INTRODUCTION À LA PROGRAMMATION PARALLÈLE

CYCLE DE FORMATION HPC@LR
6 NOVEMBRE 2018

BAPTISTE CHAPUISAT

UNIVERSITÉ DE MONTPELLIER

LOI DE MOORE

- Énonce que la puissance double tous les 18 mois.
  - Vrai jusqu'au début des années 2000.
  - Les cœurs actuels ont une puissance 20 fois inférieures à la prédiction de la loi de Moore.

LOI D'AMDAHL

$$T = (1 - p)\,T + pT$$

- *p* est la partie du programme pouvant bénéficier des améliorations technologiques.

LOI D'AMDAHL

$$T = (1 - p)\,T + pT$$

- *p* est la partie du programme pouvant bénéficier des améliorations technologiques.

$$T_n = (1 - p)T + \frac{p}{n}T$$

LOI D'AMDAHL

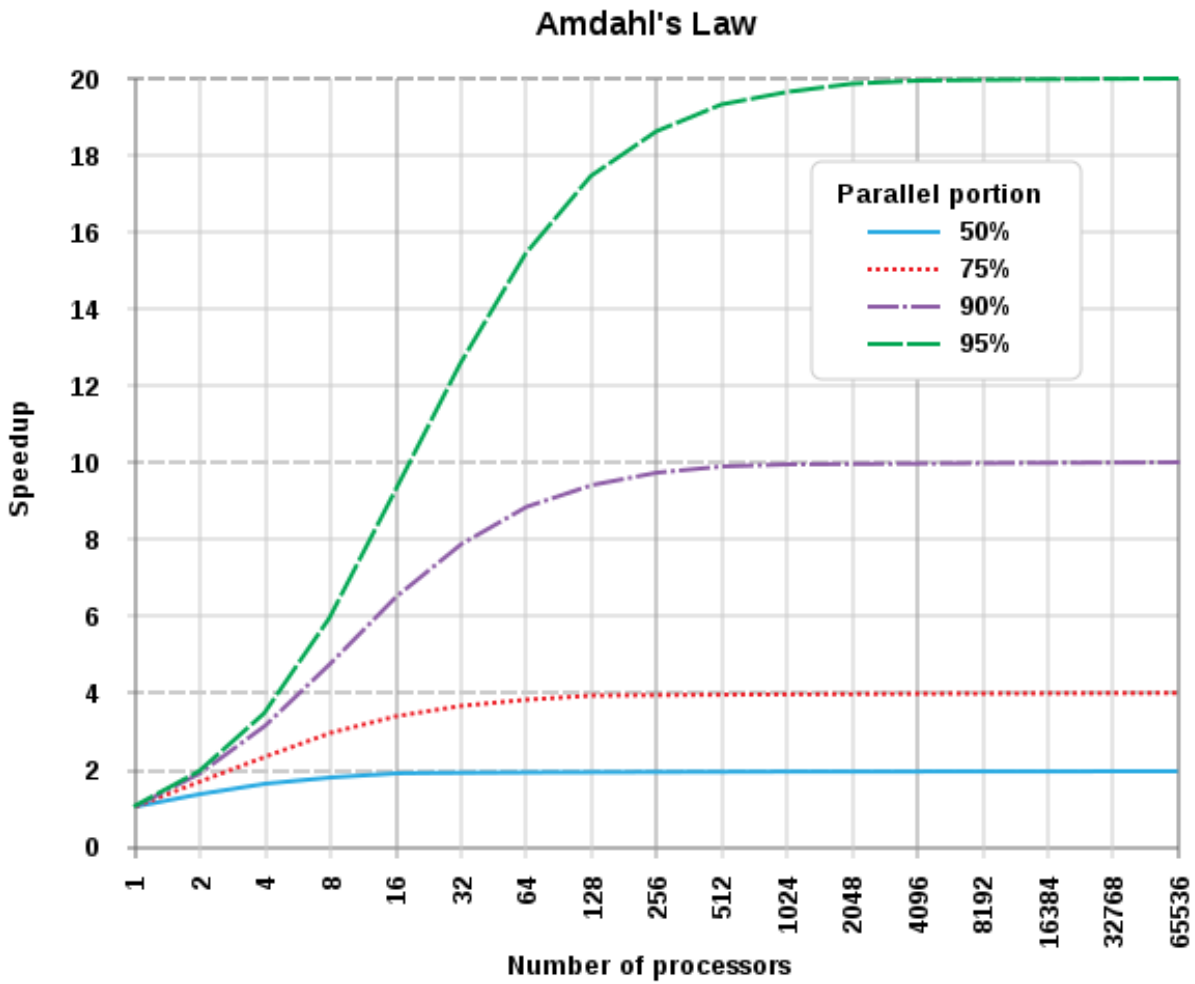
$$T = (1 - p)\,T + pT$$

- *p* est la partie du programme pouvant bénéficier des améliorations technologiques.

$$T_n = (1 - p)T + \frac{p}{n}T$$

- Accélération

$$\frac{1}{(1 - p) + \frac{p}{n}}$$


Amdahl's Law
Speedup
Number of processors
Parallel portion
50%
75%
90%
95%
0
2
4
6
8
10
12
14
16
18
20
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536

LES DIFFÉRENTS TYPES DE PARALLÉLISATION

- Les architectures
  - Les clusters de calcul (OCCIGEN, Turing, Muse)
