NTNU

Norwegian University of
Science and Technology

DEPARTMENT OF COMPUTER SCIENCE

TDT4200 - PARALLEL PROGRAMMING

Exercise 4

1 Introduction

In this exercise, we will parallelise the 2D wave equation using threads. We will explore the libraries `Pthreads` and `OpenMP`.

2 Tasks

2.1 Programming

For the programming, you will parallelise the code twice, once for Pthreads and once for OpenMP.

We have simplified the grid a to a NxN grid and substituted the spatial stepsize $dx = dy = h$. We will also not work with user input such as time iterations, snapshot frequency, and grid size.

Before you start the exercises, it is recommended to run the `make check` command as a sanity check for the baseline. Both OpenMP and Pthreads should produce identical output with a single thread as the sequential version.

A note on optimization flags in the `Makefile`. On a machine where the length of vector registers is comparable to the number of cores, an aggressively optimizing compiler can vectorize most of the computation and achieve the same speed with 1 thread as with 4-8 threads (depending on the configuration).

Therefore, -O2/-O3 optimization flags are not included in the Makefile, and increasing the thread count should produce performance improvements on any multi-core machine with the default settings.

It can be interesting to add -O2 or -O3 to see what happens on a particular computer, but if it is done without awareness of this effect, it can create the impression that the threads are not doing anything.

2.1.1 Pthreads

With Pthreads, you can run the code with a custom amount of threads using the following command

```
./parallel [size]
```

Listing 1: Run you code with different amount of threads

Note that the `simulate` function now returns a `void *` to be used in the creation of threads.

1. **Initialise Pthreads:**

   We need to include Pthreads and do the basic preparations to get started.

   a Include the Pthread header.

   b Declare variables that the threads will use. They can include, but are not limited to:

   - `n_threads`
   - `barrier`

   c Initialise the Pthreads barrier.

   d Destroy the Pthreads barrier.

2. **Run the integration loop:**

   We need to spawn threads to parallelise the simulation.

   Create threads for the `simulate` function and join them together.

3. **Time Step:**

   Currently, every thread is going through every cell. Recall that every cell is stored in one sequence in memory (pr time step). One way of dividing the workload is illustrated in Figure 1.

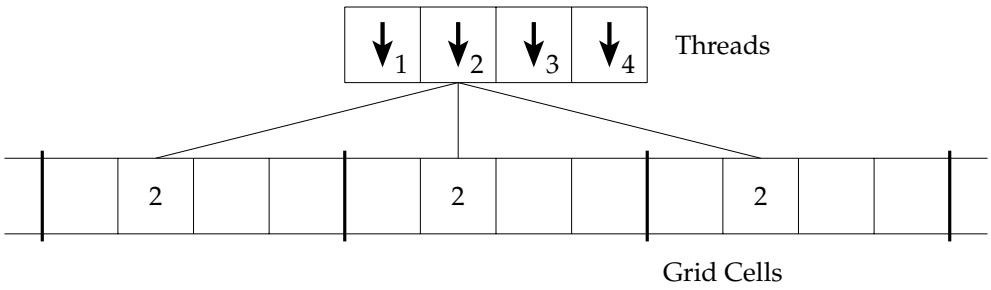

Figure 1: Thread number 2 calculates the cells one vector length apart.

   Change the `time_step` function so that the threads divide the workload.

4. **Remember the Boundary:**

   Same as with the time step, the calculation of the boundary condition is not split between the threads.

   Update the boundary condition such that the workload is divided between the threads.

5. **Simulate:**

   Now, it's time to combine the simulation.

   > Update the `simulate` function and have the threads calculate their part of the simulation and ensure that only one thread stores the simulation's state.

2.1.2 OpenMP

Now, it's time to parallelise the 2D wave equation with OpenMP. You will notice two files in the OpenMP folder, `wave_2d_barrier.c` and `wave_2d_workshare.c`. The programming tasks will only be done in the `wave_2d_workshare.c` file.

You can run the code with the following command

```
./parallel
```

Listing 2: Running the workshare program

6. **Initialise OpenMP:**

   We need to include OpenMP to get started.

   > Include the OpenMP header.

7. **Parallelise the Main Calculation:**

   OpenMP has a very powerful way of parallelising loops.

   > Parallelise the `for` loops in the `time_step` function.

2.2 Theory Questions

1. Why is there no need for a border exchange when using Pthreads?
2. What is the difference between OpenMP and MPI?
3. Comment on the difference between Pthreads and the two OpenMP implementations.
4. How would you parallelise a recursion problem with OpenMP?

3 Deliverables

Please deliver `.zip` file with the following contents containing your changes to the code:

- `wave_2d_pthreads.c`
- `wave_2d_workshare.c`
- A document with your answers to the theory questions.