



Chapter 10

Parallel Algorithms

Algorithm Theory
WS 2019/20

Fabian Kuhn

Sequential Algorithms

Classical Algorithm Design:

- One machine/CPU/process/... doing a computation

RAM (Random Access Machine):

- Basic standard model
- Unit cost basic operations
- Unit cost access to all memory cells

Sequential Algorithm / Program:

- Sequence of operations
(executed one after the other)

Parallel and Distributed Algorithms

Today's computers/systems are not sequential:

- Even cell phones have several cores
- Future systems will be highly parallel on many levels
- This also requires appropriate algorithmic techniques

Goals, Scenarios, Challenges:

- Exploit parallelism to speed up computations
- Shared resources such as memory, bandwidth, ...
- Increase reliability by adding redundancy
- Solve tasks in inherently decentralized environments
- ...

Parallel and Distributed Systems

- Many different forms
- Processors/computers/machines/... communicate and share data through
 - Shared memory or message passing
- Computation and communication can be
 - Synchronous or asynchronous
- Many possible **topologies** for message passing
- Depending on system, various **types of faults**

Algorithmic and theoretical challenges:

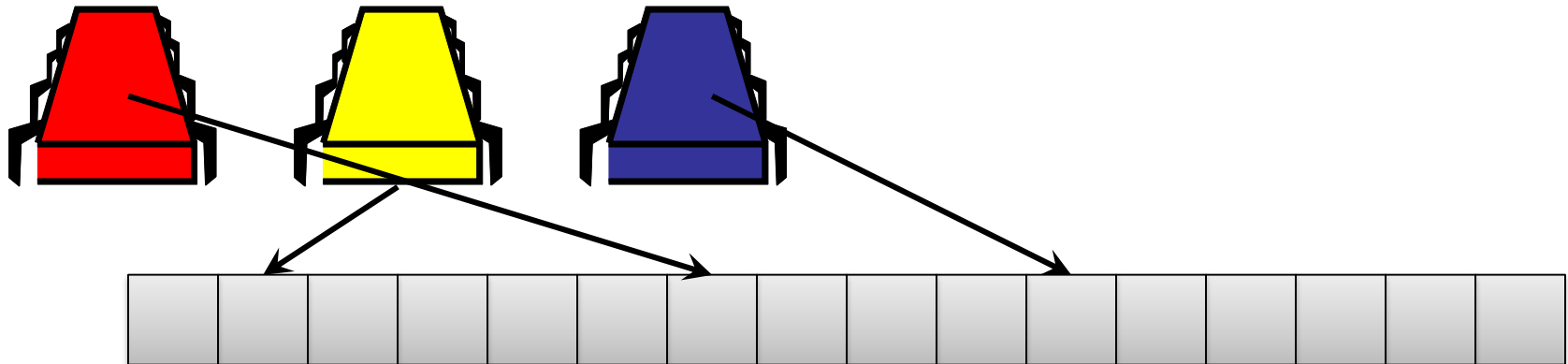
- How to parallelize computations
- Scheduling (which machine does what)
- Load balancing
- Fault tolerance
- Coordination / consistency
- Decentralized state
- Asynchrony
- Bounded bandwidth / properties of comm. channels
- ...

Models

- A large variety of models, e.g.:
- **PRAM** (Parallel Random Access Machine)
 - Classical model for parallel computations
- **Shared Memory**
 - Classical model to study coordination / agreement problems, distributed data structures, ...
- **Message Passing** (fully connected topology)
 - Closely related to shared memory models
- Message Passing in **Networks**
 - Decentralized computations, large parallel machines, comes in various flavors...
- Computations in large clusters of powerful individual machines: Massively Parallel Computations (MPC)

PRAM

- Parallel version of RAM model
- p processors, shared random access memory



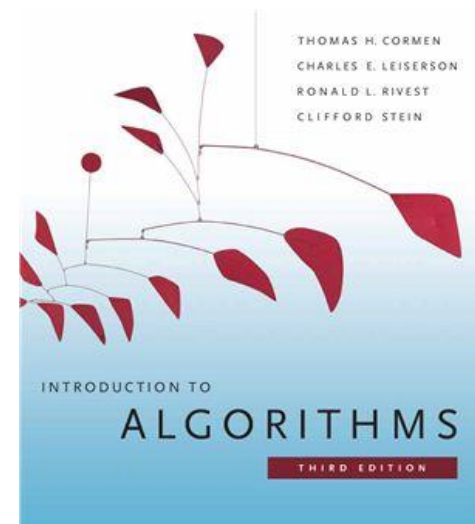
- Basic operations / access to shared memory cost 1
- Processor operations are synchronized
- **Focus on parallelizing computation** rather than cost of communication, locality, faults, asynchrony, ...