  - Les grilles (France Grille)
  - Les accélérateurs (XeonPhi, GPU)

- Les méthodes de parallélisation
  - Le calcul réparti
  - Le calcul en mémoire partagée (Multithread)
  - Le calcul en mémoire distribuée (MPI)

LE CALCUL RÉPARTI AVEC SLURM

- Exécuter un programme séquentiel plusieurs fois sur des jeux de données différents => Les processus sont indépendants.

- L'option job array

  -a, --array

  Exemple : --array=0-27
  --array=0-59%28
  --array=1,2,5-10
  --array=0-99:4

  Variables d'environnements :
  SLURM_JOB_ID, SLURM_ARRAY_JOB_ID, SLURM_ARRAY_TASK_ID

CODE R PARALLÉLISÉ

```
# code parallele
library(parallel)
load('matrice.Rdata')
cl <- makeCluster(10)
parApply(cl, mat, 2, mean)
stopCluster
```

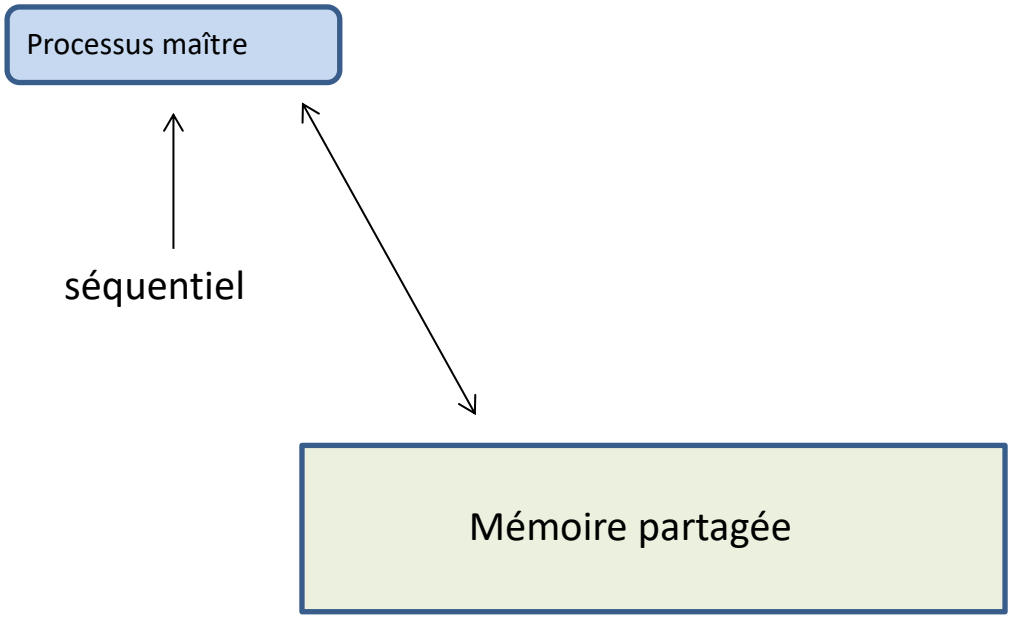
EXEMPLE DE JOB ARRAY AVEC R

```
# On recupere le numero de la tache
idtask <- as.numeric(Sys.getenv("SLURM_ARRAY_TASK_ID"))
# On genere le nom du fichier
nom_f <- paste("fichier", Sys.getenv("SLURM_ARRAY_TASK_ID"), sep=".")
# On recupere la matrice ‘mat’
load('matrice.Rdata')
sink(nom_f)
col <- mat[,idtask]
res <- mean(col)
print(res)
sink()
```

