Theory of Parallel and Distributed Systems (WS2016/17)
Kapitel 1
First Algorithms for PRAM

Walter Unger

Lehrstuhl für Informatik 1

12:00 Uhr, den 30. Januar 2017
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 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:

   ![Diagram 1: CPU and memory as a single logical unit](image)

   The difference is more on the practical side.

2. CPUs and memory are connected by a network:

   ![Diagram 2: CPUs and memory as separate units](image)
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)
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 $R_{n+1}$.
- **Model:** CRCW Arbitrary, Common oder Priority.
- **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)

103  45  30  15

P₁  P₂  P₃  P₄

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).
- Modell: EREW.
- Programm: Summe
  
  for all \( P_i \) where \( 1 \leq i \leq n/2 \) do in parallel
  
  \[ R_{2 \cdot i-1} \rightarrow P_i(x) \]
  
  for \( j = 1 \) to \( k \) do
  
  if \( (i - 1) \equiv 0 \pmod{2^{j-1}} \) then
  
  \[ R_{2 \cdot i-1+2^{j-1}} \rightarrow P_i(y) \]

  \[ x := x + y \]
  
  \[ P_i(x) \rightarrow R_{2 \cdot i-1} \]

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

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

\[ \text{Addition of Matrices} \]
Multiplication of Matrices

- 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.

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

PRAM Introduction

Efficiency

Selection

Merging

1:26 Matrices

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: $O(n^2)$. 
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: $O(n^2)$.
Compute the Prefixsum

Problem:

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

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

Programm: Summe

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

- 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 $m = \max_{j=1}^{n} x_j$ 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 an EREW PRAM.
- Question: could it be done faster? (i.e. on an 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)
### Compute the Maximum (Idea)

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

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

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

```plaintext
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.
<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</td>
<td>O(n/\log n)</td>
<td>O(t)</td>
<td>O(n)</td>
<td>O(1/n)</td>
<td>CRCW</td>
</tr>
<tr>
<td>Sum</td>
<td>O(n/\log n)</td>
<td>O(log n)</td>
<td>O(n)</td>
<td>1</td>
<td>CREW</td>
</tr>
<tr>
<td>Ranking</td>
<td>O(n/\log n)</td>
<td>O(log n)</td>
<td>O(n)</td>
<td>1</td>
<td>EREW</td>
</tr>
<tr>
<td>Prefixsum</td>
<td>O(n/\log n)</td>
<td>O(log n)</td>
<td>O(n^2)</td>
<td>1</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.sum</td>
<td>O(n^2/\log n)</td>
<td>O(log n)</td>
<td>O(n^2.276)</td>
<td>O(n^{-0.734})</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>O(n^2/ \log n)</td>
<td>O(n \log n)</td>
<td>O(n^2.276)</td>
<td>O(n^{-0.734})</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>O(n^3/\log n)</td>
<td>O(log n)</td>
<td>O(n^2.276)</td>
<td>O(n^{-0.734})</td>
<td>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>
</tbody>
</table>
Task: Compute the $k$-th ($k$-smallest) element in a unsorted sequence $S = \{s_1, \ldots, s_n\}$.

Lower bound: $n - 1$ comparisons

Start with a nice sequential algorithm

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 $H_i$
- $m := \text{Select}(\lceil |M|/2 \rceil, M)$
- $S_1 := \{s \in S \mid s < m\}$
- $S_2 := \{s \in S \mid s = m\}$
- $S_3 := \{s \in S \mid s > m\}$
- if $|S_1| \geq k$ then return $\text{Select}(k, S_1)$
- if $|S_1| + |S_2| \geq k$ then return $m$
- return $\text{Select}(k - |S_1| - |S_2|, S_3)$
### Example for the k-th Element (Slow Motion)

**Input/Data:**

<table>
<thead>
<tr>
<th>66</th>
<th>92</th>
<th>39</th>
<th>12</th>
<th>29</th>
<th>74</th>
<th>13</th>
<th>68</th>
<th>21</th>
<th>11</th>
<th>11</th>
<th>11</th>
<th>3</th>
<th>52</th>
<th>9</th>
<th>59</th>
<th>16</th>
<th>78</th>
<th>93</th>
<th>91</th>
<th>46</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>11</td>
<td>8</td>
<td>70</td>
<td>61</td>
<td>65</td>
<td>79</td>
<td>47</td>
<td>42</td>
<td>71</td>
<td>17</td>
<td>24</td>
<td>82</td>
<td>67</td>
<td>43</td>
<td>91</td>
<td>79</td>
<td>88</td>
<td>89</td>
<td>64</td>
<td>71</td>
</tr>
<tr>
<td>52</td>
<td>19</td>
<td>26</td>
<td>49</td>
<td>10</td>
<td>59</td>
<td>29</td>
<td>87</td>
<td>8</td>
<td>20</td>
<td>43</td>
<td>89</td>
<td>60</td>
<td>8</td>
<td>7</td>
<td>51</td>
<td>60</td>
<td>36</td>
<td>61</td>
<td>93</td>
<td>9</td>
</tr>
<tr>
<td>74</td>
<td>77</td>
<td>44</td>
<td>29</td>
<td>19</td>
<td>33</td>
<td>67</td>
<td>70</td>
<td>57</td>
<td>1</td>
<td>71</td>
<td>83</td>
<td>34</td>
<td>92</td>
<td>14</td>
<td>81</td>
<td>95</td>
<td>31</td>
<td>46</td>
<td>4</td>
<td>85</td>
</tr>
<tr>
<td>66</td>
<td>53</td>
<td>24</td>
<td>29</td>
<td>50</td>
<td>13</td>
<td>60</td>
<td>48</td>
<td>82</td>
<td>50</td>
<td>73</td>
<td>50</td>
<td>19</td>
<td>52</td>
<td>18</td>
<td>95</td>
<td>74</td>
<td>18</td>
<td>94</td>
<td>75</td>
<td>76</td>
</tr>
</tbody>
</table>

**M:**

| 66 | 53 | 26 | 29 | 29 | 59 | 60 | 68 | 42 | 20 | 43 | 50 | 34 | 52 | 14 | 81 | 74 | 36 | 89 | 75 | 71 |

**sorted M:**

| 14 | 20 | 26 | 29 | 29 | 34 | 36 | 42 | 43 | 50 | **52** | 53 | 59 | 60 | 66 | 68 | 71 | 74 | 75 | 81 | 89 |
Example for the k-th Element

Input/Data:

