TIRIRI paralléliseisme : hello

Pour réaliser ce TP élémentaire sur OpenMP avec le langage C,
vous aurezriy besoin des outils et logiciels suivants sur votre
machine :
1. Un Compilateur Compatible OpenMP
GCC (GNU Compiler Collection) :GCC est l'un des compilateurs les plus couramment utilisés pour
le C et supporte OpenMP.
Vous pouvez installer GCC via les gestionnaires de paquets de
votre système d'exploitation.
Installation sous Linux (Ubuntu/Debi

rakom kbar

tapez la commande suivante

lscpu

Quelles sont les caractéristiques de votre machine

Exemple1

```c
#include <omp.h>
#include <stdio.h>
int main () {
  #pragma omp parallel
  {// Création de threads (processus légers)
   int ID = omp_get_thread_num ();
  printf("Hello(%d)", ID);
  printf ("world(%d)\n", ID);
  }//Destruction de threads
  }
```

1-compiler et exécuter le code
gcc -fopenmp -o test_openmp test_openmp.c ./test_openmp
a-commenté le résultat
b-donner une explication

Exemple 2

```c
#include <omp.h>
#include <stdio.h>
void main () {
   #pragma omp parallel num_threads (10)
   {// Création de threads (processus légers)
    int ID = omp_get_thread_num ();
```

```
   printf("Hello(%d)", ID);
   printf ("world(%d)\n", ID);
   }//Destruction de threads
   }
```

1-compiler et exécuter le code
gcc -fopenmp -o test2_openmp test2_openmp.c ./test2_openmp
2-commenté le résultat
3-donner les rôles des diractives.
#pragma omp parallel num_threads (10)
 int ID = omp_get_thread_num ();

Exemple3 :

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <omp.h>
void main() {

omp_set_num_threads(2) ;
int a = 10, b = 20, c = 30 ;
   #pragma omp parallel private (a)
   {
   printf(" %d \n", a + 1) ;
   }
   #pragma omp parallel firstprivate (b)
   {
       b=b + 1;
   printf(" %d \n", b ) ;
   }
   #pragma omp parallel shared (c)
   {

   printf(" %d \n", c ) ;
   }

}
```

1-compiler et exécuter le code

2-commenté le résultat
3-donner les rôles des directives.

Exemple 4:

Soit le programme séquentiel suivant qui réalise la multiplication de deux matrice a et b.

1-Compiler et exécuter le programme en affichant les temps d'exécution de la version parallèle (utiliser l'exemple de calcul du temps d'exécution utilisé dans la version séquentielle du code).

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <omp.h>
int main() {
   const int DIM = 1000;
   int i,j,k;
   double debut, fin, temps;
   double **a, **b, **cresu, **ctest;
   a= (double**) malloc(DIM*sizeof(double*));
   b= (double**) malloc(DIM*sizeof(double*));
   cresu= (double**) malloc(DIM*sizeof(double*));
   ctest= (double**) malloc(DIM*sizeof(double*));
   // initialisations etc...
   for (i=0; i<DIM; i++)
   {
       a[i]=(double*) malloc(DIM*sizeof(double));
        b[i]=(double*) malloc(DIM*sizeof(double));
        cresu[i]=(double*) malloc(DIM*sizeof(double));
        ctest[i]=(double*) malloc(DIM*sizeof(double));
        for (j = 0; j < DIM; j++)
          {
            a[i][j] = (double)(i-j);
             b[i][j] = (double)(i+j);
             cresu[i][j] = 0.0;
             ctest[i][j] = 0.0;
          }
   } // Multiplication C = A x B (séquentiel)
             printf("Multiplication sequentielle:\n");
               debut= omp_get_wtime();
               for (i = 0; i < DIM; i++)
```

```
                    for (j = 0; j < DIM; j++)
                     for (k = 0; k < DIM; k++)
                       ctest[i][j] += a[i][k] * b[k][j];
                fin= omp_get_wtime();
                temps=fin-debut;
                printf ("Calcul séquentiel %f secondes\n", temps);
                // Multiplication C = A x B (parallèle)
                 return (0);
   }
```

2-Compiler et exécuter le programme parallèle proposé en affichant les temps d'exécution de la version parallèle (utiliser l'exemple de calcul du temps d'exécution utilisé dans la version séquentielle du code).

