row after the other).

Result in $R_{1+2\cdot n^2}$ bis $R_{3\cdot n^2}$.

Programm: MatSumme

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

$R_i \rightarrow P_i(x)$

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

$x := x + y$

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

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 $k \in \mathbb{N}$. 

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 $R_{2\cdot n^2}$ contains $B$ (one row after the other).

Result in $R_{1+2\cdot n^2}$ bis $R_{3\cdot n^2}$.

Programm: MatSumme

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

$R_i \rightarrow P_i(x)$

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

$x := x + y$

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

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 $k \in \mathbb{N}$. 

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 $R_{2\cdot n^2}$ contains $B$ (one row after the other).

Result in $R_{1+2\cdot n^2}$ bis $R_{3\cdot n^2}$.

Programm: MatSumme

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

$R_i \rightarrow P_i(x)$

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

$x := x + y$

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

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 $k \in \mathbb{N}$. 

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 $R_{2\cdot n^2}$ contains $B$ (one row after the other).

Result in $R_{1+2\cdot n^2}$ bis $R_{3\cdot n^2}$.

Programm: MatSumme

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

$R_i \rightarrow P_i(x)$

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

$x := x + y$

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

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 $k \in \mathbb{N}$. 

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 $R_{2\cdot n^2}$ contains $B$ (one row after the other).

Result in $R_{1+2\cdot n^2}$ bis $R_{3\cdot n^2}$.

Programm: MatSumme

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

$R_i \rightarrow P_i(x)$

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

$x := x + y$

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

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 $k \in \mathbb{N}$. 

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 $R_{2\cdot n^2}$ contains $B$ (one row after the other).

Result in $R_{1+2\cdot n^2}$ bis $R_{3\cdot n^2}$.

Programm: MatSumme

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

$R_i \rightarrow P_i(x)$

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

$x := x + y$

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

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 $k \in \mathbb{N}$.
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
    \[ A_{i,l} \rightarrow P_{i,j}(a) \]
    \[ B_{l,j} \rightarrow P_{i,j}(b) \]
    \[ h = h + a \cdot b \]
    \[ P_{i,j}(h) \rightarrow C_{i,j} \]
- Running time: $O(n)$.
- Number of processors: $O(n^2)$.
- Size of memory: $O(n^2)$. 

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

$A_{i,l} \rightarrow P_{i,j}(a)$

$B_{i,j} \rightarrow P_{i,j}(b)$

$h = h + a \cdot b$

$P_{i,j}(h) \rightarrow C_{i,j}$

Running time: $O(n)$.

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

Size of memory: $O(n^2)$. 

\[
\begin{align*}
A_{i,j} &= R(i-1) \cdot n + j \\
B_{i,j} &= R(i-1) \cdot n + j + n^2 \\
C_{i,j} &= R(i-1) \cdot n + j + 2 \cdot n^2 \\
P_{i,j} &= P(i-1) \cdot n + j
\end{align*}
\]
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></th>
<th>22</th>
<th>33</th>
<th>41</th>
<th>26</th>
<th>59</th>
<th>57</th>
<th>52</th>
<th>61</th>
<th>27</th>
<th>49</th>
<th>67</th>
<th>23</th>
<th>56</th>
<th>14</th>
<th>12</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</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>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>0</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>
</tr>
<tr>
<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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>
<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>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>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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)

<table>
<thead>
<tr>
<th></th>
<th>22</th>
<th>33</th>
<th>41</th>
<th>26</th>
<th>59</th>
<th>67</th>
<th>52</th>
<th>61</th>
<th>27</th>
<th>49</th>
<th>67</th>
<th>23</th>
<th>56</th>
<th>14</th>
<th>12</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<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>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</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>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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

\[
\text{for all } P_{i,1} \text{ where } 1 \leq i \leq n \text{ do in parallel}
\]
\[
P_{i,1}(1) \rightarrow W_i
\]

\[
\text{for all } P_{i,j} \text{ where } 1 \leq i, j \leq n \text{ do in parallel}
\]
\[
R_i \rightarrow P_{i,j}(a)
\]
\[
R_j \rightarrow P_{i,j}(b)
\]
\[
\text{if } a < b \text{ then } P_{i,j}(0) \rightarrow W_i
\]

\[
\text{for all } P_{i,1} \text{ where } 1 \leq i \leq n \text{ do in parallel}
\]
\[
W_i \rightarrow P_{i,1}(h)
\]
\[
\text{if } h = 1 \text{ then}
\]
\[
R_i \rightarrow P_{i,1}(h)
\]
\[
P_{i,1}(h) \rightarrow R_{n+1}
\]
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
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.

