Supercomputing in Plain English
Exercise #8: Hybrid MPI+OpenMP

In this exercise, we'll use the same conventions and commands as in Exercises #1, #2, #3, #4, #5, #6 and #7. You should refer back to the Exercise #1, #2, #3, #4, #5, #6 and #7 descriptions for details on various Unix commands.

In the exercise, you'll again be parallelizing and benchmarking, but this time you'll be parallelizing with a hybrid of MPI and OpenMP. Then you'll benchmark various numbers of MPI processes and OpenMP threads using various compilers and various levels of compiler optimization.

Here are the steps for this exercise:

1. Log in to the Linux cluster supercomputer (`sooner.oscer.ou.edu`).
2. Choose which language you want to use (C or Fortran90), and `cd` into the appropriate directory:

   `% cd ~/SIPE2011_exercises/NBody/C/`

   OR:

   `% cd ~/SIPE2011_exercises/NBody/Fortran90/`
3. Copy the `Serial` directory to a new `MPI` directory:

   `% cp -r MPI_collective/ Hybrid/`
4. Go into your `Hybrid` directory:

   `% cd Hybrid`
5. Edit the `makefile` to change the compiler to `mpicc` (regardless of the compiler family).
6. Parallelize the MPI code by adding in the OpenMP directive(s) that you used in Exercise #5.
7. In your batch script, change the number of processes per node to 1, and set the environment variable `OMP_NUM_THREADS`:

   ```
   setenv  OMP_NUM_THREADS  8
   ```
8. Also in your batch script, near the bottom, replace this:

   ```
   mpirun.lsf
   ```

   with this:

   ```
   mpirun_hybrid.lsf
   ```
9. Set the `MPI_COMPILER` and `MPI_INTERCONNECT` environment variables.
10. Compile using `make`. You may need to do this multiple times, debugging as you go.
11. Submit the batch job and let it run to completion. If it seems to take a very long time, probably you have a bug.
12. 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.
13. Continue to debug and run until you've got a working version of the code.

14. 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.
15. Also try different combinations of the number of MPI processes per node and the number of OpenMP threads per process.
16. You should also run different problem sizes, to see how problem size affects relative performance.