1. Write a shell script (in Bash) named `add.sh` that will add two integers supplied as command line arguments.

Bash script -

```
# !/bin/bash


### Used to check if we have an integer expression, we emphasize that 1 = +1 i.e. both are integers for us.
REGEX=^[+-]?[0-9]+$

### If number of arguments = 2 and both are integers then add them.
if [ $# = 2 ] && [[ $1 =~ $REGEX ]] && [[ "$2" =~ $REGEX ]] ; then
    echo "The sum of the numbers $1 and $2 is $(( $1 + $2 ))"
else
    echo "ERROR: This script is supposed to add two integers, please provide two command line arguments that are integers."
fi
```

Commit history (in short) -

```
git log --pretty=oneline add.sh

5fd11f62f4fb1041c445aa9275a913b6783911ee (HEAD -> master, origin/master, origin/HEAD) Changed the comments
7ec087addb0e75f3f5b854a1c80ae31eae0e35ab add.sh - Reducing capabilities in case test cases are run to evaluate add.sh
ae3ab7ac50143378402c86cb165bb59f06e7298e Error message changed to better serve the user.
437fea25d5bd44b8b0e04de9e34d742eeefe6419 New comments to represent the new capabilities.
5bc73fe376cda7d322d9407173f05d9c3bfcb00d add.sh can handle decimal point with zeros, +/- sign atthe beginning now.
8f92fd8db1007f6a0c71e14230a292c3fc403034 add.sh has comments now.
1734a1c340824d17e0ec8d22a0ca1cd6ea464e46 add.sh moved to new directory
```

We can easily incorporate numbers like 4.00 as integers too, but it has been avoided in case there are test cases. **Sample I/O** -

```
$ ./add.sh 4 3
The sum of the numbers 4 and 3 is 7

$ ./add.sh 4 +3
The sum of the numbers 4 and +3 is 7

$ ./add.sh 4 -3
The sum of the numbers 4 and -3 is 1

$ ./add.sh 4.0 -3
```

```
ERROR: This script is supposed to add two integers, please provide two command ↩
line arguments that are integers.

$ ./add.sh 4.01 -3
ERROR: This script is supposed to add two integers, please provide two command ↩
line arguments that are integers.
```

2. Write a shell script (in Bash) named `factorial.sh` that uses a while loop and `expr` to calculate the factorial of a supplied number.

Bash script -

```
# !/bin/bash

### To check if we have whole numbers as input
REGEX=^[0-9]+$

### Initialize factorial and the loop variable
factorial=1
i=$1

### Loop only if the input was a valid whole number and the number of inputs ↩
    were 1
if [ $# = 1 ] && [[ $1 =~ $REGEX ]]; then

    ### This loop will not run for 0, so we directly have factorial = 1
    while [ $i -gt 1 ]; do
        factorial=`expr $factorial \* $i`
        i=`expr $i - 1`
    done

    ### Print the value of factorial
    echo "The factorial of the number $1 is $factorial"

else
    ### Print error message
    echo "ERROR: This script is supposed to find factorials. Please provide ↩
       only one argument which is a whole number."
fi
```

Commit history (in short) -

```
$git log --pretty=oneline factorial.sh

718b142b4a2f199c609979c5a0b9536533c6ebec Changed regex formats
f6c0c5c784e279ec3e38e767d644266e3445a805 factorial.sh - Reducing capabilities ↩
in case test cases are run to evaluate factorial.sh
5b89f95e8693632b68cafa8bd092083a15f6afa0 modified/made addtional comments since↩
 we have capability to handle numbers such as 2.000, +2.000 now
188a3cda9421313624fb253b076b1ca706b610df modified/made addtional comments since↩
 we have capability to handle numbers such as 2.000, +2.000 now
ab15c16517dd9d7afacd578873c6d4ab7a59d447 1.0 or 1.0000 is now treated as 1
68723a0e15110fb5c0bf8efff1a0fa7a76fa1263 factorial .sh Added comments
```

```
e3fec269af4fb7ed349df5de60d885abd42a5b80 Shifted while loop into the if ↩
statement checking for valid inputs.
827c9830bb029ac4bd338894ee2d971b2fa40539 factorial.sh moved to new directory
```

