INDEX

Acknowledgment

I have completed my assignment by learning from the following sources

1. Given Support Materials
   a) https://ieeexplore.ieee.org/document/6298658
   b) https://www.csee.umbc.edu/~tsimo1/CMSC483/cs220/code/pi/pth_pi_sem.c
2. https://www.geeksforgeeks.org/multithreading-c-2/
3. https://www.geeksforgeeks.org/use-posix-semaphores-c/
4. https://www.ibm.com/docs/en/i/7.2?topic=lf-atol-atoll-convert-character-string-long-long-long-integer
5. https://www.quora.com/What-is-the-smallest-C-program-generating-pi-without-using-the-math-library
6. http://www.geom.uiuc.edu/~huberty/math5337/groupe/digits.html
7. https://www.codesansar.com/c-programming-examples/calculate-value-pi-using-leibniz-formula.htm

NOTE –

I have not copied anything from these sites, such as programs,

Only have taken the idea of the general syntax

Pre-Requisite

How To Make Executable and Run Programme

1) Unzip File at location x
2) Open Terminal and change directory to that location x
3) Now enter command make.
   This will create Executables
4) Enter Command make run to run the executables
5) Give arguments by writing ./all.out followed by arguments

Structure Of Programme

Program is made up of directory structure which includes

1) src – Source Folder, It contains all necessary .c files to make object files
   a) main.c - contains main code, which can call other .c files and is used to create and join the thread. Also used to initialize semaphores and destroy them. It also prints everything needed as an answer
   b) leibniz.c - contains code of Leibniz method, which can run on serial programming
   c) euler.c - contains code of euler method which can run on serial programming
   d) pleibniz.c - contains code of Leibniz method which can run on parallel programming
   e) peuler.c - contains code of euler method which can run on parallel programming
   f) time.c – code to note the time of an instance when Time() is called
2) obj – contains object file created from above code
3) inc – contains header file useful at the time of linking
4) lib – contains libraries which can be used in future

Programming

Serial Programming

It includes running the program as it is without any modification.

It may turn fast for one small program, but the program which has too many IO or large computation, this may turn slow.

This is because in this, when a program is doing IO, no other program is not allowed to run, or when there is large computation, only a single core of CPU may be utilized to carry out the whole process.

Parallel Programming

It includes running the program by dividing it into many threads and running on multiple processors at the same time

It may turn fast for large programs, but the program which has no IO or significantly less computation, this may turn slow.

It is faster for large programs because in this, when a program is doing IO, other programs are allowed to run. When there is large computation, several cores of CPU may be utilized to carry out the whole process by dividing the program into threads and keep them in sync with the help of semaphores/mutex.

For small programs, it gives slow performance because making threads and keeping in sync the shared variables and resources needs extra overhead of time. Thus if computation is small, unnecessary time is wasted in creating threads and controlling semaphore.

Time of execution of small programs and large programs, for both serial and parallel, is given in conclusion

Pi

Pi or π is having a value of 3.1459……, Its exact value can be seen here,

http://www.geom.uiuc.edu/~huberty/math5337/groupe/digits.html

It is usually defined as the ratio of the circumference of a circle to its twice its radius.

It is an irrational and real number which do not have any recurring part. This property of pie makes it difficult to be used in the calculation, which needs precision.

Several methods have been defined to calculate the approximate value of pi, to make a precise calculation. Two of them are Euler and Leibniz, which are described on the next page.

Leibniz Method

I have used this because it needs a very less amount of calculation at each step; thus, the error from calculation will be less. Also, as it computes pi for very large iteration thus parallel performance speedup is easy to shown in this.

It is a method to calculate the approximate value of pi.

It is based on expansion inverse of tan x, with x value as 1

As $tan\,45^\circ = tan\,(\pi/4) = 1.0$

Thus, $tan^{-1}\,1.0 = \pi/4$

Thus $\pi = 4\,tan^{-1}\,1.0$.....Eqt. 1

Now $tan^{-1}\,x = x - \frac{x^3}{3} + \frac{x^5}{5} \ldots\ldots\ldots\ldots + (-1)^n\,\frac{x^{2n+1}}{2n+1} + \ldots\ldots\ldots$

In general, we have, $tan^{-1}\,x = \sum_{n=0}^{\infty}(-1)^n\,\frac{x^{2n+1}}{2n+1}\,, -1 <= x <= 1$

Now from eqt. 1, we have,

$$\pi = 4 * \sum_{n=0}^{\infty}(-1)^n\,\frac{x^{2n+1}}{2n+1}, x = 1$$

$$\pi = 4 * \sum_{n=0}^{\infty}(-1)^n\,\frac{1}{2n+1}$$

Thus we have to calculate this equation till infinity to get our approx. value of pi.

