

**Monte Carlo Algorithm OpenMPI**

**CSC 512**

Drew Peusch, Keith Conley

April 13, 2017

**Introduction**

The Monte Carlo method is a broad class of computational algorithms that rely on repeated random sampling in order to obtain results. It is often used in physical and mathematical problems. In principle, Monte Carlo methods can be used to solve any problem that is inherently probabilistic in nature. In this case it will be used to estimate PI.

The pattern of a Monte Carlo problem follows a pattern. First, a domain of possible inputs must be defined. Second random inputs from a probability distribution over the domain must be generated. Third, Perform a deterministic computation on the inputs, and fourth, aggregate the results.

For discerning pi. First a circle will be created. Points will be scattered uniformly over the circle. The total number of points will be counted. The ratio of the two counts is an estimate of the ratio of the two areas which is Pi/4.

|  |
| --- |
| for ( i=0; i<nIterations; i++) |
| { |
| x = (double)rand()/RAND\_MAX; |
| y = (double)rand()/RAND\_MAX; |
| z = sqrt(x\*x+y\*y); |
| if (z<=1) count++; |
| } |
|  |

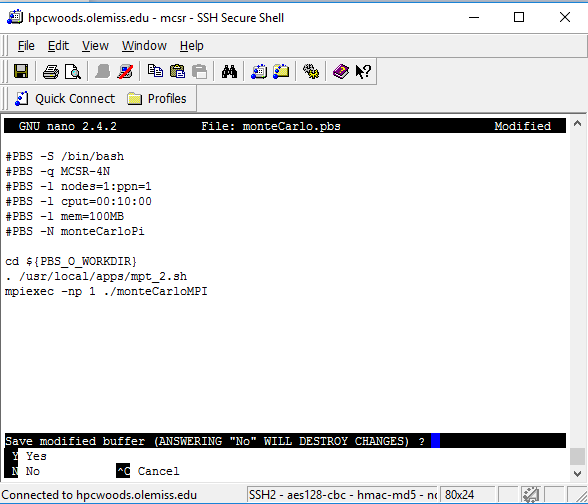
pi = (double)count/nIterations\*4.0;

In the code above, first a point is randomly generated and fit inside of the circle. This is done for nIterations which is a large number. This is then accumulted and multiplied by 4 to get PI.