We can easily incorporate numbers like 4.00 as integers too, but it has been avoided in case there are test cases. **Sample input/outputs** -

```
$ ./factorial.sh 3
The factorial of the number 3 is 6

$ ./factorial.sh 3.1
ERROR: This script is supposed to find factorials. Please provide only one ↩
argument which is a whole number.

$ ./factorial.sh -3
ERROR: This script is supposed to find factorials. Please provide only one ↩
argument which is a whole number.

$./factorial.sh
ERROR: This script is supposed to find factorials. Please provide only one ↩
argument which is a whole number.
```

3. Write a shell script (in Bash) named `tri.sh` that uses a for loop to generate the following output:

```
      1
     2 2
    3 3 3
   4 4 4 4
  5 5 5 5 5
      .
     . .
    . . .
   . . . .
  . . . . .
```

Bash script -

```
# !/bin/bash

# Print the pyramid of numbers
for ((i = 1; i<=5;i++)); do
    for ((j = 1; j<=$((5-$i));j++)); do
        echo -n " "
    done
    for  ((j = 1; j<=$i;j++)); do
        echo -n "$i "
    done
    echo
```

```
done
# Print the pyramid of dots
for ((i = 1; i<=5;i++)); do
    for ((j = 1; j<=$((5-$i));j++)); do
        echo -n " "
    done
    for  ((j = 1; j<=$i;j++)); do
        echo -n ". "
    done
    echo
done
```

Commit history (in short) -

```
$git log --pretty=oneline tri.sh

5c9a4134281218e18ca1eabc0b4c970e0a0a4fce Changed the style of for loops to more↩
 of the C++ one.
49c7c7a18a2313561208523e6957ad3d48294d54 Changed the comments again.
412301248d1cb0da7e359fc6b9ebb05e702df237 tri.sh edited comments
382fecea4c2b4c10995a69a05928699c2dabf53f tri.sh added comments
a7db7324c6f6b9867c90e7eb93a47c3da0b2d102 tri.sh prints both pyramids out just ↩
fine now.
64c9ddbd04e063c9e6ca6d7348aada40e53b53c4 tri.sh created and it prints out only ↩
the first pyramid.
```

Output -

```
$./tri.sh
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
    .
   . .
  . . .
 . . . .
. . . . .
```

4. I) Write a shell script (in Bash) named `pi.sh` to estimate the value of π using a Monte Carlo method. To do this, consider a circle of radius= 1 enclosed by a 2×2 square as shown in Figure 1. If we draw random samples for (x, y) coordinates in the upper right-hand quadrant from the range $(0, 0)$ to $(1, 1)$, each point can be classified as being within the circle (N_i, the green points), or outside the circle (N_o, the red points). The ratio of the area of the quarter circle to the area of the quarter square is: $\frac{1}{4}\pi r^2/\frac{1}{4}(4r^2) = \pi/4$. Using Monte Carlo sampling and counting how many points fall within the circle, we can approximate the same area ratio as $\pi/4 \approx N_i/N_{\text{samples}}$. Consequently, we can estimate

the value of π via:

$$\pi \approx \frac{4N_i}{N_{\text{samples}}}$$

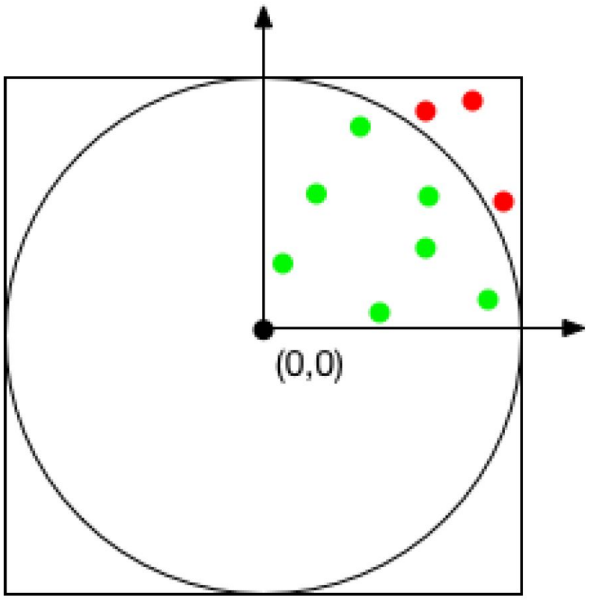

Figure 1: Circle of radius 0.5 enclosed by a 1×1 square.

