Theory of Parallel and Distributed Systems
(WS2016/17)

Kapitel 1
First Algorithms for PRAM

Walter Unger

Lehrstuhl für Informatik 1

Inhalt 1

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
   - Definition
   - Overview

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

1. There are limits to the computing power of a single computer
2. Computers become cheaper
3. Specialized computers are expensive
4. There are tasks with large data
5. Many problems are very complex
   1. Weather and other simulations
   2. Crash tests
   3. Military applications
   4. Large data: (SETI, ...)
   5. More similar problems
6. Thus there is the need for computers with more than one CPU
7. Or a quantum computer?
Motivation

1. There are limits to the computing power of a single computer
2. Computers become cheaper
3. Specialized computers are expensive
4. There are tasks with large data
5. Many problems are very complex
   1. Weather and other simulations
   2. Crash tests
   3. Military applications
   4. Large data: (SETI, ...)
   5. More similar problems
6. Thus there is the need for computers with more than one CPU
7. Or a quantum computer?
Motivation

There are limits to the computing power of a single Computer

Computers become cheaper

Specialized computers are expensive

There are tasks with large data

Manny problems are very complex

- Weather and other Simulations
- Crash tests
- Military applications
- Large data: (SETI, ...)
- More similar problems

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

or a quantum computer?
Motivation

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

6. Thus there is the need for computers with more than one CPU
7. Or a quantum computer?
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?
Motivation

1. There are limits to the computing power of a single computer.
2. Computers become cheaper.
3. Specialized computers are expensive.
4. There are tasks with large data.
5. Many problems are very complex.

   1. Weather and other Simulations
   2. Crash tests
   3. Military applications
   4. Large data: (SETI, ...)
   5. More similar problems

6. Thus there is the need for computers with more than one CPU.
7. Or a quantum computer?
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?
Motivation

1. There are limits to the computing power of a single Computer
2. Computers become cheaper
3. Specialized computers are expensive
4. There are tasks with large data
5. Many problems are very complex
   1. Weather and other Simulations
   2. Crash tests
   3. Military applications
   4. Large data: (SETI, ...)
   5. More similar problems
6. Thus there is the need for computers with more than one CPU
7. or a quantum computer?
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?
Motivation

1. There are limits to the computing power of a single computer
2. Computers become cheaper
3. Specialized computers are expensive
4. There are tasks with large data
5. Many problems are very complex
   1. Weather and other Simulations
   2. Crash tests
   3. Military applications
   4. Large data: (SETI, ...)
   5. More similar problems
6. Thus there is the need for computers with more than one CPU
7. or a quantum computer?
Motivation

1. There are limits to the computing power of a single Computer
2. Computers become cheaper
3. Specialized computers are expensive
4. There are tasks with large data
5. Many problems are very complex
   1. Weather and other Simulations
   2. Crash tests
   3. Military applications
   4. Large data: (SETI, ...)
   5. More similar problems
6. Thus there is the need for computers with more than one CPU
7. or a quantum computer?
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).
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).
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).
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).
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).
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).
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).
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).
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 synchronization.
- Processors may run simple programs.
- Advantage: really fast (for special applications).
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 Arrays

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

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

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

- Vector of processors.
- 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.
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.
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.
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.
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 summ

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

- Vector of processors.
- 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.
Example: Transputer

- Advantage: very flexible, any fixed network of degree 4 possible.
- Disadvantage: long wires may be necessary, only a fixed network possible.
Example: Transputer

- Advantage: very flexible, any fixed network of degree 4 possible.
- Disadvantage: long wires may be necessary, only a fixed network possible.
Example: Transputer

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

CPU → Bus → Memory

Link

Switch
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.
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.
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.
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.
- 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.
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.
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.
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.
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.
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.
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.
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.
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.
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.
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 II (Goodput)

- Look for good networks.
- Trees, Grids, Pyramids, ...
- $HQ(n)$, $CCC(n)$, $BF(n)$, $SE(n)$, $DB(n)$, ...
- Pancake Network and Burned Pancake Network.

**Problem:** Physical placement of the processors.

- Problem: Length of wires.
- Problem: Has the network a nice structure.
- If the network becomes too large, we may use efficiency.

**Solution:** choose a mixed network structure.
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.
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.
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 II (Goodput)

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

1. CPU and memory are one logical unit:

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

```
               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.
1. CPU and memory are one logical unit:

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

```
Network
```

2. CPUs and memory are connected by a network:

```
CPU  CPU  CPU  CPU  CPU
```

```
RAM  RAM  RAM  RAM  RAM
```

```
Network
```

The difference is more on the practical side.
Parallel Computer V (Network)

1. **CPU and memory are one logical unit:**

   ![Diagram](image)

   The difference is more on the practical side.

2. **CPUs and memory are connected by a network:**

   ![Diagram](image)
1. CPU and memory are one logical unit:

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

```
Network
```

2. CPUs and memory are connected by a network:

```
CPU  CPU  CPU  CPU  CPU
```

```
Network
```

```
RAM  RAM  RAM  RAM  RAM
```

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

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

- 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

- 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

- 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

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

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

- 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

- 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**

- 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

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

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

- For the access to the register we have the following variations:
  - EREW: Exclusive Read Exclusive Write
  - CREW: Concurrent Read Exclusive Write
  - CRCW: Concurrent Read Concurrent Write
  - ERCW: Exclusive Read Concurrent Write

- Write conflicts may be solved using the following rules:
  - Arbitrary: any processor gets access to the register.
  - Common: all processors writing to the same register have to write the same value.
  - Priority: the processor with the smallest id gets access to the register.
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.
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.
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.
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.
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:
     $$x := y \ast 5.$$ 

- 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.
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.
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:
     \[
     x := y \ast 5.
     \]

- 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.
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:
     
     \[ x := y \ast 5. \]

- 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.
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.
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.
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.
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)
Computation of an “Or” (Idea)

\[ P_1 \lor P_2 \lor P_3 \lor P_4 \lor P_5 \lor P_6 \lor P_7 \lor P_8 \]

\[ 0 \lor 1 \lor 0 \lor 0 \lor 1 \lor 0 \lor 0 \lor 1 \rightarrow 0 \]
Computation of an “Or” (Idea)

\[
\begin{align*}
P_1 & \rightarrow 0 \\
P_2 & \rightarrow 1 \\
P_3 & \rightarrow 0 \\
P_4 & \rightarrow 0 \\
P_5 & \rightarrow 1 \\
P_6 & \rightarrow 0 \\
P_7 & \rightarrow 0 \\
P_8 & \rightarrow 1 \\
\end{align*}
\]
Computation of an "Or" (Idea)

\[
\begin{align*}
x_0 &= 0 & x_1 &= 1 & x_2 &= 0 & x_3 &= 0 & x_4 &= 1 & x_5 &= 0 & x_6 &= 0 & x_7 &= 1
\end{align*}
\]

\[
0 \lor 1 \lor 0 \lor 0 \lor 1 \lor 0 \lor 0 \lor 1 \rightarrow 0
\]
Computation of an “Or” (Idea)

\[
\begin{align*}
\text{x = 0} & \quad \text{x = 1} & \quad \text{x = 0} & \quad \text{x = 0} & \quad \text{x = 1} & \quad \text{x = 0} & \quad \text{x = 0} & \quad \text{x = 1} \\
0 & \quad 1 & \quad 0 & \quad 0 & \quad 1 & \quad 0 & \quad 0 & \quad 1 \\
\end{align*}
\]
Computation of an “Or” (Idea)

\[
\begin{array}{cccccc}
0 & \lor & 1 & \lor & 0 & \lor & 1 & \lor & 0 & \lor & 0 & \lor & 1 & \rightarrow & 1 \\
\end{array}
\]
Computing an “Or”

- **Task:** Compute \( x = \bigvee_{i=1}^{n} x_i \).
- **Input:** \( x_i \) is in register \( R_i \) \((1 \leq i \leq n)\).
- **Output computed in** \( R_{n+1} \).
- **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 \( x = \text{true} \) 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”

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

- 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”

- **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.
- **Program**: Or
  
  for all $P_i$ where $1 \leq i \leq n$ do in parallel
  
  $R_i \to P_i(x)$

  if $x = \text{true}$ then $P_i(x) \to R_{n+1}$

- **Running time**: $O(1)$ (exact 2 steps).
- **Number of processors**: $n$.
- **Memory**: $n + 1$.
- **Possible models**: ERCW (Arbitrary, Common oder Priority).
Computing an “Or”

- 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"

- **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 $x = \text{true}$ 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”

- 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 \( x = true \) 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"

- 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
    \[ R_i \rightarrow P_i(x) \]
    \[ \text{if } x = \text{true then } P_i(x) \rightarrow R_{n+1} \]
- Running time: \( O(1) \) (exact 2 steps).
- Number of processors: \( n \).
- Memory: \( n + 1 \).
- Possible models: ERCW (Arbitrary, Common oder Priority).
Computing an “Or”

- 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
  ```plaintext
  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”

- 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 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 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 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)
Computing the Sum (Idea)
Computing the Sum (Idea)
Computing the Sum (Idea)

\[ M_9 \quad M_{10} \quad M_{11} \quad M_{12} \]