```
42 52 77 37 24 58 38 3 44 47 96 62 60 61 3 84 52 46 88 29 84
13 94 56 61 89 44 60 76 58 63 90 70 72 39 45 94 88 22 6 60 28
15 34 45 12 26 83 79 28 66 16 73 48 28 76 84 27 30 91 64 59 57
84 65 32 55 83 87 77 16 16 96 13 54 9 62 17 78 46 48 32 28 78
95 2 82 24 58 66 8 78 25 47 57 22 47 49 66 47 65 33 27 59 18
```

M:

```
42 52 56 37 58 66 60 28 44 47 73 54 47 61 45 78 52 46 32 59 57
```

sorted M:

```
28 32 37 42 44 45 46 47 47 52 52 54 56 57 58 59 60 61 66 73 78
```
Example for the $k$-th Element (Worst Case)

Input/Data:

```
63 93 88 87 77 92 80 91 64 63 64 58 70 62 76 87 56 66 82 66 93
90 68 91 94 89 93 51 66 82 88 61 65 63 89 84 55 71 88 60 90 82
30 32 42 0 39 10 40 6 5 32 22 71 72 86 85 77 86 73 83 87 55
10 5 36 16 13 44 23 13 23 4 18 71 58 73 65 76 50 64 86 58 62
28 24 1 21 36 15 23 21 10 23 35 76 80 83 90 51 55 53 66 61 64
```

$M$:

```
30 32 42 21 39 44 40 21 23 32 35 71 70 83 84 76 56 66 82 66 64
```

sorted $M$:

```
21 21 23 30 32 32 35 39 40 42 44 56 64 66 66 70 71 76 82 83 84
```
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 $H_i$

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

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

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

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

if $|S_1| \geq k$ then return $Select(k, S_1)$
if $|S_1| + |S_2| \geq k$ then return $m$
return $Select(k - |S_1| - |S_2|, S_3)$
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:**

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

**P 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
```

**M sorted:**

```
24 25 28 29 31 37 37 39 42 44 45 45 45 47 48 49 49 50 51 52 52 53 53 55 57 58 58 59 59 61 64 69 71 73 77
```
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 \ \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:  
   \textbf{for all} \ P_i \ \text{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) 
   
      P_i(m_1) \rightarrow R_i. 
   
   \text{Assume in the following that} \ M \ \text{is the sequence of these values.}

4:  
   \quad 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: $O(1)$
   if $|S| \leq k_1$ then $P_1$ returns $Select(k, S)$.

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

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

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

Programm: ParSelect(k,S) Steps 5

5.1a: $O(\log_2(n^{1-x}))$
- Distribute $m$ via broadcast to all $P_i$.

5.1b: $O(|S_i|)$ thus we have $O(n^x)$
  for all $P_i$ where $1 \leq i \leq \lceil n^{1-x} \rceil$ do in parallel
  - $L_i := \{ s \in S_i \mid s < m \}$
  - $E_i := \{ s \in S_i \mid s = m \}$
  - $G_i := \{ s \in S_i \mid s > m \}$

5.2: $O(\log_2(n^{1-x}))$
- Compute with Parallel Prefix:
  - $l_i := \sum_{j=1}^{i} |L_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.
  - $e_i := \sum_{j=1}^{i} |E_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.
  - $g_i := \sum_{j=1}^{i} |G_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.
- Let: $l_0 = e_0 = g_0 = 0$
Parallel k-Select (Running Time)

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

5.3: $O(n^x)$

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

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

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

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

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

Adding all up we get:

1. \( T_{\text{ParSelect}}(n) = c_1 \log n + c_2 \cdot n^x + T_{\text{ParSelect}}(n^{1-x}) + T_{\text{ParSelect}}(3/4 \cdot n) \).

2. \( T_{\text{ParSelect}}(n) = O(n^x) \) with \( P_{\text{ParSelect}}(n) = O(n^{1-x}) \).

3. \( \text{Eff}_{\text{ParSelect}}(n) = \frac{O(n)}{O(n^x) \cdot O(n^{1-x})} = O(1) \)
Sequential Merging

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

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

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

- Algorithm does not care about special cases.

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

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

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

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