

# PARALLEL AND DISTRIBUTED COMPUTING LAB

## REPORT

**NAME:** S Shyam Sundaram

**REG NO:** 19BCE1560

**PROGRAMMING ENVIRONMENT:** OpenMP

**PROBLEM:** Profiling

**DATE:** 27<sup>th</sup> September, 2021

### **HARDWARE CONFIGURATION:**

CPU NAME	:	Intel core i5 – 1035G1 @ 1.00 Ghz
Number of Sockets:	:	1
Cores per Socket	:	4
Threads per core	:	1
L1 Cache size	:	320KB
L2 Cache size	:	2MB
L3 Cache size (Shared):	:	6MB
RAM	:	8 GB

### **STATEMENT**

The performance of a piece of code is evaluated by looking at metrics such as time taken by each function to complete its task. This can give us better insights into which parts of the code can be parallelised in order to try making it more efficient. For this experiment, a sample C program is written with multiple functions that include: Matrix Addition, Matrix Multiplication, Matrix Subtraction, Prefix sum, checking if a matrix is Diagonal or not and checking if a matrix is symmetrical or not. Each of these are then subject to functional, line and hardware profiling. The same C program is used for functional and line profiling. For hardware, a parallelized code for matrix multiplication is used.

### **CODE**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define M 1000
#define N 1000
#define I 1000
#define V 10000000
```

```

int** initArray(int r,int c,int u,int l)
{
    srand(time(0));

    int** arr=(int**)malloc(r*sizeof(int*));
    for(int i=0;i<r;++i)
        arr[i]=(int*)malloc(c*sizeof(int));

    if(u!=l && l!=0 && u!=0)                //initializes each element to a random array if low
er (l) and uper (u) vales are different
    for(int i=0;i<r;++i)
        for(int j=0;j<c;++j)
            arr[i][j]=(rand()%(u-l+1))+l;

    if(l==0 && u==0)                        //initialize elements to 0 if l and u are 0
    for(int i=0;i<r;++i)
        for(int j=0;j<c;++j)
            arr[i][j]=0;

    return arr;
}

```

```

void freeArray(int** arr,int r)
{
    for(int i=0;i<r;++i)
        free(arr[i]);

    free(arr);
}

```

```

void MatAdd()
{
    //init array
    printf("Starting Matrix addition...\n");

    printf("Initialising arrays...\n");
    int** a=initArray(M,N,100,1);
    int** b=initArray(M,N,100,1);
    int** c=initArray(M,N,1,1);

    printf("Performing addition...\n");
    for(int i=0;i<M;++i)
        for(int j=0;j<N;++j)
            c[i][j]=a[i][j]+b[i][j];
}

```

```

printf("Finished addition...\n");
printf("Freeing Arrays...\n\n");
freeArray(a,M);
freeArray(b,M);
freeArray(c,M);
}

void MatMul()
{
    //init array
    printf("Starting Matrix Multiplication...\n");

    printf("Initialising arrays...\n");
    int** a=initArray(M,I,100,1);
    int** b=initArray(I,N,100,1);
    int** c=initArray(M,N,0,0);

    printf("Performing Multiplication...\n");
    for(int i=0;i<M;++i)
        for(int j=0;j<N;++j)
            for(int k=0;k<I;++k)
                c[i][j]+=a[i][k]*b[k][j];

    printf("Finished Multiplication...\n");
    printf("Freeing Arrays...\n\n");
    freeArray(a,M);
    freeArray(b,M);
    freeArray(c,M);
}

void MatSub()
{
    //init array
    printf("Starting Matrix Subtraction...\n");

    printf("Initialising arrays...\n");
    int** a=initArray(M,N,100,1);
    int** b=initArray(M,N,100,1);
    int** c=initArray(M,N,1,1);

    printf("Performing subtraction...\n");
    for(int i=0;i<M;++i)
        for(int j=0;j<N;++j)

```

```

        c[i][j]=a[i][j]-b[i][j];

printf("Finished subtraction...\n");
printf("Freeing Arrays...\n\n");
freeArray(a,M);
freeArray(b,M);
freeArray(c,M);
}

```

