PCSE MPI Homework 2

Overview

You will parallelize the code that you have written in homework 0 using MPI calls/collectives and report scaling information.

Use Stampede2 (preferred) or on Maverick (only if Stampede2 is unavailable.)

Resources:
Your own code written for homework 0
General:

You can use the login node testing and parallelizing codes, but use ***idev*** to interactively run an executable when benchmarking. You can check where the threads are running by checking the load on each core with the ***top*** utility.

```
top      # then hit the 1 character
```

The basic steps are.

1. Create a new folder named hw2
2. Copy your serial code from your hw0 folder to the hw2 folder and parallelize your code using MPI calls.
3. Debug, compile, and run the code
4. **Plot and analyze results, as requested.**

<u>Machine selection</u>
This exercise works best on Stampede2. Only use maverick if Stampede2 becomes unavailable

<u>Instructions on compiling</u>
Compile your code using mpicc or mpif90
Run your code using ibrun

Instructions: hw2

1. Your code has 3 time consuming kernels
    a. Initialization if array x with random numbers
    b. Smoothing, i.e. calculating array **y** from array **x**
    c. Counting of elements below a threshold
2. At the beginning of your code (i.e. before you initialize the first array with random numbers) print a message from each node. Having this 'dummy' parallel region will prevent your actual timings from being corrupted by extra overhead from starting the very first parallel region.
3. Parallelize the kernels b. and c using MPI. Remember to Initialize the array with random numbers on the root node
4. Employ the appropriate MPI Collective for the 2 kernels
5. Enhance the output of your code so that the number of nodes is printed in the well-organized output created at the end of the code.
6. Once you have finished modifying and testing the code increase the number of elements in both directions to 32768+2. This will increase the run-time but also the memory footprint. It is absolutely essential that you are testing and running this larger case in an idev session, if you have not already done so in the first place. You will be using a lot of resources on a node. If you run the tests on the login nodes you will be caught and blocked by the sys-admin. Moreover your timings will be wrong since you will be sharing the login node with other users.
7. Throughout this assignment (unless otherwise instructed) execute your code with ibrun

Assignment 1: This assignment focuses on the kernel that counts the number of elements in an array.

- Make sure that your code is properly debugged. Compare the number of elements below the threshold in your serial and parallel version.
- Report 3 times:
    - Parallel execution with 2 tasks
    - Parallel execution with 8 tasks
    - Parallel execution with 16 tasks
    - Discuss the difference in execution time. Speculate what may be going on.

Assignment 2: This assignment focuses on the 'smoothing' kernel. Report only times for this kernel in assignment 2. Only discuss the timings for the 'smoothing' kernel in this assignment.

- Run tests Create a plot (speedup v. node count) for 2, 4, 8, 16, 32 and 64 tasks. Add also the ideal speed-up in either plot.
- Discuss the plots. What setup performs the best? Discuss the deviation from the ideal speed-up.
- Add the output of your code for the run with 8 nodes to your report.
- In your opinion: Is the scaling good or bad?

Summary:

- Parallelize 2 kernels in your code
- Investigate the ‘counting’ kernel (assignment 1)
- Investigate the ‘smoothing’ kernel (assignment 2)
- Answer all the questions from the assignments. All text highlighted in green warrants a response in your report.
- use mpicc or mpif90 to compile
- use ibrun to execute