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

\[ M_1 \quad M_2 \quad M_3 \quad M_4 \quad M_5 \quad M_6 \quad M_7 \quad M_8 \]
Computing the Sum (Idea)

\[
\begin{array}{cccc}
\text{0} & \text{0} & \text{0} & \text{0} \\
\hline
P_1 & P_2 & P_3 & P_4 \\
\hline
12 & 6 & 34 & 5 & 7 & 23 & 4 & 11
\end{array}
\]
Computing the Sum (Idea)

```
0 0 0 0
```

```
P1   P2   P3   P4
12   6   34   5
```

```
P1   P2   P3   P4
    7   23   4
```

```
P1   P2   P3   P4
    11
```
Computing the Sum (Idea)
Computing the Sum (Idea)
Computing the Sum (Idea)

\[\begin{array}{cccc}
18 & 39 & 30 & 15 \\
\end{array}\]
Computing the Sum (Idea)

18 → P₁
39 → P₂
30 → P₃
15 → P₄

18
39
30
15

12
6
34
5
7
23
4
11
Computing the Sum (Idea)
Computing the Sum (Idea)

```
<table>
<thead>
<tr>
<th>57</th>
<th>45</th>
<th>30</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
</tr>
</tbody>
</table>
```

```
| 12 | 6 | 34 | 5 | 7 | 23 | 4 | 11 |
```
Computing the Sum (Idea)
Computing the Sum (Idea)

- 57
- 45
- 30
- 15

P1 → P2 → P3 → P4

- 12
- 6
- 34
- 5
- 7
- 23
- 4
- 11
Computing the Sum (Idea)

- $P_1$ receives the sum of 57 and 45.
- $P_2$ receives the sum of 30 and 15.
- The results are then merged into a single sum: 12, 6, 34, 5, 7, 23, 4, 11.
Computing the Sum (Idea)

- $P_1$: 103
- $P_2$: 45
- $P_3$: 30
- $P_4$: 15

$P_1, P_2, P_3, P_4$ are processes computing the sum.

- 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: $n$.

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

The sum is computed across processors $P_i$ for $1 \leq i \leq n/2$ in parallel. Each processor $P_i$ receives inputs $x_i$ and $x_j$ for $j \equiv 0 \pmod{2^{j-1}}$, and outputs the sum to $R_{2 \cdot i - 1}$. The running time is $O(k) = O(\log n)$.
Computing the sum (EREW)

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

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

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

Running time: \( O(k) = O(\log n) \) (precise \( 3 \cdot k + 1 \) steps).

Number of processors: \( n/2 \).

Size of memory: \( n \).
Computing the sum (EREW)

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

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

**Programm: Summe**

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

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

Assume w.l.o.g $n = 2^k$ for $k \in \mathbb{N}$. 
Computing the sum (EREW)

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

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

Programm: Summe

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

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

\[
\text{for } j = 1 \text{ to } k \text{ do}
\]

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

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

\[
x := x + y
\]

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

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

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

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

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

Computing the sum (EREW)

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

Programm: Summe

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

\[
\begin{align*}
R_{2 \cdot i - 1} &\rightarrow P_i(x) \\
\text{for } j = 1 \text{ to } k &\text{ do} \\
\text{if } (i - 1) \equiv 0 \pmod{2^{j-1}} &\text{ then} \\
R_{2 \cdot i - 1 + 2^{j-1}} &\rightarrow P_i(y) \\
x &:= x + y \\
P_i(x) &\rightarrow R_{2 \cdot i - 1}
\end{align*}
\]

- 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 $k \in \mathbb{N}$. 
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: $O(n^2)$.
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 $R_{3\cdot n^2}$.
- Program: 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 $R_{3\cdot n^2}$.
- Program: 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: $O(n^2)$.
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: $O(n^2)$.
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 $k \in \mathbb{N}$. 

- PRAM Introduction
- Motivation and History
- Efficiency
- Selection
- Merging
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

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 $R_{2 \cdot n^2}$ 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**: $O(n^2)$. 
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

  \begin{verbatim}
  for all $P_i$ where $1 \leq i \leq n^2$ do in parallel
  $R_i \rightarrow P_i(x)$
  $R_{i+n^2} \rightarrow P_i(y)$
  $x \gets x + y$
  $P_i(x) \rightarrow R_{i+2\cdot n^2}$
  \end{verbatim}

- 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

- 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 $k \in \mathbb{N}$. 
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.
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.
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 $R_2^{n^2}$ contains $B$ (one row after the other).
- Result in $R_1^{+2\cdot n^2}$ bis $R_3^{n^2}$
  - 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.
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 $R_{2.n^2}$ contains $B$ (one row after the other).
- Result in $R_{1+2.n^2}$ bis $R_{3.n^2}$
- Register $A_{i,j} = R_{(i-1)n+j} \ (1 \leq i, j \leq n)$.
- Register $B_{i,j} = R_{(i-1)n+j+n^2} \ (1 \leq i, j \leq n)$.
- Register $C_{i,j} = R_{(i-1)n+j+2.n^2} \ (1 \leq i, j \leq n)$.
- processor $P_{i,j} = P_{(i-1)n+j} \ (1 \leq i, j \leq n)$.
- Use the above notation to simplify the algorithm.
- Each processor has to do some hidden local computation to implement the above expressions.
Motivation and History
PRAM Introduction
Efficiency
Selection
Merging
1:25 Matrices  6/10
Walter Unger 8.11.2016 21:46  WS2016/17

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.
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 $R_{2.n^2}$ contains $B$ (one row after the other).
- Result in $R_{1+2.n^2}$ bis $R_{3.n^2}$
- Register $A_{i,j} = R_{(i-1).n+j}$ ($1 \leq i, j \leq n$).
- Register $B_{i,j} = R_{(i-1).n+j+n^2}$ ($1 \leq i, j \leq n$).
- Register $C_{i,j} = R_{(i-1).n+j+2.n^2}$ ($1 \leq i, j \leq n$).
- processor $P_{i,j} = P_{(i-1).n+j}$ ($1 \leq i, j \leq n$).
- 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 $k \in \mathbb{N}$. 
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 $R_{2,n^2}$ contains $B$ (one row after the other).
- Result in $R_{1+2,n^2}$ bis $R_{3,n^2}$
- 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.
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 $k \in \mathbb{N}$. 

\[
\begin{align*}
A &\times B = C \\
A_{i,j} &\times B_{i,j} = C_{i,j}
\end{align*}
\]
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.
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 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
   1. $A_{i,l} \rightarrow P_{i,j}(a)$
   2. $B_{l,j} \rightarrow P_{i,j}(b)$
   3. $h = h + a \cdot b$
   4. $P_{i,j}(h) \rightarrow C_{i,j}$

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

\[
\begin{align*}
A_{i,j} &= R((i-1)\cdot n+j) \\
B_{i,j} &= R((i-1)\cdot n+j+n^2) \\
C_{i,j} &= R((i-1)\cdot n+j+2\cdot n^2) \\
P_{i,j} &= P((i-1)\cdot n+j) \\
\end{align*}
\]
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: $O(n^2)$. 

\[
\begin{align*}
A_{i,j} &= R(i-1) \cdot n + j \\
B_{i,j} &= R(i-1) \cdot n + j + n^2 \\
C_{i,j} &= R(i-1) \cdot n + j + 2 \cdot n^2 \\
P_{i,j} &= P(i-1) \cdot n + j
\end{align*}
\]
Multiplication 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 CREW PRAM.

Programm: MatrProd 1

for all $P_{i,j}$ where $1 \leq i, j \leq n$ do in parallel

$\quad h = 0$

$\quad$ for $l = 1$ to $n$ do

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

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

$\quad \quad h = h + a \cdot b$

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

- 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 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 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

$a = A_{i,l}$

$b = B_{l,j}$

$h = h + a \cdot b$

$h \rightarrow C_{i,j}$

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
    \begin{align*}
    h &= 0 \\
    \text{for } l = 1 \text{ to } n \text{ do}
    \end{align*}
    \begin{align*}
    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}
    \end{align*}

- 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
  \[
  h = 0 \\
  \text{for } l = 1 \text{ to } n \text{ do} \\
  \quad A_{i,l} \rightarrow P_{i,j}(a) \\
  \quad B_{l,j} \rightarrow P_{i,j}(b) \\
  \quad h = h + a \cdot b \\
  \quad P_{i,j}(h) \rightarrow C_{i,j}
  \]
- 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.
- Program: MatrProd 2
  
  for all $P_{i,j}$ where $1 \leq i, j \leq n$ do in parallel
  
  $h = 0$
  
  for $l = 1$ to $n$ do
  
  $A_{i,l} \rightarrow P_{i,j}(a)$
  
  $B_{l,j} \rightarrow P_{i,j}(b)$
  
  $h = h + a \cdot b$
  
  $P_{i,j}(h) \rightarrow C_{i,j}$
  
- Running time: $O(n)$.
- Number of processors: $O(n^2)$.
- Size of memory: $O(n^2)$.
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: $O(n^2)$.
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: $O(n^2)$. 

