Fortran and C++ OpenMP And OpenACC Lab

OpenMP

To compile a code to apply OpenMP write：

g++ –o mybin -O –fopen mpcode.cxx

gfortran –o –O mybin –fopenmp code.f90

Project 1

Write a code in your language that sets up a parallel region. Inside and outside the region, have the code print “Hello from thread” number “of “ number, where the first number will be the thread ID and the second will be the number of threads.

Project 2

One way to compute pi is to use numerical integration. Several integral formulations exist but a simple one involves $\int dx/(1+x^2)$. In pseudocode the solution can be expressed as

```
Nsteps=Something
h=1./(double)Nsteps
for i=1 to Nsteps do
   x=(i-0.5)*h
   total=total+4./(1.+x*x)
end
pi=h*total
```

Write a program to compute pi. Run it in serial for a relatively small Nsteps to make sure it works. When you are done, parallelize it with OpenMP. Add the timing functions to return total time required. When you are satisfied that it is working, increase the number of steps to 100000000. Run with 1, 2, 4, and 8 threads and plot the scaling curve.

Project 3

The steady-state heated plate is a 2-dimensional Laplace problem. The equation is given by

$$\nabla^2 T = 0$$

In two dimensions this can be written as

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}$$

This can be discretized as w[i][j] = (u[i-1][j] + u[i+1][j] +u[i][j-1] + u[i][j+1]) / 4.0;

(similarly for Fortran) where we start with u, compute w, then swap w and u for the next iteration. Thus we have a loop over i and j for each iteration. The boundaries are set at i=0 (C/C++) or i=1 (Fortran) and i=nrows-1, j=ncols-1 (C/C++) or i=nrows, j=ncols (Fortran). We terminate the outer loop when the maximum difference between u and w is less than a specified tolerance.

This type of partial differential equation is said to be elliptical. We are essentially replacing each value with the average of its neighbors, over and over again until it settles down. This method is known as Jacobi iteration. It is very slow to converge but is easy to parallelize.

We will solve the heated-plate problem in parallel. Write a program to solve the Laplace problem on a 2-dimensional grid with 0″x″1 and 0″y″1. The edges of the plate are held at 0°C for all edges except y=1.0.

Copy the serial code corresponding to your language, heatedplate.f90, heatedplate.c, or heatedplate.cxx from /share/resources/tutorials/shmem on Rivanna. Examine the code to figure out how to run it.

Be sure you are able to run it in serial before you attempt to parallelize it. For testing use a small value of epsilon such as 0.01 to obtain convergence in a reasonable time. Use a global grid of 500x500.

Parallelize the code using OpenMP.

In parallel, start with an epsilon of at most 1.e-6 and a grid of size 500x500. Use the same epsilon for all the runs so test how long 1.e-6 requires and if possible, reduce it to 1.e-8.

Write a SLURM script to submit the job. Run your program for 1 core, 5 cores, 10 cores, and 20 cores on the bootcamp partition. Use the timing built-ins for your language to get the timing information. Plot the time for the run verses the number of cores. On the same graph plot ideal

scaling. What is the efficiency of your implementation? (See the general multicore programming notes for the definition of efficiency.)

If you know some other language that can make contour plots, make a plot of your final results for each core count. If you do not have that ability you may use the Python script I have provided along with the heated plate source files.. Run it on the frontend through FastX with the commands

module load anaconda/5.2.0-py2.7

python contour.py myoutputfile.txt

Project 5. OpenACC

Repeat Project 4 using OpenACC instead. Compile with the PGI compiler:

module load pgi

pgiCC –acc pi.cxx

pgif90 –acc pi.f90

To run this on a GPU, you will need to log in to FastX Web, or through a shell to rivanna-gpu.hpc.virginia.edu

To submit to SLURM use

#SBATCH -p gpu
#SBATCH --gres=gpu
Both are required.