Université d'El Oued | Dimanche 07 Janvier 2024

Faculté des Sciences Exactes | 1ème année Master-Informatique

Corrigé type du contrôle de CP

Questions du Cours : **(04 points)**

Q-1) Expliquer le principe de la programmation MPMD ? **(01 points)**

R-1) Dans ce modèle de programmation les processus sont obtenus des codes ou programme différents et s'exécutent en traitant des données différentes.

Q-2) Quelle la différence entre une application parallèle et une application concurrente ? **(01 points)**

R-2) Le premier cas, présente l'exécution simultané des calculs, autant que le second cas, présente la composition des exécutions indépendantes des processus.

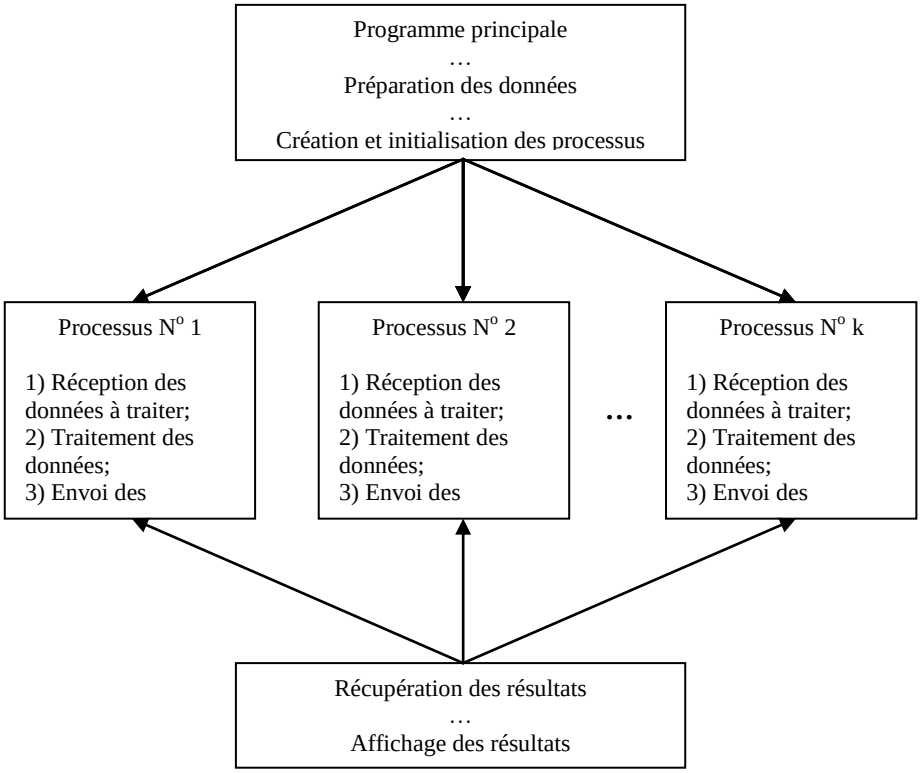
Q-3) Expliquer le principe de l'architecture SIMD ? **(01 points)**

R-3) Un système SIMD est une machine multiprocesseur capable d'exécuter la même instruction sur tous les CPU mais fonctionnant sur des flux de données différents. Les machines basées sur un modèle SIMD sont bien adaptées au calcul scientifique car elles impliquent de nombreuses opérations vectorielles et matricielles.

.

