HIGH PERFORMANCE PROGRAMMING
UPPSALA UNIVERSITY
SPRING 2017
EXAMINATION MINIPROJECT 2

1. Instructions

Your task is to optimize a given program that performs a simple quantum mechanics computation with a harmonic oscillator-like potential using a finite-difference approximation approach, and then compares the result to an analytical solution.

The code you are given to start with works, it computes the desired results, but it can hopefully be optimized. Use your knowledge from the course to optimize the code as much as you can.

The program expects the following input arguments:

- `mass` – the mass of the particle in the simulation
- `omega` – the parameter ω affecting the potential shape in the simulation
- `N` – the number of grid points to use
- `M` – the number of power-method iterations to perform
- `nThreads` – the number of threads to use

You should try to optimize the program, but you should not change the input arguments, they should still have the same meaning as in the original code.

You are allowed to change the structure of the program, the functions and data structures used etc. if you find that such changes can improve performance. However, the program should still use the power method for the given number of power-method iterations.

1.1. **Step 1: add the missing dependencies in the Makefile.**

The given Makefile can be used to build the code, but some dependencies are missing, so it does not recompile the code properly when source files have been changed. Fix this, so that the makefile works properly also when some source file(s) are changed.

1.2. **Step 2: optimize the code.**

Use all your knowledge from the course to make the code run as fast as you can; this includes both code changes and changing compiler options in the Makefile. You can choose to use either Pthreads or OpenMP for parallelization, using the number of threads that was specified as input to the program.

You are free to use any computer you want for this miniproject, you can run on the lab computers at the university, or on your own laptop, or some other computer. However, your code must anyway be portable, you should verify that it can be compiled and executed on either the lab computers or the university's Linux servers, e.g. `fredholm.it.uu.se`.

Date: May 23, 2017.

If you want to do some optimizations that mean the code will not run on the lab computers or `fredholm.it.uu.se`, then you should include that code as an option, the code you submit should by default work as a portable code. Then you can describe in your report how to enable your non-portable optimizations, and describe what the hardware requirements are.

As you work on optimizations, remember to check that the results are still correct. To check the correctness of your optimize code, compare the results to what you get with the originial, unoptimized code. The computed `energy_diff` (on the last line of output from the program) should not differ by more than 10^{-4} from the `energy_diff` given by the original code, for the test cases in Section 1.3.

1.3. **Test cases.** Check that your optimized code gives correct results for the following test cases:

Case A:

```
mass = 4.5   omega = 5.0   N = 300   M = 16000
```

Case B:

```
mass = 7.5   omega = 5.0   N = 1000   M = 4000
```

Use the above test cases to verify accuracy. When checking the efficiency of your optimized code, consider also larger test cases. In particular, try to make the code efficient for larger values of `N`.

1.4. **Report.** Write a short summary, 2-3 pages, discussing the performance of your code, showing the effectiveness of your optimizations and your parallelization. If an attempted optimization does not work and you know why it does not work, include this in your report. *Hint: figures and tables are good ways of presenting results concisely.*

Note: max 3 pages for the report! It is important to be able to present results consisely, show us that you can do that. Reports longer than 3 pages will not be accepted.

1.5. **Format of submission.** Your submission should consist of a single compressed tarball named `miniproj.tar.gz` including your code and report files:

- Runnable code in a directory called `code`
- Report in pdf format, maximum 3 pages long, filename `report.pdf`

The code you submit should be your best code with the best compiler options in the makefile. We should be able to test your code by unpacking your submitted `miniproj.tar.gz` file and going into your `code` directory and simply doing "make" and then running the program.

You should verify that your submission has the correct form in the following way: create a new directory and copy your `miniproj.tar.gz` file into that directory. Then cd into that directory so that if you do "ls" you see only those two files listed. Then type *exactly* the following commands:

```
tar -xzf miniproj.tar.gz
cd miniproj
cd code
make
./fd 4.5 5.0 300 16000 3
```

Then your optimized code should run with the given parameters, corresponding to "Case A" above, using 3 threads.

Note: your submission must follow exactly the requested format. Part of our checking of your submissions will be done using a script that automatically unpacks your file, builds your code, and runs it for some test cases, checking both result accuracy and performance. For this to work, it is necessary that your submission has precisely the requested form. Verify that your submission follows the requested format by testing it as described above.

If you want to demonstrate different versions of your code, you can include separate directories for other versions, for example as "code2", but the `code` directory should anyway contain your final result, your best code.

2. Submission and Examination

The exam will take place on Friday, June 2, in room *2507*.

On the days of the exam, you will be interviewed in turn, in front of the computer. This is your opportunity to show us what you have learned! We will ask questions about this miniproject and your code, as well as some other questions about the course contents.

If you want to take the exam, we ask you to submit the code and report under "Examinatory miniproject 2" in the Student Portal by the *May 31, 23:59*. After submitting your miniproject you will get to pick a time slot for the oral examination, some time on June 2. The Student Portal will be used for that, students get to choose time slots on a "first come, first served" basis.

Important: the exam work should be carried out *individually*. You can talk to other students, but don't do it in front of a computer and don't write anything down. The bottom line is that we have to be convinced that the handed-in code and report are done by you and understood by you.

Also important: Bring an ID card, passport or equivalent, on the exam day so that we can verify that you are who you say you are.

3. Grading

You will be graded according to a (secret) rubric based on the complete course contents. Both your miniproject submission and your answers to questions (both specific questions about this miniproject and questions about other course contents) will be taken into account. The grade requirements are as follows:

- **3** You must demonstrate an understanding of the fundamentals of the course. Your code works correctly and you have made a reasonable attempt at optimizing and parallelizing the code.
- **4** You have tried to optimize everything, and you're able to reason about the performance of your code at a high level and know why certain optimizations worked/didn't work.
- **5** Additional requirement for the highest grade: you should in addition to optimizing the given code using the power method also implement an alternative code calling a suitable LAPACK routine instead of using the power method. Briefly discuss and compare the two approaches in your report.

4. Questions

If there are any questions, email: `elias.rudberg@it.uu.se`