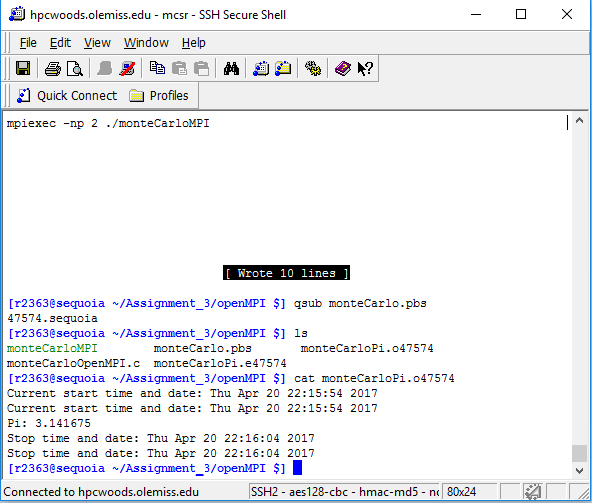
**DATA**

OpenMPI compile command: gcc -lmpi -lm -o monteCarloMPI monteCarloOpenMPI.c

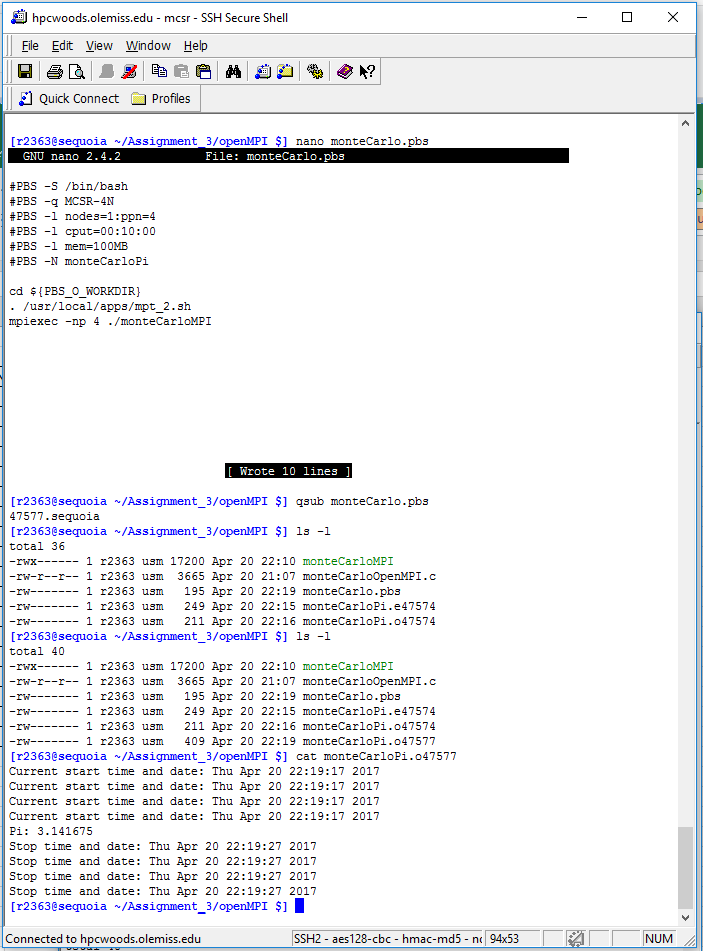
Pbs file 1 process openMPI



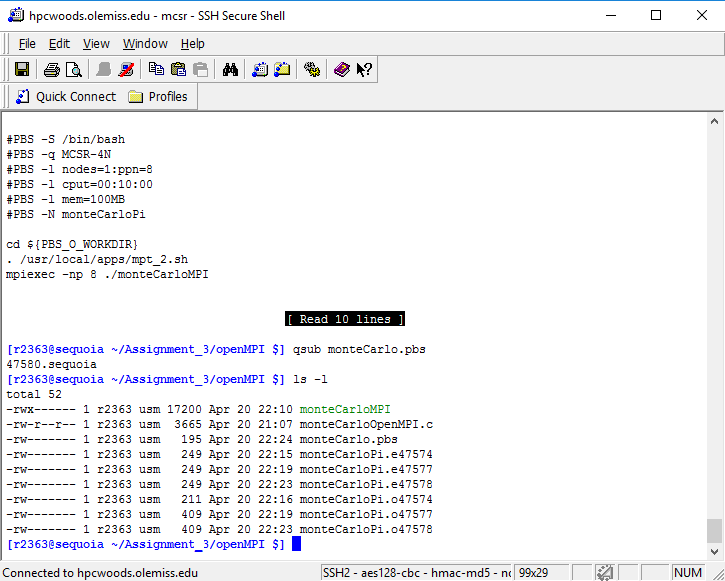
PBS file 2 Process and running openMPI

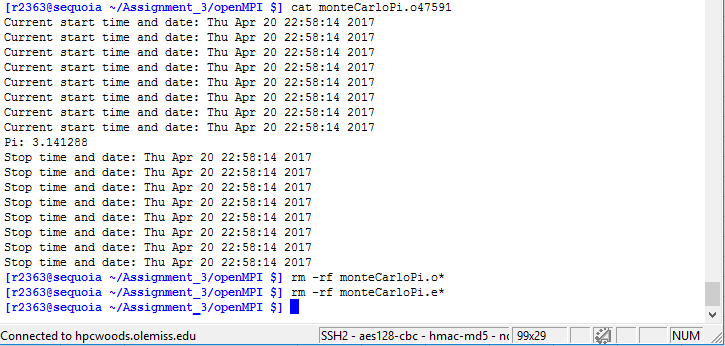


PBS file 4 process + running open MPI



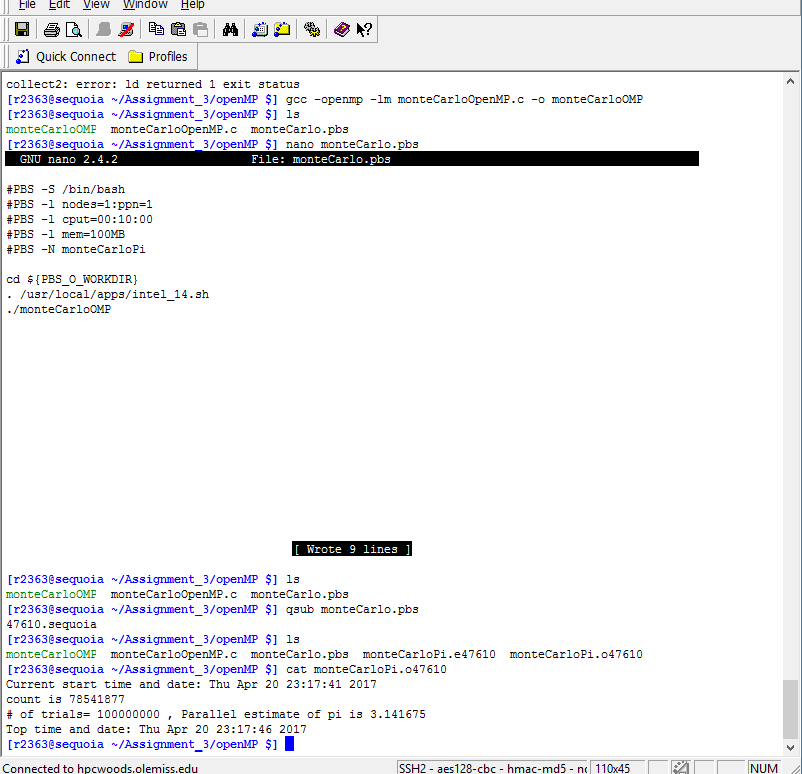
PBS file 4 process + running open MPI



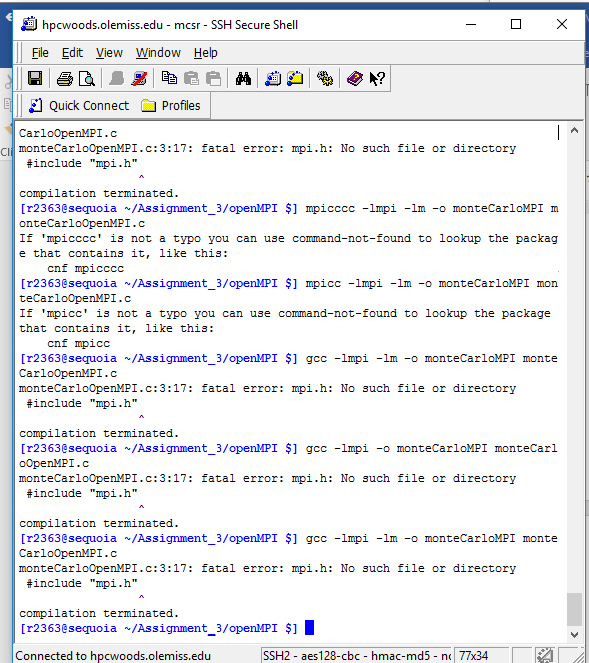


OpenMP

Processes 1



Error with collecting N2 and N1 Data



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | N1 = 1,000,000,000 | | N2 = 10,000,000 | | N3 = 100,000,000 | |
| NODE = 1 | Threads/Processes | MPI | OpenMP | MPI | OpenMP | MPI | OpenMP |
|  | 1 |  | 73(s) |  | 1(s) | 20 (s) | 5(s) |
|  | 2 |  | 36(s) |  | 1(s) | 10(s) | 5(s) |
|  | 4 |  | 24(s) |  | 1(s) | 10(s) | 5(s) |
|  | 8 |  | 22(s) |  | 1(s) | 1(s) | 5(s) |
|  | Estimated Pi OpenMP t= 4 |  | 3.141367 |  | 3.1417 |  | 3.141675 |
|  | Estimated Pi OpenMPI t= 4 |  |  |  |  | 3.141675 |  |
|  | Estimated PI Serial | 3.141537 |  |  |  | 3.1417 |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  | N3 = 100,000,000 |  |  |  |
| NODE = 2 | Threads/Processes |  |  | MPI |  |  |  |
|  | 1 |  |  | 2(s) |  |  |  |
|  | 2 |  |  | 1(s) |  |  |  |
|  | 4 |  |  | 1(s) |  |  |  |
|  | 8 |  |  | 1(s) |  |  |  |
|  |  |  |  |  |  |  |  |
|  | Estimated Pi OpenMPI t= 4 |  |  | 3.141288 |  |  |  |
|  |  |  |  |  |  |  |  |

