Homework 1

Overview

In some ways this might be the most frustrating homework we do all semester, because all we're trying to do is to submit a job. If you have access to parallel resources at work or in your lab, you can do this assignment on that machine, but most of us will be using Scholar.

Note that Scholar is a shared resource, and it will get busy as the semester goes on. Also note that the more resources your job needs the longer it will take to be scheduled, especially when Scholar begins to get busy. So always try and debug with a small number of resources. For many of us, we can debug OpenMP jobs (our first parallel programming environment that we will learn) and then just use Scholar for timing and performance tuning. Also, DO NOT run jobs on the front-end machine (scholar.rcac.purdue.edu) that we log into to submit jobs. This can get you into hot water with ITaP, and I won't be sympathetic to defending you.

Our task in this homework will be to submit and run an OpenMP job on Scholar. I will provide the program and a cookbook on how to do this as part of this homework. Additional resources on how to submit and check for job status can be found at:

Overall documentation: https://www.rcac.purdue.edu/knowledge/scholar We'll mainly be concerned with running jobs, info found at https://www.rcac.purdue.edu/knowledge/scholar/run. We will use the SLURM scheduler. This is a relatively new system and different that what was used the last time I taught this course, so this will be a learning experience for all of us.

Info on running OpenMP jobs can be found at https://www.rcac.purdue.edu/knowledge/scholar/run/examples/slurm/openmp

What needs to be done

Login to scholar immediately. You will need to you Boilerkey or SSH Keys to login in. Let me know if you don't have an account – your account should be the same as your Purdue career account password.

Transfer the files omp_hello.c to the Scholar cluster. Note that this only has a main function and no .h file. In general your programs may also need .h files and may have multiple .c files, but that won't change what we are doing.

Transfer the execution scripts to the Scholar cluster. You can use these as templates for future homework assignments.

Compile omp_hello.c.

Submit the C program to run on multiple cores.

Examine the output. You should have "Hello world!" printed by many cores.

Turn it in. Full directions follow.

How to do this:

1. Transfer the file omp_hello.c to scholar.rcac.purdue.edu using sftp or something similar. Note that if you prefer to use Fortran or C++, you can find links to versions of omp_hello.c at https://www.rcac.purdue.edu/knowledge/scholar/compile/openmp
2. Login to Scholar using ssh. At the command prompt, type `module load intel` to use the intel compilers, and `module load gcc` to use the gcc compilers. Intel's compilers generally give better performance than the gcc compilers.
3. Compile your program. If using the Intel compiler, the command is

   ```
   icc -qopenmp myprogram.c -o myprogram
   ```

   and if using the gcc compiler the command is

   ```
   gcc -fopenmp myprogram.c -o myprogram
   ```

4. Set the number of threads that you want to use to run. In bash (which was the default shell when I logged in) use

   ```
   export OMP_NUM_THREADS=20
   ```

   and for csh use

   ```
   setenv OMP_NUM_THREADS 20
   ```

5. Using sftp, transfer the submission script `c.sub` to Scholar.

   ```
   #!/bin/bash
   # FILENAME:  c.sub
   #SBATCH --nodes=1
   #SBATCH --ntasks=20
   #SBATCH --time=00:01:00

   export OMP_NUM_THREADS=20
   ./omp_hello
   ```

   Where
   - `#!/bin/bash` tells what shell should be used to execute the script;
   - `# FILENAME:  c.sub` gives the filename of the script;
   - `#SBATCH --nodes=1` tells how many nodes should be used to run the job;
   - `#SBATCH --ntasks=20` tells how many threads to start up on the node. Node that these nodes have 20 cores, so we have one thread per core;
   - `#SBATCH --time=00:01:00` says we'll use 1 minute of wall time. This is a good amount of time to set for most of our homework. If this time is too small your job may be terminated, and the output file from the job will let you know why.
   - `export OMP_NUM_THREADS=20` sets an environment variable for OpenMP so that the OpenMP knows that 20 threads are available when running our job;
   - `./omp_hello` actually executes our job.
6. Submit your job using the command `sbatch c.sub`
7. When your job has been submitted, `sbatch c.sub` shows how to get status information on it. An easy way to get status information for all of your jobs is to execute `squeue -u userid`
8. Examine the output from your job

9. Use sftp to retrieve the output that will be in a file called `slurm-<job#>.out`, rename it to `<login>.<job#>.txt` and submit it to Brightspace. The output from my run is in the file `slurm-46236.out` in the homework directory.