Computação Paralela
Oct-2020

Parallel Programming with OpenMP

These exercises aim to introduce the basic concepts of parallel programming based on shared memory using OpenMP. In all these exercises students should look at the output of the two threads and understand/explain the relative order of the *printf* outputs.

1) Parallel regions

Compile the following program (with GNU use `gcc -fopenmp -std=c99`). Execute the program multiple times and explain the results.

Questions:
1.1 The order of the output is always the same? why?
1.2 The order of the output OF EACH thread is always the same? why?
1.3 How is the loop execution divided between threads?

```
#include<omp.h>
#include<stdio.h>
int main() {
     #pragma omp parallel num_threads(2)
     for(int i=0;i<100;i++)  {
               int id = omp_get_thread_num();
               printf("T%d:i%d ", id, i);
     }
}
```

2) Work sharing & synchronization

Introduce the following directives between the *#pragma omp parallel* and the "for". Explain the results.

2.1. *#pragma omp for*

2.2. *#pragma omp master*

2.3. *#pragma omp single*

2.4. *#pragma omp critical*

Questions:
2.1/2.2/2.3 How is the loop execution divided between threads?
2.1/2.2/2.3 The loop division is always the same?

Questions?
2.4/3.1/3.2 The order of the output is always the same?
2.4/3.1/3.2 What kind of synchronization occurs?

3) Synchronization

Introduce the following changes to the **program in 1):**

3.1. Include a barrier inside the loop, after the *printf* statement (*#pragma omp barrier*).

3.2. Include the directive *#pragma omp ordered* inside the loop, before the *printf* statement. Use the program developed in 2.1, also adding *ordered* to the *#pragma omp for*

4) Loop Scheduling

Exploit the impact of the following of the loop scheduling options in program 2.1, by adding the *schedule* clause to the *for* directive:

4.1. *schedule(static)* and *schedule(static,10)*

4.2. *schedule(dynamic)* and *schedule(dynamic,10)*

4.3. *schedule(guided)*

Questions:
4.1/4.2/4.3 How is the loop execution divided (i.e., scheduled) between threads?
4.1/4.2/4.3 The loop division (i.e., scheduling) is always the same?

5) Data sharing

Compile and execute the following program several times and explain the results

```
#include<omp.h>
#include<stdio.h>
int main() {
     int w=10;
     #pragma omp parallel  num_threads(2)
     #pragma omp for
     for(int i=0;i<100;i++)  {
            int id = omp_get_thread_num();
            printf("T%d:ai%d w=%d\n", id, i, w++);
      }
      printf("W=%d\n",w);
}
```

Include the following clauses in the *for*

5.1. *private(w)*

5.2. *firstprivate(w)*

5.3. *lastprivate(w)*

5.4. *reduction(+:w)*

Questions (5.0 is the base program execution):
5.0/5.1/5.2/5.3/5.4 Explain the initial value of w (for each thread) inside the loop
5.0/5.1/5.2/5.3/5.4 Explain the final value of w inside the loop
5.0/5.1/5.2/5.3/5.4 Explain the final value of w after the loop execution

Additional notes: Access to the search cluster

Login on search1:

```
ssh aXXXXX@search1.di.uminho.pt
```

Reserve a compute node:

```
qsub -I -qcpar -lnodes=1 -lwalltime=1:00:00  # 1 processor for 1h
```

File copy to/from search1:

```
scp <<file>>  aXXXXX@search1.di.uminho.pt:
scp aXXXXX@search1.di.uminho.pt:<<file>> .
```