Runs: Create a single SLURM job script for use on Stampede2 to run your `pi.sh` script sequentially for $N_{\text{samples}} = 10, 100, 500, 1,000, 5,000, 10,000$, and $50,000$. Save the results to a file named `pi.script.log` and append the time (in seconds) required to complete each run as the 6th column entry in the file. Note: you can either add this timing incrementally, or save the timings to a temporary file and join appropriately. The `/bin/time` command we highlighted in class has an option to send the resulting timing to a file (and can also append to a file) which you may find convenient in your job script. Commit the job script and results file from a job on Stampede2 to your GitHub repo.

All relevant files -

```
pi.sh | job.sh | pi.script.log
```

Bash files - `job.sh` is the SLURM job file and `pi.sh` is the same as that in the question.

- `pi.sh` (along with short commit history) -

```
# !/bin/bash

### To check if we have whole numbers as input
REGEX=^[0-9]+$
```

```
if [ ! $# = 1 ] || [[ ! $1 =~ $REGEX ]] || [ $1 = 0 ]; then
    echo "ERROR: Please give only one positive integer as input."
    exit
fi

### Total number of points
N=$1

### Actual value of PI
PI=$(echo "scale =10; 3.14159265358979323846" | bc -l)

### Count points inside the circle
N_i=0

for i in `seq $N`;do
    ### Randomly generate x and y between 0 and 1.
    x=$(echo "scale=10; $RANDOM/32767" | bc -l)
    y=$(echo "scale=10; $RANDOM/32767" | bc -l)
    d1=$(echo "scale=10; $x * $x" | bc -l)
    d2=$(echo "scale=10; $y * $y" | bc -l)
    d=$(echo "scale=10; $d1 + $d2" | bc -l)
    ### If inside cicle, increase N_i
    if [[ $(echo "scale = 10; $d <= 1.0 " | bc -l) = 1 ]]; then
        N_i=$(( $N_i+1 ))
    fi
    piestimate=$(echo "scale=10; 4*$N_i/${N[n]}" | bc -l)
    erel=$(echo "scale=10; sqrt((($PI-$piestimate)/$PI)^2)" | bc -l)
done

### Printing in the required format, the time appending is done by job.↩
sh
echo -n "$N $N_i $(($N-$N_i)) $piestimate $erel "
```

```
$ git log --pretty=oneline pi.sh

55e8c5d751c904fc2573fb6890b4f9403d1935de Command line input verification↩
 added.
23840dec10b9107389a3756d84d0e3f0f8c7da96 Renamed COUNTER as N_i
71ac5c78a611eb8c5ad500a27dee2aac12d0b04b Added comments.
49413e8713176cb7ae146ac8c71ba024408a18b0 Added time functionality to job↩
.sh and removed the outer for loop from pi.sh since we run it for each N↩
 separately.
d3b88fb41f79a29fefd60fafcb266012b2527adb Modified pi.sh
5388b969b17f373ee5488e4c3bc84dd7d54c5748 pi.sh
```

- `job.sh` along with short commit history -

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
##SBATCH -o job_results.log
##SBATCH -e error.%j.out
#SBATCH -J PI_ESTIMATION
```

```
#SBATCH -p skx-normal
#SBATCH -A cse38018
#SBATCH -t 00:40:00

TIMEFORMAT=%R ## Change time format to give only real time value
## Empty pi.script.log before run
>pi.script.log
### Output stores stdout of ./pi.sh and stderr of time since the time ↩
command gives it's output to stderr.
{ time ./pi.sh 10 >> pi.script.log ;} 2>> pi.script.log
{ time ./pi.sh 100 >> pi.script.log ;} 2>> pi.script.log
{ time ./pi.sh 500 >> pi.script.log ;} 2>> pi.script.log
{ time ./pi.sh 1000 >> pi.script.log ;} 2>> pi.script.log
{ time ./pi.sh 5000 >> pi.script.log ;} 2>> pi.script.log
{ time ./pi.sh 10000 >> pi.script.log ;} 2>> pi.script.log
{ time ./pi.sh 50000 >> pi.script.log ;} 2>> pi.script.log
```

