IN THIS ASSIGNMENT, USE OPENMP TO PARALLELISE YOUR OPTIMISED JACOBI SOLVER CODE, AND RUN ON ALL THE CORES OF A BLUECRYSTAL PHASE 3 NODE

ASSIGNMENT 2:

OPENMP PARALLELISM

Progress: 100%

ASSIGNMENT DESCRIPTION

- Start with your optimised serial Jacobi code
- Use OpenMP loop parallelism pragmas from the lectures to run your code on all 16 cores of a node
- In particular, consider the following optimisations:
  - Which pragmas to apply
  - Which loops to parallelise
  - Anything else you can think of!

Ballpark runtimes after applying OpenMP optimisations:
2000x2000: <1 second
4000x4000: <25 seconds

- You should then produce a short report discussing your findings (3-4 pages)
  - Describe the optimisations that you tried and your approach to parallelism
  - Explain why your optimisations improved performance
  - Include an analysis of how well your code scales from 1 to 16 cores, in increments of one core

COURSEWORK SUBMISSION

- Your submission will be made via SAFE and should include:

  1. A **3 to 4 page** report in PDF form, which must include:

     a. Your name and user id

     b. A description of your OpenMP optimisations;

     c. Comparisons of your OpenMP performance vs optimised serial;

     d. An analysis of the scalability of your code from 1 core up to 16 cores;

  2. The working code you used to generate the results in your report.

- Your code must converge to the same solutions as the starting code, after the same number of iterations (within a reasonable tolerance).

GUIDANCE PART 1

- To achieve a good mark of 60%+:
    - A well-written, 3-4 page report that clearly demonstrates you understand what you did
    - Code that successfully uses OpenMP parallelism
- You should be able to get a simple OpenMP version working in less than 1 day (~7-8 hours)
- Your time on all 16 cores of Blue Crystal phase 3 should be consistent with the table on page 2 of this assignment

GUIDANCE PART 2

- To aim for a first (70%+), you'll need:
    - An excellent 4 page report
    - Code that:
        - Applies further OpenMP techniques that improve performance above those we've described. These may include code transformations beyond those discussed in class.
            - Achieves performance on 16 cores notably faster than those given in the table on page 2 of this assignment
- With ~6 weeks allocated to the OpenMP assignment, 10 hours allocated to the course each week for 10 weeks, and 4 hours per week spent in lectures and labs, don't spend more than 6 * (10 - 4) = ~36 hours on this assignment in total (twice that for the serial assignment).
    - It should only take half that time to do the simple version which, along with some interesting experiments and a decent report, should be good enough to earn 60%+

SUBMISSION REQUIREMENTS

- Your **report** which must be in a file called "**report.pdf**",
  - Lower case r: "**report.pdf**" NOT "**Report.pdf**"
- Your **source code**, i.e. "**jacobi.c**"
- Your **makefile**, called "**Makefile**"
- Don't modify the timing code in the starting code, as we'll use this to automatically extract timing information from each submission
- We must be able to reproduce any runtimes you quote in your report by compiling and running the code that you submit
- Don't zip these files up, instead submit them as separate files in SAFE

PLAGIARISM CHECKING

- The HPC assignments are all for individuals, they are **not** group work
- We will check **all** submitted code for plagiarism using the MOSS online tool
  - MOSS ignores the example code we give you
  - MOSS will spot if any of you have worked together or shared code, so **please don't!**
- We'll also check all submitted reports using the TurnItIn tool, which will find any shared text
- So please don't copy code or text from each other! You **will** get caught, and then ***both*** the copier and original provider will get a **0** for the whole assignment.

SUMMARY

- Remember, you'll get marks for:
  - A well written, comprehensive, report
  - An OpenMP code that successfully explores most of the optimisations we suggest
- Have fun exploring your first parallel programs!