Pi Computation using CWRU HPCC

Helen Zhao (hxz347), Hun Jae Lee (hxl224)

EECS338 Operating Systems Final Project

Demo due Tuesday, Dec. 6, 2016

Introduction

The four methods for computing pi all depend on different mathematical formulas and ideas.
The different formulas or algorithms were easily implemented using python:

- Ramanujan-Sato series
- Chudnovsky Algorithm
- Brent-Salamin Formula
- Monte-Carlo Method

We have implemented them in Python script and ran SLURM scripts in order to run calculation in parallel manner.
One implementation will be in time constraint, where each calculation runs for given amount of time (in minutes). Another is for digits, where each calculation runs up to certain digits.

Progress Report

Implement the following in two ways.

Running for 1 minute

In time method, with 1 minute timeframe, we were able to observe that Ramanujan-Sato series is currently the best candidate for the most accurate algorithm to Pi approximation. This method will run for 1 hour for final report.

Runtime for calculating less than 100 digits of Pi

- The Chudnovsky's and Brent-Salamin algorithms both calculated 100 digts pi in less a millisecond.
- While the Monte-Carlo method took almost a second to calculate just to calculate 5 digits.
- Furthermore, while the Nilakantha series is also a converging series like Chudnosky's and Brent-Salamin,
  the series coverged slower and took many more iterations to calculate the next digits of pi.
  The method took almost a second to calculate 10 digits of pi.

Runtime for thousands of digits of Pi

- Monte-Carlo method was not used because many iterations were required to calculate just a few more digits of pi.
  It could not keep up with the other processes.
- While the Nilakantha series was much faster than the Monte-Carlo, it required more iteration and more time per additional iteration.
  We did not use this method to calculate more digits of pi.
- The Chudnovsky's and Brent-Salamin algorithms were compared

How to run

- `make 1min` to run all 4 algorithms for 1 minute
- Running in HPC with SLURM script will be implemented soon

Result

The results are in each directory, `/time/result` and `/digits/result`

- Time result:
  - Ramanujan-Sato series
    - 1 minute
    - Calculated pi: 3.1415927
    - Difference to exact value of pi: 7.6423512..
    - Error: (approx-exact)/exact = 0.0000024326..%
    - There seems to be an error in calculating the difference. This will be evaluated and fixed.
  - Chudnovsky Algorithm
    - 1 minute
    - Calculated pi: 3.14159265..
    - Difference to exact value of pi: 3.1415926..
    - Error: (approx-exact)/exact = -2.526276..%
  - Brent-Salamin Formula
    - 1 minute
    - Calculated pi: 3.141592..
    - Difference to exact value of pi: 9.9129293
    - Error: (approx-exact)/exact = 3.155383%
    - There seems to be an error in calculating the difference. This will be evaluated and fixed.
  - Monte-Carlo Method
    - 1 minute

        - Calculated pi: 3.128579..
        - Difference to exact value of pi: -0.013013..
        - Error: (approx-exact)/exact = -0.414223.. %
- Digits result:
    - Ramanujan-Sato series
    - Chudnovsky's Algorithm
        - 10000 digits took 449s
        - 20000 digits took 3512s
        - 40000 digits took 27675s
        - We reached the limit for compute notes while trying to compute 50000 digits of pi
    - Brent-Salamin's Algorithm
        - 10000 digits took 21s
        - 20000 digits took 115s
        - 40000 digits took 421s
        - 75000 digits took 1621s
        - 120000 digits took 4465s
    - Monte-Carlo Method

Example of SLURM script

The following is a template for SLURM script, used in order to add a job to HPC:

```
#!/bin/bash
#SBATCH --mail-user=hunj@case.edu
#SBATCH --mail-type=ALL
#SBATCH --nodes=1
#SBATCH --cpus-per-task=4
#SBATCH --output=pi.output


nproc=$(expr $SLURM_JOB_CPUS_PER_NODE \* $SLURM_NNODES)
echo $nproc cores available.


cp *.py $PFSDIR/.
cd $PFSDIR
module load python
# python pi_monte-carlo.py 36000
```