\[
\begin{align*}
A_{i,j} &= R((i-1) \cdot n + j) \\
B_{i,j} &= R((i-1) \cdot n + j + n^2) \\
C_{i,j} &= R((i-1) \cdot n + j + 2 \cdot n^2) \\
P_{i,j} &= P((i-1) \cdot n + j)
\end{align*}
\]
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: $O(n^2)$. 
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 $1 \leq i \leq n$. 
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 \( 1 \leq i \leq n \).
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)

\[ P_1 \quad P_2 \quad P_3 \quad P_4 \quad P_5 \quad P_6 \quad P_7 \quad P_8 \]

\[ M_1 \quad M_2 \quad M_3 \quad M_4 \quad M_5 \quad M_6 \quad M_7 \quad M_8 \]
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)

\[ P_1 \rightarrow x_1 \]
\[ P_2 \rightarrow x_2 \]
\[ P_3 \rightarrow x_3 \]
\[ P_4 \rightarrow x_4 \]
\[ P_5 \rightarrow x_5 \]
\[ P_6 \rightarrow x_6 \]
\[ P_7 \rightarrow x_7 \]
\[ P_8 \rightarrow x_8 \]
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)

\[
\begin{align*}
&x_{1.1} & x_{1.2} & x_{2.3} & x_{3.4} & x_{4.5} & x_{5.6} & x_{6.7} & x_{7.8} \\
&x_{1.1} & x_{1.2} & x_{2.3} & x_{3.4} & x_{4.5} & x_{5.6} & x_{6.7} & x_{7.8}
\end{align*}
\]
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)

\[
\begin{align*}
X_{1..1} & \rightarrow X_{1..2} & X_{1..3} & \rightarrow X_{1..4} & X_{2..5} & \rightarrow X_{3..6} & X_{4..7} & \rightarrow X_{5..8} \\
X_{1..1} & \rightarrow X_{1..2} & X_{2..3} & \rightarrow X_{3..4} & X_{4..5} & \rightarrow X_{5..6} & X_{6..7} & \rightarrow X_{7..8}
\end{align*}
\]
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)
Computing Prefixsum (Idea)

Done!
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
  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: \( n \).
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
  
  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:** $n$. 

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: $n$. 
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: \( n \).
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**: \( n \).
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:** \( n \).
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
  
  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: $n$. 
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

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

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

PRAM Introduction

Efficiency

Selection

Merging

Walter Unger 8.11.2016 21:46  WS2016/17

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: \( n \).
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 $m = \max_{j=1}^{n} x_j$ 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

- 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

- 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

- 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

- Task: Compute $m = \max_{j=1}^{\frac{n}{2}} x_j$ 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

- 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

- Task: Compute \( m = \max_{i=1}^{j} x_j \) 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

- 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)

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 22|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 33|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 41|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 26|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 59|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 57|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 52|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 61|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 27|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 49|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 67|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 23|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 56|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 14|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 12|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 34|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

- 34 12 14 56 23 67 49 27 61 52 57 59 26 41 33 22
### Compute the Maximum (Idea)

|   | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 22|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 33|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 41|   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 26|   | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 59|   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 57|   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 52|   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 61|   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 27|   | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 49|   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 67|   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 23|   | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 56|   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 14|   | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 12|   | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 34|   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 34| 12| 14| 56| 23| 67| 49| 27| 61| 52| 57| 59| 26| 41| 33| 22|   |   |   |   |

Walter Unger 8.11.2016 21:46  WS2016/17
## Compute the Maximum (Idea)

\[
\begin{array}{cccccccccccccccccccccc}
22 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
33 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
41 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
26 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
59 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
57 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
52 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
61 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
27 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
49 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
67 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
23 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
56 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
14 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
12 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
34 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
34 & 12 & 14 & 56 & 23 & 67 & 49 & 27 & 61 & 52 & 57 & 59 & 26 & 41 & 33 & 22
\end{array}
\]
## 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></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>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></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>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></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></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>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Walter Unger 8.11.2016 21:46  WS2016/17
**Compute the Maximum (Idea)**

| 22 | 0   | 1   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0   |
| 33 | 0   | 1   | 1   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 1   | 0   | 1   | 0   |
| 41 | 1   | 1   | 1   | 0   | 1   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 1   | 1   | 1   |
| 26 | 0   | 1   | 1   | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0   | 0   |
| 59 | 1   | 1   | 1   | 1   | 1   | 0   | 1   | 1   | 0   | 1   | 1   | 1   | 1   | 1   | 1   |
| 57 | 1   | 1   | 1   | 1   | 1   | 0   | 1   | 1   | 0   | 1   | 1   | 0   | 1   | 1   | 1   |
| 52 | 1   | 1   | 1   | 0   | 1   | 0   | 1   | 1   | 0   | 1   | 0   | 0   | 1   | 1   | 1   |
| 61 | 1   | 1   | 1   | 1   | 1   | 0   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1   |
| 27 | 0   | 1   | 1   | 0   | 1   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 1   | 0   | 0   |
| 49 | 1   | 1   | 1   | 0   | 1   | 0   | 1   | 1   | 0   | 0   | 0   | 0   | 1   | 1   | 1   |
| 67 | 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   |
| 56 | 1   | 1   | 1   | 1   | 1   | 0   | 1   | 1   | 0   | 1   | 0   | 0   | 1   | 1   | 1   |
| 14 | 0   | 1   | 1   | 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   |
| 34 | 1   | 1   | 1   | 0   | 1   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 1   | 0   | 1   |

```
| 34 | 12 | 14 | 56 | 23 | 67 | 49 | 27 | 61 | 52 | 57 | 59 | 26 | 41 | 33 | 22 |
```
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
Compute the Maximum (Idea)

| 22 | 0 |
| 33 | 0 |
| 41 | 1 |
| 26 | 0 |
| 59 | 1 |
| 67 | 1 |
| 52 | 1 |
| 61 | 1 |
| 27 | 0 |
| 49 | 1 |
| 67 | 1 |
| 23 | 0 |
| 56 | 1 |
| 14 | 0 |
| 12 | 0 |
| 34 | 1 |
| 34 | 12 | 14 | 56 | 23 | 67 | 49 | 27 | 61 | 52 | 67 | 59 | 26 | 41 | 33 | 22 |
Compute the Maximum (Idea)

<table>
<thead>
<tr>
<th></th>
<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>0</th>
<th>0</th>
<th>0</th>
<th>1</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>
<td>1</td>
<td></td>
<td></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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></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>
<td>1</td>
<td>1</td>
<td></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>
<td>1</td>
<td></td>
<td></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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></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>
<td>1</td>
<td></td>
<td></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>
<td>1</td>
<td>1</td>
<td></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>
<td>0</td>
<td>1</td>
<td></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>
<td>1</td>
<td></td>
<td></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>
<td>0</td>
<td></td>
<td></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>
<td>0</td>
<td></td>
<td></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>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

34 | 12 | 14 | 56 | 23 | 67 | 49 | 27 | 61 | 52 | 67 | 59 | 26 | 41 | 33 | 22 |
Compute the Maximum (Idea)

|   | 22 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
|   | 33 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
|   | 41 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|   | 26 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
|   | 59 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
|   | 67 | 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 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
|   | 61 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
|   | 27 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
|   | 49 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
|   | 67 | 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 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
|   | 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 |
|   | 34 | 12 | 14 | 56 | 23 | 67 | 49 | 27 | 61 | 52 | 67 | 59 | 26 | 41 | 33 | 22 |
## 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></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>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></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>1</td>
</tr>
<tr>
<td></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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></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>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</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>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</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>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></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>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<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>67</td>
<td>59</td>
<td>26</td>
<td>41</td>
<td>33</td>
<td>22</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**

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

- 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

- 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

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

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

**Programm: Maximum**

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

- 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

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

- 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
  
  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

- **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

- 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

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

- 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: $O(n)$. 

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: $O(n)$. 
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: $O(n)$**.
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: $O(n)$. 
Computing the Maximum

Programm: Maximum

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

- 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

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

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

Number of processors: $O(n)$.

Memory: $O(n)$.

Model: CREW.
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
  
  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.
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

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.
Identify the Roots of a Forest

- Nodes are identified by numbers from 1 till \( n \)
- Input: Father of node \( i \) is written in register \( R_i \).
- For the roots \( i \) we have: in register \( R_i \) is written \( i \).

Programm: Ranking

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

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

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

Memory: \( O(n) \).

Model: CREW.
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.
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.
Identify the Roots of a Forest

- Nodes are identified by numbers from 1 till \( n \)
- Input: Father of node \( i \) is written in register \( R_i \).
- For the roots \( i \) we have: in register \( R_i \) is written \( i \).

