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 interact 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 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

- CPU
- Bus
- Memory
- Link
- Switch
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 IV (Network)
CPU and memory are one logical unit:

1. CPU and memory are one logical unit:
   - CPU
   - RAM
   - CPU
   - RAM
   - CPU
   - RAM
   - CPU
   - RAM

   Network

CPUs and memory are connected by a network:

2. CPUs and memory are connected by a network:
   - CPU
   - CPU
   - CPU
   - CPU
   - CPU

   Network

   RAM
   - RAM
   - RAM
   - RAM
   - RAM

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)

\[ x = 0 \quad x = 1 \quad x = 0 \quad x = 0 \quad x = 1 \quad x = 0 \quad x = 0 \quad x = 1 \]

\[ 0 \lor 1 \lor 0 \lor 0 \lor 1 \lor 0 \lor 0 \lor 1 \rightarrow 1 \]
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)

103  45  30  15

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

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 \( R_i \) for \( 1 \leq i \leq n \).
- 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 \( k \in \mathbb{N} \).
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

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

\[
\begin{align*}
R_i & \rightarrow P_i(x) \\
R_{i+n^2} & \rightarrow P_i(y) \\
x & := x + y \\
P_i(x) & \rightarrow R_{i+2.n^2}
\end{align*}
\]

- 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

1. $h = 0$
2. for $l = 1$ to $n$ do
3.   $A_{i,l} \rightarrow P_{i,j}(a)$
4.   $B_{l,j} \rightarrow P_{i,j}(b)$
5.   $h = h + a \cdot b$
6.   $P_{i,j}(h) \rightarrow C_{i,j}$

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

$$
A_{i,j} = R(i-1) \cdot n + j \\
B_{i,j} = R(i-1) \cdot n + j + n^2 \\
C_{i,j} = R(i-1) \cdot n + j + 2 \cdot n^2 \\
P_{i,j} = P(i-1) \cdot n + j
$$
Motivation and History
PRAM Introduction
Efficiency
Selection
Merging

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: $O(n^2)$. 

\[
\begin{align*}
A_{i,j} &= R(i-1) \cdot n + j \\
B_{i,j} &= R(i-1) \cdot n + j + n^2 \\
C_{i,j} &= R(i-1) \cdot n + j + 2 \cdot n^2 \\
P_{i,j} &= P(i-1) \cdot n + j
\end{align*}
\]
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
- Program: Summe
  
  \[
  \begin{align*}
  \text{for all } P_i \text{ where } 1 \leq i \leq n \text{ do in parallel} \\
  & R_i \rightarrow P_i(x) \\
  & \text{for } j = 1 \text{ to } k \text{ do} \\
  & \quad \text{if } i > 2^{j-1} \text{ then} \\
  & \quad \quad R_{i-2^{j-1}} \rightarrow P_i(y) \\
  & \quad \quad x := x + y \\
  & \quad \quad P_i(x) \rightarrow R_i
  \end{align*}
  \]
- 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 $m = \max_{j=1}^{n} x_j$ 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>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>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>
<td>1</td>
</tr>
<tr>
<td>26</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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<td>22</td>
</tr>
</tbody>
</table>
Compute the Maximum (Idea)

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 22| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |   |
| 33| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |   |
| 41| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |   |
| 26| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |   |
| 59| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |   |
| 67| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |   |
| 52| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |   |
| 61| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |   |
| 27| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |   |
| 49| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |   |
| 67| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |   |
| 23| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |   |
| 56| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |   |
| 14| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   |
| 12| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   |
| 34| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |   |
|   | 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

  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 $i$.
- Program: 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?
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>
</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 Sum</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>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 ($k$-smallest) element in a unsorted sequence $S = \{s_1, \cdots, s_n\}$.