```

void isDiagonal()
{
    //init array
    printf("Starting isDiagonal...\n");
    int flag=1;
    printf("Initialising array...\n");
    int** a=initArray(N,N,100,1);

    printf("Examining...\n");
    for(int i=0;i<N;++i)
    {
        for(int j=0;j<N;++j)
            if(i!=j && a[i][j]!=0)
            {
                flag=1;
                break;
            }
        if(flag==1)
            break;
    }
    if(flag==1)
        printf("Not a diagonal matrix\n");
    else
        printf("Is a diagonal matrix\n");
    printf("Freeing Array...\n\n");
    freeArray(a,N);
}

```

```

void isSymmetric()
{
    //init array
    printf("Starting isSymmetric...\n");
    int flag=1;
    printf("Initialising array...\n");
    int** a=initArray(N,N,100,1);

```

```

printf("Examining...\n");
for(int i=0;i<N;++i)
{
    for(int j=0;j<N;++j)
        if(a[i][j]!=a[j][i])
        {
            flag=1;
            break;
        }
    if(flag==1)
        break;
}
if(flag==1)
printf("Not a symmetric matrix\n");
else
printf("Is a symmetric matrix\n");
printf("Freeing Array...\n\n");
freeArray(a,N);
}

void prefix()
{
    printf("Initialising arrays...\n");
    long *x=(long*)malloc(V*sizeof(long));
    for(int i=0;i<V;++i)
        x[i]=i;

    long *y=(long*)malloc(V*sizeof(long));

    y[0]=x[0];

    printf("Summing prefixes...\n");
    for(int i=1;i<V;++i)
        y[i]=y[i-1]+x[i];

    printf("Summation done. Freeing arrays...\n\n");
    free(x);
    free(y);
}

int main()
{
    MatMul();
}

```

```
MatAdd();
MatSub();
isSymmetric();
isDiagonal();
prefix();

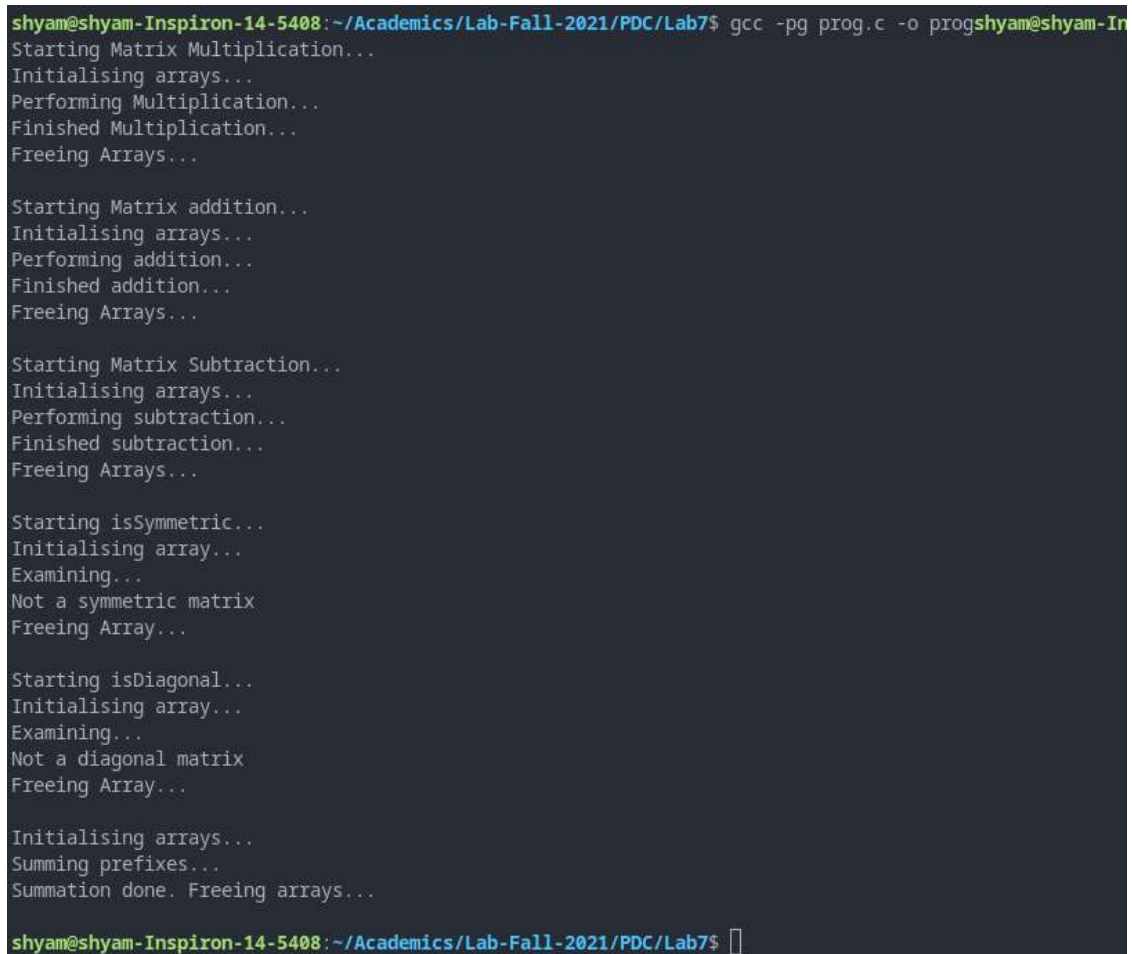
return 0;
}
```

## FUNCTIONAL PROFILING

### COMPILATION AND EXECUTION

