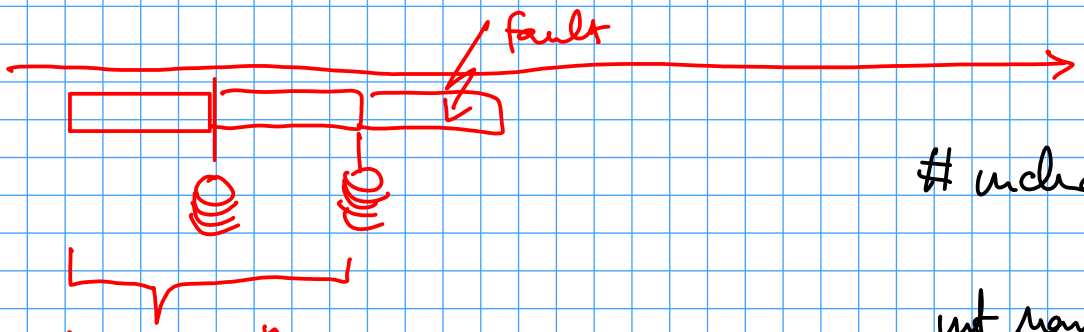


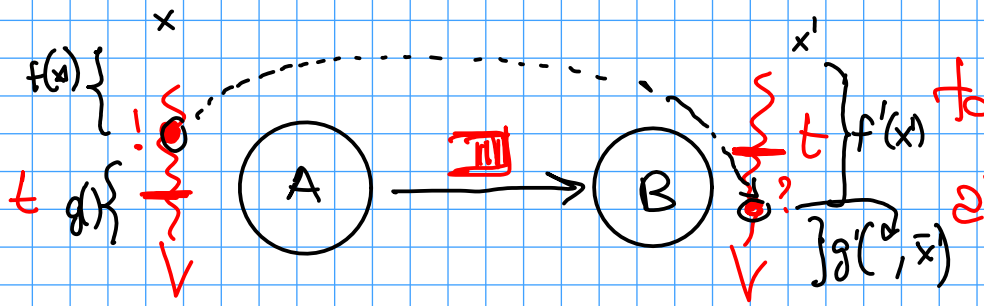

```
#include

int main( ___ ){

  for(i = ... ){
    <reading (disk)>
    <computations>
    updates (x[] y[])
    write (to disk)
  }
  do something with x, y
```

i ⟶

x, y ⟶

checkpoint ⟶

restore starts here

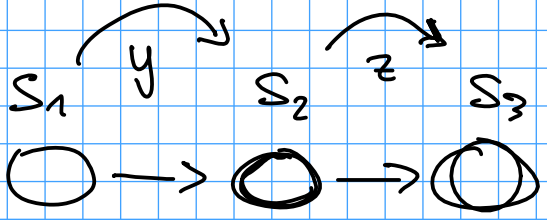


to checkpoint:

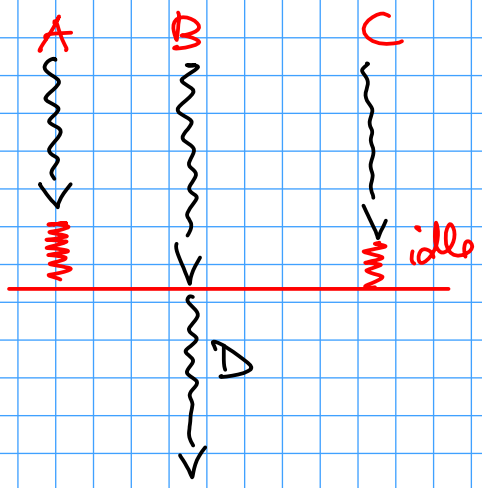
a) status A & B / all the C.A.

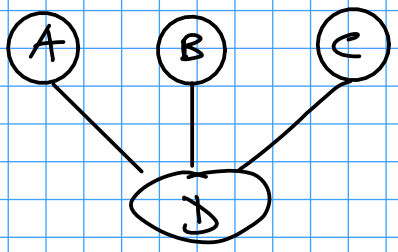
b) status of the concurrent computations (as going communicates)

A, B run in a thread




```
while (true) {   ___ ? y ;   ~~~ z = f(y) ;   ___ ! z }
```





power management (!? NF concern)