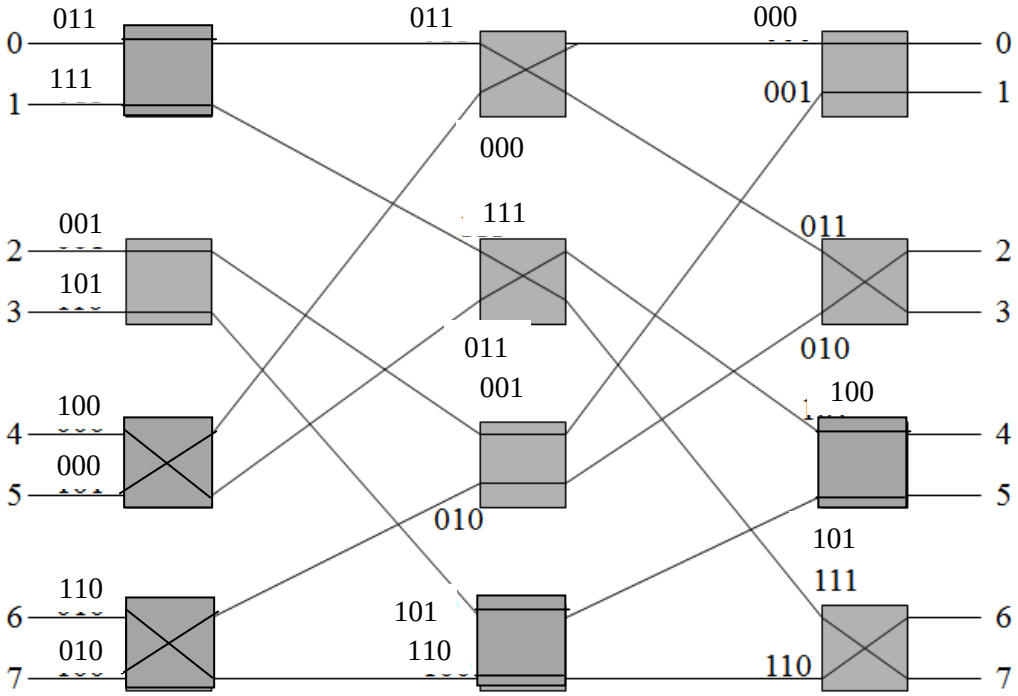
Q-4) Citer et expliquer les différentes phases utilisées par un programme parallèle ? **(01 points)**

R-4) Les différentes phases utilisées par un programme parallèle: **(01 points)**



Exercice N° 02: (06 points)

Créer un réseau Omega de 8x8, ensuite, appliquer la matrice des permutations suivante : $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 7 & 1 & 5 & 4 & 0 & 6 & 2 \end{pmatrix}$



Exercice N° 01: (06 points)

Soit le tableau ci-dessous qui représente les valeurs des temps d'exécution en secondes de six versions des programmes, avec un nombre de threads différent à chaque fois.

Q-1) Pourquoi les valeurs d'exécution de la même version du programme se diffère d'une exécution à l'autre ? **(02 points)**

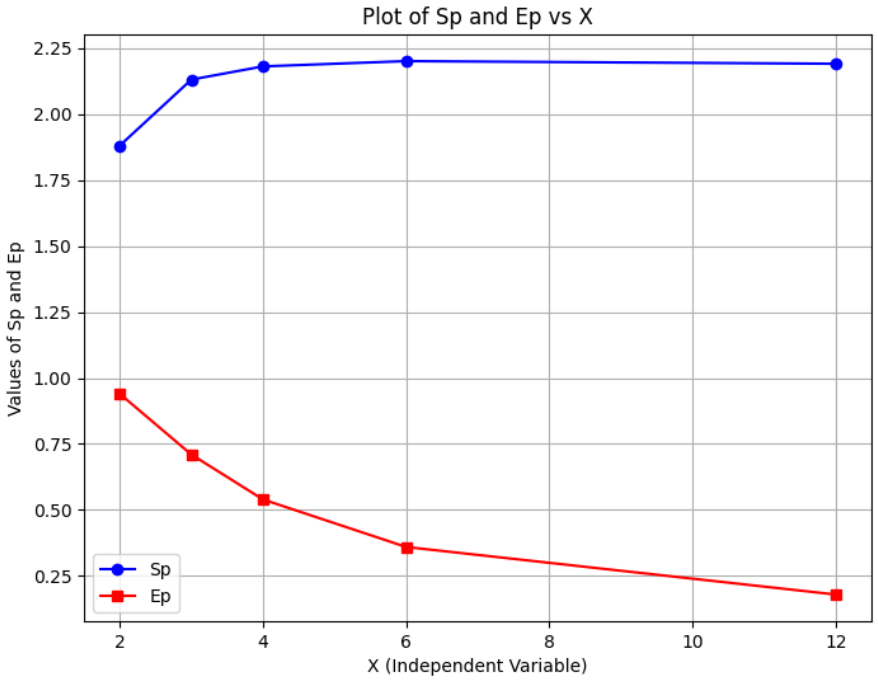
R-1) Le changement dans le temps d'exécution est dû à la variation de temps de latence (temps d'attente) pour exécuter un processus à cause de l'exécution concurrente.

