Lab Guide 5

Introduction to shared memory programming and OpenMP

Objective:
- introduce the base concepts of shared memory parallel programming
- introduce the basic constructs of OpenMP

Introduction

These exercises consist of running a small C program with OpenMP directives to enable parallel execution (with two threads). Students should run the program multiple times, looking at the output of each run and try to understand/explain its output (e.g., the relative order of the `printf` statements). Specifically, for each OpenMP directive, students should answer to several questions related to what constraints are imposed. Note that the output order probably will change from execution to execution but OpenMP defined constraints are always valid for all runs.

The compilation and execution can be performed, as usual, on a compute node of the SeARCH cluster (login into the `s7edu2.di.uminho.pt` machine). The program should be compiled with the `-fopenmp` flag.

```
module load gcc/11.2.0
gcc -O2 -fopenmp ex51.c
```

Execution can be performed on a compute node of the cluster (partition “cpar”, two cores for our task):

```
srun --partition=cpar --cpus-per-task=2 ./a.out
```

Exercise 1- The basic OpenMP construct: the parallel region

Copy & paste the following code to a new file, compile, and run the code 2 or 3 times. This code contains a small loop that displays the loop iteration number and the identification number of the thread executing each iteration. Note that each thread has a private variable *id* and the OpenMP function *omp_get_thread_num()* returns a different number for each calling thread.

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

int main() {
    printf("master thread\n");
#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);
    }
    printf("master thread\n");
}
```

a) Is the order of the output always the same across multiple runs? Why?

b) Is the order of the output **OF EACH** thread always the same? Why?

c) How is the loop execution distributed (i.e., scheduled) between threads?

Exercise 2- Work sharing and synchronization

Successively introduce **one** of the following directives between the *#pragma omp parallel* and the "*for*" in code of exercise 1 and answer the following questions:

2.1. *#pragma omp for*

2.2. *#pragma omp master*

2.3. *#pragma omp single*

a) In 2.1/2.2/2.3, how is the loop execution distributed (i.e., scheduled) between threads?

b) In 2.1/2.2/2.3, the loop division is always the same?

2.4. *#pragma omp critical*

c) In 2.4, is the order of the output always the same? What kind of synchronization occurs?

Exercise 3- Synchronization

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

a) Is the order of the output always the same? What kind of synchronization occurs?

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*

b) The order of the output is always the same? What kind of synchronization occurs?

Exercise 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)*

a) In 4.1/4.2/4.3, how is the loop execution distributed (i.e., scheduled) between threads?

b) In 4.1/4.2/4.3, is the loop distribution (i.e., scheduling) always the same?