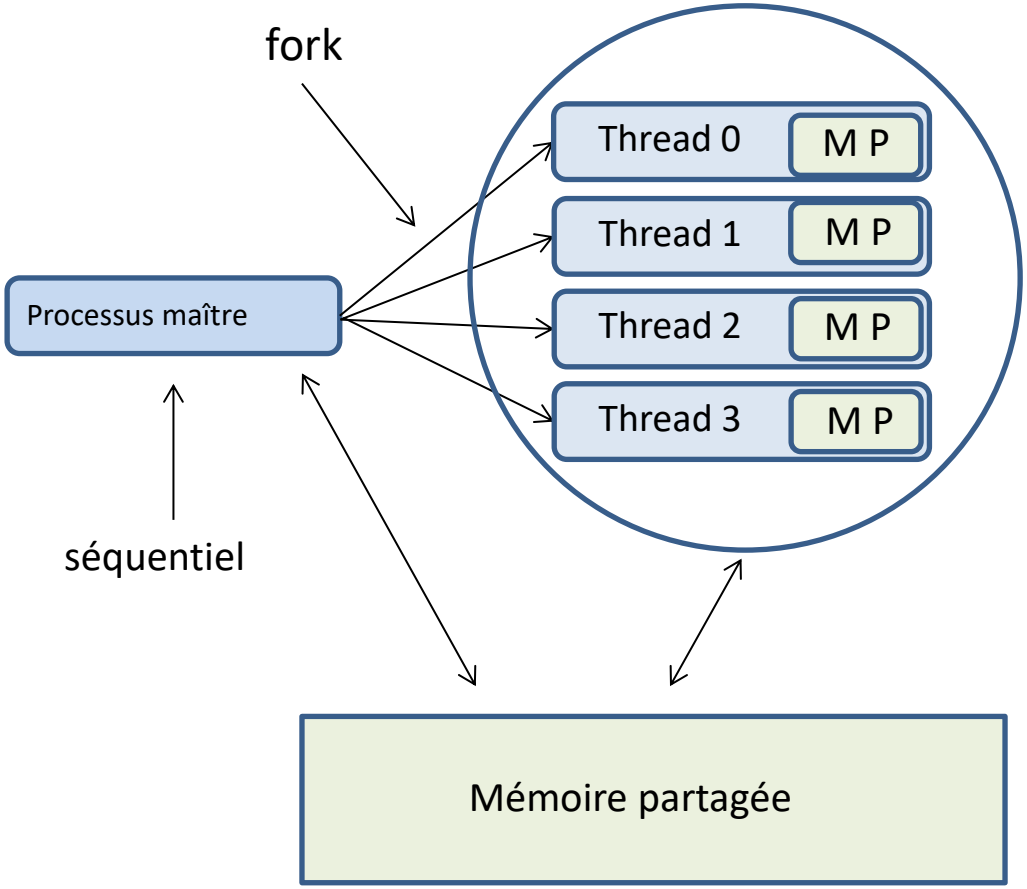
EXEMPLE DE JOB ARRAY AVEC R

```
#!/bin/sh
#SBATCH --job-name=test_array
#SBATCH -N 1
#SBATCH --array=1-10
#SBATCH -o array-%j.out
#SBATCH --account=admin
#SBATCH --partition=defq
#SBATCH --mail-type=ALL
#SBATCH --mail-user=baptiste.chapuisat@umontpellier.fr

module purge
module load cv-standard R

R CMD BATCH mon_prog.R
```

