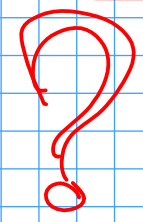
PORTABILITY



PARALLEL ARCHITECTURES

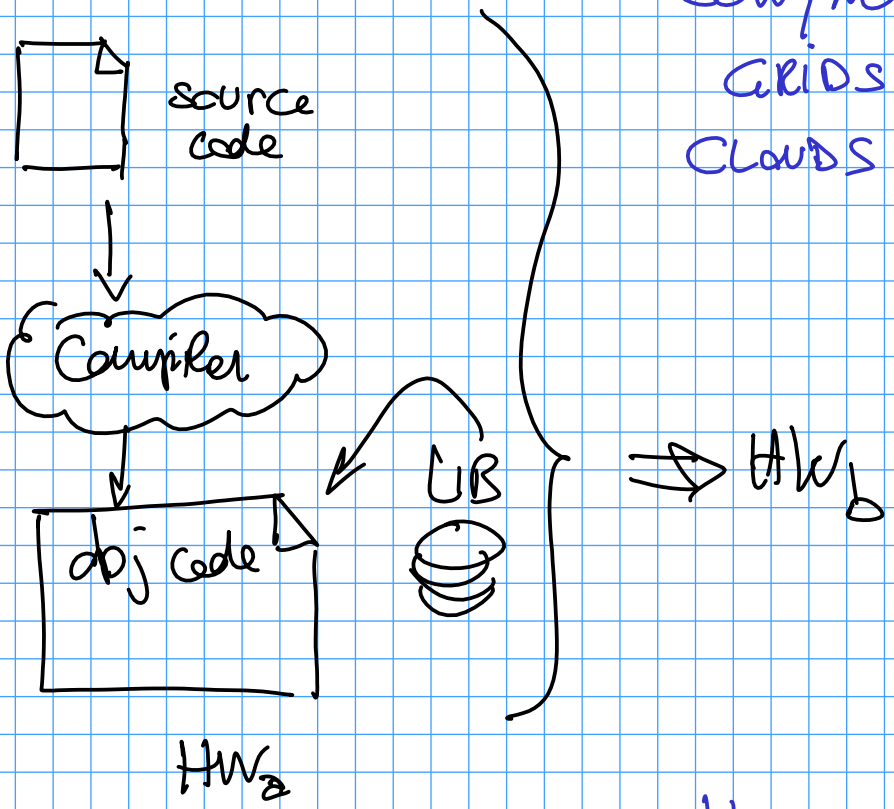
SMP → symetrice multiprocessor (MC)

COW/NOW

GRIDS

CLOUDS

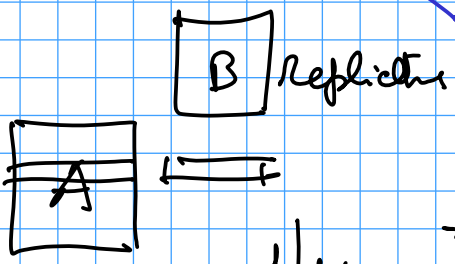
MC → MIC, GPU, FPGA



- functional portability
- performance portability

Example : MM

- HW_a — MC (SMP)
- HW_b — COW/NOW



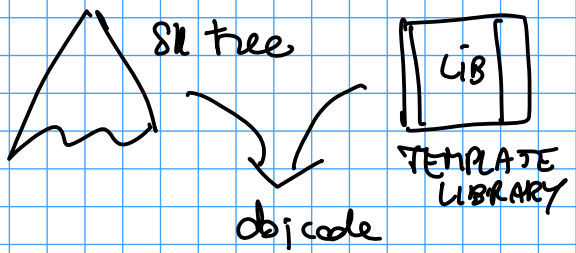
$\Rightarrow HW_b$ broadcast (B)

need to "adapt" my program

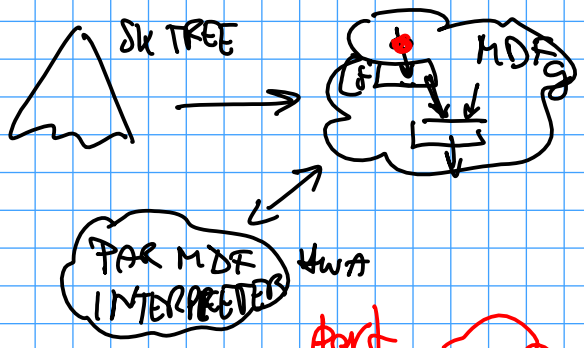
Functional | Performance

PORTABILITY

TEMPLATES in MDF



$HW_A \rightarrow HW_B$: choosing different template from the library



port

$HW_A \rightarrow HW_B$

HWB

Focus: ⊛ Template performance models

⊛ Rewritings of the skeleton tree (evaluated through performance models)

