Modèles et techniques en programmation parallèle hybride et multi-cœurs: Travail pratique 1

Francesco Clerici

The exercise consists in the parallelization through OpenMP of an algorithm to solve PDE in 3D. In particular, the algorithm presents several nested loops which are devoted to the computation of the finite difference increment du at time n. The parallelization will focus on these loops. The machine used is a MacBook Pro with an 8-Core Intel Core i9 2,3 GHz. The runs are made up to 8 threads.

Fine-grain approach

First, we parallelize the loops contained inside the `values.cxx` file. Even if this file contains only initialization methods (`Values::init()`, `Values::init(callback_t f)` and `Values::boundaries(callback_t f)`), these could be relevantly time consuming for a low number of time steps or if the finite difference grid is reduced in size through the computation. Second, we parallelize the function providing the finite difference increment du. Since we are adopting a fine-grain paradigma, we simply add the OpenMP directive at the beginning of each for-loop every time. We should only give more attention to the loop inside the function `Scheme::iteration_domaine`. In order to avoid data race while computing `du_sum` we have to add the directive `reduction(+:du_sum)`.

Coarse-grain approach

Within the coarse-grain approach we write directly a parallel block inside the main function. Then, we take advantage of the functions providing local partitions of the domain to focus on a number of `nthreads` slices of the domain, each of measure $(0, \frac{1}{\texttt{nthreads}}) \times (0,1) \times (0,1)$. No matter how we choose the dimension on which we cut the domain. Consequently, before starting any loop on the domain points, we compute `imin_loc` and `imax_loc` of each thread as

```
int tid = omp_get_thread_num();
int imin_loc = m_P.imin_local(0, tid);
int imax_loc;
int nth = m_P.nthreads();

if(tid == nth-1)
  imax_loc = m_P.imax(0);
else
  imax_loc = m_P.imin_local(0, tid+1);
```

In this way we are able to call in a parallel way any (thread-safe) function depending on the indexing, e.g.

```
iteration_domaine(imin_loc, imax_loc,
                  m_P.imin(1), m_P.imax(1),
                  m_P.imin(2), m_P.imax(1));
```