- $Eff_A(n) := \frac{ST(n)}{P_A(n) \cdot T_A(n)}$ 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>
<tr>
<td>Problem</td>
<td>processors</td>
<td>timet</td>
<td>ST(n)</td>
<td>Eff</td>
<td>Modell</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td>--------</td>
<td>---------</td>
<td>--------</td>
<td>--------</td>
</tr>
<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>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></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></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></td>
<td>CREW</td>
</tr>
</tbody>
</table>

- **ERCW** refers to Exclusive Read Exclusive Write.
- **EREW** refers to Exclusive Read Exclusive Write.
- **CRCW** refers to Concurrent Read Concurrent Write.
- **CREW** refers to Concurrent Read Exclusive Write.
**k-th Element**

- Task: Compute the $k$-th ($k$-smallest) element in an unsorted sequence $S = \{s_1, \ldots, s_n\}$.
- Lower bound: $n - 1$ comparisons
- Start with a nice sequential algorithm

Program: Select($k, S$)

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

**Input/Data:**

<table>
<thead>
<tr>
<th>65</th>
<th>21</th>
<th>40</th>
<th>79</th>
<th>32</th>
<th>53</th>
<th>59</th>
<th>96</th>
<th>35</th>
<th>87</th>
<th>42</th>
<th>27</th>
<th>92</th>
<th>85</th>
<th>79</th>
<th>93</th>
<th>86</th>
<th>16</th>
<th>23</th>
<th>75</th>
<th>52</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>78</td>
<td>70</td>
<td>58</td>
<td>55</td>
<td>82</td>
<td>35</td>
<td>11</td>
<td>37</td>
<td>93</td>
<td>5</td>
<td>58</td>
<td>31</td>
<td>50</td>
<td>90</td>
<td>42</td>
<td>24</td>
<td>92</td>
<td>21</td>
<td>82</td>
<td>65</td>
</tr>
<tr>
<td>29</td>
<td>47</td>
<td>10</td>
<td>79</td>
<td>63</td>
<td>37</td>
<td>8</td>
<td>53</td>
<td>32</td>
<td>33</td>
<td>47</td>
<td>65</td>
<td>23</td>
<td>56</td>
<td>84</td>
<td>84</td>
<td>57</td>
<td>78</td>
<td>0</td>
<td>17</td>
<td>93</td>
</tr>
<tr>
<td>84</td>
<td>35</td>
<td>14</td>
<td>67</td>
<td>24</td>
<td>57</td>
<td>18</td>
<td>37</td>
<td>5</td>
<td>32</td>
<td>79</td>
<td>31</td>
<td>75</td>
<td>51</td>
<td>73</td>
<td>20</td>
<td>85</td>
<td>1</td>
<td>85</td>
<td>54</td>
<td>59</td>
</tr>
<tr>
<td>90</td>
<td>22</td>
<td>77</td>
<td>82</td>
<td>67</td>
<td>46</td>
<td>3</td>
<td>14</td>
<td>26</td>
<td>87</td>
<td>30</td>
<td>72</td>
<td>42</td>
<td>50</td>
<td>66</td>
<td>4</td>
<td>97</td>
<td>59</td>
<td>83</td>
<td>64</td>
<td>28</td>
</tr>
</tbody>
</table>

**M:**

| 65 | 35 | 40 | 79 | 55 | 53 | 18 | 37 | 32 | 87 | 42 | 58 | 42 | 51 | 79 | 42 | 85 | 59 | 23 | 64 | 59 |

**sorted M:**

| 18 | 23 | 32 | 35 | 37 | 40 | 42 | 42 | 42 | 51 | 53 | 55 | 58 | 59 | 59 | 64 | 65 | 79 | 79 | 85 | 87 |
Example for the k-th Element

Input/Data:

|   | 97 | 8  | 21 | 50 | 67 | 38 | 80 | 56 | 36 | 54 | 1  | 54 | 3  | 89 | 60 | 67 | 93 | 15 | 25 | 9  | 8  |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 2 | 87 | 92 | 0  | 56 | 35 | 7  | 81 | 66 | 64 | 72 | 32 | 68 | 19 | 79 | 29 | 60 | 55 | 7  | 61 | 62 | 18 |
| 2 | 27 | 31 | 16 | 38 | 86 | 65 | 79 | 72 | 64 | 41 | 73 | 78 | 67 | 48 | 94 | 64 | 22 | 59 | 48 | 86 | 60 |
| 1 | 1  | 87 | 91 | 50 | 44 | 24 | 80 | 17 | 47 | 76 | 8  | 85 | 79 | 68 | 23 | 17 | 26 | 58 | 20 | 86 | 30 |
| 2 | 2  | 97 | 12 | 85 | 19 | 14 | 53 | 56 | 51 | 95 | 3  | 44 | 72 | 43 | 79 | 58 | 52 | 48 | 97 | 40 | 45 |

