Faculté des Sciences & Techniques

Université de Limoges

Master 1ère année

Parallélisme & Applications

TD n°1

Programmation OpenMP

Les différentes formes de parallélisme

1 – Commentez le programme suivant, `omp_hello.c` :

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

int main ()
{ int nthreads, tid;

#pragma omp parallel private(nthreads, tid)
  {
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  if (tid == 0)
    {  nthreads = omp_get_num_threads();
       printf("Number of threads = %d\n", nthreads);
    }
  }
}
```

2 – Quel va être le résultat de ce programme, `omp_workshare1.c` :

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define CHUNKSIZE 10
#define N 100

int main ()
{ int nthreads, tid, i, chunk;
  float a[N], b[N], c[N];

  for (i=0; i < N; i++)
     a[i] = b[i] = i * 1.0;
  chunk = CHUNKSIZE;

  #pragma omp parallel shared(a,b,c,nthreads,chunk) private(i,tid)
  {
  tid = omp_get_thread_num();
  if (tid == 0)
    { nthreads = omp_get_num_threads();
      printf("Number of threads = %d\n", nthreads);
    }
  printf("Thread %d starting...\n",tid);

  #pragma omp for schedule(dynamic,chunk)
  for (i=0; i<N; i++)
    {  c[i] = a[i] + b[i];
       printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
    }
  }  /* end of parallel section */
}
```

3 – Soit le programme suivant :

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

const int size = 90;

int main()
{
  /* long long unsigned int tab[100]; */
  uint64_t tab[size];
  tab[0] = 0;
  tab[1] = 1;

  #pragma omp parallel for shared(tab) schedule(static,10)
  for(int i=2; i<size; i++)
    tab[i] = tab[i-1] + tab[i-2];

  for(int i=2; i<size; i++)
    printf("%ld ", tab[i]);
}
```

Est-ce qu'il fournit un résultat correct ?
Pourquoi ?

4 – Quel est le résultat de ce programme, `omp_workshare2.c` :

```c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define N 50

int main ()
{ int i, nthreads, tid;
  float a[N], b[N], c[N], d[N];

for (i=0; i<N; i++) {
  a[i] = i * 1.5; b[i] = i + 22.35; c[i] = d[i] = 0.0;
  }
#pragma omp parallel shared(a,b,c,d,nthreads) private(i,tid)
  {
  tid = omp_get_thread_num();
  if (tid == 0)
    { nthreads = omp_get_num_threads();
      printf("Number of threads = %d\n", nthreads);
    }
  printf("Thread %d starting...\n",tid);

  #pragma omp sections nowait
    {
    #pragma omp section
      {
      printf("Thread %d doing section 1\n",tid);
      for (i=0; i<N; i++)
        { c[i] = a[i] + b[i];
          printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
        }
      }
    #pragma omp section
      {
      printf("Thread %d doing section 2\n",tid);
      for (i=0; i<N; i++)
        { d[i] = a[i] * b[i];
          printf("Thread %d: d[%d]= %f\n",tid,i,d[i]);
        }
      }
    }  /* end of sections */
    printf("Thread %d done.\n",tid);
  }  /* end of parallel section */
}
```

5 – Et de celui-ci, `omp_reduction.c` :

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int   i, n;
    float a[100], b[100], sum;

    /* Some initializations */
    n = 100;
    for (i=0; i < n; i++)
      a[i] = b[i] = i * 1.0;
    sum = 0.0;

    #pragma omp parallel for reduction(+:sum)
      for (i=0; i < n; i++)
        sum = sum + (a[i] * b[i]);
    printf("   Sum = %f\n",sum);
}
```

6 – Comment vont s'organiser les différentes threads dans le programme, `omp_orphan.c` :

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define VECLEN 100
float a[VECLEN], b[VECLEN], sum;

void dotprod ()
{ int i,tid;

  tid = omp_get_thread_num();
  #pragma omp for reduction(+:sum)
  for (i=0; i < VECLEN; i++)
    { sum = sum + (a[i]*b[i]);
     printf("  tid= %d i=%d\n",tid,i);
    }
}
int main ()
{  int i;

  for (i=0; i < VECLEN; i++) a[i] = b[i] = 1.0 * i;
  sum = 0.0;
  #pragma omp parallel
     dotprod();
  printf("Sum = %f\n",sum);
}
```

7 – Ce programme affiche l'ensemble des informations du contexte parallèle, `omp_getEnvInfo.c` :

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

int main (int argc, char *argv[])
{ int nthreads, tid, procs, maxt, inpar, dynamic, nested;

/* Start parallel region */
#pragma omp parallel private(nthreads, tid)
  {
  /* Obtain thread number */
  tid = omp_get_thread_num();

  /* Only master thread does this */
  if (tid == 0)
    {
    printf("Thread %d getting environment info...\n", tid);

    /* Get environment information */
    procs = omp_get_num_procs();
    nthreads = omp_get_num_threads();
    maxt = omp_get_max_threads();
    inpar = omp_in_parallel();
    dynamic = omp_get_dynamic();
    nested = omp_get_nested();

    /* Print environment information */
    printf("Number of processors = %d\n", procs);
    printf("Number of threads = %d\n", nthreads);
    printf("Max threads = %d\n", maxt);
    printf("In parallel? = %d\n", inpar);
    printf("Dynamic threads enabled? = %d\n", dynamic);
    printf("Nested parallelism supported? = %d\n", nested);

    }
  }  /* Done */
}
```

8 – Décrivez l'organisation des threads pour le programme suivant, `omp_mm.c` :

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

#define NRA 62                 /* number of rows in matrix A */
#define NCA 15                 /* number of columns in matrix A */
#define NCB 7                  /* number of columns in matrix B */

int main (int argc, char *argv[])
{ int   tid, nthreads, i, j, k, chunk;
  double   a[NRA][NCA], /* matrix A to be multiplied */
  b[NCA][NCB],  /* matrix B to be multiplied */
  c[NRA][NCB];  /* result matrix C */

  chunk = 10;                    /* set loop iteration chunk size */

/*** Spawn a parallel region explicitly scoping all variables ***/
#pragma omp parallel shared(a,b,c,nthreads,chunk) private(tid,i,j,k)
  {
  tid = omp_get_thread_num();
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Starting matrix multiple example with %d threads\n",nthreads);
    printf("Initializing matrices...\n");
    }
  /*** Initialize matrices ***/
  #pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    for (j=0; j<NCA; j++)
      a[i][j]= i+j;
  #pragma omp for schedule (static, chunk)
  for (i=0; i<NCA; i++)
    for (j=0; j<NCB; j++)
      b[i][j]= i*j;
  #pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    for (j=0; j<NCB; j++)
      c[i][j]= 0;

  /*** Do matrix multiply sharing iterations on outer loop ***/
  /*** Display who does which iterations for demonstration purposes ***/
  printf("Thread %d starting matrix multiply...\n",tid);
  #pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    {
    printf("Thread=%d did row=%d\n",tid,i);
    for(j=0; j<NCB; j++)
      for (k=0; k<NCA; k++)
        c[i][j] += a[i][k] * b[k][j];
    }
  }   /*** End of parallel region ***/

/*** Print results ***/
printf("******************************************************\n");
printf("Result Matrix:\n");
for (i=0; i<NRA; i++)
  {
  for (j=0; j<NCB; j++)
    printf("%6.2f   ", c[i][j]);
  printf("\n");
  }
printf("******************************************************\n");
printf ("Done.\n");
}
```