Lower bound: $n - 1$ comparisons

Start with a nice sequential algorithm

Program: 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 $H_i$

$m := Select(\lceil |M|/2 \rceil, M)$

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

$S_2 := \{s \in S \mid s = m\}$

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

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:

| 80 | 33 | 53 | 67 | 22 | 72 | 0 | 39 | 14 | 79 | 24 | 27 | 64 | 87 | 67 | 74 | 33 | 47 | 59 | 76 | 21 |
|----|----|----|----|----|----|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 4  | 44 | 88 | 58 | 61 | 47 | 76 | 77 | 29 | 51 | 84 | 14 | 10 | 36 | 78 | 12 | 27 | 92 | 49 | 40 | 35 |
| 15 | 79 | 65 | 40 | 97 | 8  | 3  | 28 | 61 | 25 | 75 | 7  | 26 | 86 | 94 | 39 | 50 | 23 | 41 | 8  | 30 |
| 57 | 42 | 86 | 45 | 64 | 80 | 79 | 72 | 66 | 62 | 1  | 66 | 83 | 59 | 47 | 38 | 49 | 39 | 88 | 56 | 50 |
| 61 | 90 | 6  | 27 | 45 | 53 | 19 | 61 | 93 | 69 | 72 | 13 | 18 | 19 | 43 | 61 | 97 | 23 | 3  | 92 | 39 |

M:

| 57 | 44 | 65 | 45 | 61 | 53 | 19 | 61 | 61 | 62 | 72 | 14 | 26 | 59 | 67 | 39 | 49 | 39 | 49 | 56 | 35 |

sorted M:

| 14 | 19 | 26 | 35 | 39 | 39 | 44 | 45 | 49 | 49 | 53 | 56 | 57 | 59 | 61 | 61 | 61 | 62 | 65 | 67 | 72 |
Example for the k-th Element

Input/Data:

```
|  94 |  31 |  90 |  86 |  60 |  53 |  52 |  23 |  12 |  49 |  51 |  26 |  87 |  45 |   1 |  52 |  57 |  16 |  35 |  12 |  36 |
|  83 |  27 |  93 |  70 |  68 |  45 |  55 |  26 |  45 |  95 |  32 |  31 |  93 |  24 |  78 |  78 |  59 |  50 |  62 |  17 |  40 |
|  0  |  58 |  82 |  21 |  54 |  33 |  42 |  34 |  64 |  63 |  73 |  78 |  58 |  57 |  30 |  66 |  93 |  33 |  19 |  96 |  78 |
|  47 |  57 |  91 |  59 |  43 |  54 |  81 |  88 |  60 |  36 |  7 |  42 |  58 |  66 |  80 |  78 |  59 |  43 |  79 |  62 |  46 |
|  20 |  93 |   2 |  68 |  41 |  61 |  51 |  74 |  82 |  58 | 10 | 32 |  12 |  67 |  93 |  54 |  48 |  58 |  56 |  89 |  26 |
```

M:

```
|  47 |  57 |  90 |  68 |  54 |  53 |  52 |  34 |  60 |  58 |  32 |  32 |  58 |  57 |  78 |  66 |  59 |  43 |  56 |  62 |  40 |
```

sorted M:

```
|  32 |  32 |  34 |  40 |  43 |  47 |  52 |  53 |  54 |  56 |  57 |  57 |  58 |  58 |  59 |  60 |  62 |  66 |  68 |  78 |  90 |
```
Example for the k-th Element (Worst Case)

Input/Data:

<table>
<thead>
<tr>
<th>73</th>
<th>65</th>
<th>54</th>
<th>57</th>
<th>71</th>
<th>94</th>
<th>61</th>
<th>85</th>
<th>73</th>
<th>64</th>
<th>93</th>
<th>82</th>
<th>82</th>
<th>67</th>
<th>71</th>
<th>59</th>
<th>84</th>
<th>61</th>
<th>56</th>
<th>91</th>
<th>69</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>76</td>
<td>64</td>
<td>88</td>
<td>59</td>
<td>74</td>
<td>53</td>
<td>68</td>
<td>77</td>
<td>56</td>
<td>89</td>
<td>88</td>
<td>89</td>
<td>76</td>
<td>64</td>
<td>60</td>
<td>56</td>
<td>80</td>
<td>64</td>
<td>67</td>
<td>56</td>
</tr>
<tr>
<td>29</td>
<td>17</td>
<td>10</td>
<td>42</td>
<td>33</td>
<td>10</td>
<td>34</td>
<td>3</td>
<td>19</td>
<td>42</td>
<td>4</td>
<td>69</td>
<td>84</td>
<td>89</td>
<td>89</td>
<td>83</td>
<td>85</td>
<td>70</td>
<td>52</td>
<td>54</td>
<td>77</td>
</tr>
<tr>
<td>43</td>
<td>26</td>
<td>5</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>1</td>
<td>18</td>
<td>29</td>
<td>0</td>
<td>81</td>
<td>52</td>
<td>82</td>
<td>67</td>
<td>90</td>
<td>67</td>
<td>66</td>
<td>64</td>
<td>66</td>
<td>52</td>
</tr>
<tr>
<td>12</td>
<td>40</td>
<td>13</td>
<td>11</td>
<td>11</td>
<td>5</td>
<td>42</td>
<td>6</td>
<td>44</td>
<td>4</td>
<td>16</td>
<td>77</td>
<td>73</td>
<td>85</td>
<td>78</td>
<td>78</td>
<td>70</td>
<td>55</td>
<td>73</td>
<td>58</td>
<td>60</td>
</tr>
</tbody>
</table>