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <omp.h>
int main() {
   const int DIM = 1000;
   int i,j,k;
   double debut, fin, temps;
   double **a, **b, **cresu, **ctest;
   a= (double**) malloc(DIM*sizeof(double*));
   b= (double**) malloc(DIM*sizeof(double*));
   cresu= (double**) malloc(DIM*sizeof(double*));
   ctest= (double**) malloc(DIM*sizeof(double*));
   // initialisations etc...
   for (i=0; i<DIM; i++)
   {
       a[i]=(double*) malloc(DIM*sizeof(double));
        b[i]=(double*) malloc(DIM*sizeof(double));
        cresu[i]=(double*) malloc(DIM*sizeof(double));
        ctest[i]=(double*) malloc(DIM*sizeof(double));
        for (j = 0; j < DIM; j++)
          {
            a[i][j] = (double)(i-j);
             b[i][j] = (double)(i+j);
             cresu[i][j] = 0.0;
             ctest[i][j] = 0.0;
           }
   } printf("Multiplication parallèle:\n");
```

```
   debut= omp_get_wtime();
   #pragma omp parallel for schedule (static) num_threads(4)
private(i,j,k)
    for (j = 0; j < DIM; j++)
    for (k = 0; k < DIM; k++)
    ctest[i][j] += a[i][k] * b[k][j];
    fin= omp_get_wtime();
    temps=fin-debut;
    printf ("Calcul parallèle %f secondes\n", temps);
                return (0);
   }
```

Expliquer le role de la directive:

```
#pragma omp parallel for schedule (static) num_threads(4)
private(i,j,k)
```

TIRIRI paralléliseisme

Pour réaliser ce TP élémentaire sur OpenMP avec le langage C, vous aurezriy besoin des outils et logiciels suivants sur votre machine :

1. Un Compilateur Compatible OpenMP

GCC (GNU Compiler Collection) :

GCC est l'un des compilateurs les plus couramment utilisés pour le C et supporte OpenMP.

Vous pouvez installer GCC via les gestionnaires de paquets de votre système d'exploitation.

Installation sous Linux (Ubuntu/Debi

tapez la commande suivante

lscpu

Quelles sont les caractéristiques de votre machine

Exemple1

```
#include <omp.h>
#include <stdio.h>
int main () {
 #pragma omp parallel
 {// Création de threads (processus légers)
  int ID = omp_get_thread_num ();
 printf("Hello(%d)", ID);
 printf ("world(%d)\n", ID);
 }//Destruction de threads
 }
```

1-compiler et exécuter le code
gcc -fopenmp -o test_openmp test_openmp.c ./test_openmp
a-commenté le résultat
b-donner une explication

Exemple 2

```
#include <omp.h>
#include <stdio.h>
void main () {
  #pragma omp parallel num_threads (10)
  {// Création de threads (processus légers)
   int ID = omp_get_thread_num ();
  printf("Hello(%d)", ID);
  printf ("world(%d)\n", ID);
  }//Destruction de threads
  }
```

1-compiler et exécuter le code
gcc -fopenmp -o test2_openmp test2_openmp.c ./test2_openmp
2-commenté le résultat
3-donner les rôles des diractives.
#pragma omp parallel num_threads (10)
 int ID = omp_get_thread_num ();

Exemple3 :

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <omp.h>
void main() {

omp_set_num_threads(2) ;
int a = 10, b = 20, c = 30 ;
  #pragma omp parallel private (a)
  {
  printf(" %d \n", a + 1) ;
  }
  #pragma omp parallel firstprivate (b)
```

```
  {
     b=b + 1;
  printf(" %d \n", b ) ;
  }
  #pragma omp parallel shared (c)
  {

  printf(" %d \n", c ) ;
  }

}
```

1-compiler et exécuter le code
2-commenté le résultat
3-donner les rôles des directives.