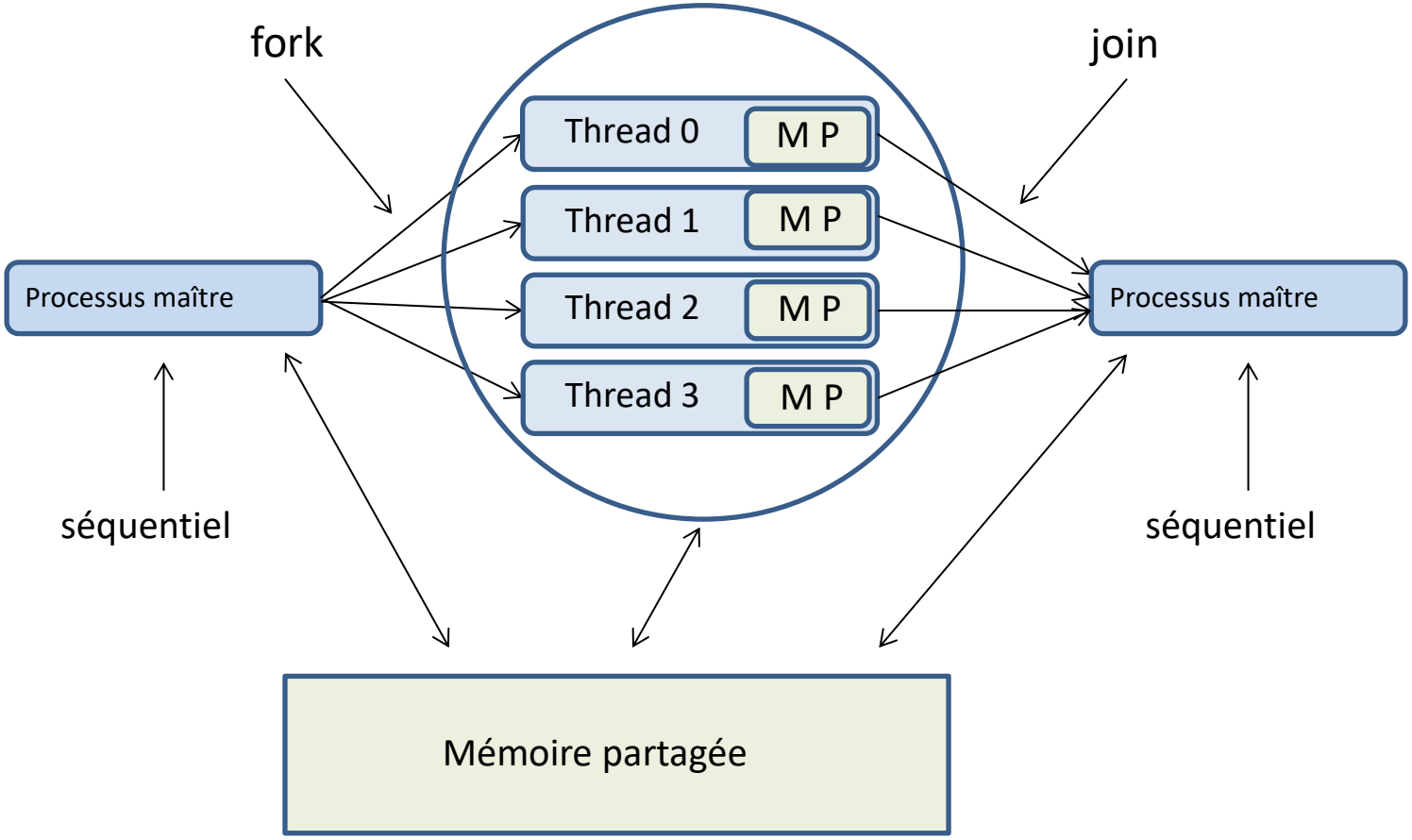
MODÈLE DE PROGRAMMATION MULTITHREAD



MODÈLE DE PROGRAMMATION MULTITHREAD



MODÈLE DE PROGRAMMATION MULTITHREAD



PROGRAMMATION MULTITHREAD

- Il existe plusieurs bibliothèques multithread
  - OpenMP (omp.h)
  - Pthread (pthread.h)
  - bibliothèque propre au langage
    - Ex : library(parallel) en R
- OpenMP est un standard "industriel" supporté par de nombreux compilateurs (gcc, IBM, Intel, Texas Instrument, Oracle, …).
  - Avantage : Simple
  - Inconvénients : Limité dans les langages (Fortran, C, C++). Nombre de processus.

EXEMPLE : PRODUIT SCALAIRE

```
#include <stdio.h>
#define SIZE 256
int main () {
   double sum , a[SIZE], b[SIZE ];
   // Initialization
   sum = 0.;
   for (size_t i = 0; i < SIZE; i++) {
      a[i] = i * 0.5;
      b[i] = i * 2.0;
   }
   // Computation
   for (size_t i = 0; i < SIZE; i++)
      sum = sum + a[i]*b[i];

   printf("sum = %g\n", sum);
   return 0;
}
```

```
#include <stdio.h>
#include <pthread.h>
#define SIZE 256
#define NUM_THREADS 4
#define CHUNK SIZE/ NUM_THREADS
int id[ NUM_THREADS ];
double sum , a[SIZE], b[SIZE ];
pthread_t tid[ NUM_THREADS ];
pthread_mutex_t mutex_sum;
void* dot(void* id) {
   size_t i;
   int my_first = *(int*)id * CHUNK;
   int my_last = (*(int*)id + 1) * CHUNK;