**Programm: Ranking**

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

\[
\text{for } j = 1 \text{ to } \lceil \log n \rceil \text{ do}
\]

\[
R_i \rightarrow P_i(h)
\]

\[
R_h \rightarrow P_i(h)
\]

\[
P_i(h) \rightarrow R_i
\]

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

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

**Memory:** \( O(n) \).

**Model:** CREW.
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.
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.
### Motivation and History

**PRAM Introduction**

**Efficiency**

**Selection**

**Merging**

1:37  Situation  1/11

---

**Z**

---

**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>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>ERCW</td>
</tr>
<tr>
<td>Or</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Maximum</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>CRCW</td>
</tr>
<tr>
<td>Sum</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Ranking</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Prefixsum</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.sum</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

**Question:** May we save some processors?
# 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>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>ERCW</td>
</tr>
<tr>
<td>Or</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Maximum</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>CRCW</td>
</tr>
<tr>
<td>Sum</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Ranking</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Prefixsum</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.sum</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

**Question:** May we save some processors?
## 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>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Maximum</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>CRCW</td>
</tr>
<tr>
<td>Sum</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Ranking</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Prefixsum</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.sum</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

**Question:** May we save some processors?
### 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>EREW</td>
</tr>
<tr>
<td>Or</td>
<td>$O(n/t)$</td>
<td>$O(n)$</td>
<td>$O(\log n + t)$</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>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Ranking</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Prefixsum</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.sum</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

**Question:** May we save some processors?
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>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Ranking</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Prefixsum</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.sum</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

**Question:** May we save some processors?
### 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.sum</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

Question: May we save some processors?
## 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>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

**Question:** May we save some processors?
## 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>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

**Question:** May we save some processors?
# 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?
### 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?
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.sum</td>
<td>$O(n^2/\log n)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^2/\log n)$</td>
<td>$O(n^2)$</td>
<td>$O(n \cdot \log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3/\log n)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>CREW</td>
</tr>
<tr>
<td>Mat.prod.</td>
<td>$O(n^3/\log n)$</td>
<td>$O(n^2)$</td>
<td>$O(\log n)$</td>
<td>EREW</td>
</tr>
</tbody>
</table>

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

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

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

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 $A$. 
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 $A$. 
Cost Measurement

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

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

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

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

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


### Efficiency

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

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

1. If $|S| \leq 50$ then return $k$-th number in $S$
2. Split $S$ in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$
3. Sort each $H_i$
4. Let $M$ be the sequence of the middle elements in $H_i$
5. $m := Select(\lceil |M|/2 \rceil, M)$
6. $S_1 := \{s \in S \mid s < m\}$
7. $S_2 := \{s \in S \mid s = m\}$
8. $S_3 := \{s \in S \mid s > m\}$
9. If $|S_1| \geq k$ then return $Select(k, S_1)$
10. If $|S_1| + |S_2| \geq k$ then return $m$
11. Return $Select(k - |S_1| - |S_2|, S_3)$
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

Programm: Select($k, S$)

if $|S| \leq 50$ then return $k$-th number in $S$

Split $S$ in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$

Sort each $H_i$

Let $M$ be the sequence of the middle elements in $H_i$

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

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

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

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

if $|S_1| \geq k$ then return $Select(k, S_1)$

if $|S_1| + |S_2| \geq k$ then return $m$

return $Select(k - |S_1| - |S_2|, S_3)$
**Task:** Compute the $k$-th ($k$-smallest) element in a unsorted sequence $S = \{s_1, \ldots, s_n\}$.

**Lower bound:** $n - 1$ comparisons

**Start with a nice sequential algorithm**

**Programm:** Select($k, S$)

```plaintext
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 $Select(k - |S_1| - |S_2|, S_3)$
```
Task: Compute the $k$-th ($k$-smallest) element in a unsorted sequence $S = \{s_1, \ldots, s_n\}$.

Lower bound: $n - 1$ comparisons

Start with a nice sequential algorithm

Programm: Select($k, S$)

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

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

Programm: Select($k, S$)

1. If $|S| \leq 50$ then return $k$-th number in $S$
2. Split $S$ in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$
3. Sort each $H_i$
4. Let $M$ be the sequence of the middle elements in $H_i$
5. $m := Select(\lceil |M|/2 \rceil, M)$
6. $S_1 := \{s \in S \mid s < m\}$
7. $S_2 := \{s \in S \mid s = m\}$
8. $S_3 := \{s \in S \mid s > m\}$
9. If $|S_1| \geq k$ then return $Select(k, S_1)$
10. If $|S_1| + |S_2| \geq k$ then return $m$
11. Return $Select(k - |S_1| - |S_2|, S_3)$
Task: Compute the $k$-th ($k$-smallest) element in a unsorted sequence $S = \{s_1, \ldots, s_n\}$.

Lower bound: $n - 1$ comparisons

Start with a nice sequential algorithm

Programm: Select($k,S$)

if $|S| \leq 50$ then return $k$-th number in $S$

Split $S$ in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$

Sort each $H_i$

Let $M$ be the sequence of the middle elements in $H_i$

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

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

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

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

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

if $|S_1| + |S_2| \geq k$ then return $m$

return $\text{Select}(k - |S_1| - |S_2|, S_3)$
k-th Element

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

- Lower bound: \( n - 1 \) comparisons

- Start with a nice sequential algorithm

Programm: Select(k,S)

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

Split \( S \) in \( \lceil n/5 \rceil \) sub-sequences \( H_i \) of size \( \leq 5 \)

Sort each \( H_i \)

Let \( M \) be the sequence of the middle elements in \( H_i \)

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

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

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

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

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

- if \( |S_1| + |S_2| \geq k \) then return \( m \)

return \( \text{Select}(k - |S_1| - |S_2|, S_3) \)
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

Programm: `Select(k,S)`

1. If $|S| \leq 50$ then return the $k$-th number in $S$.
2. Split $S$ into $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$.
3. Sort each $H_i$.
4. Let $M$ be the sequence of the middle elements in $H_i$.
5. $m := Select(\lceil |M|/2 \rceil, M)$
6. $S_1 := \{s \in S \mid s < m\}$
7. $S_2 := \{s \in S \mid s = m\}$
8. $S_3 := \{s \in S \mid s > m\}$
9. If $|S_1| \geq k$ then return $Select(k, S_1)$
10. If $|S_1| + |S_2| \geq k$ then return $m$
11. Return $Select(k - |S_1| - |S_2|, S_3)$
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

Programm: Select($k, S$)

1. if $|S| \leq 50$ then return $k$-th number in $S$
2. Split $S$ in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$
3. Sort each $H_i$
4. Let $M$ be the sequence of the middle elements in $H_i$
5. $m := Select(\lceil |M|/2 \rceil, M)$
6. $S_1 := \{s \in S \mid s < m\}$
7. $S_2 := \{s \in S \mid s = m\}$
8. $S_3 := \{s \in S \mid s > m\}$
9. if $|S_1| \geq k$ then return Select($k, S_1$)
10. if $|S_1| + |S_2| \geq k$ then return $m$
11. return Select($k - |S_1| - |S_2|, S_3$)
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

Programm: Select($k, S$)

if $|S| \leq 50$ then return $k$-th number in $S$

Split $S$ in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$

Sort each $H_i$

Let $M$ be the sequence of the middle elements in $H_i$

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

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

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

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

if $|S_1| \geq k$ then return Select($k, S_1$)

if $|S_1| + |S_2| \geq k$ then return $m$

return Select($k - |S_1| - |S_2|$, $S_3$)
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

Programm: Select($k, S$)

if $|S| \leq 50$ then return $k$-th number in $S$

Split $S$ in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size $\leq 5$

Sort each $H_i$

Let $M$ be the sequence of the middle elements in $H_i$

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

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

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

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

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

if $|S_1| + |S_2| \geq k$ then return $m$

return $\text{Select}(k - |S_1| - |S_2|, S_3)$
**k-th Element**

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

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

```plaintext
if |S| ≤ 50 then return k-th number in S
Split S in $\lceil n/5 \rceil$ sub-sequences $H_i$ of size ≤ 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:

|   51  |  16  |  86  |  93  |  44  |  65  |  66  |  74  |  95  |  66  |  48  |  70  |   2  |   41  |  77  |  27  |  68  |  41  |  44  |  51  |  86  |
|  49  |  85  |  26  |  52  |  30  |  10  |  12  |   7  |   4  |  30  |  85  |  53  |   9  |   9  |  88  |  92  |  17  |  13  |  61  |  14  |  95  |
|  64  |  61  |  82  |  26  |  96  |  10  |  14  |  91  |  19  |  29  |  34  |  43  |  12  |  31  |   4  |  95  |  10  |  42  |  14  |  21  |  81  |
|  49  |  39  |  29  |  91  |  24  |  65  |  72  |  17  |  47  |  10  |  66  |  44  |  68  |   6  |  89  |  53  |  51  |  86  |  40  |  51  |  15  |
|  34  |  30  |  38  |   5  |  27  |   6  |  32  |  58  |  94  |  28  |   0  |  38  |  23  |  43  |  66  |  33  |  15  |  70  |   0  |  22  |  21  |

M:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

sorted M:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
### Example for the k-th Element (Slow Motion)

**Input/Data:**

| 64 | 85 | 86 | 93 | 96 | 65 | 72 | 91 | 95 | 66 | 85 | 70 | 68 | 43 | 89 | 95 | 68 | 86 | 61 | 51 | 95 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 51 | 61 | 82 | 91 | 44 | 65 | 66 | 74 | 94 | 30 | 66 | 53 | 23 | 41 | 88 | 92 | 51 | 70 | 44 | 51 | 86 |
| 49 | 39 | 38 | 52 | 30 | 10 | 32 | 58 | 47 | 29 | 48 | 44 | 12 | 31 | 77 | 53 | 17 | 42 | 40 | 22 | 81 |
| 49 | 30 | 29 | 26 | 27 | 10 | 14 | 17 | 19 | 28 | 34 | 43 | 9  | 9  | 66 | 33 | 15 | 41 | 14 | 21 | 21 |
| 34 | 16 | 26 | 5  | 24 | 6  | 12 | 7  | 4  | 10 | 0  | 38 | 2  | 6  | 4  | 27 | 10 | 13 | 0  | 14 | 15 |

**M:**