M:  

|   | 27 | 87 | 16 | 50 | 44 | 24 | 80 | 56 | 51 | 72 | 8  | 68 | 67 | 68 | 60 | 60 | 52 | 48 | 48 | 62 | 30 |

sorted M:  

|   | 8  | 16 | 24 | 27 | 30 | 44 | 48 | 48 | 50 | 51 | 52 | 56 | 60 | 60 | 62 | 67 | 68 | 68 | 72 | 80 | 87 |
### Example for the k-th Element (Worst Case)

**Input/Data:**

| 68 | 69 | 89 | 72 | 91 | 84 | 53 | 73 | 57 | 56 | 65 | 94 | 81 | 78 | 77 | 60 | 88 | 93 | 55 | 79 | 81 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 59 | 56 | 50 | 57 | 67 | 52 | 51 | 80 | 62 | 58 | 76 | 54 | 68 | 52 | 89 | 80 | 61 | 57 | 75 | 91 | 74 |
| 32 | 9  | 41 | 0  | 24 | 27 | 13 | 11 | 34 | 41 | 11 | 81 | 86 | 93 | 74 | 56 | 86 | 90 | 63 | 65 | 60 |
| 42 | 10 | 24 | 14 | 44 | 17 | 13 | 17 | 15 | 13 | 11 | 64 | 62 | 74 | 94 | 51 | 65 | 69 | 56 | 57 | 59 |
| 44 | 37 | 32 | 29 | 35 | 29 | 28 | 31 | 7  | 4  | 2  | 73 | 81 | 69 | 70 | 53 | 55 | 67 | 64 | 92 | 62 |

**M:**

| 44 | 37 | 41 | 29 | 44 | 29 | 28 | 31 | 34 | 41 | 11 | 73 | 81 | 74 | 77 | 56 | 65 | 69 | 63 | 79 | 62 |

**sorted M:**

| 11 | 28 | 29 | 29 | 31 | 34 | 37 | 41 | 41 | 44 | 44 | 56 | 62 | 63 | 65 | 69 | 73 | 74 | 77 | 79 | 81 |
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 $\text{Select}(k - |S_1| - |S_2|, S_3)$
Running Time

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

Proof:

$n = 50$:

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

$n > 50$:

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

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

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:**

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

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

\[
M:\n53 \quad 38 \quad 58 \quad 35 \quad 45 \quad 37 \quad 29 \quad 54 \quad 41 \quad 52 \quad 61 \quad 58 \quad 64 \quad 53 \quad 60 \quad 28 \quad 32 \quad 36 \quad 20 \quad 49 \quad 45 \quad 50 \quad 22 \quad 48 \quad 49 \quad 48 \quad 56 \quad 43 \quad 59 \quad 63 \quad 56 \quad 54 \quad 33 \quad 46 \quad 58
\]

sorted \( M:\n20 \quad 22 \quad 28 \quad 29 \quad 32 \quad 33 \quad 35 \quad 36 \quad 37 \quad 38 \quad 41 \quad 43 \quad 45 \quad 45 \quad 46 \quad 48 \quad 48 \quad 49 \quad 49 \quad 50 \quad 52 \quad 53 \quad 53 \quad 54 \quad 54 \quad 56 \quad 56 \quad 58 \quad 58 \quad 58 \quad 59 \quad 60 \quad 61 \quad 63 \quad 64
Parallel k-Select

Programm: ParSelect(k,S)

1: 
   if |S| ≤ k₁ then \( P₁ \) returns \( \text{Select}(k, S) \).

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

3: 
   for all \( P_i \) where \( 1 ≤ i ≤ \lceil n^{1-x} \rceil \) 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: 
   \( m := \text{ParSelect}(\lceil |M|/2 \rceil, M) \).

5: 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 $1 \leq i \leq \lfloor n^{1-x} \rfloor$ 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 $1 \leq i \leq \lfloor n^{1-x} \rfloor$.
$e_i := \sum_{j=1}^{i} |E_i|$ for all $1 \leq i \leq \lfloor n^{1-x} \rfloor$.
$g_i := \sum_{j=1}^{i} |G_i|$ for all $1 \leq i \leq \lfloor n^{1-x} \rfloor$.

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: $O(1)$
   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:
\[ l_i := \sum_{j=1}^{i} |L_i| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \]
\[ e_i := \sum_{j=1}^{i} |E_i| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \]
\[ g_i := \sum_{j=1}^{i} |G_i| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \]
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:**
  
  \[ A = (a_1, a_2, \cdots, a_r) \] 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 \( n = r + s \).

- **Programm: 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).
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 $P(n) \leq n/\log n$. 
Idea for Merging (EREW)

- 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?
Legend

- : Not of relevance
- : implicitly used basics
- : idea of proof or algorithm
- : structure of proof or algorithm
- : Full knowledge