Programmation Parallèle : Examen (1h45)

Exercice 1: Questions de cours

1. Qu'est ce que la vectorisation ? Quelle est son utilité ?
2. Citez deux différences entre **OpenMP** et **MPI**.
3. Citez deux aspects importants à considérer pour obtenir des codes parallèles performants.
4. Citez un avantage et un inconvénient de l'utilisation des barrières de synchronisation.

Exercice 2: Vectorisation

Soit une machine fournissant les instructions vectorielles suivantes :

- vector float **vec_ld**(float*) : chargement aligné de 4 floats
- void **vec_st**(vector float, float*) : rangement aligné de 4 floats
- vector float **vec_splat**(float) : remplissage d'un vecteur par une constante
- vector float **vec_add**(vector float, vector float) : somme élément par élément de deux vecteurs de floats
- vector float **vec_mul**(vector float, vector float) : produit élément par élément de deux vecteurs de floats
- float **vec_hadd**(vector float) : somme des éléments d'un vecteur de floats

Toutes ces instructions ont une latence de 1 cycle.

On considère que 'vector float' est le type représentant un registre vectoriel 128 bits contenant 4 floats contigus. Leur initialisation peut s'écrire:

```
vector float f = {1,2,3,4};
```

Implémenter un code vectoriel utilisant ces instructions permettant de calculer le produit scalaire de deux tableaux de float de $4N$ éléments.

Exercice 3: OpenMP

1. Soit le code ci-dessous,

```
#include <stdio.h>
#define MAX 10000

int main(){

size_t i;
int n = 0;

#pragma omp parallel for

for ( i=0 ; i<MAX ; i++ ){
    n++;}

printf("n =%d \n", n);

return 0;
}
```

(a) Quel est le résultat attendu par ce code?

(b) Est ce que ce code affiche le résultat attendu? Justifiez votre réponse en expliquant le prblème si il en a.

(c) Si nécessaire, proposez une correction du code pour qu'il affiche le résultat attendu.

2. Soit le code ci-dessous, où **factorielle** est une fonction qui calcule la factorielle d'un entier donné.

```
#define N 1000

int main(){

size_t i;
long int* tab = (long int*) malloc(N*sizeof(long int));


for ( i=1 ; i=<N ; i++ )
    tab[i-1] = factorielle(i);

return 0;
}
```

(a) Ajoutez la directive OpenMP nécessaire pour paralléliser ce code, justifiez votre choix.

(b) Quel est le paramètre important à prendre en considération pour une parallélisation efficace de ce code?

Exercice 4: MPI

On considère une matrice carrée réelle A, de taille $n \times n$. On souhaite calculer, en parallèle et en utilisant **MPI**, la trace de la matrice A, avec
$Trace(A) = \sum_{i=1}^{n} a_{ii}$.

Ecrivez un programme MPI qui effectue les tâches suivantes :

1. le processus de rang 0 initialise la matrice A et la distribue sur p processus. On considère une distribution au long des lignes et on suppose que n est divisible par p.
2. chaque processus calcule la trace locale correspondant à sa portion de la matrice A.
3. le processus de rang 0 récupère toutes les traces locales pour calculer la trace globale de la matrice.

Vous disposez des protoptypes de certaines fonctions **MPI** :

```
int MPI_Init(int *argc, char ***argv)
int MPI_Finalize( void )
int MPI_Comm_size( MPI_Comm comm, int *size )
int MPI_Comm_rank( MPI_Comm comm, int *rank )
int MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest,
            int tag, MPI_Comm comm )
int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int source,
            int tag, MPI_Comm comm, MPI_tatus *status )
int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root,
            MPI_Comm comm )
int MPI_Barrier( MPI_Comm comm )
int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
               void *recvbuf, int recvcount, MPI_Datatype recvtype,
               int root, MPI_Comm comm)
int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
               void *recvbuf, int recvcount, MPI_Datatype recvtype,
               int root, MPI_Comm comm)
int MPI_Reduce(const void *sendbuf, void *recvbuf, int count,
               MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count,
                  MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```