① diff templates for the same skeleton ② diff SK comp. in the same tree section

① + ② in combinations

① New interpreter as efficient as possible

↳ implement as efficiently as possible the "core" ops of MDF interpreter

→ access to repository

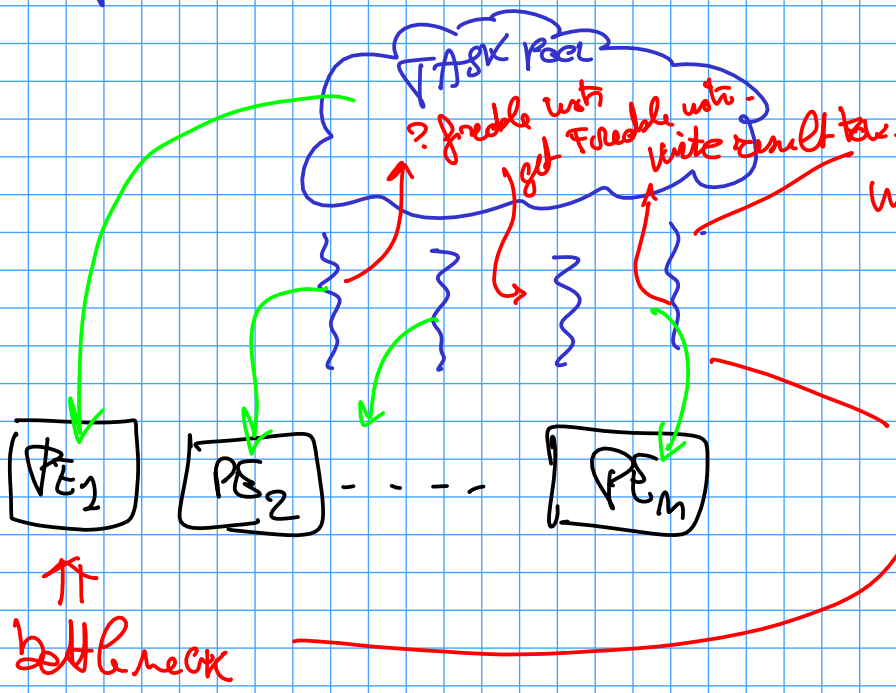
→ fireable instruction "list"

→ executing fireable instructions

MDF → CoW/NoW

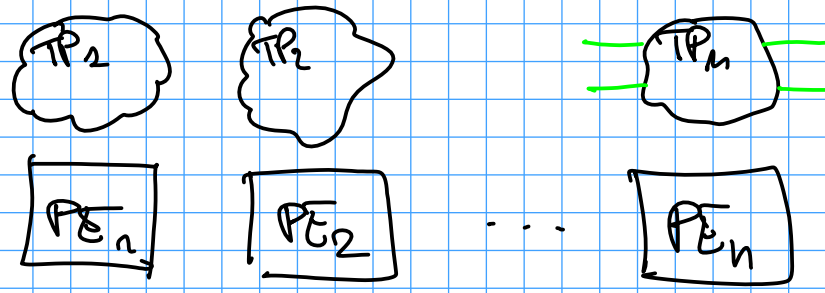
problem: implementation of the MDF instruction repository

TASK Pool



"Software" RING connecting a number of "local" TASK Pools such that they like a single global pool.

INIT: spread MDF graph over the TPi



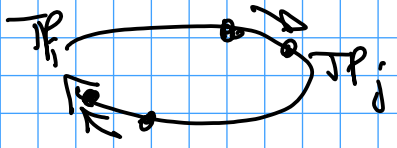

```
while (get fireable ()) {
    compute; store results }
```

"water" ≡ fireable instruction

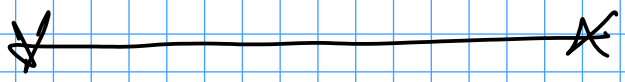
TP manager::

if (# fireable instruction < LOW. W. MARK)
    ask fireable instructions from neighbour TPi
if (# fireable instruction > HIGH. W. MARK)
    start answering fireable instructions requests from other TPi
    answer local ops if possible

