Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Parallélisme & Applications

TD n°3

Programmation OpenMP — la directive « task »

La directive « task »

- la directive « task » crée une tâche de manière **explicite** ;
- toutes les threads se **partagent le travail disponible** parmi l'ensemble des tâches créées ;
- la directive « taskwait » **garantie** que toutes les tâches enfants crées dans la même tâche courante **ont fini**.

1 – Comparez les deux programmes suivants :

```
#include <omp.h>
#include <stdio.h>

main()
{
   int i;
   #pragma omp parallel private(i)
   {   for(i=0; i<10; i++)
       {      printf("Task %d: %i\n",
              omp_get_thread_num(),i);
       }
   }
}
```

```
#include <omp.h>
#include <stdio.h>

main()
{
   int i;
   #pragma omp parallel private(i)
   {   for(i=0; i<10; i++)
       #pragma omp task
       {      printf("Task %d: %i\n",
              omp_get_thread_num(),i);
       }
   }
}
```

Combien de threads vont être utilisées et quel va être leur travail ?

2 – Que va produire le code suivant :

```
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

int main()
{
   #pragma omp parallel sections
   {
      #pragma omp section
      {   int i;
         for (i=0;i<4;i++)
         {
            #pragma omp task
            printf("Task %d\n",omp_get_thread_num());
         }
      }
      #pragma omp section
      {   int i;
         for (i=0;i<4;i++)
           {
            #pragma omp task
            printf("Task %d\n",omp_get_thread_num());
         }
      }
   }
}
```

3 – Décrivez le fonctionnement du programme suivant :

```
1  #include <stdio.h>
2  int work( int i )
3  {
4     if ( i > 0 )
5     {
6        #pragma omp parallel
7        {
8           #pragma omp task
9           {
10             work( i-1 );
11          }
12          #pragma omp task
13          {
14             work( i-1 );
15          }
16          #pragma omp taskwait
17          printf( "Completed %i\n", i );
18       }
19    }
20 }
21 void main()
22 {
23    work( 3 );
24 }
```

4 – Essayez de **paralléliser** ces morceaux de code ou expliquez **pourquoi** ce n'est pas possible :

a.

```
1 for (i = 0; i < sqrt(x); i++)
2 {
3     a[i] = 2.3 * i;
4     if (i < 10)
5         b[i] = a[i];
6 }
```

b.

```
1 flag = 0;
2 for (i = 0; i < n && !flag; i++)
3 {
4         a[i] = 2.3 * i;
5         if (a[i] < b[i])
6         flag = 1;
7 }
```