```

```

**sorted M:**

```

```
Example for the k-th Element (Slow Motion)

Input/Data:

<table>
<thead>
<tr>
<th>64</th>
<th>85</th>
<th>86</th>
<th>93</th>
<th>96</th>
<th>65</th>
<th>72</th>
<th>91</th>
<th>95</th>
<th>66</th>
<th>85</th>
<th>70</th>
<th>68</th>
<th>43</th>
<th>89</th>
<th>95</th>
<th>68</th>
<th>86</th>
<th>61</th>
<th>51</th>
<th>95</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>61</td>
<td>82</td>
<td>91</td>
<td>44</td>
<td>65</td>
<td>66</td>
<td>74</td>
<td>94</td>
<td>30</td>
<td>66</td>
<td>53</td>
<td>23</td>
<td>41</td>
<td>88</td>
<td>92</td>
<td>51</td>
<td>70</td>
<td>44</td>
<td>51</td>
<td>86</td>
</tr>
<tr>
<td>49</td>
<td>39</td>
<td>38</td>
<td>52</td>
<td>30</td>
<td>10</td>
<td>32</td>
<td>58</td>
<td>47</td>
<td>29</td>
<td>48</td>
<td>44</td>
<td>12</td>
<td>31</td>
<td>77</td>
<td>53</td>
<td>17</td>
<td>42</td>
<td>40</td>
<td>22</td>
<td>81</td>
</tr>
<tr>
<td>49</td>
<td>30</td>
<td>29</td>
<td>26</td>
<td>27</td>
<td>10</td>
<td>14</td>
<td>17</td>
<td>19</td>
<td>28</td>
<td>34</td>
<td>43</td>
<td>9</td>
<td>9</td>
<td>66</td>
<td>33</td>
<td>15</td>
<td>41</td>
<td>14</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>34</td>
<td>16</td>
<td>26</td>
<td>5</td>
<td>24</td>
<td>6</td>
<td>12</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>0</td>
<td>38</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>27</td>
<td>10</td>
<td>13</td>
<td>0</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

\[ M: \]

| 49 | 39 | 38 | 52 | 30 | 10 | 32 | 58 | 47 | 29 | 48 | 44 | 12 | 31 | 77 | 53 | 17 | 42 | 40 | 22 | 81 |

\[ \text{sorted } M: \]
### Example for the k-th Element (Slow Motion)

**Input/Data:**

| M: | 51 | 16 | 86 | 93 | 44 | 65 | 66 | 74 | 95 | 66 | 48 | 70 | 2 | 41 | 77 | 27 | 68 | 41 | 44 | 51 | 86 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|---|----|----|----|----|----|----|----|----|----|
|    | 49 | 85 | 26 | 52 | 30 | 10 | 12 | 7  | 4  | 30 | 85 | 53 | 9  | 9  | 88 | 92 | 17 | 13 | 61 | 14 | 95 |
|    | 64 | 61 | 82 | 26 | 96 | 10 | 14 | 91 | 19 | 29 | 34 | 43 | 12 | 31 | 4  | 95 | 10 | 42 | 14 | 21 | 81 |
|    | 49 | 39 | 29 | 91 | 24 | 65 | 72 | 17 | 47 | 10 | 66 | 44 | 68 | 6  | 89 | 53 | 51 | 86 | 40 | 51 | 15 |
|    | 34 | 30 | 38 | 5  | 27 | 6  | 32 | 58 | 94 | 28 | 0  | 38 | 23 | 43 | 66 | 33 | 15 | 70 | 0  | 22 | 21 |

- **M:**
  | 49 | 39 | 38 | 52 | 30 | 10 | 32 | 58 | 47 | 29 | 48 | 44 | 12 | 31 | 77 | 53 | 17 | 42 | 40 | 22 | 81 |

- **sorted M:**
  

---

Walter Unger 8.11.2016 21:46

WS2016/17

RWTH
### Example for the k-th Element (Slow Motion)

**Input/Data:**

```
51 16 86 93 44 65 66 74 95 66 48 70 2 41 77 27 68 41 44 51 86
49 85 26 52 30 10 12 7 4 30 85 53 9 9 88 92 17 13 61 14 95
64 61 82 26 96 10 14 91 19 29 34 43 12 31 4 95 10 42 14 21 81
49 39 29 91 24 65 72 17 47 10 66 44 68 6 89 53 51 86 40 51 15
34 30 38 5 27 6 32 58 94 28 0 38 23 43 66 33 15 70 0 22 21
```

**M:**

```
49 39 38 52 30 10 32 58 47 29 48 44 12 31 77 53 17 42 40 22 81
```

**sorted M:**

```
10 12 17 22 29 30 31 32 38 39 40 42 44 47 48 49 52 53 58 77 81
```
Example for the k-th Element (Slow Motion)

**Input/Data:**

<table>
<thead>
<tr>
<th>51</th>
<th>16</th>
<th>86</th>
<th>93</th>
<th>44</th>
<th>65</th>
<th>66</th>
<th>74</th>
<th>95</th>
<th>66</th>
<th>48</th>
<th>70</th>
<th>2</th>
<th>41</th>
<th>77</th>
<th>27</th>
<th>68</th>
<th>41</th>
<th>44</th>
<th>51</th>
<th>86</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>85</td>
<td>26</td>
<td>52</td>
<td>30</td>
<td>10</td>
<td>12</td>
<td>7</td>
<td>4</td>
<td>30</td>
<td>85</td>
<td>53</td>
<td>9</td>
<td>9</td>
<td>88</td>
<td>92</td>
<td>17</td>
<td>13</td>
<td>61</td>
<td>14</td>
<td>95</td>
</tr>
<tr>
<td>64</td>
<td>61</td>
<td>82</td>
<td>26</td>
<td>96</td>
<td>10</td>
<td>14</td>
<td>91</td>
<td>19</td>
<td>29</td>
<td>34</td>
<td>43</td>
<td>12</td>
<td>31</td>
<td>4</td>
<td>95</td>
<td>10</td>
<td>42</td>
<td>14</td>
<td>21</td>
<td>81</td>
</tr>
<tr>
<td>49</td>
<td>39</td>
<td>29</td>
<td>91</td>
<td>24</td>
<td>65</td>
<td>72</td>
<td>17</td>
<td>47</td>
<td>10</td>
<td>66</td>
<td>68</td>
<td>6</td>
<td>89</td>
<td>53</td>
<td>51</td>
<td>86</td>
<td>40</td>
<td>51</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>30</td>
<td>38</td>
<td>5</td>
<td>27</td>
<td>6</td>
<td>32</td>
<td>58</td>
<td>94</td>
<td>28</td>
<td>0</td>
<td>38</td>
<td>23</td>
<td>43</td>
<td>66</td>
<td>33</td>
<td>15</td>
<td>70</td>
<td>0</td>
<td>22</td>
<td>21</td>
</tr>
</tbody>
</table>

**M:**

| 49 | 39 | 38 | 52 | 30 | 10 | 32 | 58 | 47 | 29 | 48 | 44 | 12 | 31 | 77 | 53 | 17 | 42 | 40 | 22 | 81 |

**sorted M:**

| 10 | 12 | 17 | 22 | 29 | 30 | 31 | 32 | 38 | 39 | 40 | 42 | 44 | 47 | 48 | 49 | 52 | 53 | 58 | 77 | 81 |
Example for the k-th Element

Input/Data:

| 64 | 15 | 76 | 67 | 59 | 57 | 62 | 43 | 41 | 50 | 27 | 97 | 28 | 81 | 57 | 23 | 79 | 45 | 25 | 25 | 55 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 21 | 4  | 27 | 94 | 34 | 33 | 75 | 48 | 2  | 40 | 65 | 8  | 14 | 5  | 36 | 66 | 57 | 86 | 27 | 62 | 29 |
| 36 | 29 | 7  | 47 | 19 | 58 | 21 | 10 | 59 | 37 | 25 | 25 | 35 | 95 | 60 | 71 | 25 | 36 | 46 | 57 | 3  |
| 86 | 25 | 73 | 6  | 92 | 29 | 72 | 12 | 3  | 27 | 62 | 39 | 76 | 38 | 61 | 27 | 58 | 37 | 78 | 16 | 8  |
| 44 | 20 | 93 | 26 | 42 | 49 | 51 | 49 | 36 | 89 | 3  | 94 | 43 | 86 | 52 | 28 | 39 | 90 | 30 | 85 | 49 |

M:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

sorted M:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
Example for the k-th Element

Input/Data:

<table>
<thead>
<tr>
<th>64</th>
<th>15</th>
<th>76</th>
<th>67</th>
<th>59</th>
<th>57</th>
<th>62</th>
<th>43</th>
<th>41</th>
<th>50</th>
<th>27</th>
<th>97</th>
<th>28</th>
<th>81</th>
<th>57</th>
<th>23</th>
<th>79</th>
<th>45</th>
<th>25</th>
<th>25</th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>4</td>
<td>27</td>
<td>94</td>
<td>34</td>
<td>33</td>
<td>75</td>
<td>48</td>
<td>2</td>
<td>40</td>
<td>65</td>
<td>8</td>
<td>14</td>
<td>5</td>
<td>36</td>
<td>66</td>
<td>57</td>
<td>86</td>
<td>27</td>
<td>62</td>
<td>29</td>
</tr>
<tr>
<td>36</td>
<td>29</td>
<td>7</td>
<td>47</td>
<td>19</td>
<td>58</td>
<td>21</td>
<td>10</td>
<td>59</td>
<td>37</td>
<td>25</td>
<td>25</td>
<td>35</td>
<td>95</td>
<td>60</td>
<td>71</td>
<td>25</td>
<td>36</td>
<td>46</td>
<td>57</td>
<td>3</td>
</tr>
<tr>
<td>86</td>
<td>25</td>
<td>73</td>
<td>6</td>
<td>92</td>
<td>29</td>
<td>72</td>
<td>12</td>
<td>3</td>
<td>27</td>
<td>62</td>
<td>39</td>
<td>76</td>
<td>38</td>
<td>61</td>
<td>27</td>
<td>58</td>
<td>37</td>
<td>78</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>44</td>
<td>20</td>
<td>93</td>
<td>26</td>
<td>42</td>
<td>49</td>
<td>51</td>
<td>49</td>
<td>36</td>
<td>89</td>
<td>3</td>
<td>94</td>
<td>43</td>
<td>86</td>
<td>52</td>
<td>28</td>
<td>39</td>
<td>90</td>
<td>30</td>
<td>85</td>
<td>49</td>
</tr>
</tbody>
</table>

M:

| 44 | 20 | 73 | 47 | 42 | 49 | 62 | 43 | 36 | 40 | 27 | 39 | 35 | 81 | 57 | 28 | 57 | 45 | 30 | 57 | 29 |

sorted M:
Example for the k-th Element

Input/Data:

| 64 | 15 | 76 | 67 | 59 | 57 | 62 | 43 | 41 | 50 | 27 | 97 | 28 | 81 | 57 | 23 | 79 | 45 | 25 | 25 | 55 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 21 | 4  | 27 | 94 | 34 | 33 | 75 | 48 | 2  | 40 | 65 | 8  | 14 | 5  | 36 | 66 | 57 | 86 | 27 | 62 | 29 |
| 36 | 29 | 7  | 47 | 19 | 58 | 21 | 10 | 59 | 37 | 25 | 25 | 35 | 95 | 60 | 71 | 25 | 36 | 46 | 57 | 3  |
| 86 | 25 | 73 | 6  | 92 | 29 | 72 | 12 | 3  | 27 | 62 | 39 | 76 | 38 | 61 | 27 | 58 | 37 | 78 | 16 | 8  |
| 44 | 20 | 93 | 26 | 42 | 49 | 51 | 49 | 36 | 89 | 3  | 94 | 43 | 86 | 52 | 28 | 39 | 90 | 30 | 85 | 49 |

M:

| 44 | 20 | 73 | 47 | 42 | 49 | 62 | 43 | 36 | 40 | 27 | 39 | 35 | 81 | 57 | 28 | 57 | 45 | 30 | 57 | 29 |

sorted M:

| 20 | 27 | 28 | 29 | 30 | 35 | 36 | 39 | 40 | 42 | 43 | 44 | 45 | 47 | 49 | 57 | 57 | 57 | 62 | 73 | 81 |
### Example for the k-th Element

**Input/Data:**