```
$ git log --pretty=oneline job.sh

a8b750768d400e16c1fd6c2a1e1faee6975b9950 Empties pi.script.log before ↩
running rest of job.sh.
ab2737f0fe96c17eaca8a357a5dbaf50c8168df4 Suppressing output apart from ↩
our required ones.
eb45fd0c0e423e2b8d53ddd0b66fd7058696343d On second thought, will save ↩
the output log file, so uncommented SBATCH -o
cbf840d95d467c8b4153348cb2887bd7f2e70a3e Commented out SBATCH -o and ↩
SBATCH -e because those files are not relevant now.
810f4826a891c59e4240a613a88867910d22e056 Added comment explaining the ↩
timeformat
b01b29c37f9b7a98c1c708f248bfc6af5063a78d Removed the send emails feature↩
 from job scripts, also changed jobs in jobfile.
bab188623d49a36b13e8e3e296ba5f4165c292e2 Forgot to correct name of ↩
output file everywhere
c514700d74c24077938912da6423ae25ca767592 Increased runtime
45e87b3c37772a90134bc1b7fa13e4c00d8c6386 Best version so far -without ↩
ibrun and with output to pi.script.log
6573d7d7e8290bef0a1dd88d6934fb7ad4a44b9a Commented te code
ebd89756a90071e0928b0b95669aa5a561e56136 The previous code works ↩
perfectly, but without ibrun. This one is with ibrun.
462f9395bc96d96ca873fa28722c32517e87b62b Another iteration at getting ↩
correct output format for job.sh
6784998205681b5e13bdf9ec81198a566d39cdd1 Another iteration at getting ↩
correct output format for job.sh
7423b510bb835da942b40c8b07799acd6d135c45 Another iteration at getting ↩
correct output format for job.sh
b8ba12be55a8fdf5bc035d0e3c44478260435870 Another iteration at getting ↩
correct output format for job.sh
11d2ecf0c0d66fd28cc4a11e3ce791c6593aa774 Another iteration at getting ↩
correct output format for job.sh
f6a10c6d00b93b843390445ed38441049e4690fe Another iteration at getting ↩
correct output format for job.sh
9344374de1528e13fd4271c0e875ed7296f0ba87 Another iteration at getting ↩
correct output format for job.sh
75d8a1d3cd2beca95331d02747946c434e9ca04b Reversing previous attempt
dbd28cc042b98d1274fe7ce42eead7509da1d145 Attempt to solve time display ↩
```

```
by removing error output to a file.
f253ff6698214adea440ec9961be9a7958fbf3ec Attempt to solve time display ↩
by swapping ibrun time with time ibrun
69b056b88ffb5dab96c84def82826710d4975d9a Changing the log script names a↩
 bit of job.sh
2c220b78c0c0874bd5bb97285d903007dee2b301 Changing the number of cores in↩
 the job file.
49413e8713176cb7ae146ac8c71ba024408a18b0 Added time functionality to job↩
.sh and removed the outer for loop from pi.sh since we run it for each N↩
 separately.
d5f900123aa140044fd897be0f042c0908a41979 Added the C++ file for ↩
estimating pi as well as the job script for SLURM for Q.4 I
```

Log files - `pi.script.log` is the same as the one in the question. The format of the output is -

`N_samples N_i N_o pi_estimate e_rel time`.

The time is in seconds.

- `pi.script.log`

