19 nov

Parallelism

next week: lab
take computer

homework

Sem full = 0
Sem free = max

```
produce() {
    free.p();
    add;
    full.v();
}
```

```
consume() {
    full.p()
    remove();
    free.v()
}
```

How are exam graded
- is every semaphore's p & v called?
- every semaphore has a value?
- does it solve the problem?

exercice & example

```
for
from(i from 1 to N) {
    v v[i] = a + b.v[i]^2 + c.v[i]^3 + d.v[i]^9
             + e[i]^5 + + [i]^6 + g[i]^7
}
```

How to parallelize this code?

1) Divide the computation into small sub-computations

$P_0 = b.v[i]$
$P_1 = c.v[i]^2$
⋮
$P_7 = g.v[i]^7$

note: P_n = processor n

problems: - work is unbalanced between the processors
- memory access to v[i] can be a problem as well

We could divide our work differently:

```
for i from 1 to N {
// v v[i] = a + b.v[i]^2 + c.v[i]^3
//          + v[i]^4 x (d + e.v[i] + f.v[i]^2 + g.v[i]^3)
    Do in parallel {
        x = a + b.v[i]^2 + c.v[i]^3
        y = v[i]^4
        z = d + e.v[i] + f.v[i]^2 + g.v[i]^3
    } end
    v v[i] = x + y x z
}
```

// = divisions

should be done in parallel

→ control parallelism: relies on the instructions to see what can be //.

Algorithmic complexity of this loop: $O(N)$ for the first
$O(N)$ — second
on 3 processors

2) We take N processors

$P_0 \to vv[1]$
$P_1 \to vv[2]$
⋮
$P_{N-1} \to vv[N]$

Complexity: $O(1)$ on N processors

from i from 1 to N in parallel
$vv[i] = \dots$

This is called Data parallelism
You do not look at the program but at the array of data.

3) Horner sequences

$vv[i] = a \times v[i](0 + b v[i] c[i]^2 + d[i]^3 \dots)$
$= a \times v[i](0 + v[i])(b + c[i] + d \cdot v[i]^2 \dots$
⋮

$O(N)$ on 7 processors
This is a pipeline

↳ [g * v[i]] → [f · v[i] + ○] → ⋯ → → [vv[i]]

If we don't have N processors but only $p \ll N$ procs.

vv[ ] : ⟷ N

each (was processor will have to compute N/p cells

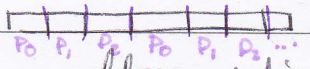
In data parallelism, how do we associate some data to a processor?

① P_1 P_2 P_3 P_4 — packs of consecutive cells
bloc distribution
⊖ could be unbalanced

② P_0 P_1 P_2 P_3 | P_0 P_1 P_2 P_3
cyclic distribution
⊖ you are messing with the cache

Parallelism

③



more than 4 parts

bloc cyclic

you can play on the size of blocs to optimize for the cache while reducing the probability of having unbalanced computations.

note: look at map-reduce algorithm

P-RAM

Parallel Random Access Machine

- It has a set of processors. $P_1, P_2 \dots P_N$, and each processor knows its id.
- Each processor has a private memory and some registers. (Program counter is THE register we can't do without ! → address of the next instruction to process)
- All processors have access to an unlimited shared memory.
- All processors have "classical" instructions +
  - halt: stops the processor
  - fork: starts a new program on a free processor
- Each instruction in the P-RAM takes 1 unit of time
- The execution model is synchronous: all processors performs 1 instruction at the same time (1 step in P_1 takes 1 step in P_2): lockstep
- Read-Write conflicts

  it depends (concurrent reads or write)

	Read	Writ
Concurrent	CR	CW
Exclusive	ER	EW

ER: No more than 1 access to a memory address

CR: Multiple accesses are allowed at the same time

EW: Exclusive write

CW: Allowed when all processors write the same value

other conditions: (1) consistency) →

(2) arbitrary: only one value will be written, but we don't know which one.

(3) priority: a processor has priority over others.

④ Fusion : applying a function to all values and the written value is the result. (Max, +, ...)

4 main types of P-RAM:

EREW — for the rest of this course,
CREW — we will study algo on these 4 P-RAMs
ERCW | 4 subtypes
CRCW |

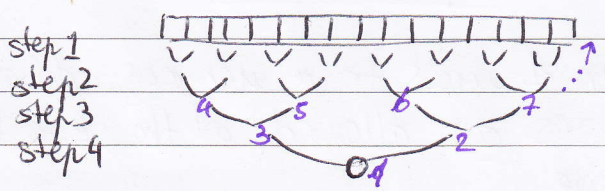
A parallel algorithm is defined as a couple (algorithm, P-RAM), P-RAM being the most restrictive.

Exercice : Max Value on P-RAM
input : array of size $n = 2^m$
output : max value

1) Sequential version :

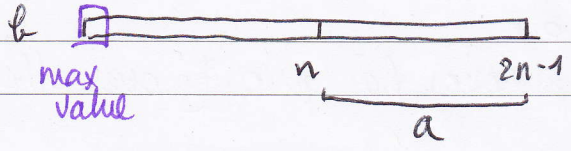
V1
```
max = a[1];
for i in a {
    if a[i] > max {
        max = a[i];
    }
}
return max;
```

V2
```
for l from m-1 to 0
    for j = 2^l do 2^(l+1) - 1 in parallel
        b[j] = max(b[2j]; b[2j+1]);
```



How to represent a tree as an array?
1) number the nodes from ~~the~~ 1 from the root, breadth first
2) given a node j :
- childs are $2j$; $2j+1$
- parent is $\lceil \frac{j}{2} \rceil$.

Size of b : (b > a)
b has a size of $2n-1$



Parallelism

On which P-RAM does it work?

→ No processor writes in the same cell
→ No processor reads in the same cell
↳ so we need an EREW P-RAM.

Complexity : $O(\log(n))$ on $\frac{n}{2}$ processors
↳ $O(n)$ (big O notation)

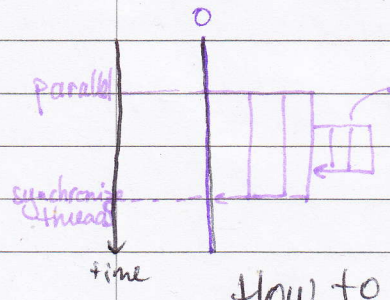
For next week

1) ↳ complexity of the sequential version of the algorithm
2) You are free to use the P-RAM you want, what is the fastest algorithm to compute the max value?

For lab session

OPEN MP → library (ext. of the compiler)
↳ indicate where shit's parallel and it deals with threads for you.

An open MP program is a single process with multiple threads (simulates a P-RAM)
Threads are created when encountering a parallel region.
The number of threads is decided at ~~one time~~ runtime by OpenMP, but can be set by developpers.
Each thread has an id and the master thread has an id of 0.




```
#pragma omp parallel for
for(i=0;i<10000;i++){
    a[i]=b[i]+c[i]
}
```

How to compile?

```
gcc -o test-para test.c -fopenmp
```

(parallel)

install gcc : Linux Subsystem
(installed last ~~year~~)