Lab 8

Math 9830 Timo Heister, heister@clemson.edu

Note: Unless specifically asked to submit a solution, just work on the exercises and keep track of your progress in your journal.

1. Familiarize yourself with `02_threads_hello` and `04_threads_ex1` and run them on your computer.

2. Determine the number of (virtual) cores in your machine (hint: `/proc/cpuinfo`). Figure out if your machine uses hyperthreading. Also report what `std::thread::hardware_concurrency()` returns. **Report in your journal.**

3. Implement multithreaded vector addition based on `05_threads_ex2` and find the optimal number of threads for your machine (try anything between 1 and twice the number of cores in your system, the command line tool `time` might help to see how fast your code runs).

4. It is of course a lot more useful to have a function for computing the sum of two vectors that does the multi-threading internally. Make that change. Your function might have the following interface
`void add(Vector &destination, const Vector &left, const Vector &right)`.

5. Vector addition does not require any communication or synchronization between threads and is therefore the easiest numerical operation to implement in a multithreaded way. Implement a parallel version of the function `norm`. You have three options to accumulate the result: 1) accumulate sequentially after all threads finish. 2) synchronize using a mutex. 3) use an atomic variable for accumulation. **Submit the final version that includes the solutions to the last three questions and report the speed up you see over a sequential computation in your journal.**

6. Run the program `mpi-hello` in parallel. What happens when you run with more MPI ranks (processes) than your computer has cores? Is this a good idea?