```
64 15 76 67 59 57 62 43 41 50 27 97 28 81 57 23 79 45 25 25 55
21 4 27 94 34 33 75 48 2 40 65 8 14 5 36 66 57 86 27 62 29
36 29 7 47 19 58 21 10 59 37 25 25 35 95 60 71 25 36 46 57 3
86 25 73 6 92 29 72 12 3 27 62 39 76 38 61 27 58 37 78 16 8
44 20 93 26 42 49 51 49 36 89 3 94 43 86 52 28 39 90 30 85 49
```

**M:**

```
44 20 73 47 42 49 62 43 36 40 27 39 35 81 57 28 57 45 30 57 29
```

**sorted M:**

```
20 27 28 29 30 35 36 39 40 42 43 44 45 47 49 57 57 57 62 73 81
```
Example for the k-th Element (Worst Case)

Input/Data:

| 74 | 70 | 50 | 64 | 83 | 64 | 53 | 83 | 62 | 93 | 89 | 51 | 55 | 69 | 50 | 84 | 67 | 83 | 92 | 64 | 73 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 80 | 90 | 94 | 68 | 63 | 76 | 51 | 50 | 87 | 79 | 61 | 89 | 87 | 81 | 82 | 92 | 86 | 75 | 93 | 78 | 85 |
| 34 | 43 | 3  | 32 | 34 | 17 | 22 | 32 | 24 | 35 | 2  | 63 | 61 | 66 | 88 | 58 | 54 | 86 | 79 | 73 | 93 |
| 12 | 16 | 36 | 16 | 27 | 19 | 42 | 20 | 15 | 12 | 2  | 54 | 86 | 60 | 92 | 53 | 90 | 71 | 64 | 72 | 73 |
| 32 | 33 | 6  | 7  | 35 | 2  | 26 | 16 | 21 | 33 | 30 | 70 | 59 | 82 | 72 | 94 | 78 | 56 | 90 | 65 | 92 |

M:

|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

sorted M:

|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
Example for the k-th Element (Worst Case)

Input/Data:

<table>
<thead>
<tr>
<th>74</th>
<th>70</th>
<th>50</th>
<th>64</th>
<th>83</th>
<th>64</th>
<th>53</th>
<th>83</th>
<th>62</th>
<th>93</th>
<th>89</th>
<th>51</th>
<th>55</th>
<th>69</th>
<th>50</th>
<th>84</th>
<th>67</th>
<th>83</th>
<th>92</th>
<th>64</th>
<th>73</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>90</td>
<td>94</td>
<td>68</td>
<td>63</td>
<td>76</td>
<td>51</td>
<td>50</td>
<td>87</td>
<td>79</td>
<td>61</td>
<td>89</td>
<td>87</td>
<td>81</td>
<td>82</td>
<td>92</td>
<td>86</td>
<td>75</td>
<td>93</td>
<td>78</td>
<td>85</td>
</tr>
<tr>
<td>34</td>
<td>43</td>
<td>3</td>
<td>32</td>
<td>34</td>
<td>17</td>
<td>22</td>
<td>32</td>
<td>24</td>
<td>35</td>
<td>2</td>
<td>63</td>
<td>61</td>
<td>66</td>
<td>88</td>
<td>58</td>
<td>54</td>
<td>86</td>
<td>79</td>
<td>73</td>
<td>93</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>36</td>
<td>16</td>
<td>27</td>
<td>19</td>
<td>42</td>
<td>20</td>
<td>15</td>
<td>12</td>
<td>54</td>
<td>86</td>
<td>60</td>
<td>92</td>
<td>53</td>
<td>90</td>
<td>71</td>
<td>64</td>
<td>72</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>6</td>
<td>7</td>
<td>35</td>
<td>2</td>
<td>26</td>
<td>16</td>
<td>21</td>
<td>33</td>
<td>30</td>
<td>70</td>
<td>59</td>
<td>82</td>
<td>72</td>
<td>94</td>
<td>78</td>
<td>56</td>
<td>90</td>
<td>65</td>
<td>92</td>
</tr>
</tbody>
</table>

M:

| 34 | 43 | 36 | 32 | 35 | 19 | 42 | 32 | 24 | 35 | 30 | 63 | 61 | 69 | 82 | 84 | 78 | 75 | 90 | 72 | 85 |

sorted M:
### Example for the k-th Element (Worst Case)

**Input/Data:**

<p>| | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>74</td>
<td>70</td>
<td>50</td>
<td>64</td>
<td>83</td>
<td>64</td>
<td>53</td>
<td>83</td>
<td>62</td>
<td>93</td>
<td>89</td>
<td>51</td>
<td>55</td>
<td>69</td>
<td>50</td>
<td>84</td>
<td>67</td>
</tr>
<tr>
<td>80</td>
<td>90</td>
<td>94</td>
<td>68</td>
<td>63</td>
<td>76</td>
<td>51</td>
<td>50</td>
<td>87</td>
<td>79</td>
<td>61</td>
<td>89</td>
<td>87</td>
<td>81</td>
<td>82</td>
<td>92</td>
<td>86</td>
</tr>
<tr>
<td>34</td>
<td>43</td>
<td>3</td>
<td>32</td>
<td>34</td>
<td>17</td>
<td>22</td>
<td>32</td>
<td>24</td>
<td>35</td>
<td>2</td>
<td>63</td>
<td>61</td>
<td>66</td>
<td>88</td>
<td>58</td>
<td>54</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>36</td>
<td>16</td>
<td>27</td>
<td>19</td>
<td>42</td>
<td>20</td>
<td>15</td>
<td>12</td>
<td>2</td>
<td>54</td>
<td>86</td>
<td>60</td>
<td>92</td>
<td>53</td>
<td>90</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>6</td>
<td>7</td>
<td>35</td>
<td>2</td>
<td>26</td>
<td>16</td>
<td>21</td>
<td>33</td>
<td>30</td>
<td>70</td>
<td>59</td>
<td>82</td>
<td>72</td>
<td>94</td>
<td>78</td>
</tr>
</tbody>
</table>

**M:**

| 34 | 43 | 36 | 32 | 35 | 19 | 42 | 32 | 24 | 35 | 30 | 63 | 61 | 69 | 82 | 84 | 78 | 75 | 90 | 72 | 85 |

**sorted M:**

| 19 | 24 | 30 | 32 | 32 | 34 | 35 | 35 | 36 | 42 | **43** | 61 | 63 | 69 | 72 | 75 | 78 | 82 | 84 | 85 | 90 |
Example for the k-th Element (Worst Case)

Input/Data:

| 74 | 70 | 50 | 64 | 83 | 64 | 53 | 83 | 62 | 93 | 89 | 51 | 55 | 69 | 50 | 84 | 67 | 83 | 92 | 64 | 73 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 80 | 90 | 94 | 68 | 63 | 76 | 51 | 50 | 87 | 79 | 61 | 89 | 87 | 81 | 82 | 92 | 86 | 75 | 93 | 78 | 85 |
| 34 | 43 | 3  | 32 | 34 | 17 | 22 | 32 | 24 | 35 | 2  | 63 | 61 | 66 | 88 | 58 | 54 | 86 | 79 | 73 | 93 |
| 12 | 16 | 36 | 16 | 27 | 19 | 42 | 20 | 15 | 12 | 2  | 54 | 86 | 60 | 92 | 53 | 90 | 71 | 64 | 72 | 73 |
| 32 | 33 | 6  | 7  | 35 | 2  | 26 | 16 | 21 | 33 | 30 | 70 | 59 | 82 | 72 | 94 | 78 | 56 | 90 | 65 | 92 |

M:

| 34 | 43 | 36 | 32 | 35 | 19 | 42 | 32 | 24 | 35 | 30 | 63 | 61 | 69 | 82 | 84 | 78 | 75 | 90 | 72 | 85 |

sorted M:

| 19 | 24 | 30 | 32 | 32 | 34 | 35 | 35 | 36 | 42 | **43** | 61 | 63 | 69 | 72 | 75 | 78 | 82 | 84 | 85 | 90 |
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

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

\[
\text{if } |S| \leq 50 \text{ then return } k\text{-th number in } S
\]
\[
\text{Split } S \text{ in } \left\lceil n/5 \right\rceil \text{ sub-sequences } H_i \text{ of size } \leq 5
\]
\[
\text{Sort each } H_i
\]
\[
\text{Let } M \text{ be the sequence of the middle elements in } H_i
\]
\[
m := \text{Select}\left(\left\lfloor |M|/2 \right\rfloor, M\right)
\]
\[
S_1 := \{s \in S \mid s < m\}
\]
\[
S_2 := \{s \in S \mid s = m\}
\]
\[
S_3 := \{s \in S \mid s > m\}
\]
\[
\text{if } |S_1| \geq k \text{ then return } \text{Select}(k, S_1)
\]
\[
\text{if } |S_1| + |S_2| \geq k \text{ then return } m
\]
\[
\text{return } \text{Select}(k - |S_1| - |S_2|, S_3)
\]
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 $O(n)$.*
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 $O(n)$. 
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 $O(n)$. 
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 $O(n)$. 
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 $O(n)$. 
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.
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.
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.
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.
Parallel k-Select

- Input $S = \{s_1, \ldots, s_n\}$.
- Processors $P_1, P_2, \ldots P_{\lceil n^{1-x} \rceil}$, 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.
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.
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.
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.
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.
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.
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.
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:**

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

**M:**

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

**sorted M:**