```
10 7 3 2.8000000000 .1087323185 0.196
100 78 22 3.1200000000 .0068731506 1.847
500 395 105 3.1600000000 .0058592320 9.267
1000 775 225 3.1000000000 .0132393504 18.574
5000 3944 1056 3.1552000000 .0043313508 94.250
10000 7866 2134 3.1464000000 .0015301960 194.124
50000 39350 10650 3.1480000000 .0020395097 1151.091
```

II) Repeat the exercise from Part I implementing the same Monte Carlo method to estimate π using a compiled language (`C/C++`or `Fortran`).

Runs: Create a second SLURM job script for use on Stampede2 to run your compiled π estimator for the same sample counts used in Part I. Save the results to a file named `pi.compiled.log` and commit the src code, job script, and results file from a job on Stampede2 to your repo.

Relevant files -

```
pi.cpp | job_cpp.sh | pi.compiled.log
```

The command to compile `pi.cpp` is given inside `job_cpp.sh` itself and creates the executable `b.out`.

Scripts - `job_cpp.sh` is the SLURM job file and `pi.cpp` is the equivalent of `pi.sh` in the previous part.

- `pi.cpp` along with short commit history -

```cpp
#include <stdio.h>      /* printf, NULL */
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include <iostream>
#include <math.h>
// Actual value of PI
#define PI 3.14159265358979323846
using namespace std;

int main(int argc, char *argv[]) {
    // Input verification
    // atoi returns integer part of input and 0 if not a numeric input
    if (argc != 2 || atoi(argv[1]) <= 0) {
        cerr << "ERROR: Give a positiver integer input for number of samples. " << endl;
        return 1;
    }
    srand(time(NULL));
    int N = atoi(argv[1]);
    double x,y,d;
    //Counter for number of points inside circle.
    int N_i = 0;
    double piestimate,erel;

    N_i = 0;
    for (int j = 0;j < N; j++){
        // Generate x and y between 0 and 1
        x = double(rand())/double(RAND_MAX);
        y = double(rand())/double(RAND_MAX);
        d = x*x + y*y;
        // If inside circle, increase N_i
        if(d<=1.0){
            N_i++;
        }
    //estimate of PI and erel
    piestimate = double(4*N_i)/double(N);
    erel = fabs(piestimate - PI)/PI;
    }
    //output in the required format
    cout<<N<<" "<<N_i<<" "<<N-N_i<<" "<<piestimate<<" "<<erel<<" ";
}
```

```
$ git log --pretty=oneline pi.cpp

98fb26cb85de152db98258186fcd39adf8aed3dc pi.cpp - argc and argv to take in inputs and input verification is also a thing now.
d1a474dfcbf7169ab45bafccf4c90ce7c85d7d95 Changed spacing and indentation.
0a5a1d1231c1145dc42178facb099b68a18bcf92 Replace variable counter by N_i
ff04e1b01d688f60d44cbeae7058249dd775a649 Added comments.
26972942cc3f95619962afefe5b4c5720d351f69 Made N a command line input
d5f900123aa140044fd897be0f042c0908a41979 Added the C++ file for estimating pi as well as the job script for SLURM for Q.4 I
```

- `job_cpp.sh` along with short commit history -

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
##SBATCH -o job_cpp_results.log
##SBATCH -e error.%j.out
#SBATCH -J PI_ESTIMATION_CPP
#SBATCH -p skx-normal
#SBATCH -A cse38018
#SBATCH -t 00:05:00

TIMEFORMAT=%R ## Change time format to give only real time value, got this from stack overflow

## Create empty pi.compiled.log file
>pi.compiled.log


### Output stores stdout of ./pi.sh and stderr of time

g++ pi.cpp -o ./b.out
{ time ./b.out 10  >> pi.compiled.log;} 2>> pi.compiled.log
{ time ./b.out 100  >> pi.compiled.log;} 2>> pi.compiled.log
{ time ./b.out 500  >> pi.compiled.log;} 2>> pi.compiled.log
{ time ./b.out 1000  >> pi.compiled.log;} 2>> pi.compiled.log
{ time ./b.out 5000  >> pi.compiled.log;} 2>> pi.compiled.log
{ time ./b.out 10000  >> pi.compiled.log;} 2>> pi.compiled.log
{ time ./b.out 50000  >> pi.compiled.log;} 2>> pi.compiled.log
```

```
$ git log --pretty=oneline job_cpp.sh

5d8a4ad4e3ac00f9d99e5a8ec7a9207a91774789 pi.compiled.log empties before every run of job_cpp.sh and slight change in input format since pi.cpp takes input from argc and argv (this is the next commit)
6aafa778ec326892d3ba527933b4ac59ecb3b4dc Removing the temporary files created within the code at the end of the code.
aae1638c3b14ad1b3d8c601214381f6d509cfbca Removed ibrun since I don't see use for it here and it is messing up the pi.compiled.log file
ab2737f0fe96c17eaca8a357a5dbaf50c8168df4 Suppressing output apart from our required ones.
6092153905ef880127c70aba3e36b2e0e0ab7abf Renamed a.out as b.out, since a.out is supposed to be the executable of last question.
b94480db22742c70ed3e481da303eeda18e8ece6 No longer getting error output since it is not useful anymore.
b01b29c37f9b7a98c1c708f248bfc6af5063a78d Removed the send emails feature from job scripts, also changed jobs in jobfile.
780b6f11d76b23deb7034603689f3f49059968b5 with ibrun now
d0a2b5205a17af6ef2f944e6511acbd00d786cc7 with ibrun now
40182d31f840fefa52ed89de789ab64debd9c296 Changed job name.
1ffa091d4c7079d3da341b91b3d98efd100addf7 Similar to job.sh, but for cpp files. The input from terminal is read into a file and fed to a.out.
```

