OpenMP Example

Day 1, afternoon session 1: We examine a serial and parallel implementation of a code solving an N-body problem with a star, a planet and many small particles near the planet's radius. The example goes through logging onto Bridges, using git, the general algorithm, and the parallel algorithm using OpenMP. Additional work is then proposed guiding a learner through issues that often come up using serial and shared memory parallel codes.

1 Preparing to Compute

1. Log onto Bridges (`psc.edu/bridges/user-guide/connecting-to-bridges`)

```
$ ssh <USERNAME>@bridges.psc.edu
# You might have to type yes if this is the first time you log on
```

2. Verify Bridges environment and submit an interactive job

```
# Shows info about the allocation we have
$ projects
# Show the currently charged group
$ id -gn
# Make sure this is as5fp9p;  If not use $ change_primary_group as5fp9p

# Submit an interactive job
$ interact -t 6:00:00 -N 1 --ntasks-per-node=4 --egress -A as5fp9p
# 6 hour interactive job, with 1 node and 4 tasks/node
# Using nodes connected to the outside world on the as5fp9p allocation
# (Wait until your begins before you continue)
```

3. Set up the compute environment

```
# Go to the correct directory
$ cd /pylon5/as5fp9p/<USERNAME>/

# Look at available modules, load gcc and verify what we are using
$ module avail
$ module load gcc/5.3.0
$ module list
$ which gcc
$ gcc --version
```

2 Using Git

1. Get the code from github. *Note: If you copy/paste the address on github and get an authenticity error, try cloning with the other of git or https.*

   ```
   $ git clone https://github.com/PSU-ICS/astroinformaticsNBodyExample.git
   ```

2. View branches

   ```
   $ cd astroinformaticsNBodyExample
   $ git branch
   ```

3. Create a new branch

   ```
   $ git branch <USERNAME>_4June2018
   ```

4. Switch branches

   ```
   $ git checkout <USERNAME>_4June2018
   $ vim README.md
   # Add your username and the date
   $ git add README.md
   ```

5. Add a file to the repository

   ```
   $ echo 'abc' > abc.txt
   $ git add abc.txt
   ```

6. Commit your changes

   ```
   $ git commit
   # Verify the new and modified files
   # Type a message for the commit and then type Esc :wq
   ```

7. View the git log

   ```
   $ git log
   ```

8. Merge with the master branch

   ```
   $ git checkout master
   $ git merge <USERNAME>_4June2018
   ```

9. Push back to the origin *Note: you can't do that without rights.*

   ```
   $ git push origin
   ```

3 Serial Code

1. Go through basic algorithm

   ```
   $ vim serial.c
   ```

   - Headers
   - Variable declarations and definitions
   - Read in points from file
   - Loop over time
     - Loop over base particle
       - Loop over interacting particles
     - Write out file
   - End

2. Things to help you
   - Lots of commented out write-statements for your own purposes
   - symmetricThree.txt is a simple 3-body problem you can do in serial and see how the symmetric system looks with few variables for you to work with
   - You can also only read in a portion of the files (change the value of stars) for debugging purposes

3. Compiling

   ```
   $ gcc -lm -Ofast -o serial serial.c
   ```

   - *gcc* compiler
   - *lm* math library
   - *Ofast* optimization
   - *-o serial* output file name
   - *serial.c* input file name

4. Running

   ```
   # Run and put all output on the screen
   $ ./serial

   # Run and put all output into a file
   $ ./serial > log.4June2018_baseline 2>&1

   # Do this all in the background
   $ ./serial > log.4June2018_baseline 2>&1 &
   ```

5. Resource management (primitive profiling)

```
# Run top in the foreground while your task runs in the background
$ ./serial > log.4June2018_baseline 2>&1 &
$ top U <USERNAME>

# Or run with verbose time
$ /usr/bin/time -v ./serial > log.4June2018_baseline 2>&1 &
```

6. Visualization

```
# Load the modules required to make the figures
$ module load python3/3.5.2_gcc_mkl
$ module load ffmpeg/3.1.1

# Set-up python if required
$ pip3 install matplotlib --user
$ mkdir -p ~/.config/matplotlib/
$ touch ~/.config/matplotlib/matplotlibrc
$ echo "backend : pdf " >> ~/.config/matplotlib/matplotlibrc

# Make the figures
$ python3 makeFigs.py

# Make a videos stringing all of the figures together
$ ffmpeg -r 20 -i figureB%05d.png -qscale 0  video_zooming.mp4
$ ffmpeg -r 20 -i figure%05d.png -qscale 0  video_full.mp4
```

7. Bring the videos back for visualization

```
# On your own computer within a terminal
$ scp <USERNAME>@data.bridges.psc.edu:/pylon5/as5fp9p/<USERNAME>/astro*/*.mp4 .

# Or use WinSCP, Filezilla or some other file-transfer program
```

4 Parallel Code

1. Go through basic algorithm

   ```
   $ vim parallel_openMP.c
   ```

   - Header
   - Lock definitions
   - Parallel for with shared and privates
   - Modify dist to save privates
   - Locks
   - Barrier

2. Compiling

   ```
   $ gcc -lm -Ofast -fopenmp -o parallel parallel_openMP.c
   ```

   - *-fopenmp* openmp

3. Running

   ```
   $ export OMP_NUM_THREADS=4
   $ ./parallel
   ```

4. Resource management (primitive profiling)
   - Same as before

5. Visualization
   - Same as before

6. Discussion
   - Why do we get rid of dx, dy and dz?
   - Is 2 locks the best?
   - How much CPU is being used? Similar to threads?
   - What happens to CPU when you export more threads that your session?

5 Other

Basic Profiling Calculate the percentage of time able to be parallelized

- Use Amdahl's law and multiple runs with different numbers of processors
- Note that this will be approximate as it isn't ideal
- Compare the timing for a case with high and low I/O frequency
- Re-calculate with the lower I/O frequency

Lock Elimination Algorithm modification

- Change the parallel algorithm to eliminate the lock
- Hint: you will be duplicating some of the computations
  - When might this be worthwhile?

Compiler Versions and flags

- Switch to using the intel module and C compiler (icc)
- Compare the timings with GCC
- Try different optimization flags
  - Suggestions: -O2, -O3, -mavx, -funroll-loops, -funroll-all-loops

Precision Floats vs. Doubles

- See how long the symmetric case takes to become unstable due to numerical error
- Change the variables between floats and doubles
- See how long the symmetric case stays stable now

Time Scheme Switch to a better temporal scheme

- Currently explicit (forward Euler) - switch to implicit (backward Euler), or semi-implicit, or Runge-Kutta
- Currently have the deltaV separated from changing the velocity to make this easier

Variable Time Step Base on interactions

- Currently using a constant time-step
- Switch to one based on $0.1 * min(Dist)/max(Speed)$ for next step
- Parallel hints: locks on a single variable or an array with a comparison before the next time-step begins