CSC4005 High Performance Computing (2018/19)

Assignment 1 - Sequential Searching and Parallel Searching using OpenMP

Contact: Dr Blesson Varghese
b.varghese@qub.ac.uk

Deadline: Thursday, October 25, 23:59

Part A - Sequential Searching

Specification

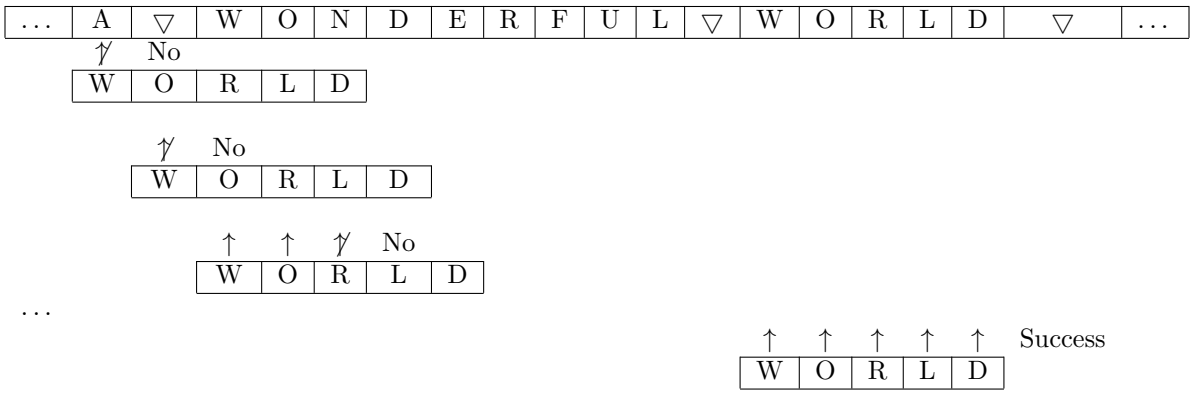
There are two non-empty sequences of characters known as the *text* and the *pattern*. The pattern is usually much smaller than the text. A program is required to determine whether the pattern occurs as a contiguous subsequence of the text and, if so, what is the position of the first character of the first occurrence.

Example

Suppose the text is `I THINK TO MYSELF WHAT A WONDERFUL WORLD`. If the pattern is `TERRIBLE` then it does not occur as a subsequence of the text. If the pattern is `WONDERFUL` then it occurs at position 25 (indexed from 0). It the pattern is `W` then it occurs at positions 18, 25 and 35 and the required answer is 18.

Algorithm

You are to use a *straightforward* pattern matching algorithm in which each potential position of the pattern is examined in turn



There are other, more sophisticated, algorithms with better worst case performance (e.g. *Boyer-Moore*), but in later assignments you will be asked to parallelise the straightforward algorithm rather than investigating other algorithms. You may NOT use techniques, such as hashing in the straightforward algorithm.

Implementation

You will be provided with a compressed tar file `Assignment1a-dir.tar.gz`. This contains a program `searching_sequential.c` (which implements the straightforward algorithm) and some example test cases.

The program looks for a folder called `inputs` in which is a collection of folders called `test0`, `test1`, `text2`, ..., one for each test case. Each test case folder contains the files `text.txt` and `pattern.txt` containing the test and pattern data. You may assume that all the characters are spaces or upper case letters. Two sample folders, `test0` and `test1`, have been provided for you, as have a compile script and a run script.

You should carry out each of the following steps:

- Transfer `Assignment1a-dir.tar.gz` to `kelvin.qub.ac.uk` using FileZilla or another scp facility.
- Uncompress `Assignment1a-dir.tar.gz` using `gunzip`, extract the contents of the archive file using `tar` and use `cd` to move into the `Assignment1a-dir` directory.

```
$ gunzip Assignment1a-dir.tar.gz
$ tar xvf Assignment1a-dir.tar
$ cd Assignment1a-dir
```

- Examine the contents of the directory and its subdirectories. In particular, examine the program `searching_sequential.c` and ensure that you understand how it works.
- Compile and run the program interactively, as follows,

```
$ ./compile searching_sequential
$ ./searching_sequential
```

- Run the program in batch using the `run.qsub` script. Remember to edit this file to use your own folder names. The output will be sent to the file `searching_sequential.out`. You can check on the status of the job using the command `qstat`.

```
$ qsub  run.qsub
$ qstat -u username
```

Assignment

- Determine 20 patterns and texts which will result in the worst case performance and note the CPU execution time for different length(text)*length(pattern).

  The product, length(text)*length(pattern), should be approximately equal to, 10^2, 10^4, 10^6, 10^8 and 10^{10}, and for each product you should construct 4 pattern-text examples. For example, for a product of 1,000,000 the pattern lengths could be 1, 10, 100 and 1,000 with the corresponding text lengths being 1,000,000, 100,000, 10,000 and 1,000.

