LAB: OpenMP Affinity

Introduction

What you will learn

- Experiment with different affinity settings for
  - Simple OpenMP code
  - Nested OpenMP code
  - Hybrid OpenMP code (OpenMP and MPI)

Accessing Lab Files

- Logon to Stampede using your account
- Untar the file lab_OpenMP_affinity.tar (in ~train00)
- The new directory (lab_openmp_affinity) contains subdirectories for exercises
- cd into the appropriate subdirectory for an exercise

```
ssh <name>@stampede.tacc.utexas.edu
tar xvf ~train00/lab_OpenMP_affinity.tar
cd lab_OpenMP_affinity
```

IDEV session and 2 windows

- Create 2 windows on Stampede that you can look at side-by-side
- First window: **`idev`** session
  - Start an idev session
  - **`idev -m 60 -A TRAINING-HPC`**
  - Once the session has started type **`hostname`**
- Second window: **`top`**
  - ssh into the node of the idev session
  - **`ssh <node-name>`** (from other window)
  - Type **`top`**
  - Change update interval: Press '**`s`**', then '**`1`**', then 'enter'
  - Press '**`1`**' to see all the cores/hardware threads
  - Note, window must have enough lines

First Exercise (1)

- '**`cd`**' into the subdirectory openmp
- Source the env.sh file: **`source env.sh`**
- Run the code: **`make run`**
- Watch for the threads in the '**`top`**' window

- You should see 4 threads at 'maximum' distance

- Inspect the **`env.sh`** file

First Exercise (2)

- Task 1
  - Edit the env.sh file
  - Change bind policy from spread to close
  - Source env.sh file: **`source env.sh`**
  - Run code again (make run) and observe the threads in the 'top' window
- Task 2
  - Run code with:
  - Bind policy = spread
  - Number of threads = 8
- Task 3
  - Run code with 16 threads

First Exercise (3)

- Inspect the code **`stencil.f90`**
- Stencil update in 2-D
  - New values are calculated from the 4 neighbors in line 51
  - Simple parallel region starts in line 47
  - Lines just above the parallel region show example statements for parallel regions with clauses that change:
    - The number of threads
    - The binding policy
    - Remember, the ‘places’ cannot be changed

Second Exercise (1)

- '**`cd`**' into the subdirectory nested2
- Source the env.sh file: **`source env.sh`**
- Run the code: **`make run`**
- Watch for the threads in the '**`top`**' window

- Inspect the **`env.sh`** file

Second Exercise (2)

- Inspect the code stencil.f90
  - Outer parallel region starts at line 58
  - Note that the number of threads is hard-wired: num_threads(2)
  - Work division is done in the code by calculating indices in lines 59-67
  - Inner parallel region starts in line 71
  - Loop nest in lines 73-77. Note that variables ind_start and ind_end are used
  - Code executes 40 iterations
  - First 20 iterations: both outer threads are executing
  - Last 20 iterations: only one outer thread continues
  - Initially you will see 8 threads. After a while 3 threads will vanish
  - Let's discuss what you see in class

Second Exercise (3)

- Change the binding policy in env.sh to ‘spread’
- Rerun the test
- Do not forget to source env.sh every time you change it

- Change the number of threads to 8
- Now all cores should be utilized

Third Exercise (1)

- This is a second exercise with nested OpenMP
- This time 4 threads are used by the outer parallel region
- '`cd`' into the subdirectory nested4

- Run the code
- Make changes to env.sh as you see fit to utilize all cores