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`:

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 int main ()
 6 { int nthreads, tid;
 7
 8 #pragma omp parallel private(nthreads, tid)
 9   {
10   tid = omp_get_thread_num();
11   printf("Hello World from thread = %d\n", tid);
12
13   if (tid == 0)
14     {  nthreads = omp_get_num_threads();
15        printf("Number of threads = %d\n", nthreads);
16     }
17   }
18 }
```

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

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #define CHUNKSIZE 10
 5 #define N 100
 6
 7 int main ()
 8 { int nthreads, tid, i, chunk;
 9   float a[N], b[N], c[N];
10
11   for (i=0; i < N; i++)
12      a[i] = b[i] = i * 1.0;
13   chunk = CHUNKSIZE;
14
15   #pragma omp parallel shared(a,b,c,nthreads,chunk) private(i,tid)
16   {
17   tid = omp_get_thread_num();
18   if (tid == 0)
19     { nthreads = omp_get_num_threads();
20       printf("Number of threads = %d\n", nthreads);
21     }
22   printf("Thread %d starting...\n",tid);
23
24   #pragma omp for schedule(dynamic,chunk)
25   for (i=0; i<N; i++)
26     {  c[i] = a[i] + b[i];
27        printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
28     }
29   }  /* end of parallel section */
30 }
```

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

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #define N 50
 5
 6 int main ()
 7 { int i, nthreads, tid;
 8   float a[N], b[N], c[N], d[N];
 9
10 for (i=0; i<N; i++) {
11   a[i] = i * 1.5; b[i] = i + 22.35; c[i] = d[i] = 0.0;
12   }
13 #pragma omp parallel shared(a,b,c,d,nthreads) private(i,tid)
14   {
15   tid = omp_get_thread_num();
16   if (tid == 0)
17     { nthreads = omp_get_num_threads();
18       printf("Number of threads = %d\n", nthreads);
19     }
20   printf("Thread %d starting...\n",tid);
21
22   #pragma omp sections nowait
23     {
24     #pragma omp section
25       {
26       printf("Thread %d doing section 1\n",tid);
27       for (i=0; i<N; i++)
28         { c[i] = a[i] + b[i];
29           printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
30         }
31       }
32     #pragma omp section
33       {
34       printf("Thread %d doing section 2\n",tid);
35       for (i=0; i<N; i++)
36         { d[i] = a[i] * b[i];
37           printf("Thread %d: d[%d]= %f\n",tid,i,d[i]);
38         }
39       }
40     }  /* end of sections */
41     printf("Thread %d done.\n",tid);
42   }  /* end of parallel section */
43 }
```

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

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 int main ()
 5 {
 6     int   i, n;
 7     float a[100], b[100], sum;
 8
 9     /* Some initializations */
10     n = 100;
11     for (i=0; i < n; i++)
12       a[i] = b[i] = i * 1.0;
13     sum = 0.0;
14
15     #pragma omp parallel for reduction(+:sum)
16       for (i=0; i < n; i++)
17         sum = sum + (a[i] * b[i]);
18     printf("   Sum = %f\n",sum);
19 }
```

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

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #define VECLEN 100
 5 float a[VECLEN], b[VECLEN], sum;
 6
 7 void dotprod ()
 8 { int i,tid;
 9
10   tid = omp_get_thread_num();
11   #pragma omp for reduction(+:sum)
12   for (i=0; i < VECLEN; i++)
13     { sum = sum + (a[i]*b[i]);
14       printf("  tid= %d i=%d\n",tid,i);
15     }
16 }
17 int main ()
18 {  int i;
19
20   for (i=0; i < VECLEN; i++) a[i] = b[i] = 1.0 * i;
21   sum = 0.0;
22   #pragma omp parallel
23     dotprod();
24   printf("Sum = %f\n",sum);
25 }
```

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

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 int main (int argc, char *argv[])
 6 { int nthreads, tid, procs, maxt, inpar, dynamic, nested;
 7
 8 /* Start parallel region */
 9 #pragma omp parallel private(nthreads, tid)
10   {
11   /* Obtain thread number */
12   tid = omp_get_thread_num();
13
14   /* Only master thread does this */
15   if (tid == 0)
16     {
17     printf("Thread %d getting environment info...\n", tid);
18
19     /* Get environment information */
20     procs = omp_get_num_procs();
21     nthreads = omp_get_num_threads();
22     maxt = omp_get_max_threads();
23     inpar = omp_in_parallel();
24     dynamic = omp_get_dynamic();
25     nested = omp_get_nested();
26
27     /* Print environment information */
28     printf("Number of processors = %d\n", procs);
29     printf("Number of threads = %d\n", nthreads);
30     printf("Max threads = %d\n", maxt);
31     printf("In parallel? = %d\n", inpar);
32     printf("Dynamic threads enabled? = %d\n", dynamic);
33     printf("Nested parallelism supported? = %d\n", nested);
34
35     }
36   }  /* Done */
37 }
```

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

```
 1 #include <omp.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 #define NRA 62                  /* number of rows in matrix A */
 6 #define NCA 15                  /* number of columns in matrix A */
 7 #define NCB 7                   /* number of columns in matrix B */
 8
 9 int main (int argc, char *argv[])
10 { int   tid, nthreads, i, j, k, chunk;
11   double   a[NRA][NCA], /* matrix A to be multiplied */
12   b[NCA][NCB],  /* matrix B to be multiplied */
13   c[NRA][NCB];  /* result matrix C */
14
15   chunk = 10;                     /* set loop iteration chunk size */
16
17 /*** Spawn a parallel region explicitly scoping all variables ***/
18 #pragma omp parallel shared(a,b,c,nthreads,chunk) private(tid,i,j,k)
19   {
20   tid = omp_get_thread_num();
21   if (tid == 0)
22     {
23     nthreads = omp_get_num_threads();
24     printf("Starting matrix multiple example with %d threads\n",nthreads);
25     printf("Initializing matrices...\n");
26     }
27   /*** Initialize matrices ***/
28   #pragma omp for schedule (static, chunk)
29   for (i=0; i<NRA; i++)
30     for (j=0; j<NCA; j++)
31       a[i][j]= i+j;
32   #pragma omp for schedule (static, chunk)
33   for (i=0; i<NCA; i++)
34     for (j=0; j<NCB; j++)
35       b[i][j]= i*j;
36   #pragma omp for schedule (static, chunk)
37   for (i=0; i<NRA; i++)
38     for (j=0; j<NCB; j++)
39       c[i][j]= 0;
40
41   /*** Do matrix multiply sharing iterations on outer loop ***/
42   /*** Display who does which iterations for demonstration purposes ***/
43   printf("Thread %d starting matrix multiply...\n",tid);
44   #pragma omp for schedule (static, chunk)
45   for (i=0; i<NRA; i++)
46     {
47     printf("Thread=%d did row=%d\n",tid,i);
48     for(j=0; j<NCB; j++)
49       for (k=0; k<NCA; k++)
50         c[i][j] += a[i][k] * b[k][j];
51     }
52   }   /*** End of parallel region ***/
53
54 /*** Print results ***/
55 printf("******************************************************\n");
56 printf("Result Matrix:\n");
57 for (i=0; i<NRA; i++)
58   {
59   for (j=0; j<NCB; j++)
60     printf("%6.2f   ", c[i][j]);
61   printf("\n");
62   }
63 printf("******************************************************\n");
64 printf ("Done.\n");
65 }
```