**Code**

monteCarloSerial.c

|  |
| --- |
| #include <stdlib.h> |
| #include <stdio.h> |
| #include <math.h> |
| #include <string.h> |
| #include <time.h> |
|  |
|  |
| #define SEED 35791246 |
| void startTime(void); |
| void stopTime(void); |
|  |
| int main(int argc, char\*\* argv) |
| { |
| startTime(); |
|  |
| double x,y, z; |
| long int i,count=0; /\* # of points in the 1st quadrant of unit circle \*/ |
| double pi; |
|  |
| long int nIterations = 100000000; |
| /\* initialize random numbers \*/ |
| srand(SEED); |
| count=0; |
| for ( i=0; i<nIterations; i++) |
| { |
| x = (double)rand()/RAND\_MAX; |
| y = (double)rand()/RAND\_MAX; |
| z = sqrt(x\*x+y\*y); |
| if (z<=1) count++; |
| } |
|  |
| pi = (double)count/nIterations\*4.0; |
|  |
| printf("count is %ld \n", count); |
| printf("# of trials= %ld , Serial estimate of pi is %6.6f \n",nIterations,pi); |
|  |
| stopTime(); |
| } |
|  |
| void startTime(void) |
| { |
| time\_t rawtime; |
| struct tm \* timeinfo; |
|  |
| time ( &rawtime ); |
| timeinfo = localtime ( &rawtime ); |
| printf ( "Current start time and date: %s", asctime (timeinfo) ); |
| } |
|  |
| void stopTime(void) |
| { |
| time\_t rawtime; |
| struct tm \* timeinfo; |
|  |
| time ( &rawtime ); |
| timeinfo = localtime ( &rawtime ); |
| printf ( "Top time and date: %s", asctime (timeinfo) ); |

}

monteCarloOpenMP.c

|  |
| --- |
| #include <stdlib.h> |
| #include <stdio.h> |
| #include <math.h> |
| #include <string.h> |
| #include <time.h> |
|  |
|  |
| #define SEED 35791246 |
| void startTime(void); |
| void stopTime(void); |
|  |
| int main(int argc, char\*\* argv) |
| { |
| startTime(); |
|  |
| double x,y, z; |
| long int i,count=0; /\* # of points in the 1st quadrant of unit circle \*/ |
| double pi; |
|  |
| long int nIterations = 1000000000; |
| /\* initialize random numbers \*/ |
| srand(SEED); |
| count=0; |
|  |
| int numthreads=8; |
| #pragma omp parallel private(x, y, z, i) reduction(+:count) num\_threads(numthreads) |
| { |
| for ( i=0; i<nIterations; i++) |
| { |
| x = (double)rand()/RAND\_MAX; |
| y = (double)rand()/RAND\_MAX; |
| z = sqrt(x\*x+y\*y); |
| if (z<=1) count++; |
| } |
| } |
|  |
| pi = (double)count/nIterations\*4.0; |
|  |
| printf("count is %ld \n", count); |
| printf("# of trials= %ld , Parallel estimate of pi is %6.6f \n",nIterations,pi); |
|  |
| stopTime(); |
| } |
|  |
| void startTime(void) |
| { |
| time\_t rawtime; |
| struct tm \* timeinfo; |
|  |
| time ( &rawtime ); |
| timeinfo = localtime ( &rawtime ); |
| printf ( "Current start time and date: %s", asctime (timeinfo) ); |
| } |
|  |
| void stopTime(void) |
| { |
| time\_t rawtime; |
| struct tm \* timeinfo; |
|  |
| time ( &rawtime ); |
| timeinfo = localtime ( &rawtime ); |
| printf ( "Top time and date: %s", asctime (timeinfo) ); |