		Numéro d'exécution										**Moyen**
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Séquentiel		8.80	7.13	7.07	7.19	7.85	6.71	7.51	6.68	7.39	7.19	7.35
Parallèle	**2**	3.24	3.81	4.08	3.90	3.81	3.78	3.96	3.74	4.10	4.10	3.89
	3	3.04	3.68	3.40	3.39	3.44	3.54	3.40	3.43	3.43	3.68	3.44
	4	3.34	3.49	3.46	3.30	3.37	3.35	3.34	3.37	3.28	3.42	3.36
	6	2.95	3.23	3.33	3.41	3.40	3.30	3.20	3.42	3.30	3.35	3.33
	12	3.94	3.52	3.18	3.13	3.16	3.39	3.29	3.11	3.29	3.37	3.35

Q-2) Calculer les valeurs de l'**accélération** et de l'**efficacité** dans chaque cas, ensuite dessiner des courbes ? **(04 points)**

R-2)

	2	**3**	**4**	**6**	**12**
$\mathbf{S_p}$	1.88	2.13	2.18	2.20	2.19
$\mathbf{E_p}$	0.94	0.71	0.54	0.36	0.18



Q-3) Donnez vos remarques sur les différentes courbes ? **(02 points)**

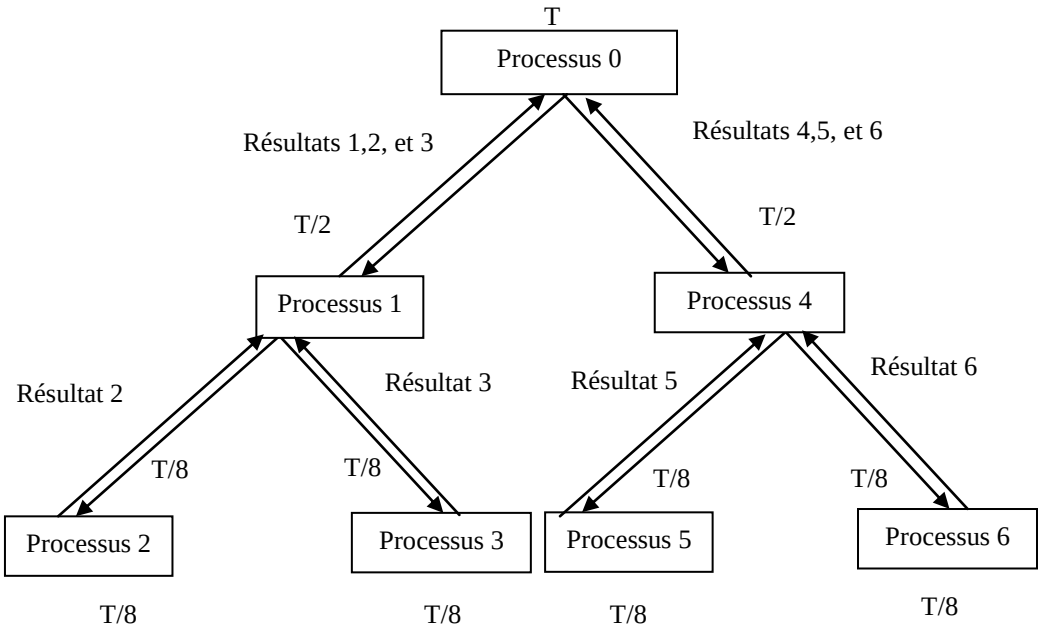
R-3)) Nous pouvons remarquer le suivant :

- L'utilisation d'un ensemble de threads permet de gagner un temps important de traitement.
- L'ajout de threads pour une exécution parallèle est limité par la capacité de la mémoire et d'exécution de la machine.

