Parallelization Sensation!

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

You’ll be parallelizing an existing serial code, similar to the Game of Life program featured Wed June 29.

If you produce a fast MPI version, you may earn your way onto the Leader Board on the workshop webpage. The person with the best final score will receive a modest gift.

Here are the steps for the Parallelization Sensation!

1. Log in to the Linux cluster supercomputer (sooner.oscer.ou.edu).
2. Confirm that you’re in your home directory:
   ```
   % pwd
   /home/yourusername
   ```
3. Check that you have a NCSIPARI2011_exercises subdirectory inside your home directory:
   ```
   % ls
   NCSIPARI2011_exercises
   ```
4. Copy Henry’s Transport directory into your NCSIPARI2011_exercises directory:
   ```
   % cp -r ~hneeman/NCSIPARI2011_exercises/Transport/ ~/NCSIPARI2011_exercises/
   ```
5. Go into your NCSIPARI2011_exercises subdirectory:
   ```
   % cd NCSIPARI2011_exercises
   ```
6. Confirm that you’re in your NCSIPARI2011_exercises subdirectory:
   ```
   % pwd
   /home/yourusername/NCSIPARI2011_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 NCSIPARI2011_exercises subdirectory:
   ```
   % pwd
   /home/yourusername/NCSIPARI2011_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. Go into your Serial subdirectory:
    ```
    % cd Serial
    ```
13. Edit the batch script transport_serial.bsub to use your username and e-mail address.

14. Examine transport.c (or transport.f90).

15. Compile using make:

   ```make```

16. Submit the batch script file transport_serial.bsub to the batch scheduler:

   ```bash
   % bsub < transport_serial.bsub
   ```

   **NOTICE** the less than symbol `<` which is **EXEMPLARY IMPORTANT**.

   You should get back output something like this:

   ```bash
   Job <###> is submitted to queue <pari_q>.
   ```

   where `###` is replaced by the batch job ID for the batch job that you’ve just submitted.

17. Check the status of your batch job:

   ```bash
   % bjobs
   ```

   You’ll get one of the following outputs, either:

   *No unfinished job found*

   (if you get this right after the `bjobs` command, try it several more times, because sometimes there’s a pause just before the batch job starts showing up, as below),

   **OR:**

   ```bash
   JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
   4081250 yourusername PEND pari_q sooner1 transport Oct 17 14:58
   ```

   where `###` is replaced by a batch job ID number, and `yourusername` is replaced by your user name, and where `PEND` is short for “pending,” meaning that your job is waiting to start,

   **OR:**

   ```bash
   JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
   4081250 yourusername RUN pari_q sooner1 c127 transport Oct 17 14:58
   ```

18. You may need to check the status of your batch job repeatedly, using the `bjobs` command, until it runs to completion. **This may take several minutes (occasionally much longer).**

   You’ll know that the batch job has finished when it no longer appears in the list of your batch jobs:

   *No unfinished job found*

19. Once your job has finished running, find the **standard output** and **standard error** files from your job:

   ```bash
   % ls -ltr
   ```

   Using this command, you should see files named

   ```bash
   transport_#####_stdout.txt
   ```

   and

   ```bash
   transport_#####_stderr.txt
   ```

   (where `#####` is replaced by the batch job ID).

   These files should contain the output of `transport`. 
20. Look at the contents of the standard output file:
   \[
   \% \text{ cat transport}_{####}_stdout.txt
   \]
   (where #### is replaced by the batch job ID).
   You may want to look at the stderr file as well:
   \[
   \% \text{ cat transport}_{####}_stderr.txt
   \]

21. If this run had ANY problems, then send e-mail to:
   \text{support@oscer.ou.edu}
   which reaches all OSCER staff (including Henry), and attach the following files:
   - make_cmd
   - makefile
   - transport.c
   - transport_serial.bsub
   - transport_{####}_stdout.txt
   - transport_{####}_stderr.txt

22. When the batch job has finished, examine the output files, including the following file:
   \text{data_xline.txt}

23. Profile the executable:
   \[
   \% \text{ gprof transport} \rightarrow \text{ transport_serial_gprof.txt}
   \]

24. Examine the profile output in the file named transport_serial_gprof.txt to determine which routine most of the runtime is spent in. That’s where you should focus your speedup efforts.

25. Go up to the parent of the Serial directory (that is, to the NBody directory):
   \[
   \% \text{ cd ..}
   \]

26. Copy the Serial directory to a new MPI directory:
   \[
   \% \text{ cp -r Serial/ MPI/}
   \]

27. Copy the new batch script into the new directory:
   \[
   \% \text{ cp transport_mpi.bsub MPI/}
   \]

28. Go into your MPI collective communications directory:
   \[
   \% \text{ cd MPI}
   \]

29. Edit your batch script transport_mpi.bsub to use your username and e-mail address.

30. Edit your makefile to change gcc or pgcc or icc to mpicc (or to change gfortran or pgf90 or ifort to mpif90).

31. Parallelize the code using MPI. We recommend using MPI_Sendrecv, but that’s not a requirement.

32. Set the environment variables named MPI_COMPILER and MPI_INTERCONNECT; for example:
   \[
   \% \text{ setenv MPI_COMPILER gnu}
   \% \text{ setenv MPI_INTERCONNECT ib}
   \]

33. Compile using your makefile. You may need to do this multiple times, debugging as you go.
34. Submit the batch job and let it run to completion. Once it starts actually running (that is, no longer pending in the queue waiting to start), if it seems to take a very long time, probably you have a bug.

35. 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.

36. Continue to debug and run until you’ve got a working version of the code.

Parallelization Sensation Rules

- You may participate as an individual or as part of a team of collaborators.
- You’re welcome to submit questions to us via Piazza. While we’ll make our best effort to respond in a timely manner, we cannot promise to do so.
- Your code MUST compile and run on Sooner. If it runs on everything except Sooner, it will be discarded.
- Submit your source code, makefile and batch script by e-mail to Henry Neeman (hneeman@ou.edu) by no later than 12:00 noon Pacific Time this coming Friday (July 1 2011).
- Late submissions will be ignored.
- Be sure to tell us what values you used for MPI_COMPILER and MPI_INTERCONNECT.
- We will compile and run your code as you have set it up, using the values for those environment variables that you’ve specified.
- Your MPI version MUST be able to run successfully on 32 MPI processes.
- You MUST choose inputs such that the original serial version of the Transport code runs for at least 10 minutes on your set of inputs before completing (runtime only, not pending time in the queue, nor startup and shutdown overhead time in the batch system, etc).
- The values in your MPI version’s output (data_xline.txt) must be just about the same as the original serial version’s output produced by the same compiler family on the same input data. Here, “just about the same” means that the relative error is less than $10^{-3}$ when calculated like so:

\[
\frac{(\text{parallel_value}[i][j][k] - \text{serial_value}[i][j][k])}{\text{serial_value}[i][j][k]} < 10^{-3}
\]

For each value of the final timestep of your run:

Note that, where $\text{serial_value}$ is zero, we won’t calculate the relative error.

Your overall relative error is the maximum of the individual relative error values that you obtain.

- Your score will be the original serial version’s time divided by your MPI version’s time on 32 processes, for the same input dataset (your choice) on both.
- You may submit multiple times up until the deadline (but not after). We only promise to judge the last one submitted, though we reserve the right to try at any time your most recently submitted version.
- The score values will be posted to the Parallelization Sensation Leader Board on the workshop webpage. You may submit anonymously or by individual name or team name.
- The final top entry will receive acknowledgement and possibly a small gift. Results will be posted by the end of this workshop.