Type count

February 20, 2023

Let's assume we have a big array containing `ndata` different data each being of a particular type `type` $= \{0, 1, ..., \texttt{ntype}\}$. These data must be analyzed to figure out which type they are. The objective of this exercise is to parallelize a code that analyzes all the data and counts how many data of each type exist in the array.

1 Assignment

- The `work` routine contains reference code showing how the typecount is achieved sequentially. The main loop of the `work` routine can be executed in parallel by multiple threads. Note that the `data` array can much larger than the number of types which means that there can be conflicts. Three OpenMP techniques can be used to resolve this conflict: critical sections, atomics and locks.
  - Write a parallel version of the `work` routine called `work_par_critical` where the main loop is parallelized and the conflict is resolved using critical sections.
  - Write a parallel version of the `work` routine called `work_par_atomic` where the main loop is parallelized and the conflict is resolved using atomic instructions.
  - Write a parallel version of the `work` routine called `work_par_locks` where the main loop is parallelized and the conflict is resolved using locks.

2 Package content

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

- `main.c`: this file contains the main program, the `work`, `work_par_critical`, `work_par_atomic` and `work_par_locks` described above. **This is the only file to be modified**.
- `aux.c, aux.h`: these files contain auxiliary routines and can be safely ignored.

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

```
$ ./main ndata ntype
```

where ndata is the number of data to be analyzed and ntype the number of possible types. When executed, it will initialize the data, call the work routine and then call the work_par_critical, the work_par_atomic and work_par_locks. Upon execution of each of these routines, the main program will print a message saying whether the result is correct or not and the time taken by the routine.

3 Assignment

- ⌨ At the beginning, the work_par_critical, work_par_atomic and work_par_locks routines are, simply, a copy of the work one. Modify these three routines to achieve the parallelization, respectively, using critical sections, atomic instructions and locks to resolve conflicts.