+ 2 RINGS



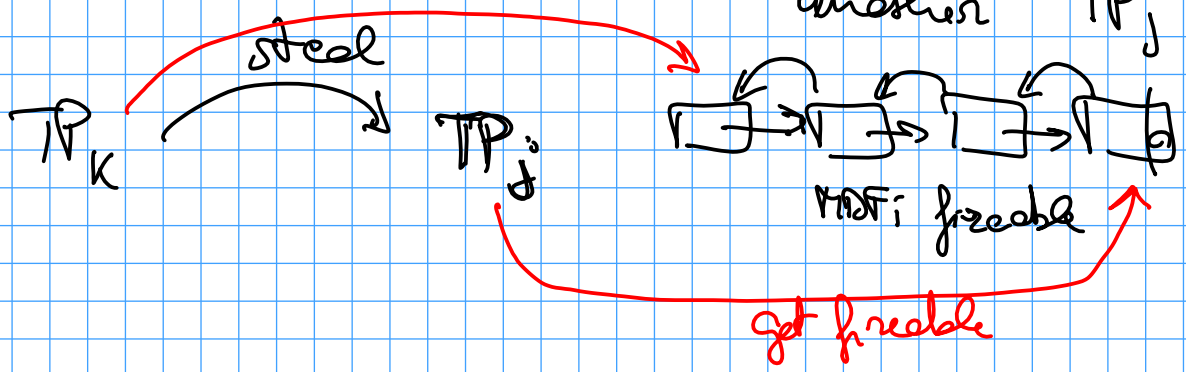
COW (16 nodes, single core, with TCP/IP)

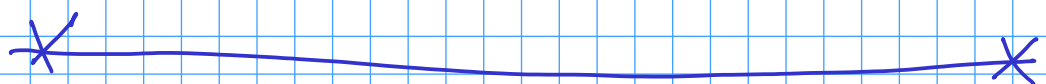


WORK STEALING

if TP_i is requested to deliver a fireable instruction with NO fireable instruction in TP_i:
then try to steal a fireable instruction from another TP_j

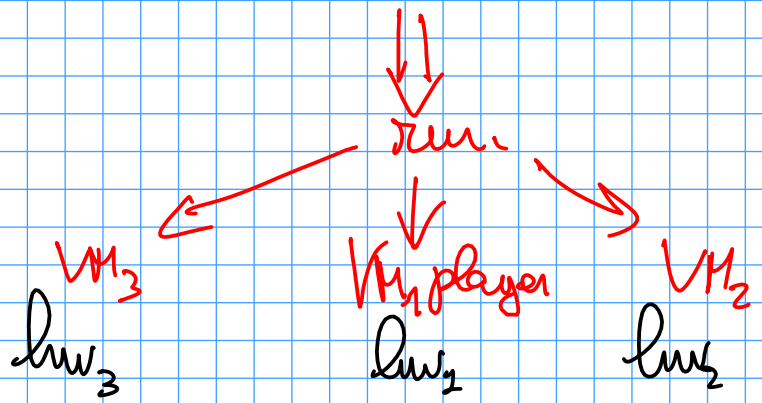
(random victim selection)





Portability through virtual machines

Skeleton program ⟹ compile ⟹ VM object code



JVM

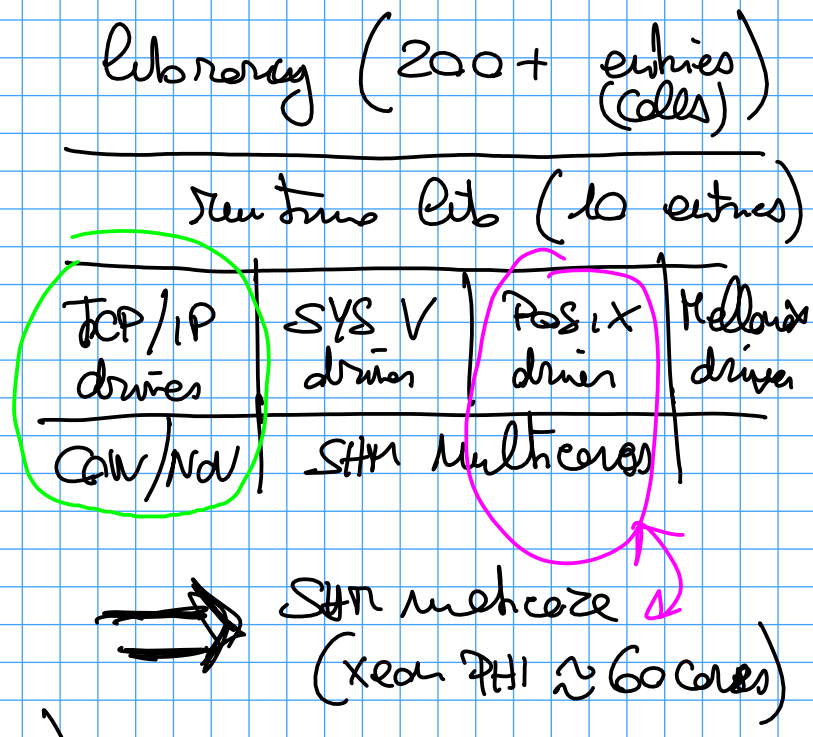
executes bytecode

what happens to performances?