It is seen that this method very closely converges to the value of pi when iterated over 1 million times or more.

Relation between the value of n, Accuracy, the time needed, and value of pi can be seen from the plot and table given below

From this, we can conclude that for increasing value of n, our Accuracy is also growing, as the error is decreasing, but also time is increasing drastically. It is for thread value as 10

Accuracy is found the same in both parallel and serial computation.

The optimal n keeping in mind time and Accuracy would be n= 10^9

No of iterations	Error	Time (serial)	Time (parallel)	Value of pi(stored in Double)
10	0.099753	0.000000	0.000927	3.04183961892
100	0.00999975	0.000001	0.000753	3.13159290355
1000	0.001	0.000010	0.000668	3.14059265383
10000	0.0001	0.000097	0.000707	3.14149265359
100000	1E-05	0.001008	0.001419	3.14158265358
1000000	1E-06	0.010096	0.009241	3.14159165358
10000000	1E-07	0.100438	0.093092	3.14159255358
100000000	1.00005E-08	1.009978	0.650355	3.14159264358
1000000000	1.00174E-09	9.966874	5.265899	3.14159265258
10000000000	1.01447E-10	100.959642	53.04798	3.14159265348

Table1. Leibniz computation

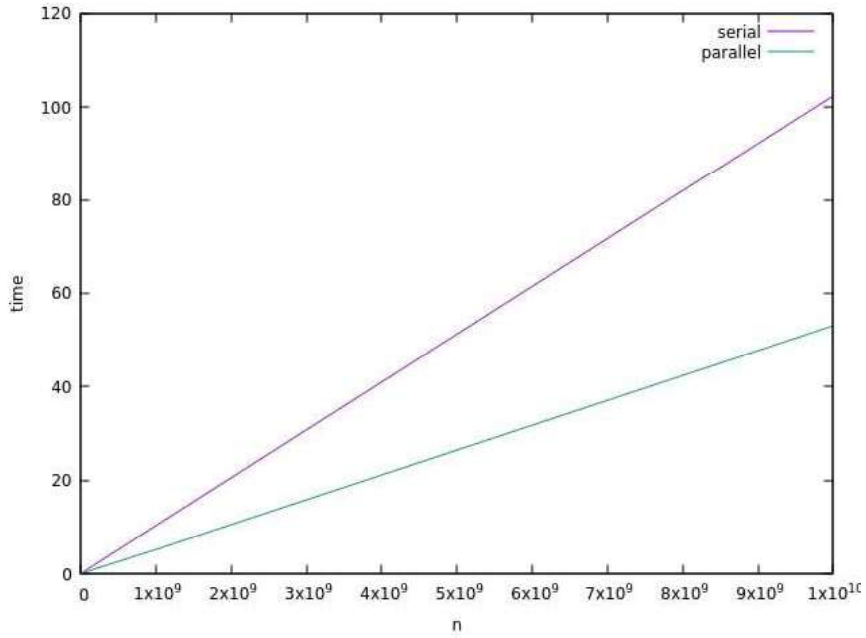

Fig1. Serial V/S. Parallel for Leibniz

Euler Method

I have used this because it needs a very less amount of calculation at each step; thus, the error from calculation will be less. Also, as it computes pi for very small iteration thus, serial performance is easy to show in this.

It is a method to calculate the approximate value of pi.

It is based on the summation of expansion inverse of tan x, with x value as of 1/3 and 1/2

As $tan^{-1}\frac{1}{2} + tan^{-1}\frac{1}{3} = \pi/4$

Thus $\pi = 4\ (tan^{-1}\frac{1}{2} + tan^{-1}\frac{1}{3})$……..Eqt. 1

Now $tan^{-1}x = x - \frac{x^3}{3} + \frac{x^5}{5} \ldots\ldots\ldots\ldots + (-1)^n \frac{x^{2n+1}}{2n+1} + \ldots\ldots\ldots$

In general, we have, $tan^{-1}x = \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{2n+1}, -1 <= x <= 1$

Now from eqt. 1, we have,

$$\pi = 4 * \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{2n+1}, x = \frac{1}{2}, \frac{1}{3}$$

$$\pi = 4 * \sum_{n=0}^{\infty}(-1)^n \frac{(\frac{1}{2})^{2n+1} + (\frac{1}{3})^{2n+1}}{2n+1}$$

Thus we have to calculate this equation till infinity to get our approx. value of pi.

