1 Parallel computing

Doing things concurrently ≅ more complex than sequential programming. Central questions:

- BASICS: How do we split up tasks?
- BASICS: How do the different parts communicate?
- PERFORMANCE: How much time does it take to run the application? (User's question)
- PERFORMANCE: Is the system fully utilized? (Adim's question)

A few important termas are:

Computational Model:	Model that can be solved with a computer
Model of Computation:	The description of how to solve a Computational Model with a computer
Concurrent:	Parallel / At the same time

Typical steps in development are:
Specification →Design →Implementation →Adaption to system →Adaption to Machine

1.1 Invisible Parallelism

This is parallelism that we don't see while programming like:

- In compiler for the hardware and the OS
- Pipelining
- On the Bit-/Instruction-/Memory-Level

1.2 Amdahl's Law

Describes how much a program can be sped up:

$$S=\frac{1}{s+(1-s)/p} \geq \frac{1}{s}$$

S:	speed up
s:	time used for sequential part
1-s:	time used for parallel part
p:	number of processors

1.3 Granularity

- Splitting up processes into pieces
- How can they communicate? How can I combine them to run on 1 processor?
- How small can and should I make them?

1.4 Data Locality

- How to minimize data retrieval time?
- How to keep data as local as possible?

1.5 Load Imbalane

- Trying to balance load across processors
- Balancing the load of regular and irregular processes

2 Parallel Algorithms

How do you identify parallelism?

- Look up algorithms
- Look at code
- Look at data structures

2.1 Design

1. Decompose (Machine Dependent)
   - What level of parallelism? (static / dynamic)
   - How many processors?
   - How big is the problem?
2. Clustering (Machine Independent)
   - How to bring tasks together?
3. Implement (Machine Dependent)
4. Map work to processes (Machine Dependent)
   - Doing it static / dynamic?
   - Being aware of data locality and possible load imbalance.

2.2 Memory

There are different kinds of memory:

Distributed:	Programmer has to manually communicate data. This is most common.
Shared:	Uniform location of data →easier for programmer. The bus can get overloaded here.
Virtual Shared:	Is distributed but pretends to be shared.

2.3 Solving Concepts

1. Farmer Worker
   - Farmer organizes and schedules
   - Workers do work
   - Farmer can push work to workers (Push-Model)
   - Workers can pull work from farmer (Pull-Model)
2. Divide & Conquer
   - Recursive split up (for example)
   - Hardest part is the Administration
   - First divide problem then conquer subparts
3. Data Parallelism
   - When data is splitable
   - Different nodes work on different parts of data
4. Task Parallelism
   - Different tasks for different nodes
   - Task can start when input is available (data dependent)
   - Task can start when its output is needed (demand-flow)
5. Bulk Synchronisation
   - Every part waits at a certain point for every other part and continues together

2.4 System Level Communication

With Shared Memory

- Mutual exclusion per data (Only one can access)
- (Un)locking mechanisms put in place
- Bariers put in place where processes wait

With Distributed Memory

- Communication has to happen
- After sending data, it is duplicate →which is the original?
- General structure of send command: SEND(Buffer*, element-count, destination)
- General structure of receive command: RECEIVE(Buffer*, element-count, owner)