

If communication is cheap,

- then distributed memory can be replaced by **shared memory**: processors access a global, shared memory instead of using point-to-point communication. What model should be chosen?
 - ▶ **Exclusive-read, exclusive-write** (EREW): a register may be read or modified by a single processor.
 - ▶ **Concurrent-read, exclusive-write** (CREW): a register can be read by any number of processors, but modified by a single processor only.
 - ▶ **Concurrent-read, concurrent-write** (CRCW): any register may read or modified by any number of processors.
- So far we have demanded optimal parallelizations, i.e., constant efficiency. Which speedups can be obtained, if processors are cheap, i.e., if we work with a polynomial number of processors?

Are there efficiently solvable problems which do not allow strong speedups?

The Random Access Machine (RAM)

The **RAM** is a model of today's computers. A RAM is composed of:

- an input device,
- a *CPU* which stores the program and a label referring to the current command,
- a storage composed of an unbounded number of local registers, where each register stores an integer,
- an accumulator which
 - ▶ may access a register directly (by its address) or indirectly (by the address as stored in a register)
 - ▶ and perform arithmetic or boolean operations,
- and an output device.

The Parallel Random Access Machine (PRAM)

- A **PRAM** is a collection of RAM's, where each RAM (or processor) may access a **shared memory** of an unbounded number of global registers.
 - ▶ The input of length n is stored in the first n registers of the shared memory. Each processor receives also the input length n and the number of processors, which are stored in special local registers.
 - ▶ All other local or global registers are initialized to be zero.
 - ▶ All processors use the same program.
 - ▶ Processors compute **synchronously**.
- One distinguishes **EREW-PRAMs**, **CREW-PRAMs** and **CRCW-PRAMs**.
- How to cope with **write conflicts**, i.e., if several processors try to write into the same register?

Conflict Resolution

- The most restrictive model is the **common model**: all processors writing into the same register at the same time have to write the same value.
 - The **arbitrary model** chooses an arbitrary processor from among the processors trying to write into a fixed register and writes its value. The PRAM program has to work correctly for all choices.
 - The **priority model** is the most liberal model: the processor with the smallest address wins and its value is written.
- A “common PRAM” is also an “arbitrary PRAM” and an “arbitrary PRAM” is also a “priority PRAM”.
 - Does the conflict resolution scheme matter?

$p, t : \mathbb{N} \rightarrow \mathbb{N}$ are two functions.

- $\text{EREW}(p(n), t(n))$: all languages $L \subseteq \{0, 1\}^*$ recognizable by an EREW-PRAM in time $O(t(n))$ with $O(p(n))$ processors.
- The classes $\text{CREW}(p(n), t(n))$ and $\text{CRCW}(p(n), t(n))$ are defined analogously.
- Nick's class is defined as

$$\mathcal{NC} = \bigcup_{k, l \in \mathbb{N}} \text{CRCW}(n^k, (\log_2 n)^l)$$

and consists of all languages L recognizable with super-fast algorithms: **poly-logarithmic time** with a **polynomial number of processors**.

Simulating a CRCW-PRAM via Message Passing

Simulate a CRCW-PRAM P with $p(n)$ processors by a message passing algorithm Q with $p(n)$ processors.

- Q uses the hypercube as communication pattern.
- The simulation setup:
 - ▶ processor ρ of P is simulated by node $v(\rho)$ of the hypercube.
 - ▶ Shared memory cells are distributed among the nodes of the hypercube via a good [hash function \$h\$](#) .
- Simulating a step of processor ρ :
 - ▶ if ρ does not access the shared memory: no problem, $v(\rho)$ has all the information it needs.
 - ▶ What to do, if ρ accesses the shared memory?

Simulating A Shared Memory Access

Use our solution of the routing problem via **randomized routing on the hypercube**.

