CMDA 3634 Fall 2017 Homework 06

Kevin Jiang

December 14, 2017

You must complete the following task by 11:59pm on 12/12/17.

Your write up for this homework should be presented in a LaTeX formatted PDF document. You may copy the LaTeX used to prepare this report as follows

1. Click on this link
2. Click on Menu/Copy Project.
3. Modify the HW06.tex document to respond to the following questions.
4. Remember: click the Recompile button to rebuild the document when you have made edits.
5. Remember: Change the author

Each student must individually upload the following files to the CMDA 3634 Canvas page at https://canvas.vt.edu

1. `firstnameLastnameHW06.tex` LaTeX file.
2. Any figure files to be included by `firstnameLastnameHW06.tex` file.
3. `firstnameLastnameHW06.pdf` PDF file.
4. `mcpi.r`, `mcpi_mpi.r`, and `mcpi_gpu.r` text file with student code.

You must complete this assignment on your own.

100 points will be awarded for a successful completion.
Extra credit will be awarded as appropriate.

Q1 *(10 points) Serial Monte Carlo approximation of Pi.*

10 points will be awarded for submitting a serial Pi approximation method written in R. Submit code to canvas and push code to GitHub.

Q2 *(20 points) Parallel Monte Carlo approximation of Pi.*

20 points will be awarded for submitting a parallel Pi approximation method written in R (using pbdMPI). Submit code to canvas and push code to GitHub.

Q3 *(20 points) Strong Scaling Study of Parallel Pi Approximation Method.*

For the strong scaling study, fix the total number of testpoint. Divide the total number of tests evenly among all processors. (You may have to alter your program slightly.)

1. Create a batch script that will build and run the code using 1, 2, 3, ..., 16 processors
2. Submit your batch script on the newriver1 login node using qsub.
3. Plot a graph that shows how long it takes to run the Monte Carlo Pi approximation on the vertical y-axis, and number of processors used on the horizontal x-axis.
4. Comment on your results. Does the code strong-scale perfectly ?

Copy and paste the contents of the batch script file used to submit a job to newriver into your LaTeX report. Include the plot mentioned above.

Batch script:

```
#PBS -ACMDA3634

cd $PBS_O_WORKDIR

#load R and R-parallel
module purge
module load intel mkl R/3.2.0
module load openmpi hdf5 netcdf R-parallel/3.2.0

#run the serial version
#Rscript run_mcpi.r

#run the parallel version with parRapply
#Rscript mcpi_parallel_apply.r

#run the parallel version with mclapply
#Rscript mcpi_parallel_mc.r

#run the pbdr version
for i in {1..16}
do
mpiexec -np $i Rscript mcpi_pbdr.r
done
```

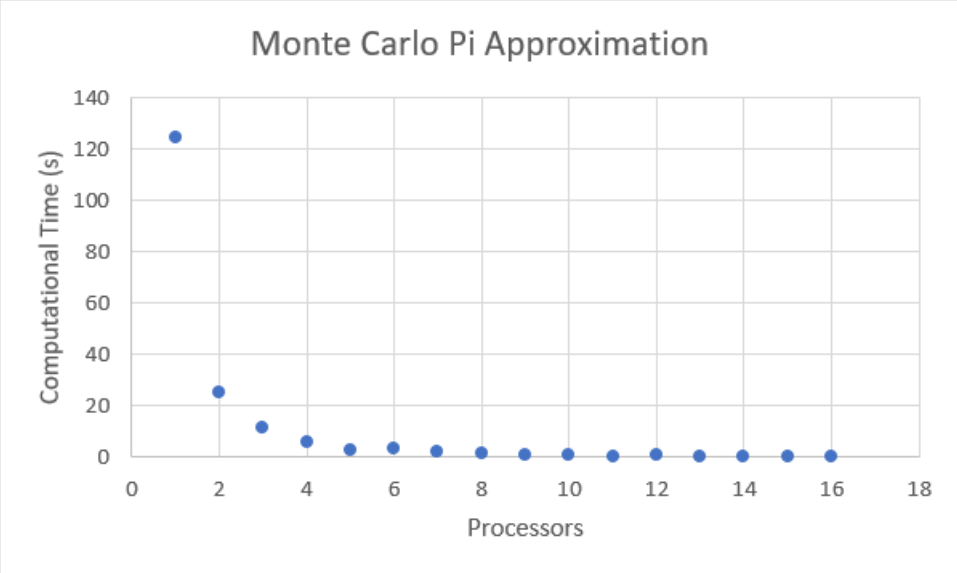

Figure 1: Monte Carlo Pi strong scaling

Based on the graph produced, the code does strong-scale perfectly although the times are not as consistent despite increasing the number of processors. However, overall, the model fairly produces an accurate trend line.