```
gcc -pg prog.c -o prog
./prog
gprof prog
```

### SCREENSHOTS



```
shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ gcc -pg prog.c -o progshyam@shyam-In
Starting Matrix Multiplication...
Initialising arrays...
Performing Multiplication...
Finished Multiplication...
Freeing Arrays...

Starting Matrix addition...
Initialising arrays...
Performing addition...
Finished addition...
Freeing Arrays...

Starting Matrix Subtraction...
Initialising arrays...
Performing subtraction...
Finished subtraction...
Freeing Arrays...

Starting isSymmetric...
Initialising array...
Examining...
Not a symmetric matrix
Freeing Array...

Starting isDiagonal...
Initialising array...
Examining...
Not a diagonal matrix
Freeing Array...

Initialising arrays...
Summing prefixes...
Summation done. Freeing arrays...

shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ █
```

## Flat profile:

```
shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ gprof prog
Flat profile:
```

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
96.17	4.58	4.58	1	4.58	4.60	MatMul
2.32	4.69	0.11	1	0.11	0.11	prefix
1.69	4.77	0.08	11	0.01	0.01	initArray
0.21	4.78	0.01	1	0.01	0.03	MatAdd
0.00	4.78	0.00	11	0.00	0.00	freeArray
0.00	4.78	0.00	1	0.00	0.02	MatSub
0.00	4.78	0.00	1	0.00	0.01	isDiagonal
0.00	4.78	0.00	1	0.00	0.01	isSymmetric

% the percentage of the total running time of the program used by this function.

cumulative a running sum of the number of seconds accounted for by this function and those listed above it.

self the number of seconds accounted for by this function alone. This is the major sort for this listing.

calls the number of times this function was invoked, if this function is profiled, else blank.

self the average number of milliseconds spent in this

## Call Graph:

Call graph (explanation follows)

granularity: each sample hit covers 2 byte(s) for 0.21% of 4.78 seconds

index	% time	self	children	called	name
<spontaneous>					
[1]	100.0	0.00	4.78		main [1]
		4.58	0.02	1/1	MatMul [2]
		0.11	0.00	1/1	prefix [3]
		0.01	0.02	1/1	MatAdd [5]
		0.00	0.02	1/1	MatSub [6]
		0.00	0.01	1/1	isDiagonal [7]
		0.00	0.01	1/1	isSymmetric [8]
[2]	96.3	4.58	0.02	1/1	main [1]
		4.58	0.02	1	MatMul [2]
		0.02	0.00	3/11	initArray [4]
		0.00	0.00	3/11	freeArray [9]
[3]	2.3	0.11	0.00	1/1	main [1]
		0.11	0.00	1	prefix [3]
		0.01	0.00	1/11	isDiagonal [7]
		0.01	0.00	1/11	isSymmetric [8]
		0.02	0.00	3/11	MatAdd [5]
		0.02	0.00	3/11	MatMul [2]
		0.02	0.00	3/11	MatSub [6]
[4]	1.7	0.08	0.00	11	initArray [4]
[5]	0.7	0.01	0.02	1/1	main [1]
		0.01	0.02	1	MatAdd [5]
		0.02	0.00	3/11	initArray [4]
		0.00	0.00	3/11	freeArray [9]
[6]	0.5	0.00	0.02	1/1	main [1]
		0.00	0.02	1	MatSub [6]
		0.02	0.00	3/11	initArray [4]
		0.00	0.00	3/11	freeArray [9]
[7]	0.2	0.00	0.01	1/1	main [1]
		0.00	0.01	1	isDiagonal [7]
		0.01	0.00	1/11	initArray [4]

-----					
[5]	0.7	0.01	0.02	1/1	main [1]
		0.01	0.02	1	MatAdd [5]
		0.02	0.00	3/11	initArray [4]
		0.00	0.00	3/11	freeArray [9]
-----					
[6]	0.5	0.00	0.02	1/1	main [1]
		0.00	0.02	1	MatSub [6]
		0.02	0.00	3/11	initArray [4]
		0.00	0.00	3/11	freeArray [9]
-----					
[7]	0.2	0.00	0.01	1/1	main [1]
		0.00	0.01	1	isDiagonal [7]
		0.01	0.00	1/11	initArray [4]
		0.00	0.00	1/11	freeArray [9]
-----					
[8]	0.2	0.00	0.01	1/1	main [1]
		0.00	0.01	1	isSymmetric [8]
		0.01	0.00	1/11	initArray [4]
		0.00	0.00	1/11	freeArray [9]
-----					
[9]	0.0	0.00	0.00	1/11	isDiagonal [7]
		0.00	0.00	1/11	isSymmetric [8]
		0.00	0.00	3/11	MatAdd [5]
		0.00	0.00	3/11	MatMul [2]
		0.00	0.00	3/11	MatSub [6]
		0.00	0.00	11	freeArray [9]
		-----			

## **INFERENCES AND OBSERVATIONS**

From the profile generated, we see that matrix multiplication takes the most amount of the total execution time, 4.58 seconds, which forms about 96% of the total time. This is also observed in the 2<sup>nd</sup> entry of call graph. We also see that the functions called the most are initArray and freeArray, each called 11 times. From this, we see that we could speedup our program by improving upon matrix multiplication, which is parallelizable.



## LINE PROFILING

### COMPILATION AND EXECUTION

gcc -fprofile-arcs -ftest-coverage prog.c -o prog

./prog

gcov prog.c

### SCREENSHOTS