CPU

Throttling

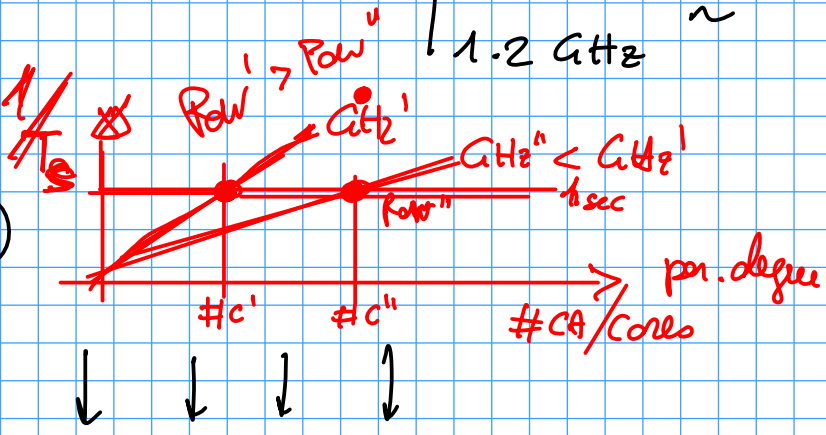
clock { 2.4 GHz, 1.2 GHz }

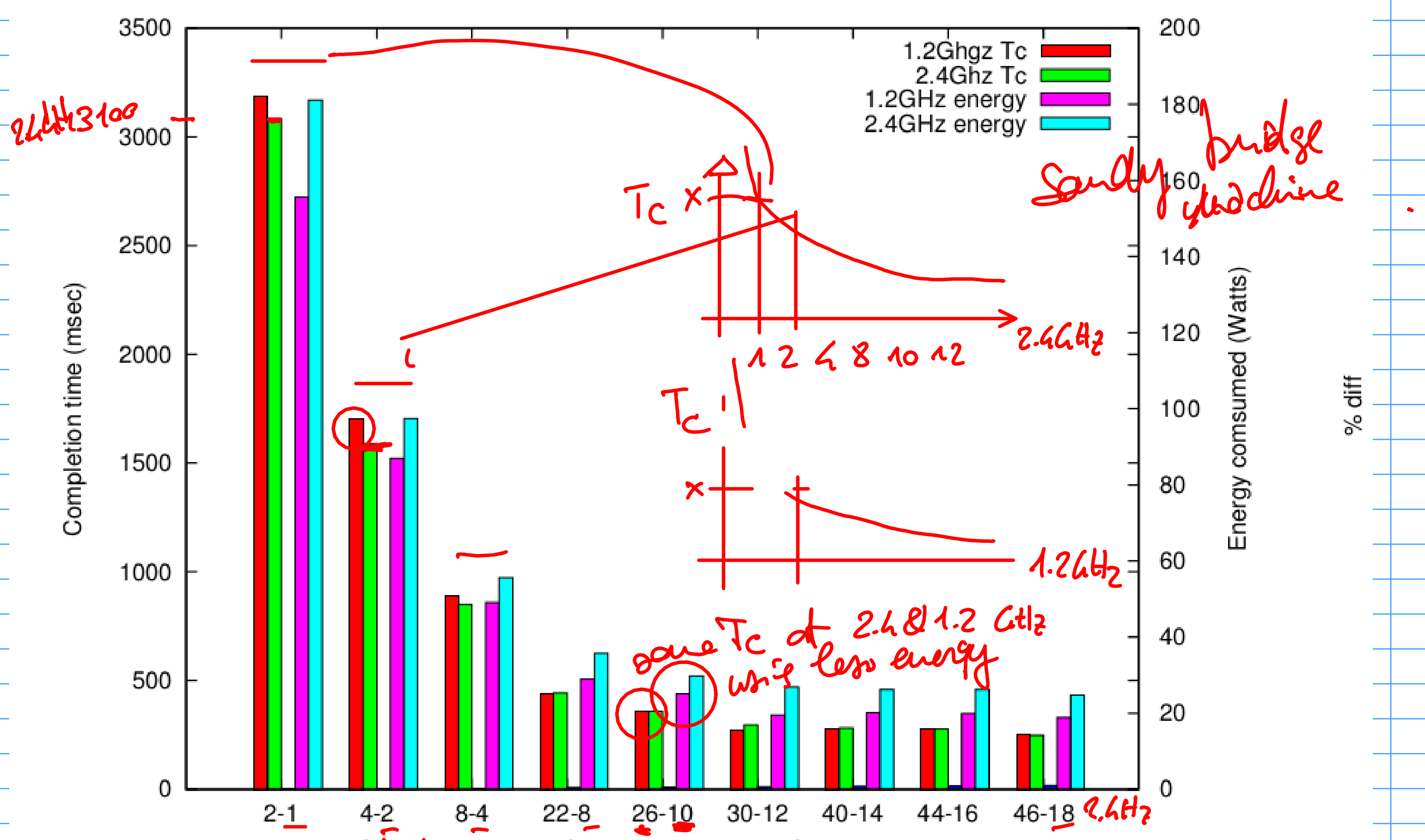
2.8 GHz ~ 1.6 ~

large number of cores

16 (2× hyperthreading)