```
```
**Example for the k-th Element**

**Input/Data:**

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

**P:**

- $P_1$ to $P_{35}$

**M:**

- $M$ as a list

**sorted M:**

- Sorted list of elements
Example for the k-th Element

<table>
<thead>
<tr>
<th>Input/Data:</th>
</tr>
</thead>
<tbody>
<tr>
<td>36 34 91 91 10 27 69 10 34 51 15 44 59 53 26 26 56 42 43 20 78 5 82 83 45 89 61 64 64 0 93 72 2 73 33</td>
</tr>
<tr>
<td>8 10 73 33 31 10 90 51 36 48 29 8 37 75 1 52 53 55 46 19 2 84 1 94 71 35 57 86 26 51 13 24 76 18 93</td>
</tr>
<tr>
<td>3 71 51 41 67 75 60 27 22 49 78 65 12 93 61 94 54 70 21 36 10 40 36 81 5 71 88 93 38 7 71 40 33 77 33</td>
</tr>
<tr>
<td>37 83 90 8 77 85 96 68 86 6 49 66 32 88 11 48 29 18 96 53 82 15 30 8 72 82 65 71 72 39 43 55 81 55 93</td>
</tr>
<tr>
<td>13 56 26 5 76 20 48 90 95 91 70 40 95 38 3 93 15 74 55 60 60 72 26 82 85 69 35 41 19 21 1 24 60 37 16</td>
</tr>
<tr>
<td>13 30 93 30 37 1 32 48 93 97 55 3 97 78 21 6 7 97 59 18 16 87 30 16 21 83 65 22 70 69 57 24 88 56 29</td>
</tr>
<tr>
<td>34 16 83 71 7 50 84 36 35 8 94 1 26 70 81 59 24 23 20 70 48 53 34 88 66 4 10 10 81 23 46 55 36 38 40</td>
</tr>
<tr>
<td>50 62 95 82 96 95 79 49 51 23 45 27 15 54 8 12 78 66 29 75 34 16 3 93 59 77 88 9 2 11 77 85 56 23 5</td>
</tr>
<tr>
<td>82 79 72 51 23 95 30 11 11 61 47 94 29 89 25 71 23 49 91 4 36 73 72 57 77 70 2 88 25 79 8 61 55 16 40</td>
</tr>
<tr>
<td>68 34 21 30 66 89 92 99 26 51 41 1 94 67 52 68 60 65 20 88 87 65 71 70 5 57 64 34 60 91 27 59 12 92 87 13</td>
</tr>
<tr>
<td>83 92 79 54 2 47 97 22 10 28 97 20 8 52 86 37 77 80 55 11 61 57 77 43 82 93 49 90 61 8 51 30 46 10 57</td>
</tr>
<tr>
<td>31 0 14 15 6 38 5 12 19 16 35 9 23 84 85 23 41 52 66 67 67 14 11 35 40 35 82 62 3 50 31 18 5 41 68</td>
</tr>
<tr>
<td>28 20 47 47 67 34 38 66 38 80 89 18 93 39 22 73 13 13 43 19 54 90 48 43 83 68 41 97 3 2 60 25 86 29 79 59</td>
</tr>
<tr>
<td>8 53 84 52 3 14 47 74 50 82 94 46 91 30 24 14 30 15 13 55 37 94 29 43 25 61 95 46 33 69 86 43 16 83 45</td>
</tr>
<tr>
<td>18 95 65 25 75 15 62 57 84 93 30 38 50 8 92 9 85 16 37 71 49 89 29 45 58 39 38 29 45 72 1 78 85 84 7</td>
</tr>
</tbody>
</table>

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

M:

\[ 31 53 73 41 37 38 62 48 38 49 49 38 37 70 25 48 41 49 46 54 48 57 30 57 59 69 61 60 38 39 46 43 55 55 40 \]

sorted M:

\[ 25 30 31 37 37 38 38 38 39 40 41 41 43 46 46 48 48 48 49 49 49 53 54 55 55 57 57 59 60 61 62 69 70 73 \]
**Example for the k-th Element**

**Input/Data:**

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

**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 |
| 31 | 53 | 73 | 41 | 37 | 38 | 62 | 48 | 38 | 49 | 49 | 38 | 37 | 70 | 25 | 48 | 41 | 49 | 46 | 54 | 48 | 57 | 30 | 57 | 59 | 69 | 61 | 60 | 38 | 39 | 46 | 43 | 55 | 55 | 40 |

**sorted M:**

| 25 | 30 | 31 | 37 | 37 | 38 | 38 | 38 | 39 | 40 | 41 | 41 | 43 | 46 | 46 | 48 | 48 | 48 | 49 | 49 | 49 | 53 | 54 | 55 | 55 | 57 | 57 | 59 | 60 | 61 | 62 | 69 | 70 | 73 |
Parallel $k$-Select

Programm: ParSelect($k, S$)

1: if $|S| \leq k_1$ then $P_1$ 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: 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: \[ m := ParSelect(\lceil |M|/2 \rceil, M). \]

5: More to come!
Parallel k-Select

Programm: ParSelect(k,S)

1: 
   if $|S| \leq k_1$ then $P_1$ 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: 
   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: 
   $m := ParSelect(\lceil |M|/2 \rceil, M)$.

5: More to come!
Parallel k-Select

Programm: ParSelect(k,S)

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

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

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

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

5: More to come!
Parallel k-Select

Programm: ParSelect(k,S)
1: \(\text{if } |S| \leq k_1 \text{ then } P_1 \text{ returns } \text{Select}(k, S).\)
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: \(\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: \(m := \text{ParSelect}(\lceil |M|/2 \rceil, M)\).
5: More to come!
Parallel k-Select

Programm: ParSelect(k,S)
1: if $|S| \leq k_1$ then $P_1$ 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: 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: $m := ParSelect(\lceil |M|/2 \rceil, M)$.
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 $1 \leq i \leq \lceil n^{1-x} \rceil$ do in parallel
$\quad L_i := \{ s \in S_i \mid s < m \}$
$\quad E_i := \{ s \in S_i \mid s = m \}$
$\quad G_i := \{ s \in S_i \mid s > m \}$

5.2:
Compute with Parallel Prefix:
$\quad l_i := \sum_{j=1}^{i} |L_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.
$\quad e_i := \sum_{j=1}^{i} |E_i|$ for all $1 \leq i \leq \lceil n^{1-x} \rceil$.
$\quad g_i := \sum_{j=1}^{i} |G_i|$ 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

5.1:
Distribute $m$ via broadcast to all $P_i$.

\textbf{for all} $P_i$ where $1 \leq i \leq \lfloor n^{1-x} \rfloor$ \textbf{do in parallel}

\begin{align*}
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 \}
\end{align*}

5.2:
Compute with Parallel Prefix:

\begin{align*}
l_i &:= \sum_{j=1}^{i} |L_j| \text{ for all } 1 \leq i \leq \lfloor n^{1-x} \rfloor. \\
e_i &:= \sum_{j=1}^{i} |E_j| \text{ for all } 1 \leq i \leq \lfloor n^{1-x} \rfloor. \\
g_i &:= \sum_{j=1}^{i} |G_j| \text{ for all } 1 \leq i \leq \lfloor n^{1-x} \rfloor.
\end{align*}

Let: $l_0 = e_0 = g_0 = 0$

5.3:
Even more to come!
Parallel k-Select

Programm: ParSelect(k,S) Steps 5

5.1:
Distribute $m$ via broadcast to all $P_i$.

\begin{align*}
\text{for all } P_i \text{ where } 1 \leq i \leq \lceil n^{1-x} \rceil \text{ 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 \}
\end{align*}

5.2:
Compute with Parallel Prefix:

\begin{align*}
l_i &:= \sum_{j=1}^{i} |L_j| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \\
e_i &:= \sum_{j=1}^{i} |E_j| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \\
g_i &:= \sum_{j=1}^{i} |G_j| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \\
\text{Let: } l_0 = e_0 = g_0 = 0
\end{align*}

5.3:
Even more to come!
Parallel k-Select

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

5.3:

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

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

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

6:

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

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:  
   \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}
   m_i := Select(\lceil |S_i|/2 \rceil, S_i)
   P_i(m_1) \rightarrow R_i.
   \text{ Assume in the following that } M \text{ is the sequence of these values }

4:  
   m := ParSelect(\lceil |M|/2 \rceil, M).
Programm: ParSelect(k,S)

1: $O(1)$
   
   if $|S| \leq k_1$ then $P_1$ 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:
   
   for all $P_i$ where $1 \leq i \leq \lceil n^{1-x} \rceil$ 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:
   
   $m := \text{ParSelect}(\lceil |M|/2 \rceil, M)$. 


Parallel k-Select (Running Time)

Programm: ParSelect(k,S)

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

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

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

4:  
   \[m := ParSelect(\lceil |M|/2 \rceil, M).\]
Parallel k-Select (Running Time)

Programm: ParSelect(k,S)

1: $O(1)$
   \[\text{if } |S| \leq k_1 \text{ then } P_1 \text{ returns } Select(k, S).\]
2: $O(\log_2(|S|^{1-x}))$ thus we have $O(\log n)$
   \[S \text{ is split into } \lceil |S|^{1-x} \rceil \text{ sub-sequences } S_i \text{ with } |S_i| \leq \lceil n^x \rceil\]
   \[P_i \text{ stores the start-address of } S_i.\]
3: $O(n^x)$
   \[\text{for all } P_i \text{ where } 1 \leq i \leq \lceil n^{1-x} \rceil \text{ do in parallel}\]
   \[m_i := Select(\lceil |S_i|/2 \rceil, S_i)\]
   \[P_i(m_1) \rightarrow R_i.\]
   \[\text{Assume in the following that } M \text{ is the sequence of these values}\]
4: $m := ParSelect(\lceil |M|/2 \rceil, M)$. 
Parallel k-Select (Running Time)

Programm: ParSelect(k,S)

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

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

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

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

Programm: ParSelect(k,S) Steps 5

5.1a:
Distribute \( m \) via broadcast to all \( P_i \).

5.1b:
for all \( P_i \) where \( 1 \leq i \leq \lceil n^{1-x} \rceil \) do in parallel

