

Assignment 3: Outer-loop parallelization

This assignment deals with the parallelization of different coding variants of matrix-vector multiplication using OpenMP. Assume, you are given $M \in \mathbb{R}^{N,N}$ and $x \in \mathbb{R}^N$. The product $y := Mx$ can be computed as

- $y_i = \sum_{j=1}^N M_{ij} x_j$, $1 \leq i \leq N$ ("**dot product variant**")
Strategy for parallelization: Instead of having one CPU computing all N components of y alone, the work can be split into p chunks, and every chunk can be processed by its own CPU.
- $y = \sum_{j=1}^N x_j M_{*j}$, where M_{*j} refers to the j -th column of M ("**saxpy variant**")
Strategy for parallelization: Each CPU computes its own part of the sum (= partial sum) and stores the result in a private variable. Then, the p partial sums are added together to get y .

To do:

- Download the file `mv_mult.c` from the FTP download site of the course and copy it to your account on Brutus.
- Set up your environment using module. Compile the file by either

```
gcc -O3 -fopenmp mv_mult.c -lm
```

or

```
icc -O3 -openmp mv_mult.c -lm
```

- Next, you should set the environment variable `OMP_NUM_THREADS` to p , where p is the number of CPUs you want to use later. The command for this purpose is either `export OMP_NUM_THREADS=p` or `setenv OMP_NUM_THREADS p`, depending on your shell. Find out which shell you use by `env |grep SHELL`. The man page of the shell will tell you whether to take `export` or `setenv`. Make sure the environment variable `OMP_NUM_THREADS` has been properly set by entering `env |grep OMP_NUM_THREADS`.
- Submit your program to the queues using: `bsub -n p -o output ./a.out`
Here p is no. of processors. Please note that this number must be equal to number of threads you have requested using `OMP_NUM_THREADS`. Check file `output` for the results.
Don't be surprised when the indicated speed-up is about 1.0. This is because the parallel versions for computing y_2 and y_4 , which are mentioned in the output, are so far mere copies of the serial ones.
- Change the code so that it uses outer-loop parallelization for both **dot product** and **saxpy** variant (the related strategies are outlined above). To this end, familiarize yourself with the OpenMP *directives* `parallel`, `for`, `critical` and `parallel for`. As *clauses*, you will need `private`, `shared` and maybe `schedule`. See the OpenMP documentation for details.

Bitte wenden!

- When you have made the changes, try to run your program for different values of p as described. Please do not use more than 16 processors. Check for the numerical correctness of your results. What is the best speed-up you can achieve?

If you encounter any problems, don't hesitate to contact the assistant via the discussion board on the lecture's homepage. When you have finished the assignment, mail your solution (the whole C file and distilled results) to harish@math.ethz.ch with subject `pnc exercise`.

Due Date: 30th April 2010

Webpage: <http://www.math.ethz.ch/education/bachelor/lectures/fs2010/math/pnc>

Organizers: Sohrab Kehtari (sohrab.kehtari@math.ethz.ch)

Harish Kumar (harish@math.ethz.ch)