Prof. Jingke Li (FAB 120-06, li@cs.pdx.edu); Class: MW 2:00-3:50pm @ FAB 40-07; Office Hr: MW 1-2pm & by appt.

Lab 1: Getting Started

For this and all future labs, you will be working on the CS linux systems. You can either use a lab machine, or remotely log on to one (connect to `linuxlab.cs.pdx.edu`).

Download and unzip the file `lab1.zip` from D2L. You'll see a `lab1` directory with multiple versions of the sum program:

```
sum.c sum-pthd.c sum-omp.c sum-mpi.c sum1.chpl sum2.chpl
```

plus a couple of other files: `Makefile`, `linuxhosts`.

1 Set up Your Environment

- Check which shell you are using (`bash`, `csh`, `ksh`, or other):

  ```
  linux> echo $0
  /bin/bash
  ```

  This info will be useful for setting up environment variables (see below).

- Check that you have the latest version of `gcc`:

  ```
  linux> gcc --version
  gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609
  ```

- Check that you have access to the MPI compiler, `mpicc`:

  ```
  linux> which mpicc
  /usr/bin/mpicc
  ```

- Use `addpkg` to add the latest version of Chapel compiler to your environment:

  ```
  linux> addpkg
  ... (a listing of available packages)
  ```

  Use arrow keys and the tab key to select `chapel-1.14.0`; then select `<OK>` and press `enter`. To effect the selection, you need to logout your session and re-login.

2 Compile and Run Pthreads Programs

To compile Pthreads programs, use `gcc` with `-pthread` flag:

```
linux> gcc -pthread -g -o sum-pthd sum-pthd.c
```

The compiled programs are run just like regular C programs:

```
linux> ./sum-pthd
```

Note that for this `sum-pthd.c` program, the number of threads is hardwired in the program. In the future, we'll see how to make that adjustable.

Exercise Add a `printf` statement in the `worker()` routine to show the id and the work range of each individual thread. You may want to place this statement under a control flag:

```
#ifdef DEBUG
  printf(...);
#endif
```

This way, the same program can be compiled to two different versions:

```
linux> gcc -pthread -g -o sum-pthd sum-pthd.c                 # regular version
linux> gcc -DDEBUG -pthread -g -o sum-pthd-debug sum-pthd.c  # debug version
```

3 Compile and Run OpenMP Programs

To compile OpenMP programs, use `gcc` with `"-fopenmp"` flag:

```
linux> gcc -fopenmp -g -o sum-omp sum-omp.c
```

Again, the compiled programs are run just like regular C programs:

```
linux> ./sum-omp
```

Exercises

1. To see the non-intrusive nature of OpenMP, compile the program without the `"-fopenmp"` flag, and save it in a different target:

   ```
   linux> gcc -g -o sum-omp0 sum-omp.c
   ```

   The result is an identical copy to the sequential version.

   One way to confirm that this target code is different from the previous one is to generate and compare their assembly code:

   ```
   linux> gcc -fopenmp -S sum-omp.c       # generate openmp code
   linux> gcc -S -o sum-omp0.s sum-omp.c  # generate sequential code
   linux> wc sum-omp.s sum-omp0.s         # compare their sizes
   linux> diff sum-omp.s sum-omp0.s       # compare their contents
   ```

2. To further confirm that the program `sum-omp` is indeed running with multiple threads, insert a `printf` statement inside the `for` loop to print out the current thread id, which can be obtained by a call to `omp_get_thread_num()`. For this to work, you also need to include the OpenMP header file:

   ```
   #include <omp.h>
   ```

   Question: How many threads are being used?

4 Compile and Run MPI Programs

To compile MPI programs, use the command `mpicc` (which is a `gcc` wrapper):

```
linux> mpicc -g -o sum-mpi sum-mpi.c
```

Before running MPI programs, you need to setup a host file. Copy `linuxhosts` to your home directory, and set the following environment variable (different shells use different syntax):

```
linux> export OMPI_MCA_orte_default_hostfile = ~/linuxhosts  # bash, ksh
linux> setenv OMPI_MCA_orte_default_hostfile ~/linuxhosts    # csh, tsch
```

You should include this line in your shell startup file to avoid typing it in every time. For bash, ksh, csh, and tcsh, respectively, the file names are `.bash_profile`, `.kshrc`, `.cshrc`, and `.tcshrc`.

An MPI program is run with the command `mpirun`, with a flag `-n <#copies>` indicating the number of copies you'd like to execute:

```
linux> mpirun -n 4 ./sum-mpi    // running 4 copies of the program
```

Note that for this program, the number of program copies is specified externally at the time of execution.

Exercise Add a `printf` statement in `sum-mpi.c` to show the rank and size of the current program.

5 Compile and Run Chapel Programs

Unlike the above three cases, where `gcc` handles all the compilations, to compile Chapel programs, a separate compiler, `chpl`, is needed:

```
linux> chpl -g -o sum1 sum1.chpl
linux> chpl -g -o sum2 sum2.chpl
```

For running Chapel programs, you need to set the following env variables in your shell startup file:

```
# for bash, ksh
export GASNET_SPAWNFN = S
export GASNET_SSH_SERVERS = "bevatron boson ..."  # list of host names
export SSH_CMD = ssh
# for csh, tcsh
setenv GASNET_SPAWNFN S
setenv GASNET_SSH_SERVERS "bevatron boson ..."    # list of host names
setenv SSH_CMD ssh
```

The list of host names need to be manually copied from the file `linuxhosts`.

A Chapel program is run with a flag `-nl <#locales>` indicating the number of locales (*i.e.* hosts) you'd like to use:

```
linux> ./sum1 -nl 1        // running the program over 1 locale
linux> ./sum2 -nl 4        // running the program over 4 locales
```

Exercises

1. In both programs, the problem domain size `N` is a configurable constant. Try to change it at the time of execution:

   ```
   linux> ./sum1 --N=2000 -nl 1
   ```

2. Chapel view threads as a lower-level concept, hence does not provide a facility to show which thread a specific code piece in executed by. However, it does provide a facility to show locale information. Add a `writeln` statement inside the `compute()` function in `sum2.chpl` to show where the computation takes place. Use `here.id` to refer to the current locale's name.

Conclusion

Summarize your experience with these four languages and tools. Which one is your favorite at this point? Remember your answer, and we'll see if you'll change your mind at the end of this course.