\[
\begin{align*}
L_i & := \{ s \in S_i \mid s < m \} \\
E_i & := \{ s \in S_i \mid s = m \} \\
G_i & := \{ s \in S_i \mid s > m \}
\end{align*}
\]

5.2:
Compute with Parallel Prefix:

\[
\begin{align*}
l_i & := \sum_{j=1}^{i} |L_i| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \\
e_i & := \sum_{j=1}^{i} |E_i| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \\
g_i & := \sum_{j=1}^{i} |G_i| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil.
\end{align*}
\]

Let:

\( l_0 = e_0 = g_0 = 0 \)
Parallel k-Select (Running Time)

Programm: ParSelect(k,S) Steps 5

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

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

5.1b:

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: $l_0 = e_0 = g_0 = 0$
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

\[
\begin{align*}
L_i & := \{ s \in S_i \mid s < m \} \\
E_i & := \{ s \in S_i \mid s = m \} \\
G_i & := \{ s \in S_i \mid s > m \}
\end{align*}
\]

5.2:

Compute with Parallel Prefix:

\[
\begin{align*}
l_i & := \sum_{j=1}^{i} |L_i| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \\
e_i & := \sum_{j=1}^{i} |E_i| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \\
g_i & := \sum_{j=1}^{i} |G_i| \text{ for all } 1 \leq i \leq \lceil n^{1-x} \rceil. \\
\text{Let: } l_0 = e_0 = g_0 = 0
\end{align*}
\]
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:
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) 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:

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

Programm: ParSelect(k,S) 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 $R_{l_i+1}, \ldots, R_{l_i}$.
- $P_i$ writes $E_i$ in $R_{e_i+1}, \ldots, R_{e_i}$.
- $P_i$ writes $G_i$ in $R_{g_i+1}, \ldots, R_{g_i}$.

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

- 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. \(\Omega(n)\).
Sequential Merging

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

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

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

- Algorithm does not care about special cases.

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

- Lower bound on the number of comparisons is \( r + s \), i.e. \( \Omega(n) \).
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
  
  \[
  \begin{align*}
  i &:= 1; \quad j := 1; \quad n := r + s \\
  \text{for } k := 1 \text{ to } n \text{ do} & \\
  \quad \text{if } a_i < b_j & \\
  \quad \quad \text{then } c_k := a_i; \quad i := i + 1; & \\
  \quad \quad \text{else } c_k := b_j; \quad j := j + 1; &
  \end{align*}
  \]

- 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. \( \Omega(n) \).
Sequential Merging

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

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

- **Program:** Merge
  
  \[
i := 1; j := 1; n := r + s \]
  
  for \( k := 1 \) to \( n \) do
    
    if \( a_i < b_j \)
    
    then \( c_k := a_i; i := i + 1; \)
    
    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. \( \Omega(n) \).
Sequential Merging

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

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

- **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. \( \Omega(n) \).
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**
  
  \[ i := 1; j := 1; n := r + s \]
  
  for \( k := 1 \) to \( n \) do
    
    if \( a_i < b_j \)
      
      then \( c_k := a_i; i := i + 1 \);
    
    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. \( \Omega(n) \).
Sequential Merging

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

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

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

- 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)

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

- 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).
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)

- 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).
Motivation and History
PRAM Introduction
Efficiency
Selection
Merging

1:58 Parallel Merging  1/6
Walter Unger 8.11.2016 21:46  WS2016/17

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 $\hat{B}_i \setminus C$. 
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 $\hat{B}_i \setminus C$. 

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 $\hat{B}_i \setminus C$. 
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 $\hat{B}_i \setminus C$. 
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 $\hat{B}_i \setminus C$. 
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 \( P(n) \leq n/\log n \).
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 \( P(n) \leq n/\log n \).
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 $P(n) \leq n/\log n$. 
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 $P(n) \leq n/\log n$. 
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 $P(n) \leq n/\log n$. 
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 $P(n) \leq n/\log n$. 
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 $P(n) \leq n/\log n$. 
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 $P(n) \leq n/\log n$. 

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.
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.
- 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.
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.
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.
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.
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.
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.
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.
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.
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

| 4 | 5 | 10 | 11 | 20 | 21 | 22 | 23 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 38 | 39 | 41 | 42 | 45 | 46 | 47 | 48 | 49 | 50 | 54 | 55 | 56 | 58 | 64 |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

| 1 | 2 | 3 | 6 | 7 | 8 | 9 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 24 | 25 | 26 | 36 | 37 | 40 | 43 | 44 | 51 | 52 | 53 | 57 | 59 | 60 | 61 | 62 | 63 |

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

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

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

- Sequences A and B are sorted.
- Compute median \( a \) of A and median \( b \) of B.
Example for the Median for two Sorted Sequences

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

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

Sequences $A$ and $B$ are sorted.

Compute median $a$ of $A$ and median $b$ of $B$. 
Sequences $A$ and $B$ are sorted.

Compute median $a$ of $A$ and median $b$ of $B$. 
Example for the Median for two Sorted Sequences

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

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

- Sequences $A$ and $B$ are sorted.
- Compute median $a$ of $A$ and median $b$ of $B$. 
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.
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.
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.
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.
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.
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 \( B \).
3. Split the sequences \( A \) and \( B \) in two sub-sequences each.
4. Continue recursively, till all sub-sequences are smaller than \( n/P(n) \).
5. Merge in the same way as before.

- 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 \( P(n) < \frac{n}{(\log n)^2} \).
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 $n/P(n)$.
5. Merge in the same way as before.

- 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 $P(n) < \frac{n}{(\log n)^2}$. 
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.

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 $P(n) < \frac{n}{(\log n)^2}$. 
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 \( P(n) < \frac{n}{(\log n)^2} \).
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 $P(n) < \frac{n}{(\log n)^2}.$
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 $P(n) < \frac{n}{(\log n)^2}$. 
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?
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?
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?
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