- If ρ wants to write into or read from cell c , then $v(\rho)$ sends a message to its colleague $h(c)$ using the bit fixing method of randomized routing. If two packets “meet on the way” and
 - ▶ if both try to write into the same cell, then “kill” the packet coming from the processor with larger address.
 - ▶ if both try to read from the same cell, then “stop” the packet coming from the processor with the larger address. The surviving packets
 - ★ try to get thru to their destination,
 - ★ return to the sender by running their path backwards
 - ★ and wake up all stopped packets.
 - ▶ With high probability, the random routing on the hypercube has no bottlenecks.
- With high probability, the CRCW-PRAM algorithm can be simulated by a message passing algorithm with the same number $p(n)$ processors and slowdown $O(\log_2 p(n))$. ◀

Comparing PRAMs of Different Types

- $\text{RAM}(t)$ is the class of all languages recognizable by a sequential random access machine in time $t(n)$ for inputs of length n . Then
$$\text{EREW}(p(n), t(n)) \subseteq \text{CREW}(p(n), t(n)) \subseteq \text{CRCW}(p(n), t(n)) \subseteq \text{RAM}(p(n) \cdot t(n)).$$
- We need the following facts:
 - ▶ The input for the **zero-counting** problem is a **sorted 0/1 sequence** A . We have to determine the position i with $A[i] = 0$ and $A[i + 1] = 1$. Any EREW-PRAM requires time $\Omega(\log_2 \frac{n}{p})$ with p processors. ◀
 - ▶ Any CREW-PRAM algorithm requires time $\Omega(\log_2 n)$ to compute the **Boolean OR** of n bits, even with an **unlimited** number of processors. ◀
 - ▶ The **parity function** on n bits requires time $\Omega(\frac{\log_2 n}{\log_2 \log_2 n})$ on a priority CRCW-PRAM with a polynomial number of processors. ◀

Exclusive versus Concurrent Read I

In the **parallel search problem** a sorted array X of n keys and an additional key y is given.

- Set $X[0] = -\infty$ and $X[n+1] = +\infty$.
- $\text{rank}(y \mid X)$ is the position j with $X[j] < y \leq X[j+1]$.

Determine $\text{rank}(y \mid X)$ with p processes.

- (1) $m = n + 1$; $l = 0$; $X[0] = -\infty$, $X[m] = +\infty$;
// The invariant $y \in [X[l], X[l+m]]$ will be maintained.
- (3) repeat $\lceil \log_p(n+2) \rceil$ times
 for $i = 1$ to p pardo
 if $X[l + (i-1) \cdot \frac{m}{p}] \leq y \leq X[l + i \cdot \frac{m}{p} - 1]$ then
 $l = l + (i-1) \cdot \frac{m}{p}$; $m = \frac{m}{p} - 1$;
 if $X[l + (i-1) \cdot \frac{m}{p}] < y < X[l + (i-1) \cdot \frac{m}{p} + 1]$ then
 $\text{rank}(y \mid X) = l + (i-1) \cdot \frac{m}{p}$; stop;

Exclusive versus Concurrent Read II

- No write conflicts, but there are read conflicts, since all processes have to know y and the starting point l of the next search interval.
- Time $O(\log_p(n)) = O(\frac{\log_2 n}{\log_2 p})$ for a CREW-PRAM.
- How much time for EREW-PRAMs?
 - ▶ We can solve the zero counting problem with the parallel search problem. 
 - ▶ Any EREW-PRAM requires time $\Omega(\log_2 \frac{n}{p})$ for the parallel search problem.
 - ▶ For $p = \sqrt{n}$: a solution in time $O(1)$ with a CREW-PRAM algorithm, whereas time $\Omega(\log_2 n)$ is required for EREW-PRAM algorithms.

Exclusive Versus Concurrent Write: OR

Compute the Boolean OR of n Bits x_1, \dots, x_n with n processors.

- With a ERCW-PRAM algorithm:
 - ▶ processor i reads x_i
 - ▶ and, if $x_i = 1$, writes a 1 into register 1 of the shared memory.
- Time $O(1)$ with n processors for an **ERCW-PRAM**, assuming the common model.
- For CREW-PRAM algorithms: ▶
 - ▶ time $\Omega(\log_2 n)$, even with an unlimited number of processors, is required
 - ▶ and also sufficient, if we use the complete binary tree as communication pattern.

Computing The Minimum

Determine the minimum of n integers $x[1], \dots, x[n] \in \{0, \dots, n-1\}$.