CA_1 .... CA_n $n \gg 16$





suitable for

data parallel | any parallel

512 ÷ 1024 core
flop/cycle

GPUs vs CPUs

higher | lower

FLOP / WATT

completely different

8 ÷ 16 cores
2 hyperthreading ⇒ 16 ÷ 32 contexts

8 ÷ 16 FLOP/cycle

$\forall x_i$ $y_i = f(x_i)$

↑

less no conditional branches inside
or the branches go the same way $\forall i$

GOOD CODE $y_i = x_i * 3 + x_i^2$

BAD CODE if (x_i % 23) == 0)
then ___
else ___

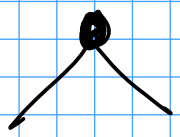
```
for(i=0; i<N; i++)
    x[i] = y[i] * z[i];
```

embarrassingly data parallel

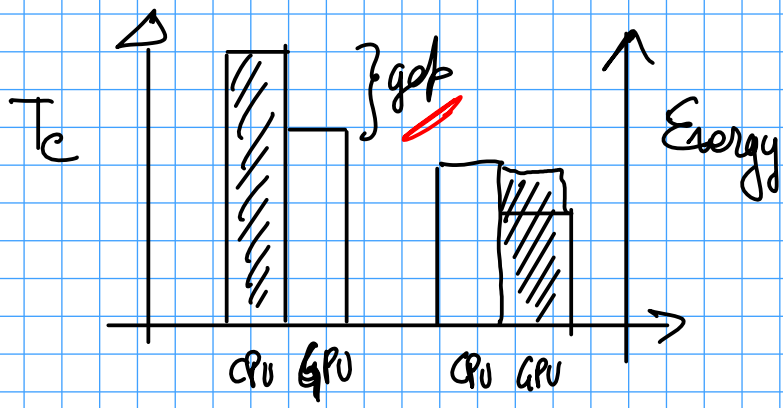
⇓

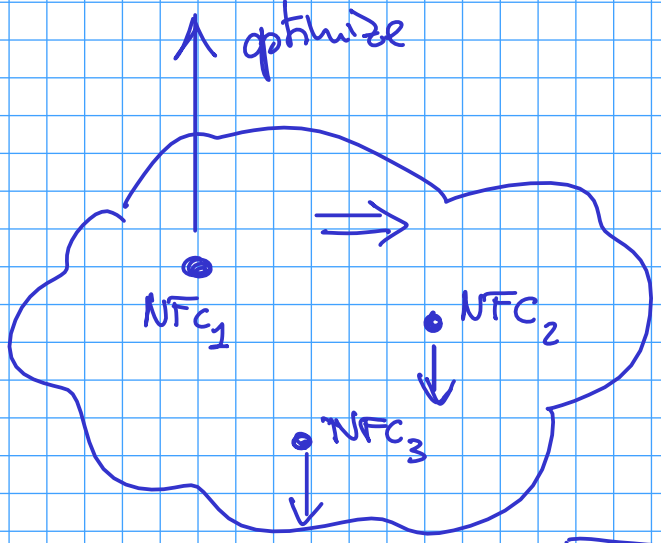
good for GPUs !

implement a CPU (threads)



implement a GPU





NEW FUNCTIONAL CONCERNS

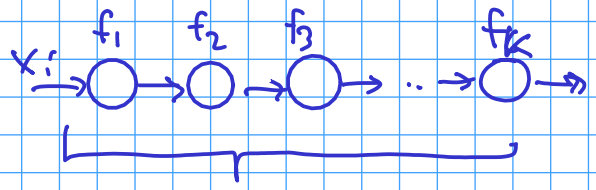
REPARA FP7

⟶ 2.6 M €

STREAM PARALLEL PATTERNS

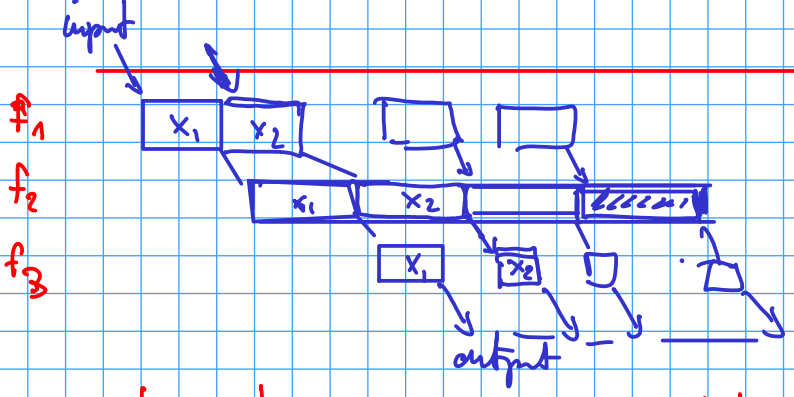
PIPELINE : computations in stages (or streams)

$$x_i \longrightarrow f_k(\dots f_3(f_2(f_1(x_i))))$$



C.A. graph

T_c T_s



m input tasks

$$T_c = m(\text{slowest stage}) + \sum(\text{non slowest stages}) + T_{comm}(\#\text{ of stages})$$

$$T_s = \max\{T_{s_1} \dots T_{s_k}\}$$

with $m \gg k$

T_c is dominated by $m(T_s)$

$$T_{seq} = m\left(\sum_1^k T_{s_i}\right)$$

$$Speedup(k) = \frac{m\left(\sum T_{s_i}\right)}{m\, T_s} \cong k$$

if I assume that stages take more or less the same time to compute a task