Assignment1: Heatdiffusion in openMP

Given is a sequential C program `stencil.c` for implementing the approximation of a simple heat diffusion scheme in 1D (three-point stencil).

It iteratively computes a vector of size n from an initial vector of size n. That vector has all values set to `0.0` but the first and the last value. These values are initialised by heat values `100.0` and `1000.0`, respectively. The iteration step recomputes all non-boundary elements by computing a weighted sum of the current values of the element itself and its two immediate neighbouring elements. The number of iterations is provided as a parameter of the program.

We also provide an optimised version `stencil_opt.c` which is a more cache-friendly implementation. If you are looking for a challenge, try to parellelise this version as well for better scaling and absolute performance.

You have to hand in the assignment on Brightspace in groups. You will get feedback, but no grade.

This assignment explores how this algorithm can be executed on a multicore system using openMP.

The programs, as well as a hello world program, Makefile, and some example scripts for running programs on the cluster are provided on `https://gitlab.science.ru.nl/parallel-computing2/parallel_programs.git/`. There is a README, read it!

Your tasks are to:

- analyse the performance of the sequential program
- develop an openMP version and explore the effectiveness of your version
- provide an extensive performance analysis

This assignment should be done by teams of 2-3 students. How you distribute the work within the team is up to you. However, you need to declare who did which part. All performance measurements need to be done on a single system with at least 4 cores. You have access to a 32 core machine via slurm22. You need to make sure you provide all relevant technical details of that system. Make sure that you:

- specify exactly what hardware is being used (CPU version, clock frequency, memory, etc.)
- specify exactly what software is being used (compiler version, compiler flags, etc.)
- specify exactly which parameter (size n and number of iterations) you are using
- repeat each experiment at least 3 times and report average time as well as the variability (error bars)

Task 1: Sequential Evaluation

Evaluate the performance of the sequential code. Use the highest level of compiler optimisation on your machine. Typically, this is `-Ofast` but you should look into the man pages of your compiler; Look for other flags such as `-march=native`. Present the wallclock time of the work-loop as a function of the vector-size n and the number of iterations. Present the absolute performance in `GFLOP/s` as a function of the vector-size n and number of iterations. Make sure that you vary the parameter n of your vectors so that the sizes of your vectors vary from a few kB to something as large as 5GB. Note that one `double` requires 8 bytes. You may have to adjust the iter parameter for your machine to obtain a reasonable range of runtimes.

Analyse any anomalies this function may show. You may want to use tools such as gprof, perf stat, valgrind, or the gperftools to find out what is going on. Summarize your findings in a few paragraphs.

Task 2: openMP

Add openMP pragmas and library calls to your sequential C version. Repeat the evaluations from Task 1 for the openMP version with different numbers of threads. Make sure that you instruct slurm to provide you with sufficient cores and sufficient memory!. Run several experiments to figure out what impact the different scheduling strategies have and present all these in a single graph. Try to analyse the performance and try to explain your findings. You may want to read up details of the openMP version you are using on www.openmp.org.

Task 3: Performance

Provide a discussion of your overall findings. This should include figures reporting speedups, scaling (strong and weak), and efficiency. Can you explain what limits your speedups? Use a roofline plot to explain your findings.

Furthermore, you should discuss and compare the effort that was required to achieve these figures (programming effort and debugging effort).

Task 4: Team

Provide a short description on how you divided up the work, i.e., who did what? Attribute percentages to your overall contributions.

Page limit:

At most 4 pages excluding tables and figures.