Other Parallel Models

- **Message passing:** Fully connected network, local memory and information exchange using messages
- **Dynamic Multithreaded Algorithms:** Simple parallel programming paradigm
 - E.g., used in Cormen, Leiserson, Rivest, Stein (CLRS)

```

FIB( $n$ )
1  if  $n < 2$ 
2    then return  $n$ 
3   $x \leftarrow$  spawn FIB( $n - 1$ )
4   $y \leftarrow$  spawn FIB( $n - 2$ )
5  sync
6  return ( $x + y$ )
  
```



Parallel Computations

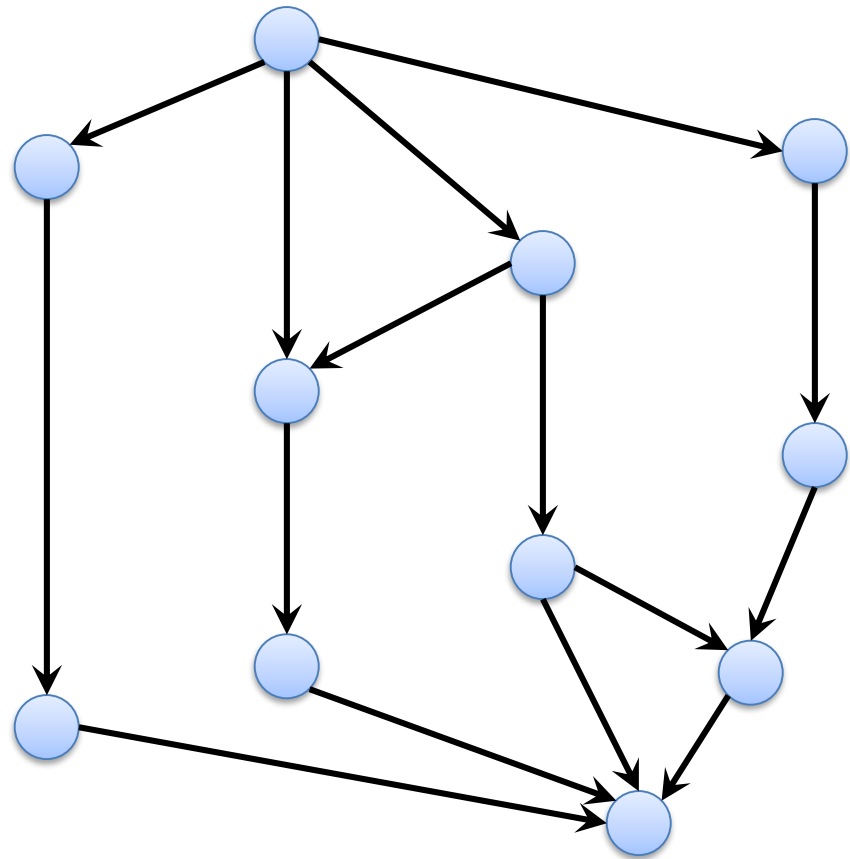
Sequential Computation:

- Sequence of operations



Parallel Computation:

- Directed Acyclic Graph (DAG)



