1 Setup

The machine onyx.boisestate.edu (running Fedora Core Linux) will be used for submission and grading of programs. Make a directory for the class (say cs430 or cs530). Make subdirectories (say p1, p2, ...) underneath the class directory.

2 Computing \(\pi\) using Monte Carlo Simulation

- Start by copying the sequential Monte Carlo code for calculating \(\pi\). It is available in the `~amit/cs430/lab/MPI/monte_carlo` folder. Your job is to parallelize this sequential program to enable faster calculations.

- Your program should be written in SPMD style. The executable must be called `pmcpi`. Process 0 should act as the coordinator. Each process runs its share of the Monte Carlo simulation. At the end each process sends its number of successful experiments to Process 0, which then combines the results and then computes the value of \(\pi\).

- We want to make sure that the set of random numbers generated in your parallel program is the same set as generated in the sequential Monte Carlo code. For this purpose use the PRAND parallel random number library that was discussed in class. See the example `~amit/cs430/lab/MPI/random/random.c` on how to use the PRAND library.

- Note that the value of \(\pi\) obtained should be the same with any number of processes since we are using exactly the same random numbers for each run.

- **Timing your parallel program.** Use `MPI_Wtime()` to time the program. Process 0 should handle the timing. Make sure that it waits for a message from all other processes signifying that they have finished their part of the program before you measure the total time taken.

- **Measuring performance.** Measure your speedups with respect to the sequential program. Make sure to use the optimize option to the compiler (-O) and turn off debugging before making speedup measurements. Report your runtime and speedup for the parameters in two tables of the following format.
<table>
<thead>
<tr>
<th>#iterations</th>
<th>serial</th>
<th>p = 2</th>
<th>p = 4</th>
<th>p = 8</th>
<th>p = 16</th>
<th>p = 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2,000,000,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4,000,000,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8,000,000,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Please report any observations that you have. In particular, please observe how well does your program scale with number of processors. If you do not get linear speedup, then explain what may be the reason.

- You may use the Linux Cluster lab interactively while you develop, test and debug your code. To run performance measurements, you must use PBS scripts to run your tests. A sample script has been provided to you along with the sequential source code. *Please submit your PBS script as part of your assignment.*

## 3 Submitting the Assignment

The parallel program must have the following name and command line arguments:

```
[amit@kohinoor monte-carlo.private]: pmcpi
Usage: pmcpi <number of iterations> <random seed> <checkpoint interval>
```

Create a README file that contains your name, date, assignment number, results and observations about the assignment.

Put all the relevant files in a directory (say ~/cs430/p1). Change directory to this directory and execute the following command (on onyx) to submit the assignment.

```
submit amit cs430 p1
```

or

```
submit amit cs530 p1
```

This command will pick up all files in the current directory (as well as any subdirectories recursively) and time-stamp them before transferring the combined files to the instructor’s account.