It is seen that this method very closely converges to the value of pi when iterated over 500 times or more.

Relation between the value of n, Accuracy, the time needed, and value of pi can be seen from the plot and table given below

From this, we can conclude that for increasing value of n, our Accuracy is also growing, as the error is decreasing, but also time is increasing drastically. It is for thread=10.

Accuracy is found the same in both parallel and serial computation.

The optimal n keeping in mind time and Accuracy would be n= 500

No of iterations	Error	Time (serial)	Time (parallel)	Value of pi(stored in Double)
10	7.39834E-08	0.000002	0.000741	3.141593
110	4.44089E-16	0.000009	0.000630	3.141593
210	8.88178E-16	0.000016	0.000639	3.141593
310	8.88178E-16	0.000024	0.000672	3.141593
410	8.88178E-16	0.000033	0.000608	3.141593
510	8.88178E-16	0.000056	0.000577	3.141593
610	8.88178E-16	0.000051	0.000897	3.141593
710	8.88178E-16	0.000058	0.000584	3.141593
810	8.88178E-16	0.000064	0.000553	3.141593
910	8.88178E-16	0.000069	0.000601	3.141593
29810	8.88178E-16	0.001476	0.001179	3.141593

Table2. Euler computation

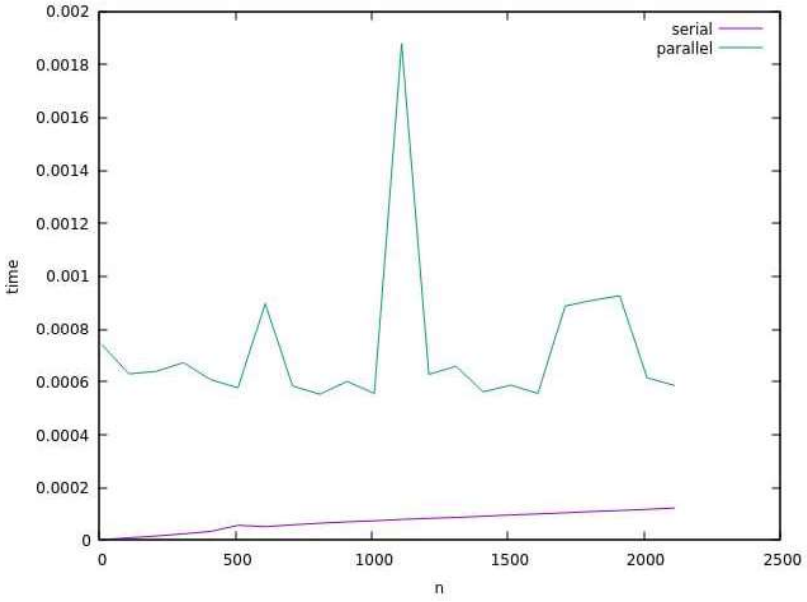

Fig2. Serial V/S. Parallel for Euler

Result

1) I have implemented Euler and Leibniz method in both serial and parallel form
2) I have tweaked parallel computation to avoid unnecessary computation by taking out variables that are calculated every time in each thread
3) Accuracy of serial and parallel in Euler for the same value of iteration is found to be the same
4) Accuracy of serial and parallel in Leibniz for the same value of iteration is found to be the same
5) The Accuracy of Euler is found to be more on n =200 as compared to the Accuracy of Leibniz on n = 10^9
6) Speed_UP of running Euler in for parallel is negative as compared to running Euler in Series
   this is because less no of computation is involved and large overhead of threading and concurrency
7) Speed_UP of running Leibniz in for parallel is positive as compared to running Leibniz in Series.
   It is found to be 2X times for a large value of n = 10^{10}
   This is because other cores of computers are also utilized for the calculation
8) A sample running screenshot of the Programme is given below
9) Time for Euler to calculate pi is also less than the time to calculate pi for Leibniz
10) We found no of bits which are same in both serial and parallel computation is same in both algo
11) The result given below is for n =1000000000 for Leibniz and thread number =10
12) The result given below is for n =500 for Euler and thread number =10

