Lab 1. OpenMP – generalities and simple examples

Main goals of the assignment

- Learn what tools to use for multi-cores and what for multiple servers;
- Learn how to use the multiple cores of a computer;
- Understand the problem of thread concurrency when writing in a global memory;
- Learn about the need of the sequential code changes when the multiple cores are used.

Overview

OpenMP versus MPI

We will use in our labs two software libraries: OpenMP and MPI:

OpenMP: to use all cores available on our deskop/laptops/servers to speedup the response time of a particular application;

MPI: to use the computation capacity of multiple computers (even millions of them, see Top500).

What is OpenMP

Please read the presentation from the textbook - page 226 (bottom of the page).

Instalation

For example, you can use the following guides:

- *In Linux/MacOs for command lines:* get the latest version of gcc, e.g. get the version gcc 4.6 that implements version 3.0 of OpenMP standard: `sudo apt-get install gcc-4.6`. On MacOs you can use also: `brew install clang-omp`
- *In Windows for command lines:* get Cygwin64 Terminal from https://cygwin.com/install.html that implements a Linux simulator and select OpenMP (and MPI) at installation
- *In Linux/MacOS/Windows for developing environment:* get Eclipse PTP for C/C++ from https://www.eclipse.org/ptp/ (both for OpenMP and MPI).

First example

1. Write the Hello world code from the textbook - page 250 (first in the section 9.4.1)
2. Compile the code; in command line, for example: > `gcc hello.c -o hello -fopenmp`
3. Run the code, e.g. > `./hello`. Uderstand the default value of number of threads and the role of `private` (run it with and without). Modify the number of threads with (a) environment settings (b) the specific function (c) the option on the pragma line.

Second example

1. The problem to solve: scalar multiplication between two vectors of dimension n, x and y:

$$< x, y >= \sum_{i=0}^{n-1} x_i y_i \tag{1}$$

   How to do it faster? Partial sums on different cores! It there are p cores, than core k $(0 \leq i < p)$ will compute

$$\sum_{i=k\cdot n/p}^{(k+1)\cdot n/p-1} x_i y_i \tag{2}$$

   if n can be divided exactly by p.

2. A `#pragma omp parallel for`, followed by a `for` that has independent steps, splits the cycle so that each thread receiving an almost equal number of cycle variable values. I.e. can be use to implement the partial sums (2).

3. Implement the parallel code. See the code from the same page as the first example. See also the small difference from the sequential code.

4. To understand the clause `reduction(+; dot_prod)` run the code with and without it for a large dimension of the vectors (e.g. 10 millions, dimension that is set, not read). [Hint: memmory congestion when the writting of `dot_prod` is performed by multiple cores]

5. In order to avoid the sitting of threads at queue to write in the global memory variable, the partial sums should be computed and using variables that are owned by the threads. Please rewrite the parallel code accordingly. Note that after all partial sums are computed, their sum should be done probably in a sequential manner.

6. Is there is a response time improvement? Hints: (1) Run the three versions of the programme (sequential – without pragma, parallel – with pragma and reduction, parallel with independent partial sums) for very large dimensions (e.g. 10 millions and random values instead 1.) and use `time ./executable-code` as command line.