   double sum_local = 0.;
   // Computation
   for (i = my_first; i < my_last; i++)
      sum_local = sum_local + a[i]*b[i];
   pthread_mutex_lock (& mutex_sum );
   sum = sum + sum_local;
   pthread_mutex_unlock (& mutex_sum );
   return NULL;
   }
```

```
int main () {
   size_t i;
   // Initialization
   sum = 0.;
   for (i = 0; i < SIZE; i++) {
      a[i] = i * 0.5;
      b[i] = i * 2.0;
   }
   pthread_mutex_init (& mutex_sum , NULL );
   for (i = 0; i < NUM_THREADS; i++) {
      id[i] = i;
      pthread_create (& tid[i], NULL , dot ,
      (void*)& id[i]);
   }
   for (i = 0; i < NUM_THREADS; i++)
      pthread_join (tid[i], NULL );
   pthread_mutex_destroy (& mutex_sum );
   printf("sum = %g\n", sum);
   return 0;
}
```

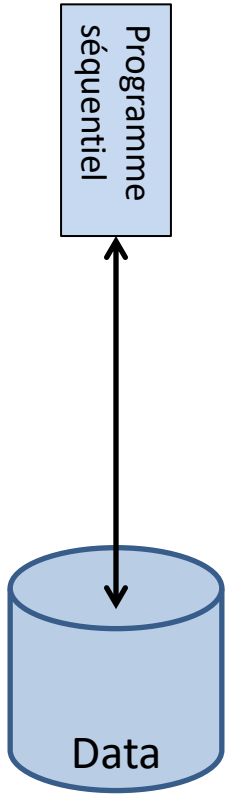
OpenMP

```
#include <stdio.h>
#include <omp.h>
#define SIZE 256
int main () {
   omp_set_num_threads(4);
   double sum , a[SIZE], b[SIZE ];
   // Initialization
   sum = 0.;
   # pragma omp parallel for
   for (size_t i = 0; i < SIZE; i++) {
      a[i] = i * 0.5;
      b[i] = i * 2.0;
   }
   // Computation
   #pragma omp parallel for reduction(+: sum)
   for (size_t i = 0; i < SIZE; i++) {
      sum = sum + a[i]*b[i];
   }
   printf("sum = %g\n", sum);
   return 0;
}
```