Log files - `pi.compiled.log` is the same as the one in the question. The format of the

output is

`N_samples N_i N_o pi_estimate e_rel time`.

- `pi.compiled.log`

```
10 6 4 2.4 0.236056 0.004
100 75 25 3 0.0450703 0.003
500 371 129 2.968 0.0552563 0.003
1000 767 233 3.068 0.0234253 0.003
5000 3915 1085 3.132 0.00305344 0.003
10000 7836 2164 3.1344 0.00228949 0.003
50000 39113 10887 3.12904 0.00399563 0.004
```

We see that the compiled code takes a much lesser time to execute. Also, no appreciable difference is seen in time as we increase the number of samples. But for a sufficiently large number of samples, the time will increase as the number of samples is increased.

III) Using the compiled binary created for Part II and the parametric job launcher demo'd in class, create a third SLURM job script and any necessary files/scripts to run your binary (in parallel) until a certain level of precision is obtained for the π estimate. Recall that we can use the job launcher as a convenience mechanism to aggregate multiple independent jobs by specifying individual tasks to perform in a text file. The general approach is as follows:

- Configure your job script to request 1 Skylake node and 48 cores
- For efficiency, design your job to perform samples in chunks and continue iterating until the desired level of precision is obtained. Let the chunk size be $N_{\text{chunk}} = 960,000,000$ samples (note that this value is evenly divisible by 48).
- At each iteration, run your binary in parallel to complete the N_{chunk} samples. Once completed, use the aggregate results to compute the current estimated value for π (using results from current and all previous iterations). If the value is not within the desired tolerance, execute another iteration. Repeat until the desired tolerance is achieved.
- Since we expect each run of your executable to take roughly the same amount of time, use a simple scheduler with the parametric job launcher by including the following in your job script: `export LAUNCHER_SCHED=interleaved`
- Stopping criteria: the desired tolerance to achieve is: $e_{\text{rel}} < 5.0 \times 10^{-6}$
- At each iteration, append the current iteration, $N_{\text{samples}}, \pi_{\text{est}}, e_{\text{rel}}$, and the accumulated runtime (in secs) to a file named `iter.log`.

Runs: Run your parametric job on Stampede2 and commit the job script, launcher related files, `iter.log`, and the SLURM job output to your repo. The `iter.log` file should have a comment in the first line highlighting the order of the variables. An

overview of the expected file format is shown below. Note that spacing between values is not important, just make sure to have at least 1 space between values.

```
# iter num_samples num_i pi relative_error time_accum
1 960000000 xxx xxx xxx xx
2 1920000000 xxx xxx xxx xx
.
.
.
```