Q4 *(20 points) Weak Scaling Study of Parallel Pi Approximation Method.*

For the weak scaling study, have each processor run on do the same amount of work. That is, the total number of points tested should increase linearly with number of processors run on. (You may have to alter your program slightly.)

1. Create a batch script that will build and run the code using 1, 2, 3, ..., 16 processors
2. Submit your batch script on the newriver1 login node using qsub.
3. Plot a graph that shows how long it takes to run the Monte Carlo Pi approximation on the vertical y-axis, and number of processors times testpoints per node used on the horizontal x-axis.
4. Comment on your results. Does the code weak-scale perfectly ?

Copy and paste the contents of the batch script file used to submit a job to newriver into your LaTeX report. Include the plot mentioned above.

Batch script:

```
#PBS -ACMDA3634

cd $PBS_O_WORKDIR

#load R and R-parallel
module purge
module load intel mkl R/3.2.0
module load openmpi hdf5 netcdf R-parallel/3.2.0
```

```
#run the serial version
#Rscript run_mcpi.r

#run the parallel version with parRapply
#Rscript mcpi_parallel_apply.r

#run the parallel version with mclapply
#Rscript mcpi_parallel_mc.r

#run the pbdr version
for i in {1..16}
do
mpiexec -np $i Rscript mcpi_pbdr.r
done
```

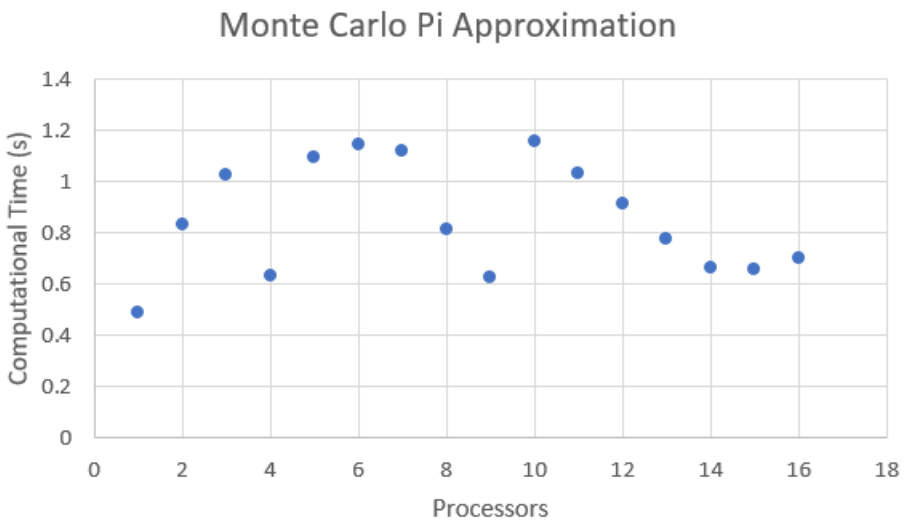

Figure 2: Monte Carlo Pi weak scaling

NOTE: I reduced the points size from 1e7 to 1e5 in the interest of time.

Based on the graph produced, the code does weak-scale, although it seems a bit sporadic.

Q5 *(30 points) GPU-R Monte Carlo approximation of Pi.*

Use GPU-R to create a Monte Carlo approximation of Pi. Document the time it takes to run the program. How does it compare to the serial program? How does it compare to the parallel R program run on various numbers of processors?

Submit code to canvas and push code to GitHub.

I was unable to figure this question since I am still unfamiliar with R's syntax.

Extra Credit: *(50 points) Vectorized Monte Carlo Estimation of Pi with Scaling Study .*

Vectorize the mcpi.r program for a further speed up. (Hint: Some default R functions, for instance `apply`, use for loops under the hood.) If the program is vectorized correctly, expect a speed up factor of about 50.

Once the program is vectorized, repeat the scaling studies performed in **Q3** and **Q4** with the vectorized code. In addition to plotting the results on there own plots, graph the results from the strong scaling study of the non-vectorized and vectorized programs on a single plot. (You might consider taking the log of the y-axis for this plot. Label axis clearly.) Comment on your results. Compare and contrast the scaling of the vectorized and non-vectorized codes.