10

Parallel Algorithms

17 December 2024

Prof. Dr. Sebastian Wild

CS566 (Wintersemester 2024/25)
Philipps-Universität Marburg

version `2024-12-17 13:51`

Learning Outcomes

Unit 10: ***Parallel Algorithms***

1. Know and apply *parallelization strategies* for embarrassingly parallel problems.
2. Identify *limits of parallel speedups*.
3. Understand and use the *parallel random-access-machine* model in its different variants.
4. Be able to *analyze* and compare simple shared-memory parallel algorithms by determining *parallel time and work*.
5. Understand efficient parallel *prefix sum* algorithms.
6. Be able to devise high-level description of *parallel quicksort and mergesort* methods.

Outline

10 Parallel Algorithms

10.1 Parallel Computation

Clicker Question

Have you ever written a concurrent program (explicit threads, job pools library, or using a framework for distributed computing)?

- A Yes
- B No
- C Concur. . . what?

→ sli.do/cs566

Types of parallel computation

£££ can't buy you more time . . . but more computers!

$\rightsquigarrow$ Challenge: Algorithms for *parallel* computation.

Types of parallel computation

£££ can't buy you more time . . . but more computers!

$\rightsquigarrow$ Challenge: Algorithms for *parallel* computation.

There are two main forms of parallelism:

1. **shared-memory parallel computer** $\leftarrow$ *focus of today*
   - p *processing elements* (PEs, processors) working in parallel
   - **single** big memory, **accessible from every PE**
   - communication via shared memory
   - think: a big server, 128 CPU cores, terabyte of main memory
2. **distributed computing**
   - p PEs working in parallel
   - each PE has **private** memory
   - communication by sending **messages** via a network
   - think: a cluster of individual machines

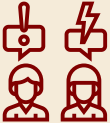
PRAM – Parallel RAM

- extension of the RAM model (recall Unit 2)
- the p PEs are identified by ids $0, \ldots, p-1$
  - like w (the word size), p is a parameter of the model that can grow with n
  - $p = \Theta(n)$ is not unusual   maaany processors!
- the PEs all **independently** run the same RAM-style program
  (they can use their id there)
- each PE has its own registers, but `MEM` is shared among all PEs
- computation runs in **synchronous** steps:
  in each time step, every PE executes one instruction

PRAM – Parallel RAM

- extension of the RAM model (recall Unit 2)
- the p PEs are identified by ids $0, \ldots, p-1$
  - like w (the word size), p is a parameter of the model that can grow with n
  - $p = \Theta(n)$ is not unusual   maaany processors!
- the PEs all **independently** run the same RAM-style program (they can use their id there)
- each PE has its own registers, but `MEM` is shared among all PEs
- computation runs in **synchronous** steps:
  in each time step, every PE executes one instruction
- As for RAM:
  - assume a basic "operating system"
  - $\rightsquigarrow$ write algorithms in pseudocode instead of RAM assembly
  - **NEW:** loops and commands can be run **"in parallel"**   (examples coming up)

PRAM – Conflict management



Problem: What if several PEs simultaneously overwrite a memory cell?

- **EREW**-PRAM (exclusive read, exclusive write)
  any **parallel access** to same memory cell is **forbidden** (crash if happens)
- **CREW**-PRAM (concurrent read, exclusive write)
  parallel **write** access to same memory cell is **forbidden**, *but reading is fine*
- **CRCW**-PRAM (concurrent read, concurrent write)
  concurrent access is allowed,
  need a rule for write conflicts:
  - common CRCW-PRAM:
    all concurrent writes to same cell must write **same** value
  - arbitrary CRCW-PRAM:
    some unspecified concurrent write wins
  - (more exist . . .)

- no single model is always adequate, but our default is CREW

PRAM – Execution costs

Cost metrics in PRAMs

- **space:** total amount of accessed memory
- **time:** number of steps till all PEs finish assuming sufficiently many PEs!
  sometimes called *depth* or *span*
- **work:** total #instructions executed on **all** PEs

PRAM – Execution costs

Cost metrics in PRAMs

- **space:** total amount of accessed memory
- **time:** number of steps till all PEs finish assuming sufficiently many PEs!
  sometimes called *depth* or *span*
- **work:** total #instructions executed on **all** PEs

Holy grail of PRAM algorithms:

- minimal time (=span)
- work (asymptotically) no worse than running time of best sequential algorithm
  $\rightsquigarrow$ *"work-efficient"* algorithm: work in same Θ-class as best sequential

Clicker Question

Does every computational problem allow a work-efficient algorithm?

A Yes

B No

→ sli.do/cs566

Clicker Question

Does every computational problem allow a work-efficient algorithm?

A Yes ✓

B ~~No~~

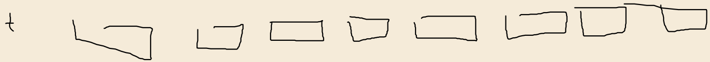
→ sli.do/cs566

The number of processors

Hold on, my computer does not have $\Theta(n)$ processors! Why should I care for span and work!?

Theorem 10.1 (Brent's Theorem)
If an algorithm has span T and work W (for an arbitrarily large number of processors), it can be run on a PRAM with p PEs in time $O(T + \frac{W}{p})$ (and using $O(W)$ work).

Proof: schedule parallel steps in round-robin fashion on the p PEs.



⇝ span and work give guideline for *any* number of processors