Synchronization

07/12/2016

1 Synchronization

This exercise is about implementing different ways to prevent data hazards occurring when multiple threads access the same data. The code to be parallelized consists of the following simple loop:

```
for(i=0; i<NIT; i++){
  j = rand() % NE;
  data[j] += func();
}
```

At each of the `NIT` iterations of the loop one element of the `data` array of size `NE` is chosen randomly and summed with the result on the `func()` function which is assumed to be expensive and take a relatively long time.

2 Package content

In the `synchronizations` directory you will find the following files:

- `main.c`: this file contains the main program that first calls the `sequential` routine containing the simple loop presented above and then calls three routines `parallel_critical`, `parallel_atomic` and `parallel_locks` that have to be developed and are meant to contain three different parallel implementations of the loop above as described below. **Only this file has to be modified for this exercise**.
- `aux.c, aux.h`: these two files contain auxiliary routines and **must not be modified**.

The code can be compiled with the `make` command: just type `make` inside the `synchronizations` directory; this will generate a `main` program that can be run like this:

```
$ ./main
```

3 Assignment

- ⌨ At the beginning, `parallel_critical`, `parallel_atomic` and `parallel_locks` are a copy of `sequential` routine. Modify these routine in order to parallelize the loop presented above; make sure that potential data access conflicts are avoided: in the first routine use the OpenMP `critical` construct, in the second use the OpenMP `atomic` construct and in the third use OpenMP locks to achieve this. Make sure that the result computed by the three parallel routines is consistently (that is, at every execution of the parallel code) the same as the sequential code.

- ✎ Report in the `responses.txt` file execution times for the sequential code and the three parallel routines using 1, 2 and 4 threads. Which parallle version is fastest? Can you explain these results? Report your comments and answers in the `responses.txt`.

Advice

- Different `atomic` operations are available in OpenMP: think about which type is better suited to the operation you have to protect.