```
shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ gcc -fprofile-arcs -ftest-coverage prog.c -o prog
shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ ./prog
Starting Matrix Multiplication...
Initialising arrays...
Performing Multiplication...
Finished Multiplication...
Freeing Arrays...

Starting Matrix addition...
Initialising arrays...
Performing addition...
Finished addition...
Freeing Arrays...

Starting Matrix Subtraction...
Initialising arrays...
Performing subtraction...
Finished subtraction...
Freeing Arrays...

Starting isSymmetric...
Initialising array...
Examining...
Not a symmetric matrix
Freeing Array...

Starting isDiagonal...
Initialising array...
Examining...
Not a diagonal matrix
Freeing Array...

Initialising arrays...
Summing prefixes...
Summation done. Freeing arrays...

shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$
```

```
shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ gcov prog.c
File 'prog.c'
Lines executed:98.44% of 128
Creating 'prog.c.gcov'

shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ gcov -b -a prog.c
File 'prog.c'
Lines executed:98.44% of 128
Branches executed:100.00% of 62
Taken at least once:80.65% of 62
Calls executed:96.72% of 61
Creating 'prog.c.gcov'
```

```
shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ cat prog.c.gcov
--: 0:Source:prog.c
--: 0:Graph:prog.gcno
--: 0:Data:prog.gcda
--: 0:Runs:1
--: 1:#include <stdio.h>
--: 2:#include <stdlib.h>
--: 3:#include<time.h>
--: 4:
--: 5:#define M 1000
--: 6:#define N 1000
--: 7:#define I 1000
--: 8:#define V 10000000
--: 9:
function initArray called 11 returned 100% blocks executed 100%
11: 10:int** initArray(int r,int c,int u,int l)
--: 11:{
11: 12:    srand(time(0));
11: 12-block 0
call 0 returned 100%
call 1 returned 100%
--: 13:
11: 14:    int** arr=(int**)malloc(r*sizeof(int*));
11011: 15:    for(int i=0;i<r;++i)
11011: 15-block 0
branch 0 taken 100%
branch 1 taken 1% (fallthrough)
11000: 16:        arr[i]=(int*)malloc(c*sizeof(int));
11000: 16-block 0
--: 17:
--: 18:
11: 19:    if(u!=l && l!=0 && u!=0) //initializes each element to a random value
11: 19-block 0
branch 0 taken 73% (fallthrough)
branch 1 taken 27%
8: 19-block 1
branch 2 taken 100% (fallthrough)
branch 3 taken 0%
8: 19-block 2
branch 4 taken 100% (fallthrough)
branch 5 taken 0%
8008: 20:    for(int i=0;i<r;++i)
8: 20-block 0
8000: 20-block 1
8008: 20-block 2
```