Exemple 4:

Soit le programme séquentiel suivant qui réalise la multiplication de deux matrice a et b.

1-Compiler et exécuter le programme en affichant les temps d'exécution de la version parallèle (utiliser l'exemple de calcul du temps d'exécution utilisé dans la version séquentielle du code).

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <omp.h>
int main() {
  const int DIM = 1000;
  int i,j,k;
  double debut, fin, temps;
  double **a, **b, **cresu, **ctest;
  a= (double**) malloc(DIM*sizeof(double*));
  b= (double**) malloc(DIM*sizeof(double*));
  cresu= (double**) malloc(DIM*sizeof(double*));
  ctest= (double**) malloc(DIM*sizeof(double*));
  // initialisations etc...
  for (i=0; i<DIM; i++)
```

```
 {
    a[i]=(double*) malloc(DIM*sizeof(double));
     b[i]=(double*) malloc(DIM*sizeof(double));
     cresu[i]=(double*) malloc(DIM*sizeof(double));
     ctest[i]=(double*) malloc(DIM*sizeof(double));
     for (j = 0; j < DIM; j++)
       {
         a[i][j] = (double)(i-j);
          b[i][j] = (double)(i+j);
          cresu[i][j] = 0.0;
          ctest[i][j] = 0.0;
        }
 } // Multiplication C = A x B (séquentiel)
          printf("Multiplication sequentielle:\n");
           debut= omp_get_wtime();
           for (i = 0; i < DIM; i++)
               for (j = 0; j < DIM; j++)
                for (k = 0; k < DIM; k++)
                  ctest[i][j] += a[i][k] * b[k][j];
           fin= omp_get_wtime();
           temps=fin-debut;
           printf ("Calcul séquentiel %f secondes\n", temps);
           // Multiplication C = A x B (parallèle)
            return (0);
 }
```

2-Compiler et exécuter le programme parallèle proposé en affichant les temps d'exécution de la version parallèle (utiliser l'exemple de calcul du temps d'exécution utilisé dans la version séquentielle du code).

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <omp.h>
int main() {
  const int DIM = 1000;
  int i,j,k;
  double debut, fin, temps;
  double **a, **b, **cresu, **ctest;
```

```
  a= (double**) malloc(DIM*sizeof(double*));
  b= (double**) malloc(DIM*sizeof(double*));
  cresu= (double**) malloc(DIM*sizeof(double*));
  ctest= (double**) malloc(DIM*sizeof(double*));
  // initialisations etc...
  for (i=0; i<DIM; i++)
  {
      a[i]=(double*) malloc(DIM*sizeof(double));
       b[i]=(double*) malloc(DIM*sizeof(double));
       cresu[i]=(double*) malloc(DIM*sizeof(double));
       ctest[i]=(double*) malloc(DIM*sizeof(double));
       for (j = 0; j < DIM; j++)
         {
           a[i][j] = (double)(i-j);
            b[i][j] = (double)(i+j);
            cresu[i][j] = 0.0;
            ctest[i][j] = 0.0;
          }
  } printf("Multiplication parallèle:\n");
  debut= omp_get_wtime();
  #pragma omp parallel for schedule (static) num_threads(4)
private(i,j,k)
   for (j = 0; j < DIM; j++)
   for (k = 0; k < DIM; k++)
   ctest[i][j] += a[i][k] * b[k][j];
   fin= omp_get_wtime();
   temps=fin-debut;
   printf ("Calcul parallèle %f secondes\n", temps);
              return (0);
  }
```

Expliquer le role de la directive:

```
#pragma omp parallel for schedule (static) num_threads(4) private(i,j,k)
```