MPI

provides a set of comm. instructions

drivers for different hw



MPI program written for Eth + TCP/IP

↳ asynch comm

SHM multicore (Xeon PHI ≈ 60 cores)

```
hdl = start_send(...)
compute something else
if (check(hdl) == completed) go on
else wait(hdl)
```

OpenMP

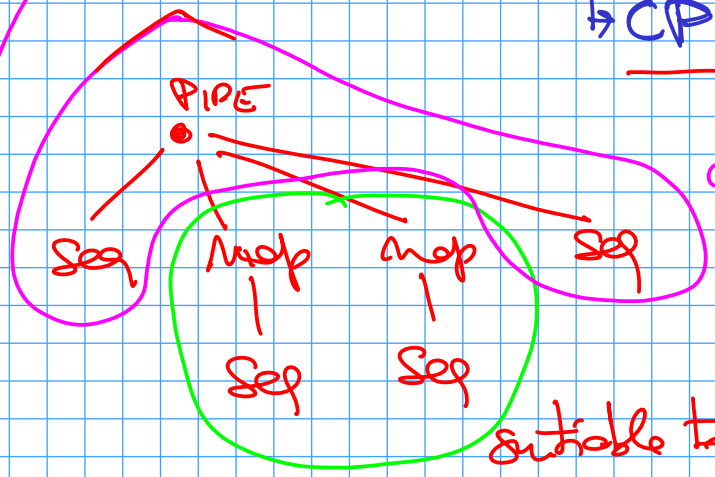
hw1 shared M

hw2 shared M

```
#pragma omp parallel for (....)
for(....) { ... }
```

PORTABILITY → heterogeneous architectures

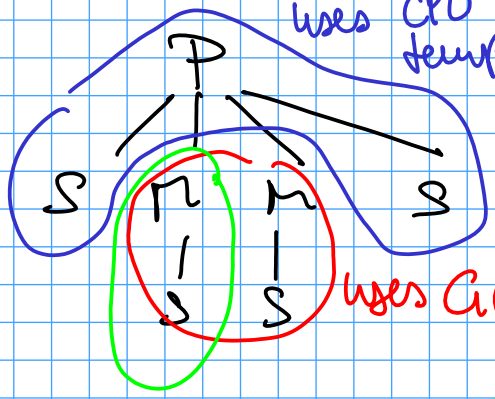
→ CPU + GPU



only suitable to target CPU cores

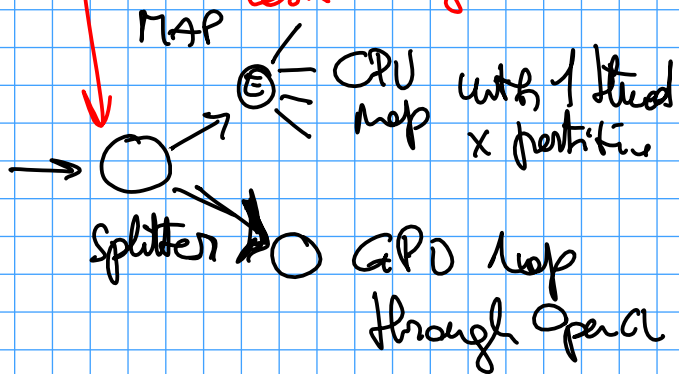
suitable to be executed on GPU cores

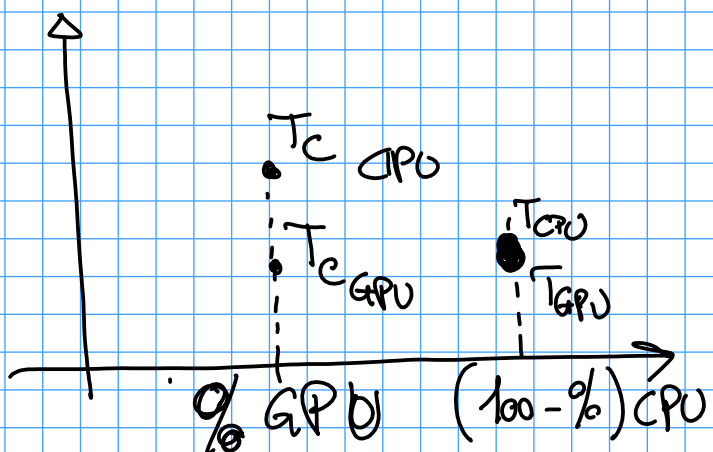
targetability : use techniques supporting portability to support usage of different target hw at the source time !



needs to somehow know to split the overall task among CPU & GPU

hw: i3 + GPU Intel (openCL)





#task

$T_{C_{CPU}}(\#t) = f(\cdot)$

$T_{C_{GPU}}(\#t) = g(\cdot)$