Scalable Cross-Platform Software Design: *Optimisation Coursework#2*

25% of the module mark in total: Marking Scheme and submission deadline is on Moodle

Part 1: Code Optimisation.

In this task the assessment allows you to demonstrate an appreciation of and an ability to improve computational performance by changes in serial coding style. The split into 3 parts below specifically separates your ability to (a) identify the issues, (b) to implement a better solution and (c) to succinctly and clearly report your work.

Summary: On Moodle there is a very poorly written computational program, ToBeOptimisedCode.cpp. It is a “made up“ code that doesn’t actually do anything useful, but it has many features common to a range of large-scale numerical simulation codes in engineering. This is a rather poorly written code for a number of reasons!

Objective: Your task is to improve the C coding of the algorithm to minimise the run time; you must not change the code’s function in any way. Changing language, direct use of machine code or parallelisation is not allowed.

Please make sure you get exactly the same results: run the original code as above and make a copy of the file produced, *data_out*. Then for any new version check for identical results by doing,

diff data_out original_ data_out

You should get no differences at all. (To install the *diff* utility do *pacman -S diffutils* in MSYS2)

Specific Assessment Requirements:

Submission through the Moodle page by the deadline shown there. Submit a single zip file containing *all* files required to simply load and run your project. A brief pdf document is required that briefly describes:

a) A copy of the original listing annotated with brief comments to identify poor programming style and where you think there is scope to improve the code: comments should cover both “coding bad practice” and “speedup” possibilities.

b) Your optimised code (suitably documented).

c) A brief (1/2 page) pdf summarising the results of your optimisation including discussion of modifications that you tried that did not yield benefits and any resources (e.g. profiling) that you drew on.

Part 2: Characterisation of your compute platform.

In this task the assessment allows you to demonstrate an understanding of how the performance of a simple code is impacted by the details of the hardware configuration. Moreover, you can show your skills at interpreting the results of simple tests to understand scaling issues with codes. Your ability to implement and assess the performance of basic parallelisation to improve speed is being explored and in particular, your skills at making choices regarding the parameters of the parallelisation method are being considered. As usual, your ability to succinctly and clearly present your work forms part of the assessment

Setup a very simple serial C program to add two double precision arrays.

a[i]=b[i]+c[i+offset]   for all 0<i<n-offset.

The arrays are to be dynamically allocated. The arrays b and c are to be initialised so that b[i]=c[i]=i

Time the execution of the loop above (i.e. excluding the array setup and initialisation).

To get reasonable timing accuracy it may be necessary to wrap an outer loop around the above loop so as to repeat the above sum-loop many times.

To do the timings you might use *gettimeofday*, but there are alternatives – if you trust them and they are sufficiently accurate.

```
#include <sys/time.h>

struct timeval start_time,end_time;

gettimeofday(&start_time,NULL);

// Your code to time

gettimeofday(&end_time,NULL);

cout<<"\n\ngettimeofday wall time="<<
        end_time.tv_sec - start_time.tv_sec+(end_time.tv_usec-start_time.tv_usec)/1e6;
```

Specific Assessment Requirements:

Clearly state the *relevant* hardware details of the platform on which you obtained the results

a) With offset=0, investigate and plot the timings for different n.

b) Based upon the previous results, select a few "interesting" values of n and repeat for different offset values.

c) Now repeat (a) and (b) using OpenMP to parallelise the loop.

d) You should expect that static scheduling will give the best performance for (c). Please design, implement and characterise an alternative loop-based test where dynamic scheduling might prove advantageous.

Required results: Submit both your test codes and the results obtained and in a brief report (< 2 pages excluding graphs) summarise, discuss and explain what you observe.

Marks and feedback will be provided 15 working days after the submission deadline; each student will receive an individual copy of the marking rubric posted on Moodle by email. If required, further individual feedback is then available by requesting a one-to-one meeting.