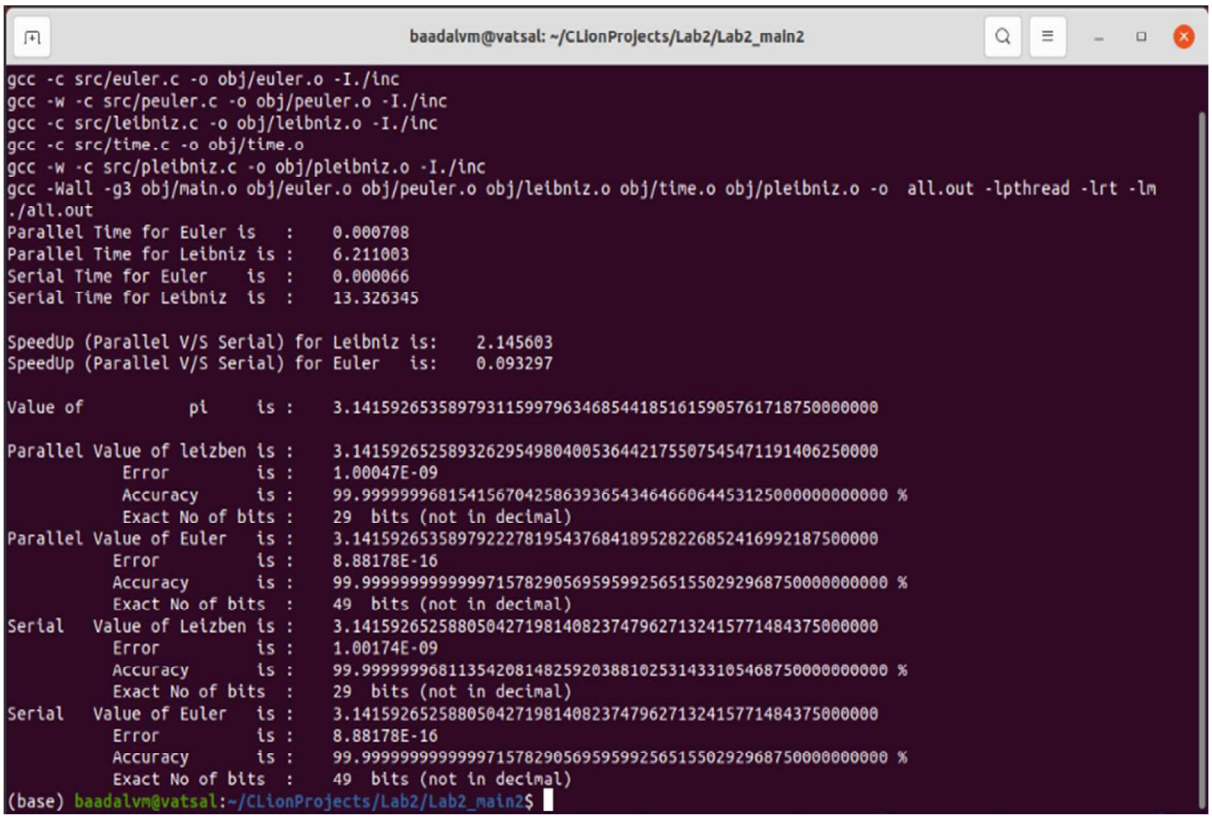

Fig3. Running Screenshot 1

Note Accuracy $= \left|\frac{\text{Observed value}}{\text{Actual value}}\right| * 100$

$= 100 - ((\frac{Error}{Actual\ Value}) * 100)$

Error $= |\text{Actual Value} - \text{Observed Value}|$

frexp() – is used to get the trailing no of 0 after the decimal. Also, it gives an exponent of the error, which is expressed in the power of 2

Conclusion

1) Both Parallel and Serial versions give the same accuracy/error almost for the same no of iteration for the same method – if the implementation is the same
2) Leibniz converges very slow in comparison to Euler for the value of pi
3) For the small value of n, Euler has significantly less error, while for the very large value of n, Leibniz has a significant error
4) For a very large value of n, Leibniz does not come as close to pi as Euler comes in only 200 iteration
5) For small iteration in Leibniz, it does not give the correct value of pi
6) The parallel speedup can be obtained if we tweak algo such that useless calculation is not calculated in every thread in Leibniz
7) Euler can show speedup for parallel for very large value of n, which is not necessary
8) Serial compuatation is preferred for Euler while parallel computation is preferred for Leibniz
9) Leibniz show speed up in its computation due to large value of n
10) Exact no of bits is more in Euler as compared to Leibniz
11) Also, the exact number of bits is found to be the same for both serial and parallel computation in both algo
12) Time taken to run Euler to get high accuracy is very much less than the time taken by Leibniz to get that accuracy
13) The accuracy which is achieved by Euler in 1 s is not achieved by Leibniz in 100 sec also
14) From the above graph and table, it is found that a value of 500 for iteration of Euler would be perfect for getting high accuracy up to 15 decimal digits
15) From the above graph and table, it is seen that a value of 1000000000 for iteration of Leibniz would be perfect for getting

high accuracy up to 8 decimal digits keeping in mind the time taken to compute it

