COMP1680 (2019/20)	**Clouds, Grids and Virtualisation**		**Contribution: 60% of course**
Course Leader: Dr Catherine Tonry	**OpenMP Coursework**		**Deadline Date: Friday 11/12/2020**

This coursework should take an average student who is up-to-date with tutorial work approximately 30 hours

Feedback and grades are normally made available within 15 working days of the coursework deadline

Learning Outcomes:

Characterise and critically evaluate high performance computing based architectures and their suitability for given applications.

Implement and execute applications using shared and distributed memory programming paradigms.

Describe and critically discuss the roles and applications of cloud and grid computing.

Plagiarism is presenting somebody else's work as your own. It includes: copying information directly from the Web or books without referencing the material; submitting joint coursework as an individual effort; copying another student's coursework; stealing coursework from another student and submitting it as your own work. Suspected plagiarism will be investigated and if found to have occurred will be dealt with according to the procedures set down by the University. Please see your student handbook for further details of what is / isn't plagiarism.

All material copied or amended from any source (e.g. internet, books) must be referenced correctly according to the reference style you are using.

Your work will be submitted for plagiarism checking. Any attempt to bypass our plagiarism detection systems will be treated as a severe Assessment Offence.

Coursework Submission Requirements

• An electronic copy of your work for this coursework must be fully uploaded by 23:55 on the Deadline Date of **Friday 11/12/2020** using the link on the coursework Moodle page for COMP1680.

• For this coursework you must submit a PDF document and a zip file of your code. In general, any text in the document must not be an image (i.e. must not be scanned) and would normally be generated from other documents (e.g. MS Office using "Save As .. PDF"). An exception to this is hand written mathematical notation, but when scanning do ensure the file size is not excessive.

• There are limits on the file size (see the relevant course Moodle page).

• Make sure that any files you upload are virus-free and not protected by a password or corrupted otherwise they will be treated as null submissions.

• Your work will not be printed in colour. Please ensure that any pages with colour are acceptable when printed in Black and White.

• You must NOT submit a paper copy of this coursework.

• All courseworks must be submitted as above. Under no circumstances can they be accepted by academic staff

The University website has details of the current Coursework Regulations, including details of penalties for late submission, procedures for Extenuating Circumstances, and penalties for Assessment Offences. See http://www2.gre.ac.uk/current-students/regs

Detailed Specification

This coursework is to be completed individually.

To complete this assignment you will need the source code provided at the following URL.
https://moodlecurrent.gre.ac.uk/pluginfile.php/1585301/mod_resource/content/1/jacobi2d.c

https://moodlecurrent.gre.ac.uk/pluginfile.php/1585302/mod_resource/content/1/gauss2d.c

You are provided with a two C program codes (called jacobi2d.c and gauss2d.c) that solve a rectangular 2 dimensional heat conductivity problem using the Jacobi and Gauss-Seidel iterative methods.

This code can be compiled and linked to produce a conventional executable files called jacobiSerial and gaussSerial by using the following commands:

gcc jacobi2d.c –o jacobiSerial

gcc gauss2d.c –o gaussSerial

To run the executable type in the executable name: jacobiSerial or gaussSerial

As you implement each of the following 4 steps make sure that you retain and do not overwrite previous versions of your solutions.

Step 1 (25 Marks)

You are required to compute a temperature distribution for a rectangular 2D problem with boundary conditions set at top 30°C, bottom 60°C, left 110°C and right 140°C with a range of problem sizes. To do this you are required to modify the codes to:

- reflect the boundary conditions described above
- report the execution time Record the run-time of your code under a range of problem sizes using different levels of compiler optimization.

Be advised that:

- it is possible that aggressive optimization will break the code
- you will need to stop the results from printing if you are to obtain realistic measurements of the execution time.

Step 2 (30 Marks)

You are then required to modify the applications you created in step 1 to produce a basic parallel version of the codes using OpenMP. The following commands will compile your parallel version on a platform that has OpenMP installed:

gcc -fopenmp jacobiOpenmp.c –o jacobiOpenmp

gcc -fopenmp gaussOpenmp.c –o gaussOpenmp

The parallel codes must include timers to report the parallel run-time of the code. This version must be tested to establish correct operation using 1, 2, 4 and 8 threads/processors, regardless of performance. (These versions may run on any platform you choose as performance is not an issue at this stage.)

Run the Gauss-Seidel code for only 1 iteration using 1 and 2 threads for a 20x20 problem size. Output the result along with the timings. Discuss the differences in the solutions.

Step 3 (30 Marks)

Using the cms-grid machines you are to run performance tests with the OpenMP implementation you created in step 2. This will require that you remove most of the print output from the code and increase the problem size to provide sufficient work to demonstrate useful speedup. You are expected to provide speedup results:

• for a range of problem sizes

• for a range of number of threads (from 2 up to 8 threads) In calculating the speedup of your parallel code you should use the optimized single processor version of your code you produced in step 1. You will need to apply similar optimizations to your parallel code.

Step 4 (15 Marks)

Using different OpenMP directives and clauses you are to further modify your OpenMP application to improve the parallel performance. You are expected to provide results that permit comparison with those you obtained in step 3. Comment on the differences between optimising the Jacobi and Gauss-Siedel Methods.

Deliverables

• A PDF file with your report

•A ZIP file with the source code for your solutions.

Your report is required to provide details of your implementation of steps 1 to 4 as described above. The report should include discussion of your solutions and provide a clear description of; the code changes you have implemented, your compilation and execution processes and your test cases. For steps 3 and 4 you are expected to provide tabular and graphical results. Comment on the differences between the two methods and the effect on parrelisation. Your zip file should provide suitably named source code files for each of your implementations.

Grading Criteria

To achieve a pass mark it is expected that an outline solution will be provided in which at least a basic attempt is evident with some progress.

To achieve a mark in the merit range it is expected that a good solution is provided in which there is clear evidence of progress and understanding.

To achieve a distinction mark it is expected that high quality solutions and reports are provided in which there is clear evidence of competence in practical, theoretical and presentation skills.

If you are unsure about any of these instructions then please email your tutor or make an appointment to see your tutor as soon as possible.