Assignment 2

Objectives

In this assignment, we continue to explore parallel programming in OpenMP. We will apply different software optimizations we learned to algorithms operating on two-dimensional arrays.

Assignment

In this assignment, you will implement a heat transfer simulation. Assume we have a square surface with a heat generator at its center and a heat sink at its edges. The hot core transmits energy and remains at maximum temperature. The heat sink temperature remains zero. Initially, all other points on the surface are at zero degrees. This is illustrated in figure 1. Over time, the temperature changes at each point on the surface. Figure 2 illustrates what the temperature distribution will look like after a period of time.

We provided a reference single threaded implementation of the heat transfer algorithm in the handout. The algorithm iteratively updates the temperature grid, with each element temperature being calculated from a linear combination of the temperature of its neighbors. You will write a parallel program using OpenMP that simulates the heat transfer example presented above and optimizes it using the various software optimization techniques you learned in the fourth lecture. The program will operate on a variable sized square-shaped two-dimensional array and will run the simulation for `N` iterations. For simplicity, the side length of the square is always even. The program must write the value of each element in the two-dimensional array in a comma-separated values (CSV) file, to check for correctness. We already provided all the code necessary to generate the CSV file. The program must also report the time it takes to run the algorithm for `N` iterations to measure performance.

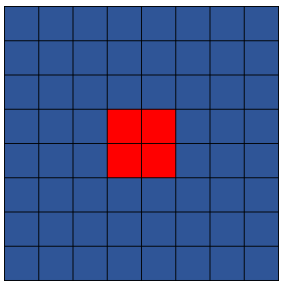

Figure 1: Initial state of the surface

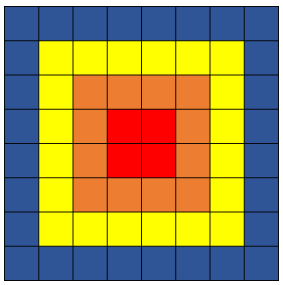

Figure 2: The surface after a period of time

Development Infrastructure

On the course's Moodle page, we have provided the tarball `A2.tgz`, which contains the following files:

- `assignment2.c` – the file containing the `main()` function that will call your code.
- `utility.h` – which contains a set of helper functions:
  - `set_clock` & `elapsed_time`: to measure the start and end of execution.
  - `init`: initializes the two-dimensional array's values and sets the initial state.
  - `save`: saves the two-dimensional array in a CSV file.
- `algorithm.c` – which contains the `simulate` function. This function applies the algorithm in question on an input two-dimensional array for `N` iterations. It updates the values of the array accordingly and writes them to an output two-dimensional array. It is useful to interchange the arrays between cycles, using one to hold the output of the previous iteration and the other to hold the output of the current iteration. **You must parallelize and optimize this function.**
- `Makefile` – which compiles your code; please refer to Assignment 1 to make sure the `Makefile` is suitable for your environment.
- `execute.sh` – A sample script to submit jobs to the SCITAS cluster.

The environment will operate as follows:

1. The program file is called `assignment2.c`.
2. Compiling the program using the provided `Makefile` produces the `assignment2` binary executable file.
3. The input arguments are, in this specified order:
   a. Number of threads
   b. Side length
   c. Number of iterations (`N`)
   d. Output file name
4. The program will save the output, after running `N` iterations, in `<OutputFileName>`

An example of running the program and the output it produces is shown below:

```
# /bin/bash
$ ./assignment2 4 1000 5000 output.csv
Running the algorithm with 4 threads on 1000 by 1000 array for
5000 iterations took 10.73 seconds
$ls
output.csv ...
```

Using the environment provided will guarantee that your program will comply to all specifications mentioned.

Deliverables

To submit your code and your report, create a tarball archive **(this is the only accepted format!)** called `a2_<yourGroupID>.tgz` and upload it on Moodle. Your file must contain:

- A parallelized and optimized version of `algorithm.c`.
- A report, with the name `a2_<yourGroupID>.pdf` , that complies to the following description.

Report

In the report, we expect you to perform the following tasks:

1- Explain how you parallelized and optimized `simulate`. List the optimizations you applied and the reasons you chose to apply them. Note that the program is trivially parallelizable, so we are more interested in the memory optimizations than in parallelizing the program itself. Identify the different ways of splitting the work among threads, and which ways result in more data locality within threads. Also identify whether there is any issue with load balancing when dividing the work.

2- Report how much each optimization improves performance and explain the result obtained. Design and run experiments that isolate the effect of each optimization/thread organization you tried out and then report the results.

3- Present the execution time you measured in a graph for the following set of thread counts {1,2,4,8,16} running for 100 iterations and side length of 10000.

The report should be as precise as possible, addressing the three questions above. Keep the descriptions of the algorithm implemented and of your reasoning in parallelizing the program short. A regular report should not be much longer than 2-3 pages.

Grading

The code will be graded automatically by a script and checked for plagiarism. The script will check how the running time scales with number of threads and if the results returned are consistent with what was expected. Plagiarized code will receive 0. We will escalate plagiarism cases to the student section.

The reports will be read and graded by humans and checked for plagiarism automatically.

The grade breakdown is as follows:

- Correctness: 50%
- Report: 50%