16) Most of the time, the speedup is of about 2 in Leibniz
17) From the graph and table below, we can clearly see that max speed up for Leibniz at $n=10^9$ is achieved at thread = 8
18) But also, it can be seen that for almost any value of thread >2, the speed is almost the same for Leibniz
This is because our CPU has two cores
19) Also, for Euler at n=500, we can clearly see that max speed up is obtained at thread = 4
20) But also it can be seen that for almost any value of thread >2, the speed is almost the same for Euler also

thread	Parallel Euler	Serial Euler	Parallel Leibniz	Serial Leibniz	Speed Up Euler	Speed Up Leibniz
1	0.000453	0.000058	13.124493	11.517509	0.127895	0.87758
2	0.000188	0.000035	5.333881	10.130044	0.186548	1.899188
3	0.000173	0.000032	5.527050	10.454529	0.184828	1.891521
4	0.000145	0.000038	5.732114	10.945011	0.261084	1.909420
5	0.000186	0.000035	5.547630	10.524869	0.188462	1.897183
6	0.000293	0.000054	5.413779	10.269988	0.184703	1.897009
7	0.000329	0.000042	5.681848	10.530343	0.127536	1.853331
8	0.000294	0.000032	5.343040	10.718980	0.108678	2.006158
9	0.000295	0.000035	5.728081	10.640249	0.118836	1.857559
10	0.000341	0.000037	5.658199	10.480041	0.108392	1.852187
11	0.000414	0.000032	5.624445	10.663451	0.077189	1.895912
12	0.000441	0.000031	5.804573	11.017770	0.070270	1.898119
13	0.000558	0.000044	5.759726	10.723555	0.079026	1.861817
14	0.000411	0.000041	5.235549	10.931312	0.099768	2.087902
15	0.000506	0.000050	5.460580	10.655608	0.098917	1.951369
16	0.000615	0.000036	5.747603	10.742886	0.058550	1.869107
17	0.000636	0.000033	5.820568	10.531898	0.051724	1.809428