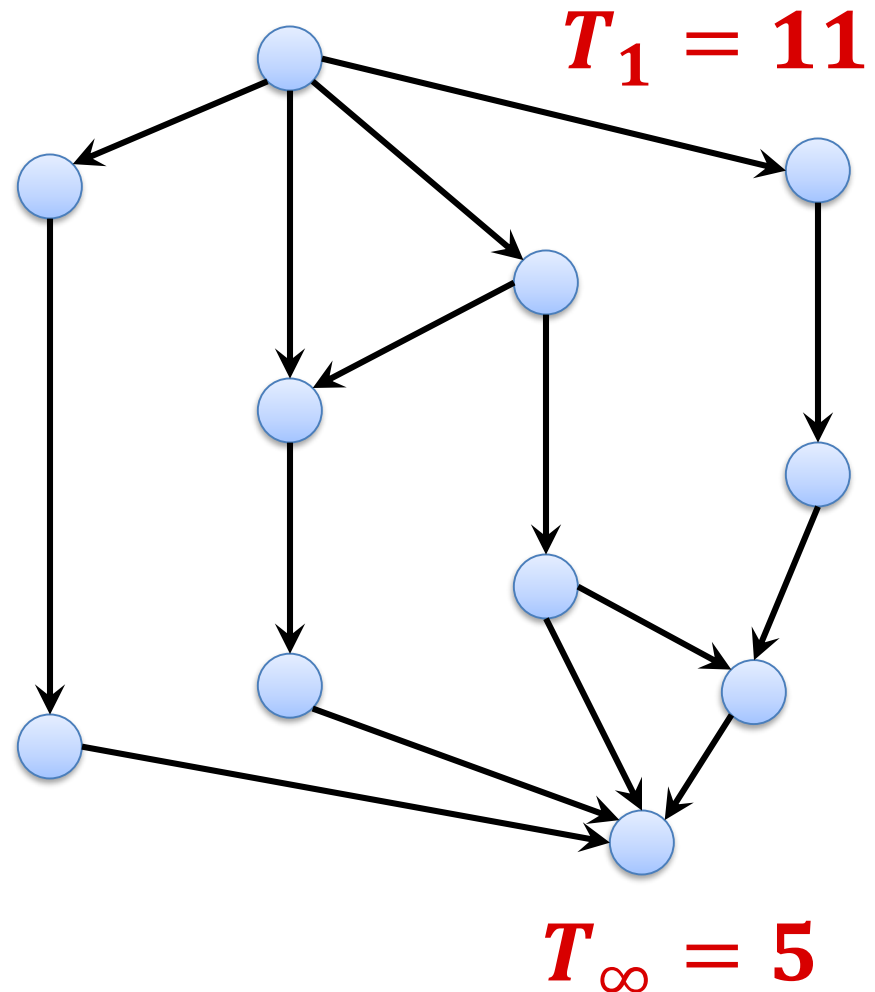
Parallel Computations

T_p : time to perform comp. with p procs

- T_1 : **work** (total # operations)
 - Time when doing the computation sequentially
- T_∞ : **critical path / span**
 - Time when parallelizing as much as possible

• **Lower Bounds:**

$$T_p \geq \frac{T_1}{p}, \quad T_p \geq T_\infty$$



Parallel Computations

T_p : time to perform comp. with p procs

- **Lower Bounds:**

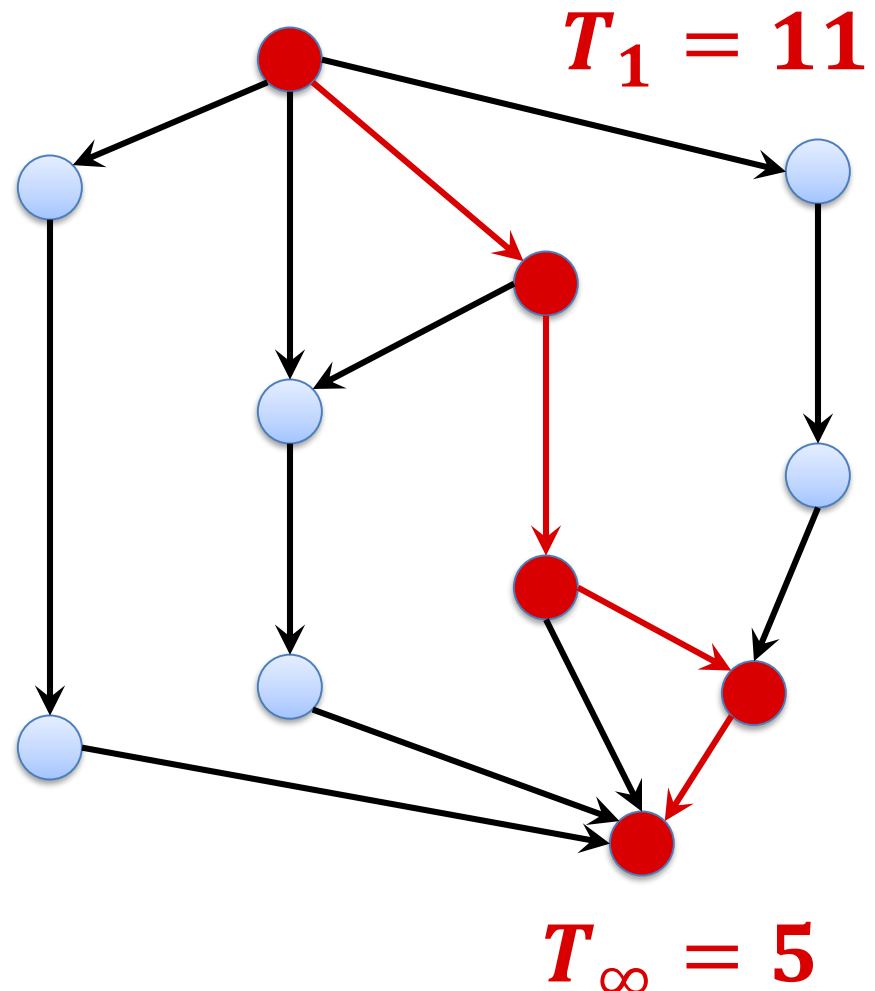
$$T_p \geq \frac{T_1}{p}, \quad T_p \geq T_\infty$$

