Exercise: OpenMP

In this exercise, we'll use the same conventions and commands as in previous exercises. You should refer back to the exercise descriptions for details on various Unix commands.

In the exercise, you'll first benchmark an existing code without parallelization, using various compilers and various levels of compiler optimization. Then you'll parallelize the code using OpenMP. Then you'll benchmark various numbers of OpenMP threads using various compilers and various levels of compiler optimization.

Here are the steps for this exercise:

1. If you haven't done the Catch-up Exercises, read through the first one quickly, to see the conventions for how these descriptions express various concepts and tasks.
2. Log in to the Linux cluster supercomputer (`sooner.oscer.ou.edu`).
3. If you don't yet have a `NCSIPARII2011_exercises` directory, create one:

   `mkdir NCSIPARII2011_exercises`

   (If you did the first catch-up exercise, you do have one already; if not, you probably don't.)
4. Copy the `Transport` directory:

   `cp -r ~hneeman/NCSIPARII2011_exercises/Transport/ ~/NCSIPARII2011_exercises/`
5. Go into your `NCSIPARII2011_exercises` subdirectory:

   `cd NCSIPARII2011_exercises`
6. Confirm that you're in your `NCSIPARII2011_exercises` subdirectory:

   `pwd`
   `/home/yourusername/NCSIPARII2011_exercises`
7. See what files or subdirectories (if any) are in the current working directory:

   `ls`
8. Go into your `Transport` subdirectory:

   `cd Transport`
9. Confirm that you're in your `NCSIPARII2011_exercises` subdirectory:

   `pwd`
   `/home/yourusername/NCSIPARII2011_exercises/Transport`
10. See what files or subdirectories (if any) are in the current working directory:

    `ls`
11. Choose which language you want to use (C or Fortran90), and `cd` into the appropriate directory:

    `cd C`

    OR:

    `cd Fortran90`
12. Copy the `Serial` directory to a new `OpenMP` directory:

    `cp -r Serial/ OpenMP/`

13. Go into your `Serial` directory.
14. Via the usual method, compile and run the code.
15. After the run completes, get a profile of the run using the `gprof` command:

    `gprof transport > transport_serial_icc_O2_gprof.txt`
16. Examine the profile output to see (a) what routine the bulk of runtime is being spent in and (b) what the total runtime is, for the serial run.
17. Now try rerunning the code with a variety of compilers and optimization levels, profiling each, outputting the profile results into slightly different filenames. What do you learn from doing this?
18. You should also run different problem sizes, to see how problem size affects relative performance.
19. Go into the parent directory of the current working directory:

    `cd ..`

    Note the two dots, which mean "the parent directory of this subdirectory."
20. Copy the batch script from the current working directory into the `OpenMP` subdirectory:

    `cp transport_openmp.bsub OpenMP/`
21. Go into your `OpenMP` directory:

    `cd OpenMP/`
22. Using your favorite text editor (for example, `vi`, `emacs`, `nano`, `pico`), edit the `makefile` to add an option to `CFLAGS` or `FFLAGS` so that compilation uses OpenMP. For each compiler family, the additional option to add is:

    - GNU: `-fopenmp`
    - Intel: `-openmp`
    - Portland Group Inc (PGI): `-mp`

13. Parallelize the code using OpenMP.

    HINTS

    a. In OpenMP, a good strategy is to focus the parallelization on those parts of the code where most of runtime is spent.

    How can we find out which routine(s) most of runtime is spent in?

    b. Probably the most straightforward approach is to parallelize one or more loops.

    How is that expressed in OpenMP?

    c. If you're going to parallelize a nested loop (a loop inside another loop), you're likely to get better performance by parallelizing the outermost loop, not an inner loop.

    This is because parallelizing the outermost loop gives the greatest ***granularity*** (size of individual subtasks that run in parallel), so overhead due to parallelization is reduced.

    d. Pay close attention to which variables should be shared and which should be private.

    - Which variables are shared by default and which variables are private by default?
    - Which variables that are shared by default should instead be private, if any? How is that expressed in OpenMP?

- Which variables that are private by default should instead be shared, if any? How is that expressed in OpenMP?
- Are there any reductions in the loop(s) that you're parallelizing? If so, how is that expressed in OpenMP?

14. Via the usual method, compile and run the code. (If you haven't done the catch-up exercises, read through them to learn how).
15. Edit your batch script to change the value of the environment variable

    ```
    OMP_NUM_THREADS
    ```

    You should try values between 1 and 8. For each such value, rerun the code and then get a profile.

    DON'T GO ABOVE 8 THREADS! Each of the compute nodes (servers) on OSCER's Linux cluster supercomputer has 2 CPU chips of 4 cores each, for a total of 8 cores per node, so the maximum number of OpenMP threads you should use should be 8.
16. For each run, once the batch job completes:
    a. Examine the various output files to see the timings for your runs with executables created by the various compilers under the various levels of optimization.
    b. Profile, as described above, but with an appropriately modified output filename.
17. Use your favorite graphing program (for example, Microsoft Excel) to create graphs of your various runs, so that you can compare the various methods visually.
18. You should also run different problem sizes, to see how problem size affects relative performance.