- We work with a priority **ERCW-PRAM** with n processors.
 - ▶ Each processor i sets $x[x[i]] = 1$.
 - ▶ Hence $x[q] = 1$ iff q is one of the numbers to be minimized.
 - ▶ If $x[q] = 1$, then processor q tries to set $x[0] = q$.
 - ▶ The processor with smallest address wins and $x[0]$ stores the smallest value.
- Time $O(1)$ with n processors is sufficient.
- Computing the minimum is at least as hard as computing the OR-function:
 - ▶ Flip the bits $b[1], \dots, b[n]$ in parallel.
 - ▶ Determine the minimum of the flipped bits.
 - ▶ The minimum is one iff the OR is zero.
- Any **CREW-PRAM** algorithm requires time $\Omega(\log_2 n)$ to determine the minimum, even with an unlimited number of processors.

Randomized EREW-PRAMs Versus CRCW-PRAMs

- We have simulated a CRCW priority PRAM by a randomized message passing algorithm. 
- Just observe that an EREW-PRAM algorithm is at least as powerful as a message passing algorithm:
EREW-PRAMs (with p shared memory registers and p processes) and message passing algorithms are identical concepts!

$CRCW(p, t) \subseteq EREW(p, t \cdot \log_2 p)$, if we assume the priority mode for CRCW algorithms and allow randomized EREW computations.

Write Resolution Schemes I

Any **priority-CRCW PRAM P** with p processors and time t can be simulated by a **common-CRCW PRAM Q** with $O(p \cdot \log_2 p)$ processors in time $O(t)$.

- Hence the weakest resolution scheme is as fast as the strongest scheme, if the number of processors is slightly increased.
- The simulation:
 - ▶ We only have to worry about simulating a priority-write operation of P by a common-write operation of Q .
 - ▶ A processor i_P of P is simulated by the processor i_Q of Q and its $\log_2 p$ assistants.
 - ▶ A shared memory register of P is simulated by an interval of $2p - 1$ shared memory registers of Q .

Write Resolution Schemes II

- The $2p - 1$ registers are used to simulate a complete binary tree of depth $\log_2 p$ with p leaves and $p - 1$ inner nodes.
- If processor i_P writes into shared memory register r , then
 - ▶ i_Q and its assistants write a one in the “leaf i_P ” of r and in all ancestors of i_P , for which the leaf belongs to the left subtree.
 - ▶ Afterwards, i_Q and its assistants check whether an ancestor, for which i_Q belongs to the right subtree, has received a one.
 - ▶ If so, then i_Q has been beaten: there is a processor with smaller address trying to write into the same register.
- The processors of the simulating PRAM write only ones: the common model is used.
- Drawback: the number of used registers increases by a factor of p and the number of processors increases logarithmically.

The Complexity of the Prefix Problem

Any priority CRCW PRAM requires at least $\Omega\left(\frac{\log_2 n}{\log \log n}\right)$ steps to compute the XOR $x_1 \oplus \cdots \oplus x_n$ with a polynomial number of processors. 

- \oplus is an associative operation and hence the **prefix problem** cannot be sped up beyond $O\left(\frac{\log_2 n}{\log \log n}\right)$,
 - ▶ even if the number of processors is considerably increased
 - ▶ and message passing is replaced by a CRCW algorithm.
- A message passing algorithm has to spend $\Omega(\log_2 n)$ steps for the prefix problem, even with an unbounded number of processors, since the OR is hard for EREW-PRAMs.

- Message Passing versus PRAMs:
 - ▶ Randomized Message Passing algorithms, and hence randomized EREW-PRAM algorithms, can simulate CRCW-PRAM algorithms with only a logarithmic delay.
 - ▶ Message Passing and EREW-PRAMs coincide, if an EREW-PRAM with p processors uses only p shared memory registers.
- Exclusiveness versus concurrency:
 - ▶ CREW-PRAMS are “logarithmically faster” than EREW-PRAMS for the parallel search problem.
 - ▶ ERCW-PRAMS are logarithmically faster than CREW-PRAMS when computing the minimum or computing the OR.
- Write resolution schemes:
 - ▶ The common-model can simulate the priority model with no delay, if the number of shared memory registers is increased by a factor of p and if the number of processors is increased logarithmically.