- **Parallelism:** $\frac{T_1}{T_\infty}$

– maximum possible speed-up

- **Linear Speed-up:**

$$\frac{T_p}{T_1} = \Theta(p)$$



- How to assign operations to processors?
- Generally an online problem
 - When scheduling some jobs/operations, we do not know how the computation evolves over time

Greedy (offline) scheduling:

- Order jobs/operations as they would be scheduled optimally with ∞ processors (topological sort of DAG)
 - Easy to determine: With ∞ processors, one always schedules all jobs/ops that can be scheduled
- Always schedule as many jobs/ops as possible
- Schedule jobs/ops in the same order as with ∞ processors
 - i.e., jobs that become available earlier have priority

Brent's Theorem

Brent's Theorem: On p processors, a parallel computation can be performed in time

$$T_p \leq \frac{T_1 - T_\infty}{p} + T_\infty.$$

Proof:

- Greedy scheduling achieves this...
- #operations scheduled with ∞ processors in round i : x_i

Brent's Theorem

Brent's Theorem: On p processors, a parallel computation can be performed in time

$$T_p \leq \frac{T_1 - T_\infty}{p} + T_\infty.$$

Proof:

- Greedy scheduling achieves this...
- #operations scheduled with ∞ processors in round i : x_i

Brent's Theorem

Brent's Theorem: On p processors, a parallel computation can be performed in time

$$T_p \leq \frac{T_1 - T_\infty}{p} + T_\infty.$$

Corollary: Greedy is a 2-approximation algorithm for scheduling.

Corollary: As long as the number of processors $p = O(T_1/T_\infty)$, it is possible to achieve a linear speed-up.

Back to the PRAM:

- Shared random access memory, synchronous computation steps
- The PRAM model comes in variants...

EREW (exclusive read, exclusive write):

- Concurrent memory access by multiple processors is not allowed
- If two or more processors try to read from or write to the same memory cell concurrently, the behavior is not specified

CREW (concurrent read, exclusive write):

- Reading the same memory cell concurrently is OK
- Two concurrent writes to the same cell lead to unspecified behavior
- This is the first variant that was considered (already in the 70s)

The PRAM model comes in variants...

CRCW (concurrent read, concurrent write):

- Concurrent reads and writes are both OK
- Behavior of concurrent writes has to be specified
 - Weak CRCW: concurrent write only OK if all processors write 0
 - Common-mode CRCW: all processors need to write the same value
 - Arbitrary-winner CRCW: adversary picks one of the values
 - Priority CRCW: value of processor with highest ID is written
 - Strong CRCW: largest (or smallest) value is written

- The given models are ordered in strength:

weak \leq common-mode \leq arbitrary-winner \leq priority \leq strong

Theorem: A parallel computation that can be performed in time t , using p proc. on a strong CRCW machine, can also be performed in time $O(t \log p)$ using p processors on an EREW machine.

- Each (parallel) step on the CRCW machine can be simulated by $O(\log p)$ steps on an EREW machine

Theorem: A parallel computation that can be performed in time t , using p proc. on a strong CRCW machine, can also be performed in time $O(t \log p)$ using p processors on an EREW machine.

- Each (parallel) step on the CRCW machine can be simulated by $O(\log p)$ steps on an EREW machine

Some Relations Between PRAM Models



Theorem: A parallel computation that can be performed in time t , using p proc. on a strong CRCW machine, can also be performed in time $O(t \log p)$ using p processors on an EREW machine.

- Each (parallel) step on the CRCW machine can be simulated by $O(\log p)$ steps on an EREW machine

Theorem: A parallel computation that can be performed in time t , using p probabilistic processors on a strong CRCW machine, can also be performed in expected time $O(t \log p)$ using $O(p/\log p)$ processors on an arbitrary-winner CRCW machine.

- The same simulation turns out more efficient in this case

Some Relations Between PRAM Models



Theorem: A computation that can be performed in time t , using p processors on a strong CRCW machine, can also be performed in time $O(t)$ using $O(p^2)$ processors on a weak CRCW machine

Proof:

- **Strong:** largest value wins, **weak:** only concurrently writing 0 is OK