PHYS:5905 Homework #11a

Please submit your solutions as a single PDF file with answers to the questions asked.
Please complete required problems before lecture on Tuesday, April 16, 2019.

The following exercises may be completed in either C or Fortran.

1. Compiling and running OpenMP codes on Argon

   (a) Ensure the that the intel module is loaded. `module list` will show modules that are loaded. If the intel compiler is not loaded, use `module load intel`

   (b) Compile your OpenMP code using the Intel Fortran compiler with the OpenMP flag `-qopenmp`
   `ifort -qopenmp -o program.e program.f90`

   (c) Although one could run a multi-threaded OpenMP code on the Argon log-in nodes, this is not allowed. One should submit a batch script to dedicate one of the Argon compute nodes to your job. Output from your job will go into any output files written to disk as well as output to the screen written to a log file that you specify. Note that Argon is a heterogeneous system with nodes that have a different number of cores, with the number of cores being possibly 24, 32, 40, 56, or 64.

   (d) Create a Batch submission script `#!/bin/sh`

```
# Job Submission script

#$ -q UI
#$ -l h_rt=00:10:00
#$ -pe smp 32
#$ -N program
#$ -o program.log
#$ -j y
#$ -V
#$ -cwd

echo "Job begin:"`date`
echo "Run sample1 on Argon, 32 proc"
echo "Run hw11 on Argon, 1 proc, 32 threads"
# Set environment variable for number of threads
OMP_NUM_THREADS=32 export OMP_NUM_THREADS
echo Number of threads is:  $OMP_NUM_THREADS
time ./program.e
echo "Job end:"`date`
```

   (e) Note that the shared memory parallel environment (for running multi-threaded codes) is specified by the job submission flag `#$ -pe smp 32`, where the number of cores requested is 32 in this case.

   (f) The environmental variable `OMP_NUM_THREADS` must be set *on the compute node* in order to run multiple threads. Its default value is 1, so it needs to be set in the batch submission script to ensure that you will actually use the correct number of threads.

   (g) Here I use the unix command `time` to return information on how long it takes for my job to run.

   (h) Note that, if you want to run the same program with different numbers of threads, these can all be put sequentially into the batch submission script.

   (i) You can also run OpenMP on any multi-core laptop with an appropriate compiler. To compile with OpenMP, use `-fopenmp` (GNU compilers) or `-qopenmp` (Intel compilers).

2. Hello World in OpenMP

   (a) Write an OpenMP version of the parallel Hello World program you wrote previously and run it on Argon to observe its output. Set the environment variable OMP_NUM_THREADS=8 to use 8 threads. You will need to define a parallel region and call OpenMP library functions to get the current thread ID and the total number of threads. Have your program produce output like this: Hello World. I am thread 2 of a team of 8 threads.

   (b) (Return) Study the order with which the output appears on the screen. In what order do the messages appear on the screen? Is it reproducible/predictable?

   (c) (Return) Modify your code to produce ordered output. Hints: You wont need any additional OpenMP directives to achieve this, but may need to introduce private and shared variables. Come up with a way of keeping track of which thread has already said hello and make sure that thread with ID n speaks up only after thread with ID $n-1$ has done so. Thread 0 starts.

3. Computing π

   The value of π can be computed (perhaps not in the most efficient way) via the Gregory-Leibniz series:

$$\pi = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} \tag{1}$$

   (a) Implement the Gregory-Leibniz series in a serial code. Make sure it works (i.e., with 500 iterations you should get something not too far off from the real value of π. Add code that computes the absolute and relative error of your result to $\pi = \cos^{-1}(1.0)$.

   (b) (Return) Parallelize your code using OpenMP. Use n =1,000,000,000 and measure how long it takes to run using the `time` unix command before your program executable name (see above). On Argon, find the time it takes to run as you successively double the number of threads from OMP_NUM_THREADS=1 (serial) to OMP_NUM_THREADS=32. Make a plot of the run time vs. number of threads. Is there any speed up? Try to run with just n =1,000,000 and see how the speedup of OMP_NUM_THREADS=32 threads compares to the serial run with OMP_NUM_THREADS=1. What is the speedup in this case?