Assignment1: Heatdiffusion openMP and MPI

Given is a C program for implementing the approximation of a simple heat diffusion scheme in 1D (three-point stencil).
This assignment explores how this algorithm can be executed on a multicore system using two different technologies: openMP and MPI.
Your tasks are to:

- explore how to optimise the sequential program
- develop an openMP version and explore the effectiveness of your version
- develop an MPI version and explore the effectiveness of your version
- provide an extensive performance analysis

This assignment should be done by teams of 4 students. How you distribute the work within the team is up to you. However, you need to declare who did which part. Ideally, you perform all performance measurements on a single system. If that constitutes a logistic problem, you can use different hardware provided you clearly specify which hardware has been used for which measurements.
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 parameters (HEAT and EPS) you are using
- repeat each experiment at least 5 times and report average time as well as the variability (error bars)

1 Task 1: Sequential Evaluation

Evaluate the performance of the sequential code. Use the highest level of compiler optimisation on your machine. Typically, this is -O3 but you should look into the man pages of your compiler. Present the wallclock time of the work-loop as a function of the length of the input vector. Make sure that you vary the length of your vectors from sizes of a few kB to something as large as your machine permits with reasonable runtimes. You may have to adjust the HEAT parameter for your machine to obtain a reasonable range of runtimes. Higher HEAT values lead to longer runtimes.

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

Try to improve your program. Ideas: you can combine the loops for the relaxation and the stability check, you can stop checking for stability as soon as the first difference bigger than EPS is found. Feel free to come up with further ideas. Analyse the impact of your optimisations by repeating the previous exercise.

2 Task 2: openMP

Add openMP pragmas and library calls to your sequential C versions (non-optimised and optimised). Repeat the evaluations from Task 1 for the openMP version. Run several experiments to figure out what impact the different scheduling strategies have and present all these in a single graph. Again, try to analyse the performance and try to explain your findings using profiling tools.

3 Task 3: MPI

Rewrite the program to leverage MPI. Try to optimise your version considering both sequential versions. Repeat the evaluations from Task 1 for your MPI versions quantifying the effects that your own optimisation attempts have (scheduling, placement, communication pattern, etc.).

4 Task 4: Performance

Provide a discussion of your overall findings. This should include figures reporting speedups, scaling (strong and weak), and efficiency. Furthermore, you should discuss and compare the effort that was requires to achieve these figures (programming effort and debugging effort). Finally, you should try to explain your findings and try to come up with possible further directions of investigation.

5 Task 5: Team

Provide a short description on how you divided up the work, i.e., who did what?

Hand-out: 20/05/2020 Submission deadline: 02/06/2020 24:00