Table3. Thread Vs. Time

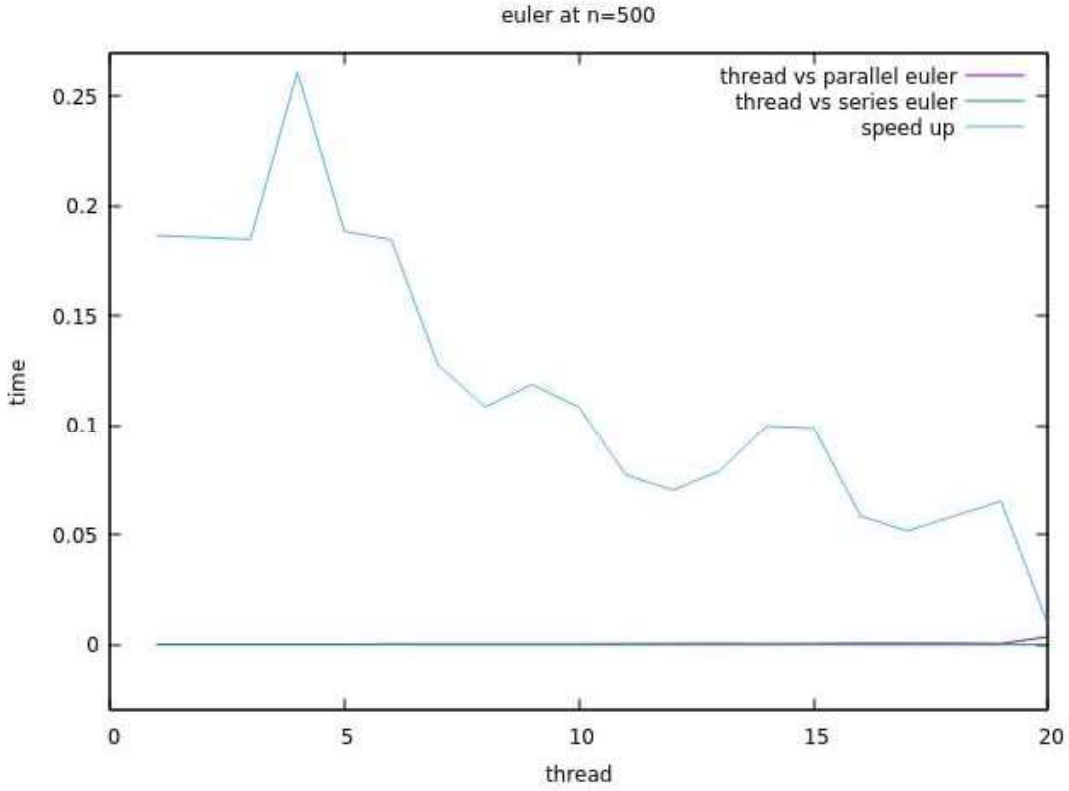

Fig4. Time and speedup for Euler at n=500

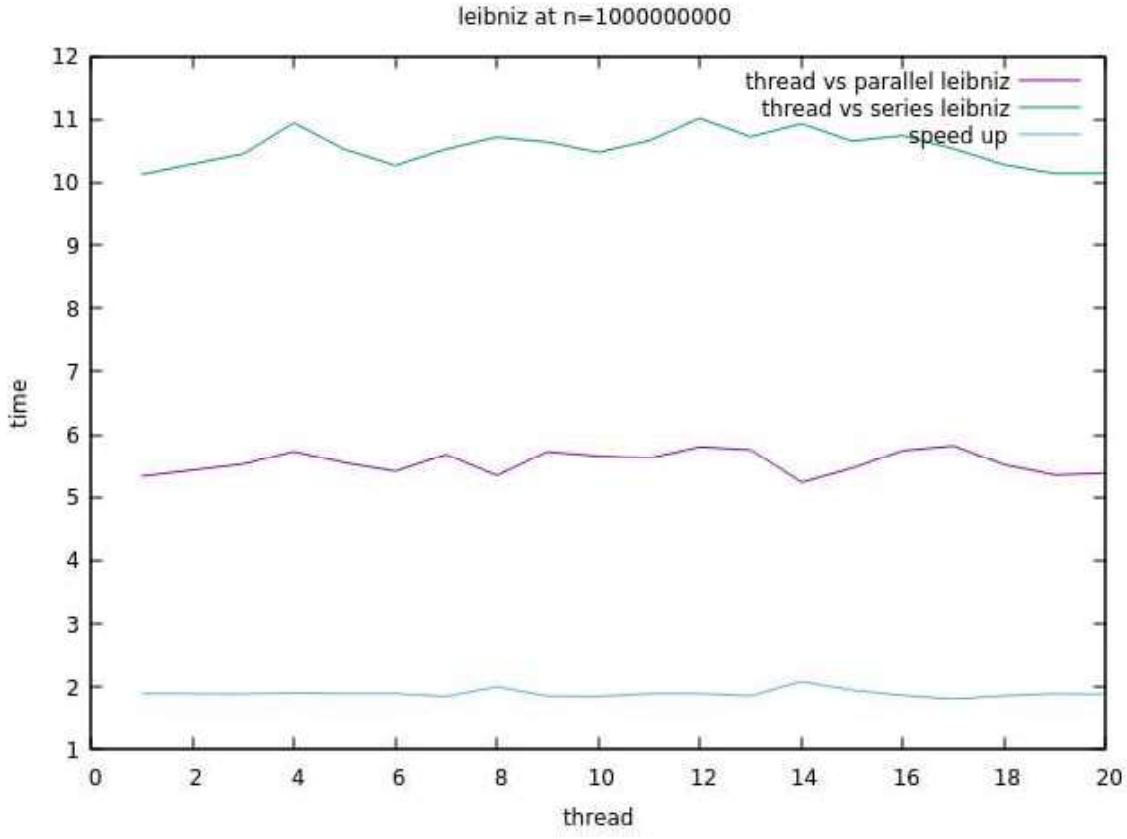

Fig4. Time and speedup for Leibniz at n=1000000000

Future Scope

1) Serial implementation can be made faster by avoiding costlier operations like multiplication by using addition instead of it
2) In most of the cases in serial computation, the value calculated at n^{th} step is dependent on $n-1^{th}$ step
   So further serial computation time can be reduced by taking the help of previously calculated info
3) Pi value can be calculated more accurately and fast by using methods like Ramanujan Method
4) Speed in parallel can be obtained by some redundancy in the equation
5) More accuracy can be achieved by using a precision library