EXEMPLE DE FICHIER SBATCH

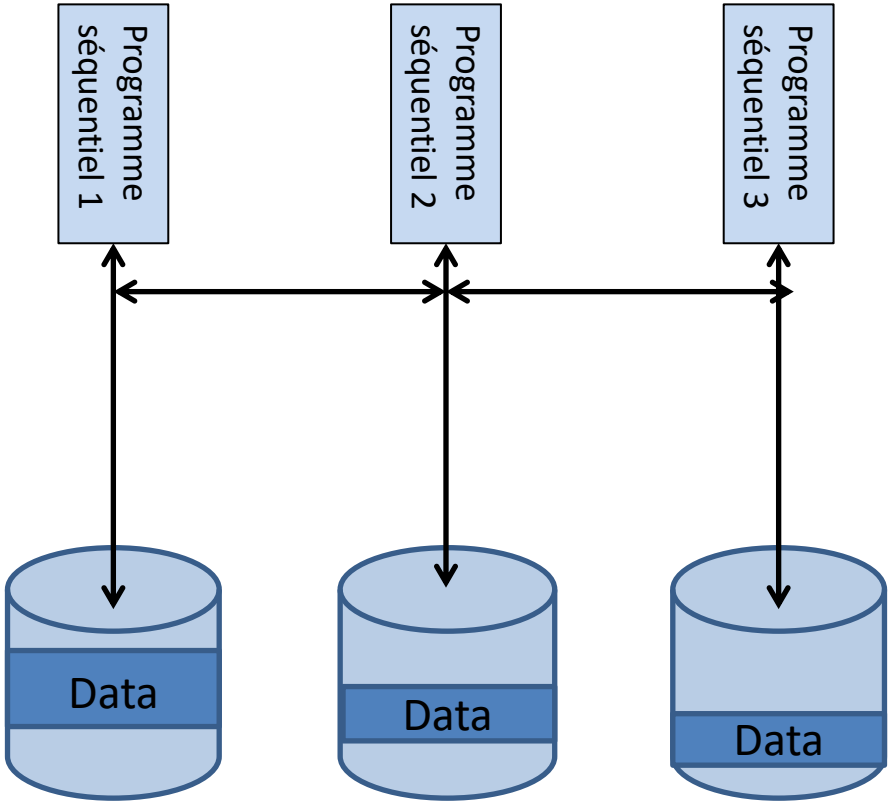
```
#!/bin/sh
#SBATCH -n 4
#SBATCH --account=admin
#SBATCH --partition=defq

export OMP_NUM_THREADS=4

./prog.exe
```

MODÈLE DE PROGRAMMATION EN MÉMOIRE DISTRIBUÉE



MODÈLE DE PROGRAMMATION EN MÉMOIRE DISTRIBUÉE

Échange de messages



MPI (MESSAGE PASSING INTERFACE)

- MPI est une norme. Il existe plusieurs implémentations (OpenMPI, MVAPICH, Intel MPI, MPICH, …).

- Fortran, C, C++
- R (Rmpi), Python (mpi4py)

EXEMPLE DE FICHIER SBATCH

```
#!/bin/sh
#SBATCH -N 2
#SBATCH -n 56
#SBATCH --account=admin
#SBATCH --partition=defq

module load cv-standard openmpi python

mpiexec -n 56 python prog.py
```