```
branch 0 taken 100% (fallthrough)
branch 1 taken 0%
1: 158:    printf("Not a symmetric matrix\n");
1: 158-block 0
call 0 returned 100%
--: 159:    else
#####: 160:    printf("Is a symmetric matrix\n");
%%%%%%: 160-block 0
call 0 never executed
1: 161:    printf("Freeing Array...\n\n");
1: 161-block 0
call 0 returned 100%
1: 162:    freeArray(a,N);
call 0 returned 100%
1: 163:}
--: 164:
--: 165:
function prefix called 1 returned 100% blocks executed 100%
1: 166:void prefix()
--: 167:{
1: 168:    printf("Initialising arrays...\n");
1: 168-block 0
call 0 returned 100%
1: 169:    long *x=(long*)malloc(V*sizeof(long));
10000001: 170:    for(int i=0;i<V;++i)
10000001: 170-block 0
branch 0 taken 100%
branch 1 taken 1% (fallthrough)
10000000: 171:    x[i]=i;
10000000: 171-block 0
--: 172:
1: 173:    long *y=(long*)malloc(V*sizeof(long));
--: 174:
1: 175:    y[0]=x[0];
--: 176:
1: 177:    printf("Summing prefixes...\n");
1: 177-block 0
```

## **INFERENCES AND OBSERVATIONS**

Gcov helps us profile our code line by line. We see that 98.44% of 128 lines are executed. This is attributed to the fact the certain conditions were not met and those statements are hence not executed. We also see how many times each line was executed in the last two screenshots (the number prefixed at each line is the count).

## **HARDWARE PROFILING**

### **CODE (Parallelized matrix multiplication)**

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

#define M 2500
#define N 250
#define L 300

int main()
{
    int chunk = 10;

    printf("Name: Shyam Sundaram\nReg num: 19BCE1560\nPDC Lab:\n\n");

    float a[M*L],b[L*N],c[M*N];

    for(int i=0;i<M;++i)
    for(int j=0;j<L;++j)
    a[j+i*L]=10*j+i;

    for(int i=0;i<L;++i)
    for(int j=0;j<N;++j)
    b[j+i*N]=10*j+i;

    for(int i=0;i<M;++i)
    for(int j=0;j<N;++j)
    c[j+i*N]=0;

    omp_set_num_threads(32);
    float start=omp_get_wtime();

    int i,j,k;
    #pragma omp parallel private(i,j,k) shared(a,b) reduction(+:c)
    {
        #pragma omp for schedule(static,chunk) collapse(3)
        for(i=0;i<M;++i)
```

```

    {
        for(j=0;j<N;++j)
        {
            for(k=0;k<L;++k)
            {
                c[j+i*N]+=a[k+i*L]*b[j+k*N];
            }
        }
    }
}

float end=omp_get_wtime();
float exec=end-start;
printf("Thread count: %d Time taken is: %f\n",32,exec);

return 0;
}

```

### **COMPILATION AND EXECUTION**

gcc -fopenmp parallel.c -o parallel

./parallel

likwid-topology

likwid-pin -c 0,1,2,3 ./parallel

### **SCREENSHOTS**

```

shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ gcc -fopenmp parallel.c -o parallel
shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ ./parallel
Name: Shyam Sundaram
Reg num: 19BCE1560
PDC Lab:

```

```

Thread count: 32 Time taken is: 0.232422

```

```

shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ likwid-topology

```

```

-----
CPU name:      Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz
CPU type:      Intel Icelake processor
CPU stepping:  5