}

monteCarloOpenMPI.c

|  |
| --- |
| #include <stdio.h> |
| #include <stdlib.h> |
| #include "mpi.h" |
| #include <math.h> |
| #include <string.h> |
| #include <time.h> |
| #define SEED 35791246 |
|  |
| void startTime(void); |
| void stopTime(void); |
|  |
| int main(int argc, char\* argv[]) |
| { |
| long niter = 10000; |
| int myid; //holds process's rank id |
| double x,y; //x,y value for the random coordinate |
| int i, count=0; //Count holds all the number of how many good coordinates |
| double z; //Used to check if x^2+y^2<=1 |
| double pi; //holds approx value of pi |
| int nodenum; |
|  |
|  |
| startTime(); //Prints starting time |
|  |
|  |
| MPI\_Init(&argc, &argv); //Start MPI |
| MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid); //get rank of node's process |
| MPI\_Comm\_size(MPI\_COMM\_WORLD, &nodenum); |
| int recieved[nodenum]; |
| long recvniter[nodenum]; |
| srand(SEED); //Give rand() a seed value |
|  |
| if(myid != 0) |
| { |
| for (i=0; i<niter; ++i) //main loop |
| { |
| x= ((double)rand())/RAND\_MAX; //gets a random x coordinate |
| y =((double)rand())/RAND\_MAX; //gets a random y coordinate |
| z = sqrt(x\*x+y\*y); //Checks to see if number in inside unit circle |
| if (z<=1) |
| { |
| count++; //if it is, consider it a valid random point |
| } |
| } |
| for(i=0; i<nodenum; ++i) |
| { |
| MPI\_Send(&count, |
| 1, |
| MPI\_INT, |
| 0, |
| myid, |
| MPI\_COMM\_WORLD); |
| MPI\_Send(&niter, |
| 1, |
| MPI\_LONG, |
| 0, |
| myid, |
| MPI\_COMM\_WORLD); |
| } |
| } |
| else if (myid == 0) |
| { |
| for(i=0; i<nodenum; ++i) |
| { |
| MPI\_Recv(&recieved[i], |
| nodenum, |
| MPI\_INT, |
| MPI\_ANY\_SOURCE, |
| MPI\_ANY\_TAG, |
| MPI\_COMM\_WORLD, |
| MPI\_STATUS\_IGNORE); |
| MPI\_Recv(&recvniter[i], |
| nodenum, |
| MPI\_LONG, |
| MPI\_ANY\_SOURCE, |
| MPI\_ANY\_TAG, |
| MPI\_COMM\_WORLD, |
| MPI\_STATUS\_IGNORE); |
| } |
| } |
|  |
| if (myid == 0) //if root process |
| { |
| int finalcount = 0; |
| long finalniter = 0; |
| for(i = 0; i<nodenum; ++i) |
| { |
| finalcount += recieved[i]; |
| finalniter += recvniter[i]; |
| } |
|  |
| pi = ((double)finalcount/(double)finalniter)\*4.0; //p = 4(m/n) |
| printf("Pi: %f\n", pi); //Print the calculated value of pi |
|  |
| } |
|  |
| MPI\_Finalize(); //Close the MPI instance |
|  |
| stopTime(); //Prints Final time at stop |
|  |
| return 0; |
| } |
|  |
|  |
| void startTime(void) |
| { |
| time\_t rawtime; |
| struct tm \* timeinfo; |
|  |
| time ( &rawtime ); |
| timeinfo = localtime ( &rawtime ); |
| printf ( "Current start time and date: %s", asctime (timeinfo) ); |
| } |
|  |
| void stopTime(void) |
| { |
| time\_t rawtime; |
| struct tm \* timeinfo; |
|  |
| time ( &rawtime ); |
| timeinfo = localtime ( &rawtime ); |
| printf ( "Top time and date: %s", asctime (timeinfo) ); |

}

Conclusion

In conclusion, the Monte Carlo method works as intended.

|  |  |  |
| --- | --- | --- |
| Questions | Yes/No | Note |
| Can the Serial  Program Be  compiled and  executed? |  |  |
| Is the Serial Result Correct? |  |  |
| Are the results from the OpenMP program correct? |  |  |
| Are the results from the MPI program correct? |  |  |
| Are the parallel results consistent with the serial results?  Present the results properly using tables ad figures to analyze the results. |  |  |