Relevant files -

```
pi.cpp | job3.sh | iters.log | jobfile
```

- `jobfile` (with short commit history)

```
./b.out 20000000 >> temp1.log
./b.out 20000000 >> temp2.log
./b.out 20000000 >> temp3.log
./b.out 20000000 >> temp4.log
./b.out 20000000 >> temp5.log
./b.out 20000000 >> temp6.log
./b.out 20000000 >> temp7.log
./b.out 20000000 >> temp8.log
./b.out 20000000 >> temp9.log
./b.out 20000000 >> temp10.log
./b.out 20000000 >> temp11.log
./b.out 20000000 >> temp12.log
./b.out 20000000 >> temp13.log
./b.out 20000000 >> temp14.log
./b.out 20000000 >> temp15.log
./b.out 20000000 >> temp16.log
./b.out 20000000 >> temp17.log
./b.out 20000000 >> temp18.log
./b.out 20000000 >> temp19.log
./b.out 20000000 >> temp20.log
./b.out 20000000 >> temp21.log
./b.out 20000000 >> temp22.log
./b.out 20000000 >> temp23.log
./b.out 20000000 >> temp24.log
./b.out 20000000 >> temp25.log
./b.out 20000000 >> temp26.log
./b.out 20000000 >> temp27.log
./b.out 20000000 >> temp28.log
./b.out 20000000 >> temp29.log
./b.out 20000000 >> temp30.log
./b.out 20000000 >> temp31.log
./b.out 20000000 >> temp32.log
./b.out 20000000 >> temp33.log
./b.out 20000000 >> temp34.log
./b.out 20000000 >> temp35.log
./b.out 20000000 >> temp36.log
```

```
./b.out 20000000 >> temp37.log
./b.out 20000000 >> temp38.log
./b.out 20000000 >> temp39.log
./b.out 20000000 >> temp40.log
./b.out 20000000 >> temp41.log
./b.out 20000000 >> temp42.log
./b.out 20000000 >> temp43.log
./b.out 20000000 >> temp44.log
./b.out 20000000 >> temp45.log
./b.out 20000000 >> temp46.log
./b.out 20000000 >> temp47.log
./b.out 20000000 >> temp48.log
```

```
$ git log --oneline jobfile
a6f140e Chnaged name of executable to b.out
7768496 pi3.cpp has input verification and accepts input through argc ↩
and argv, hence the jobfile changed too.
23c5d23 Resolving potential race conditions
a7b023d Corrected output format, output filename and some other minor ↩
changes.
324321a Minor corrections
cfb9c8d Minor corrections
217c7be Filled jobfile with 48 processes, 20000000 points in pi3.cpp and↩
 minor modification to job3.sh
b01b29c Removed the send emails feature from job scripts, also changed ↩
jobs in jobfile.
bc50b41 Using launcher now
```

Scripts - `job3.sh` is the SLURM job file and `pi.cpp` is the same as the last part, whose executable runs on individual nodes and sends results to `temp${i}.log` which is then post-processed by `job3.sh`. The final results, of course, are collected in `iters.log`. The command to compile `pi.cpp` is given inside `job3.sh` itself and creates the executable `b.out`.

- `job3.sh` (with short commit history)

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 48
##SBATCH -o job.log
##SBATCH -e error.%j.out
#SBATCH -J PI_PARALLEL
#SBATCH -p skx-normal
#SBATCH -A cse38018
#SBATCH -t 00:10:00

#Buit in bash variable that automatically updates after initialization
SECONDS=0
# Actual value of PI
PI=$(echo "scale =10; 3.14159265358979323846" | bc -l)
#Compile C++ code
g++ pi.cpp -o ./b.out
```

```
#setup launcher environment
module purge
module load TACC
module load launcher
export LAUNCHER_SCHED=interleaved
export LAUNCHER_PLUGIN_DIR=$LAUNCHER_DIR/plugins
export LAUNCHER_RMI=SLURM
#define the file with the commands to run in parallel
export LAUNCHER_JOB_FILE=./jobfile

# Delete temp*.log files if they exist and then create a new one
for((i = 1; i<=48;i++)); do
if [ -e "temp${i}.log" ]; then
  rm "temp${i}.log"
fi
touch "temp${i}.log"
done