```

```

*****
Hardware Thread Topology
*****

```

```

Sockets:      1
Cores per socket: 4
Threads per core: 2

```

```

-----
HWThread      Thread      Core      Socket      Available
0              0              0          0            *
1              0              1          0            *
2              0              2          0            *
3              0              3          0            *
4              1              0          0            *
5              1              1          0            *
6              1              2          0            *
7              1              3          0            *

```

```

-----
Socket 0:      ( 0 4 1 5 2 6 3 7 )
-----
*****

```

```

-----
Socket 0:      ( 0 4 1 5 2 6 3 7 )
-----
*****
Cache Topology
*****
Level:         1
Size:          48 kB
Cache groups:  ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
-----
Level:         2
Size:          512 kB
Cache groups:  ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
-----
Level:         3
Size:          6 MB
Cache groups:  ( 0 4 1 5 2 6 3 7 )
-----
*****
NUMA Topology
*****
NUMA domains:  1
-----
Domain:        0
Processors:    ( 0 1 2 3 4 5 6 7 )
Distances:     10
Free memory:   1509.04 MB
Total memory:  7723.63 MB
-----

```

```

shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ likwid-pin -c 0,1,2,3 ./parallel
Sleeping longer as likwid_sleep() called without prior initialization
Name: Shyam Sundaram
Reg num: 19BCE1560
PDC Lab:

[pthread wrapper]
[pthread wrapper] MAIN -> 0
[pthread wrapper] PIN_MASK: 0->1 1->2 2->3
[pthread wrapper] SKIP_MASK: 0x0
  threadid 140014309209856 -> core 1 - OK
  threadid 140014300817152 -> core 2 - OK
  threadid 140014292424448 -> core 3 - OK
Roundrobin placement triggered
  threadid 140014284031744 -> core 0 - OK
  threadid 140014275639040 -> core 1 - OK
  threadid 140014267246336 -> core 2 - OK
  threadid 140014258853632 -> core 3 - OK
  threadid 140014250460928 -> core 0 - OK
  threadid 140014242068224 -> core 1 - OK
  threadid 140014233675520 -> core 2 - OK
  threadid 140014225282816 -> core 3 - OK
  threadid 140014216890112 -> core 0 - OK
  threadid 140014208497408 -> core 1 - OK
  threadid 140014200104704 -> core 2 - OK
  threadid 140014191712000 -> core 3 - OK
  threadid 140014183319296 -> core 0 - OK
  threadid 140014174926592 -> core 1 - OK
  threadid 140014166533888 -> core 2 - OK
  threadid 140014158141184 -> core 3 - OK
  threadid 140014149748480 -> core 0 - OK
  threadid 140014141355776 -> core 1 - OK
  threadid 140014132963072 -> core 2 - OK
  threadid 140014124570368 -> core 3 - OK
  threadid 140014116177664 -> core 0 - OK
  threadid 140014107784960 -> core 1 - OK
  threadid 140014099392256 -> core 2 - OK
  threadid 140014090999552 -> core 3 - OK
  threadid 140014082606848 -> core 0 - OK
  threadid 140014074214144 -> core 1 - OK
  threadid 140014065821440 -> core 2 - OK
  threadid 140014057428736 -> core 3 - OK
Thread count: 32 Time taken is: 0.964355

```

```
shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$ likwid-perfctr -C S0:1 -g BRANCH ./parallel
Cannot access directory /usr/share/likwid/perfgroups/ICL
-----
CPU name:      Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz
CPU type:      Intel Icelake processor
CPU clock:     1.19 GHz
ERROR - [./src/perfmon.c:perfmon_init_maps:1095] Unsupported Processor
ERROR - [./src/perfmon.c:perfmon_init_funcs:1526] Unsupported Processor
Cannot get access to MSRs. Please check permissions to the MSRs
shyam@shyam-Inspiron-14-5408:~/Academics/Lab-Fall-2021/PDC/Lab7$
```

## **INFERENCES AND OBSERVATIONS**

Likwid is a suite of command line applications used to estimate a program's performance. With 'likwid-topology', we are shown the thread, cache, NUMA and GPU topologies, properties and information. With likwid-pin, we pin each software thread to hardware and evaluate the code. This means we specify which threads to use (as seen in third screenshot command and output).

## **CONCLUSION**

We have used three different profiling techniques and tools to evaluate the performance of code written in C. Doing so has given us better understanding and identification of the parts of program which have room for improvement and reduce overall time taken to execute the program.