Code

Main.c File

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include "leibniz.h"
4: #include "euler.h"
5: #include "pleibniz.h"
6: #include "peuler.h"
7: #include "time.h"
8: #include <pthread.h>
9: #include <semaphore.h>
10: #include <math.h>
11: typedef long long int lli;
12:
13: sem_t s1,s2;
14: lli ml=1000000000,me=1000;
15: double spil=0.0,spie=0.0,ppie=0.0,ppil=0.0;
16: int t1=10,t2=10;
17: lli ra=0,rb=0;
18: double b1=1.0/2.0;
19: double c1=1.0/3.0;
20: const double pie=atan(1.0)*4;
21:
22:
23: int main(int argc, char **argv){
24: if (argc==3){
25: t1=atoll(argv[1]);
26: t2=atoll(argv[1]);
27: me=atoll(argv[2]);
28: ml=atoll(argv[2]);
29: }
30: if (argc==5){
31: ml=atoll(argv[3]);
32: me=atoll(argv[4]);
33: t1=atoi(argv[1]);
34: t2=atoi(argv[2]);
35: }
36: if (argc==4){
37: ml=atoll(argv[2]);
38: me=atoll(argv[3]);
39: t1=atoi(argv[1]);
40: t2=atoi(argv[1]);
41: }
42: //for(me=10;me>0;me=me+100){
43: spil=0.0,spie=0.0,ppie=0.0,ppil=0.0;
44: int i=0;
45: ra=ml/t1;
46: rb=me/t2;
47:
48: double pstl,penl,pttl1,pttl2,psei;
49: pthread_t th1[t1],th2[t2];
```

```
50: sem_init(&s1,0,1);
51: sem_init(&s2,0,1);
52: pstl=Time();
53: for(i=0;i<t1;i++){
54: pthread_create(&(th1[i]),NULL,pleibniz,(void*)i);
55: }
56:
57: for(i=0;i<t1;i++){
58: pthread_join(th1[i],NULL);
59: }
60: psei=Time();
61: for(i=0;i<t2;i++){
62: pthread_create(&(th2[i]),NULL,peuler,(void*)i);
63: }
64: for(i=0;i<t2;i++){
65: pthread_join(th2[i],NULL);
66: }
67: penl=Time();
68: pttl2=penl-psei;
69: printf("Parallel Time for Euler is : %lf\n",pttl2);
70: pttl1=psei-pstl;
71: printf("Parallel Time for Leibniz is : %lf\n",pttl1);
72:
73:
74: double sstl,senl,sttl1,sttl2,stli;
75: sstl=Time();
76: leibniz();
77: stli=Time();
78: euler();
79: senl=Time();
80: sttl2=senl-stli;
81: printf("Serial Time for Euler is : %lf\n",sttl2);
82: sttl1=stli-sstl;
83: printf("Serial Time for Leibniz is : %lf\n",sttl1);
84: printf("\n");
85:
86:
87: int e=0;
88: double su1=sttl1/pttl1;
89: double su2=sttl2/pttl2;
90: printf("SpeedUp (Parallel V/S Serial) for Leibniz is: %lf\n",su1);
91: printf("SpeedUp (Parallel V/S Serial) for Euler is: %lf\n",su2);
92: printf("\n");
93: printf("Value of pi is : %0.55lf\n",pie);
94: printf("\n");
95: ppil=4*ppil;
96: printf("Parallel Value of leibniz is : %0.55lf\n",ppil);
97: double error=pie-ppil;
98: double acc=(ppil/pie)*100.0;
99: frexp(error,&e);
100: printf(" Error is : %G\n",error);
101: printf(" Accuracy is : %0.55lf %% \n",acc);
102: printf(" Exact No of bits : %d bits (not in decimal)\n", abs(e));
103: ppie=4*ppie;
104: printf("Parallel Value of Euler is : %0.55lf\n",ppie);
105: error=pie-ppie;
106: acc=(ppie/pie)*100.0;
```