Exercice N° 03 : (04 points)

Donnez le programme Python utilisant la bibliothèque MPI, qui permet de calculer la valeur max d'un tableau T[1600] en suivant ce scénario :

- Le processus 0 prépare le tableau T (initialisation du tableau avec des valeurs aléatoires), ensuite, divise le tableau en deux (il divise les cases du tableau) et envoi une moitié au processus 1 et l'autre moitié au processus 4.
- Les deux processus 1 et 4 vont à ses tours divisés les paries obtenus en 02, gardent une moitié et renvoyés l'autre moitié divisé en deux à d'autres processus de calcul (2 et 3 pour processus 1, et 5 et 6 pour processus 4). Les résultats doivent être renvoyés au processus 0 pour qu'ils soient afficher (voir le schéma suivant) en suivant le chemin inverse.
- Finalement, processus 0 affiche la valeur max parmi toutes les autres valeurs.




```
from mpi4py import MPI
import random

def main:()
   NC = 1600
   NCD2 = NC // 2
   NCD4 = NC // 4
   NCD8 = NC // 8
   maxF = 0
   maxval[0] =

#    Initialize the matrix T
   T = [0] * NC
```

```
#    Get the rank of the process and the number of processes
   comm = MPI.COMM_WORLD
   rank = comm.Get_rank()
   size = comm.Get_size()

   if rank == 0:
#       Initialize the array T with random values
      T = [random.randint(0, 100) for _ in range(NC)]

#       Send the two halves of T to the processes
      comm.Send([T[:NCD2], MPI.INT], dest=1, tag=0)
      comm.Send([T[NCD2:], MPI.INT], dest=2, tag=0)

#       Receive results from processes 1, 2, and 3
      for i in range:(3)
          comm.Recv([maxval, MPI.INT], source=1, tag=i+1)
          maxF = max(maxF, maxval[0])

#       Receive results from processes 4, 5, and 6
      for i in range:(3)
          comm.Recv([maxval, MPI.INT], source=4, tag=i+1)
          maxF = max(maxF, maxval[0])

#       Print the final result
      print("max table =", maxF)

   elif rank == 1 or rank == 4:
#       Process 1 and 4
      comm.Recv([T[:NCD2], MPI.INT], source=0, tag=0)

      comm.Send([T[NCD4:NCD4 + NCD8], MPI.INT], dest=rank + 1, tag=0)
      comm.Send([T[NCD4 + NCD8:NCD4 + 2 * NCD8], MPI.INT], dest=rank + 2, tag=0)

#       Find the max value of the portion of T
      maxval[0] = max(T[:NCD4])

#       Send the result back to process 0
      comm.Send([maxval, MPI.INT], dest=0, tag=0)

#       Receive and forward the max values from other processes
      comm.Recv([maxval, MPI.INT], source=rank + 1, tag=0)
      comm.Send([maxval, MPI.INT], dest=0, tag=1)

      comm.Recv([maxval, MPI.INT], source=rank + 2, tag=0)
      comm.Send([maxval, MPI.INT], dest=0, tag=2)

   elif rank == 2 or rank == 5:
#       Process 2 and 5
      comm.Recv([T[:NCD8], MPI.INT], source=rank - 1, tag=0)

#       Find the max value of the portion of T
      maxval[0] = max(T[:NCD8])

#       Send the result back to the previous process
      comm.Send([maxval, MPI.INT], dest=rank - 1, tag=0)

   elif rank == 3 or rank == 6:
#       Process 3 and 6
```

```
        comm.Recv([T[:NCD8], MPI.INT], source=rank - 2, tag=0)

#       Find the max value of the portion of T
        maxval[0] = max(T[:NCD8])

#       Send the result back to the previous process
        comm.Send([maxval, MPI.INT], dest=rank - 2, tag=0)

#    Finalize MPI
    MPI.Finalize()

if __name__ == "__main:"__
    main()
```

Bon Courage **Bonne Chance**