Parallelism

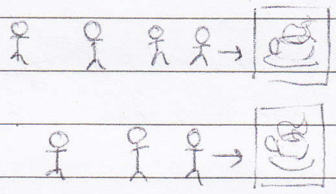
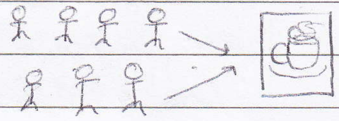
Concurrent prog = parallel prog ≠ sequential prog.

Definitions

Sequential prog: a set of tasks which are executed one after the other.

Parallelism: at least two tasks are executing simultaneously

Concurrency: at least two tasks progress ~~[illegible]~~ at the same time. (not always ~~at the~~ simultaneously)

Parallelism (hardware level)	Concurrency (programming level)
	

Why would we use parallelism?

Going faster and going larger

↳ Processing data faster

↳ distributed computing

↳ Processing more data in the same amount of time

Tools

- Multi-core / multi-processors / shared memory system
- Graphic cards / GPU
- Multiple machines

Flynn Taxonomy (1966)

Classified all machines (tools) depending on wether they go larger② or faster①

1: instructions   2: data

in the same time span	Single instruct°	Multiple instruction
Single data	SISD unicore processor	MISD pipeline, fault tolerance system
Multiple data	SIMD vector processor	MIMD cluster of computers, GPU

Note: GPU = Graphical Processing Unit

GPGPU = General Purpose GPU.

Another taxonomy: depending on the location of data:
- Shared memory system: all data & instructions are accessible directly
- Private memory system: data & instructions are located in ≠ address spaces.

Let's discuss pipelines (in CPUs)

Basic steps in processing instructions

The great CPU war
RISC vs CISC

1) fetch : take instruction from memory
2) decode : figure out which operations to perform (instruct in mem are numbers.
3) execute :
4) write back

note: when a data is being decoded, another one is being fetched, so we have 4 instructions being processed at the same time.

	F	D	E	W
t_1	i1			
t_2	i2	i1		
t_3	i3	i2	i1	
t_4	i4	i3	i2	i1
t_5		i4	i3	i2
t_6			i4	i3
t_7				i4

A pipeline does not reduce the time for 1 instruction to be processed. It increases the IPC by grouping threads.
The more steps you have, the more efficient it is.

Hazards:

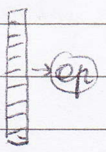
Things that happens that breaks the benefits of having a pipeline
1) structural hazard: 2 or more stages require the same part of the CPU.
exemple: integer processing in fetch and execute. Solution: have several integer processing units
2) data hazard: data dependency - i2 needs the result of i1
solution: enter a bubble instead of i2 for i2 to wait that the result of i1 is written. Not the best because it slows the pipeline
• reorder the instructions to avoid the situation. If i3 is independent of i1 and i2, insert it instead of the bubble.
3) control hazard: if you have this kind of problem: test + jump (if i2 else i3) you can't know if you need to enter i3 or i2 after.

Parallelism

3) Control hazard

solution: speculating (branch prediction)
the CPU takes a guess and executes i3 or i2, and if it ends up being wrong, it will flush the data and start over. It is right about 90% of the time.

Vector processor (let's discuss...)



Benefits: - you only have to fetch and decode 1 instruction
- caching data is made easier because the fetch is made by chunks

Moore's Law

The number of transistors in a CPU can be doubled every 24 month (then 18 month).
As there was a direct relation between the performance and the number of transistors, the performance of CPU doubles every 18 month.

$$\begin{tikzcd}
 & \text{Put more transistors} \arrow[r] & \text{not enough space on CPUs} \arrow[d] \\
 & \text{there is a limit to the speed} & \text{make bigger CPUs} \arrow[l]
\end{tikzcd}$$

at which instructions travel from one end to another of the CPU.
→ tunnel effect (electrons jumps above resistances)
→ heat and power consuption

" Put more things, it goes faster. "

You can increase the frequency, but it did not scale either.
We moved to multicore to avoid these problems
" Free lunch is over " → period of time where you had to actually make the effort to optimise your code instead of waiting for the faster next generation of CPU.

Writing parallel / concurrent code

- Low level: close to the machine (C...)
- High level: what even is a machine? (java...)

Tools

- Low level: Threads, locks, semaphores, monitors
- High level: Bag of tasks, map-reduce...

Semaphores (Dijkstra)

Low level tools to allow parallelisation.
Abstract type made of a value (usually integer) and 2 operations: P() and V().

```
-P() : if value ≤0, block;
       otherwise, value--;
-V() : increase value;
       ~~if value becomes~~
       wake-up blocked processes;
```

Thread 1	Thread 2	Semaphore
s.P()	s.P()	s = 1
x++;	x--;	
s.V()	s.V()	

P() takes the semaphore, V() gives it back.

Basic mistakes: s=0 → code crashs, no one can take the sem.
fail to give back the semaphore
s=2: dangerous but could work

note: with s=1, it is called mutual exclusion semaphore.

You can lock only the parts of the code that you need to be secure.

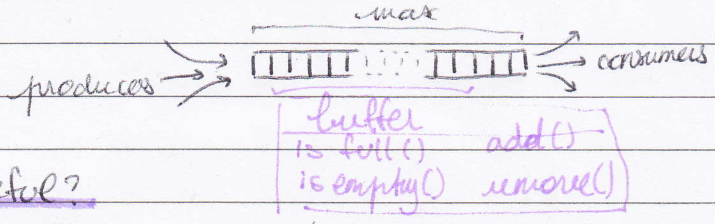
But the operations performed by P() and V() can be interrupted as well, HOW DO WE DO? We need help from the CPU. It gives a "test and set" that cannot be interrupted.

Parallelism

Programming model: producer-consumer

aka (read - write) has a max size

- 1 shared data structure (array);
- A set of producers which add data to the structure;
- A set of consumers which remove data.



1) when is this pattern useful?

When : production speed ≠ consumption speed

↳ you can increase the number of processes on one side or the other to have the data consumed as it is produced

2) A producer can put something in the array if there is room in it.

3) A consumer can consume iff the array is not empty.

```
Producer:                      Consumer
while (true)                   while (true)
      produce();                     consume();
```

wrong Solution: sem. s := 1

If there is no parallelism:

```
produce () {        s.P()          consume () {  s.P()
    if (! b.isfull()) {                if (! b.isempty) {
        b.add(.)                           b.remove();
    }}                                 }}
  s.V()                                s.V()
```

If there is parallelism AND all buffer method are atomic

↳ they can't be interrupted.

⚠ a serie of atomic instructions is not atomic.

for next week: find the right solution

hints: 2 semaphores, use MAX as a value for one of them