```
107: frexp(error,&e);
108: printf(" Error is : %G\n",error);
109: printf(" Accuracy is : %0.55lf %% \n",acc);
110: printf(" Exact No of bits : %d bits (not in decimal)\n",abs(e));
111: spil=4*spil;
112: printf("Serial Value of Leibniz is : %0.55lf\n",spil);
113: error=pie-spil;
114: acc=(spil/pie)*100.0;
115: frexp(error,&e);
116: printf(" Error is : %G\n",error);
117: printf(" Accuracy is : %0.55lf %% \n",acc);
118: printf(" Exact No of bits : %d bits (not in decimal)\n",abs(e));
119: spie=4*spie;
120: printf("Serial Value of Euler is : %0.55lf\n",spil);
121: error=pie-spie;
122: acc=(spie/pie)*100.0;
123: frexp(error,&e);
124: printf(" Error is : %G\n",error);
125: printf(" Accuracy is : %0.55lf %% \n",acc);
126: printf(" Exact No of bits : %d bits (not in decimal)\n",abs(e));
127:
128:
129: /*double error=pie-(4*ppie);
130: printf("%lli %lf %lf %lf %G\n",me,pttl,sttl,4*spie,error);*/
131: sem_destroy(&s1);
132: }
```

Euler.c File for Serial Euler Compuatation

```
1: #include <stdio.h>
2: #include <math.h>
3: typedef long long int lli;
4:
5: lli me;
6: double spie;
7: int t2;
8: double b1,c1;
9:
10: void euler() {
11: lli i=0;
12: double a,b,c;
13: for(i=0;i<me;i++){
14: a=(2*i)+1;
15: b=pow(b1,a);
16: c=pow(c1,a);
17: if(i%2==0){
18: spie=spie+((b+c)/a);
19: }
20: else{
21: spie=spie-((b+c)/a);
22: }
23: }
24: }
```

pEuler.c File for Parallel Euler Compuatation

```
1: #include <stdio.h>
2: #include <math.h>
3: #include <semaphore.h>
4: typedef long long int lli;
5:
6: sem_t s2;
7: lli me;
8: double ppie;
9: int t2;
10: double b1,c1;
11: lli rb;
12:
13: void *peuler(void* i) {
14: int ii=(int)i;
15: double a,b,c,pi=0.0;
16: lli lb=ii*rb;
17: lli ub=lb+rb;
18: for(;lb<ub;lb++){
19: a=(2*lb)+1;
20: b=pow(b1,a);
21: c=pow(c1,a);
22: if(lb%2==0){
23: pi=pi+((b+c)/a);
24: }
25: else{
26: pi=pi-((b+c)/a);
27: }
28: }
29: sem_wait(&s2);
30: fflush(stdout);
31: ppie=ppie+pi;
32: sem_post(&s2);
33: }
```

Leibniz.c File for Serial Leibniz Compuatation

```
1: #include <stdio.h>
2: typedef long long int lli;
3:
4: lli ml;
5: double spil;
6: int t1;
7:
8: void leibniz() {
9: lli i=0;
```

```
10: double a;
11: for(i=0;i<ml;i++){
12: if(i%2==0){
13: a=(2*i)+1;
14: spil=spil+(1/a);
15: }
16: else{
17: a=(2*i)+1;
18: spil=spil-(1/a);
19: }
20: }
21: }
```

pLeibniz.c File for parallel Leibniz Compuatation

```
1: #include <stdio.h>
2: #include <semaphore.h>
3: typedef long long int lli;
4:
5: sem_t s1;
6: lli ml;
7: double ppil;
8: int t1;
9: lli ra;
10:
11: void *pleibniz(void* i) {
12: int ii=(int)i;
13: double pi=0.0,a;
14: lli lb=ii*ra;
15: lli ub=lb+ra;
16: for(;lb<ub;lb++){
17: if(lb%2==0){
18: a=(2*lb)+1;
19: pi=pi+(1/a);
20: }
21: else{
22: a=(2*lb)+1;
23: pi=pi-(1/a);
24: }
25: }
26: sem_wait(&s1);
27: fflush(stdout);
28: ppil=ppil+pi;
29: sem_post(&s1);
30: }
```

Time.c File for time note of a instant

```
1: #include <stdio.h>
2: #include <sys/time.h>
3: #include <unistd.h>
4:
5: double Time(){
6: const double ms = 1.0e-6;
7: struct timeval seco;
8: int ss= gettimeofday(&seco, NULL);
9: if(ss==-1){
10: printf("error");
11: return 0;
12:
13: }
14: double ct = ((double)seco.tv_sec) + (((double)seco.tv_usec))*ms;
15: return ct;
16: }
```

**********END**********