  You will need to write a simple C program to generate the text and pattern files. As a guide you should find that the worst case execution time for a product length of 10^{10} is over 10 seconds.

  The total file size should NOT be more than 100 Megabytes (MBs). There will be penalties for exceeding 100 MBs.

- Edit the compile script so that no compiler optimisation is used i.e. use -O0 rather than -O2. Repeat your 20 tests.
- Plot a maximum of 4 scatter graphs of CPU execution time against length(text)*length(pattern) to best capture the results from the above tests.

  Hint: You will need to think of the best way to organise 4 graphs given the data points you collected. You may not be able to plot all data points you collected.

Part B - Parallel Searching using OpenMP

This second part builds on Part A: please re-read carefully the specification given in Part A. The primary aim is to design, implement and experiment with OpenMP programs to perform the same search process. You are provided with three new test case folders (`test0`, `test1` and `test2`) in `Assignment1b-dir.tar.gz` to perform your experiments.

Note: Programs **must** be executed using batch jobs on Kelvin.

Test0

In this test case you may assume that there is only one occurrence of the pattern in the text.

1. Run the program `searching_sequential.c` on `test0` as follows, `$time ./searching_sequential` Note the elapsed (i.e. wall-clock) time taken and the number of comparisons reported (which may be negative because of integer overflow). The elapsed time will be used as a reference point for comparison with your OpenMP implementation.
2. Re-engineer the `searching_sequential.c` into an OpenMP program, `searching_OMP_0.c`, based around a parallel for loop in function `hostMatch`. Note: Only two shared variables may be set in the `for` loop: one to compute the total number of comparisons made; and one to record the position of the matching point. There will be penalties. Experiment with your program using differing numbers of threads and scheduling strategies. Ensure that the number of comparisons reported and the position of the pattern match the sequential program.
3. Run `searching_OMP_0.c` with 1, 2, 4, 8, 16, 32 and 64 threads for different scheduling strategies.
   a. Tabulate the elapsed times and numbers of comparisons.
   b. Draw a graph of the parallel speedup (PS) vs the number of threads. PS is the elapsed time taken by the sequential program divided by the elapsed time taken using P threads.
   c. Draw a graph of the parallel efficiency (PE) vs the number of threads. PE is simply PS/P.

Test1

In this test case you may also assume that there is one occurrence of the pattern in the text.

1. Run the program `searching_sequential.c` on `test1` and note the elapsed time taken and the number of comparisons reported.
2. Run your OpenMP program, `searching_OMP_0.c`, on the same test case using two threads.
3. Experiment with the two programs, adding extra debug statements as you deem necessary. On the basis of your experiments explain the difference in the observed performance.
4. Create an OpenMP program `searching_OMP_1.c` in which you should attempt to reduce significantly the number of comparisons performed. You will need to describe the parallel algorithm used.

Test2

In this test case you may not assume that there is only one occurrence of the pattern in the text.

1. Experiment with `searching_sequential.c` and `searching_OMP_1.c` on the final test case, `test2`. Vary the number of threads and add extra debug statements as you deem necessary. Comment on your results and on the characteristics of `test2`.
2. Create `searching_OMP_2.c` which should attempt to minimise the number of comparisons performed and correctly detect the positions of the pattern as required by the specification in Part A.
3. Run `searching_OMP_2.c` with 1, 2, 4, 8, 16, 32 and 64 threads. You will need to report your results taking performance into account and comment on them.

Submission

You should submit your assignment to Queen's Online as a zip file named ⟨`student-number`⟩`.zip`. This should contain:

- A folder `assignment-1a` which contains:
  - A folder named `data` containing your test cases. Remember, there should be 20 as organised in the original folder provided to you. The total file size should be no more than 100MBytes.
  - A folder named `source` containing the source code and any scripts used.
- A folder `assignment-1b` which contains:
  - A `source` folder, which contains the source code of three OpenMP programs.
  - An `output` folder, which contains the output of executing your three programs and/or screenshots of the output.
- A high quality report in which:
  - For Part A of the assignment you describe the characteristics of a worst case pattern and text, selection of patterns and text, present your 4 graphs, and interesting things done or observed; and draw any conclusions.
  - For Part B of the assignment you must address all points raised in the tests.

  The report must be submitted in PDF format. The content must be organised in relevant sections.

Note: The folder and program names should be as specified. There will be penalties for not adhering to the naming convention provided. For Part B do not include the input folder containing the test cases with your submission.

The code and associated data or output carries 60% of the mark and the report has a 40% mark.

All the best!