TIMEFORMAT=%R ## Change time format to give only real time value, got this from stack overflow

# Epsilon is the desired accuracy
epsilon=$(echo "scale=10; 0.000005" | bc -l)
erel=1
iter=0 #Count number of iterations
pi_average=0 #Average value over all iterations
num_i=0 #Total number of points inside circle across all iterations
pi_iter_value=0 #pi value for this iteration

## empty iter.log file
>iter.log


echo "# iter num_samples num_i pi relative_error time_accum">>iter.log

while [ 1 ];do

    if [[ $(echo "scale=10; $erel < $epsilon" | bc -l) = 1 ]]; then
        break
    fi
    #empty the temp.log files before running the launcher again
    for((i = 1; i<=48;i++)); do
        >temp${i}.log
    done
    #run the launcher
    $LAUNCHER_DIR/paramrun
    # update iteration number and find new value of pi_average and relative error
        #Each core outputs to a different file to prevent race conditions.
    # awk will read from all temp* files at once, which is cool.
    iter=$(($iter+1))
    pi_iter_value=$(awk 'BEGIN{z=0;}{z = z + $4;}END{z = z/48.0;print z;}' temp*.log| bc -l)
        num_i=$(($num_i+$(awk 'BEGIN{z=0;}{z = z + $2;}END{print z;}' temp*.log| bc -l)))
```

```
        pi_average=$(echo "scale=10; ($pi_average*($iter-1)+↩
        $pi_iter_value)/$iter" | bc -l)
        pi_average_2=$(echo "scale=10; 4*$num_i/$(($iter*960000000))" | ↩
        bc -l)
    erel=$(echo "scale=20; sqrt((($pi_average_2-$PI)/$PI)^2)" | bc -l)
    # Print output in desired format
    echo "$iter $(($iter*960000000)) $num_i $pi_average_2 $erel $SECONDS↩
    ">>iter.log
done

# Remove the temporary log file
rm temp*
```

```
$ git log --oneline job3.sh

2e4d85d (HEAD -> master, origin/master, origin/HEAD) Changed sequence of↩
 code slightly so that the header can be seen in iter.log
a6f140e Chnaged name of executable to b.out
2c19635 iter.log empty file is created before the while loop.
2ffb2b1 Resolving race conditions, removing a.out from git and getting a↩
 new iter.log file.
23c5d23 Resolving potential race conditions
809d82b time_accum is calculated now
6bb6250 Resolving conflicts
f2fa676 Commented the files and renamed the counter variables as N_i.
bb0c3e5 Commented the files and renamed the counter variables as N_i.
76e3b6c Minor mistakes corrected in job3.sh in using temp.log and iter.↩
log, SBATCH -o -e supressed for now
a7b023d Corrected output format, output filename and some other minor ↩
changes.
2d3e209 Properly working code, only needs to be timed and the output ↩
format needs to be corrected but the results are correct.
5ddbd8d job3.sh that works pretty well, a few issues though with ↩
breaking the while loop.
cfb9c8d Minor corrections
217c7be Filled jobfile with 48 processes, 20000000 points in pi3.cpp and↩
 minor modification to job3.sh
b01b29c Removed the send emails feature from job scripts, also changed ↩
jobs in jobfile.
bc50b41 Using launcher now
116b870 Modified job3.sh and pi3.cpp, will run trial tests now.
b6577d1 Last part of last question.
```

Log files `iters.log` is the same as the one mentioned in the question. The time is in seconds.

- `iters.log`

```
# iter num_samples num_i pi relative_error time_accum
1 960000000 754003007 3.1416791958 .00002754724108091407 26
2 1920000000 1507976297 3.1416172854 .00000784054870337529 51
3 2880000000 2261985383 3.1416463652 .00001709693653143743 77
4 3840000000 3015975224 3.1416408583 .00001534403581917091 102
```

```
5 4800000000 3769969826 3.1416415216 .00001555517076666148 127
6 5760000000 4523956069 3.1416361590 .00001384820217104010 154
7 6720000000 5277921612 3.1416200071 .00000870689272013845 178
8 7680000000 6031865265 3.1415964921 .00000122183574591677 204
```