M:

|  43 |  40 |  13 |  42 |  33 |  18 |  42 |   6 |  44 |  42 |  16 |  81 |  82 |  82 |  71 |  78 |  70 |  66 |  64 |  66 |  60 |

sorted M:

|   6 |  13 |  16 |  18 |  33 |  40 |  42 |  42 |  43 |  44 |  60 |  64 |  66 |  66 |  70 |  71 |  78 |  81 |  82 |  82 |
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:**

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

**P:** P₁ P₂ P₃ P₄ P₅ P₆ P₇ P₈ P₉ P₁₀ P₁₁ P₁₂ P₁₃ P₁₄ P₁₅ P₁₆ P₁₇ P₁₈ P₁₉ P₂₀ P₂₁ P₂₂ P₂₃ P₂₄ P₂₅ P₂₆ P₂₇ P₂₈ P₂₉ P₃₀ P₃₁ P₃₂ P₃₃ | P₃₄ P₃₅

**M:**

| 63 53 | 31 50 43 41 | 31 58 | 58 47 43 44 52 32 35 26 55 54 58 63 | 22 43 62 | 49 48 40 44 29 66 44 41 | 27 46 51 26 |

sorted M:

| 22 26 26 27 29 31 31 32 35 40 41 41 43 43 44 44 44 44 | 46 47 48 49 50 51 52 53 54 55 58 58 58 68 62 63 63 66 |
Parallel k-Select

Programm: ParSelect(k,S)
1:  
   \textbf{if} \ |S| \leq k_1 \ \textbf{then} \ P_1 \ \textbf{returns} \ Select(k, S).
2:  
   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:  
   \textbf{for all} \ P_i \ where \ 1 \leq i \leq \lceil n^{1-x} \rceil \ \textbf{do in parallel} \\
   \quad m_i := \ Select(\lceil|S_i|/2\rceil, S_i) \\
   \quad P_i(m_1) \rightarrow R_i. \\
   \quad \text{Assume \ in \ the \ following \ that} \ M \ \text{is \ the \ sequence \ of \ these \ values.}
4:  
   m := \ ParSelect(\lceil|M|/2\rceil, M).
5:  
   \textbf{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:

$l_i := \sum_{j=1}^{i} |L_i|$ 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

\[ P_i \text{ writes } L_i \text{ in } R_{l_i-1+1}, \ldots, R_{l_i}. \]
\[ P_i \text{ writes } E_i \text{ in } R_{e_i-1+1}, \ldots, R_{e_i}. \]
\[ P_i \text{ writes } G_i \text{ in } R_{g_i-1+1}, \ldots, R_{g_i}. \]

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: $T_{\text{ParSelect}}(n^{1-x})$
   \[m := \text{ParSelect}(\lceil |M|/2 \rceil, M).\]
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
  
  \[
  \begin{align*}
  i &:= 1; j := 1; n := r + s \\
  \text{for } k := 1 \text{ to } n \text{ do} \\
  &\quad \text{if } a_i < b_j \\
  &\quad \quad \text{then } c_k := a_i; i := i + 1; \\
  &\quad \quad \text{else } c_k := b_j; j := j + 1;
  \end{align*}
  \]

- 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 $P(n) \leq n / \log n$. 
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 $B$.
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