**PARALLEL AND DISTRIBUTED COMPUTING LAB**

**REPORT**

**NAME:** S Shyam Sundaram

**REG NO:** 19BCE1560

**PROGRAMMING ENVIRONMENT:** OpenMP

**PROBLEM:** Scheduling Algorithms with Prime number count and matrix multiplication

**DATE:** 1st 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 |

**PRIME NUMBER COUNT**

**CODE**

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

int sieve(int x)

{

for(int i=2;i\*i<=x;++i)

{

if(x%i==0)

return -1;

}

return 1;

}

int main()

{

int N[]={1000,10000,100000,1000000};

int chunk = 10;

int thread[]={1,2,4,8,16,32,64,128,256,512};

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

for(int i=0;i<4;++i)

{

printf("-------\nN: %d\n",N[i]);

for(int t=0;t<10;++t)

{

omp\_set\_num\_threads(thread[t]);

float start=omp\_get\_wtime();

int cnt=0;

int n=N[i];

#pragma omp parallel for schedule(dynamic,chunk) reduction(+:cnt)

for(int j=2;j<n;++j)

{

if(sieve(j)==1)

cnt+=1;

}

float end=omp\_get\_wtime();

float exec=end-start;

printf("Count: %d Thread count: %d Time taken is: %f\n",cnt,thread[t],exec);

}

}

return 0;

}

**NOTE:** For Static, replace schedule clause (in orange) argument from ‘dynamic’ to ‘static’. For default, remove schedule clause.

**COMPILATION AND EXECUTION**

gcc -fopenmp prime.c

./a.out

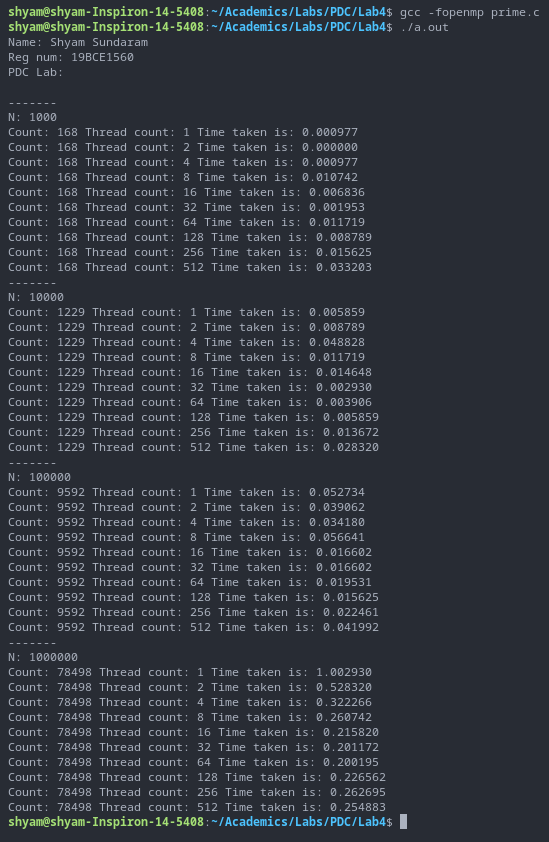
**OBSERVATIONS**

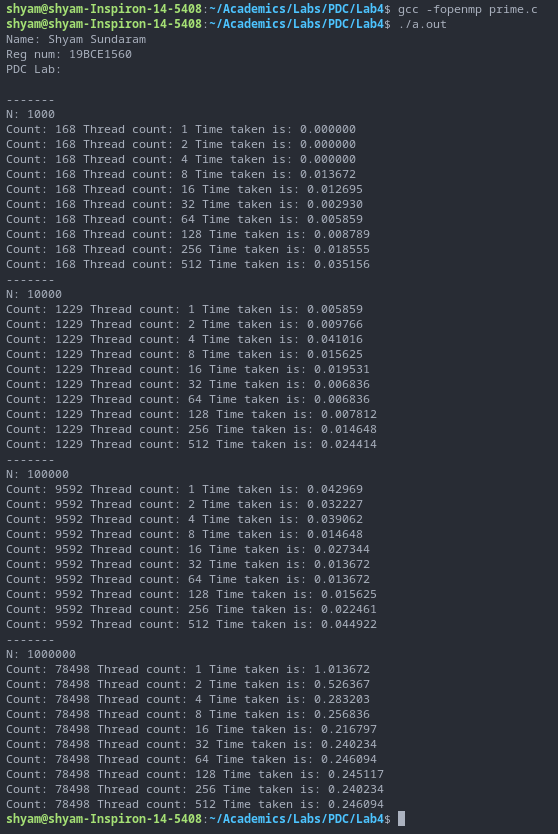
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **NUMBER OF THREADS** | **DEFAULT EXECUTION TIME** | **STATIC EXECUTION TIME** | **DYNAMIC EXECUTION TIME** |
| 10000 | 1 | 0.005859 | 0.005859 | 0.005859 |
| 2 | 0.008789 | 0.019531 | 0.009766 |
| 4 | 0.048828 | 0.057617 | 0.041016 |
| 8 | 0.011719 | 0.058594 | 0.015625 |
| 16 | 0.014648 | 0.026367 | 0.019531 |
| 32 | 0.002930 | 0.003906 | 0.006836 |
| 64 | 0.003906 | 0.005859 | 0.006836 |
| 128 | 0.005859 | 0.011719 | 0.007812 |
| 256 | 0.013672 | 0.017578 | 0.014648 |
| 512 | 0.028320 | 0.028320 | 0.024414 |
| 100000 | 1 | 0.052734 | 0.043945 | 0.042969 |
| 2 | 0.039062 | 0.030273 | 0.032227 |
| 4 | 0.034180 | 0.027344 | 0.039062 |
| 8 | 0.056641 | 0.048828 | 0.014648 |
| 16 | 0.016602 | 0.048828 | 0.027344 |
| 32 | 0.016602 | 0.012695 | 0.013672 |
| 64 | 0.019531 | 0.012695 | 0.013672 |
| 128 | 0.015625 | 0.023438 | 0.015625 |
| 256 | 0.022461 | 0.033203 | 0.022461 |
| 512 | 0.041992 | 0.043945 | 0.044922 |
| 1000000 | 1 | 1.002930 | 0.834961 | 1.013672 |
| 2 | 0.528320 | 0.399414 | 0.526367 |
| 4 | 0.322266 | 0.293945 | 0.283203 |
| 8 | 0.260742 | 0.258789 | 0.256836 |
| 16 | 0.215820 | 0.232422 | 0.216797 |
| 32 | 0.201172 | 0.227539 | 0.240234 |
| 64 | 0.200195 | 0.211914 | 0.246094 |
| 128 | 0.226562 | 0.221680 | 0.245117 |
| 256 | 0.262695 | 0.242188 | 0.240234 |
| 512 | 0.254883 | 0.254883 | 0.246094 |

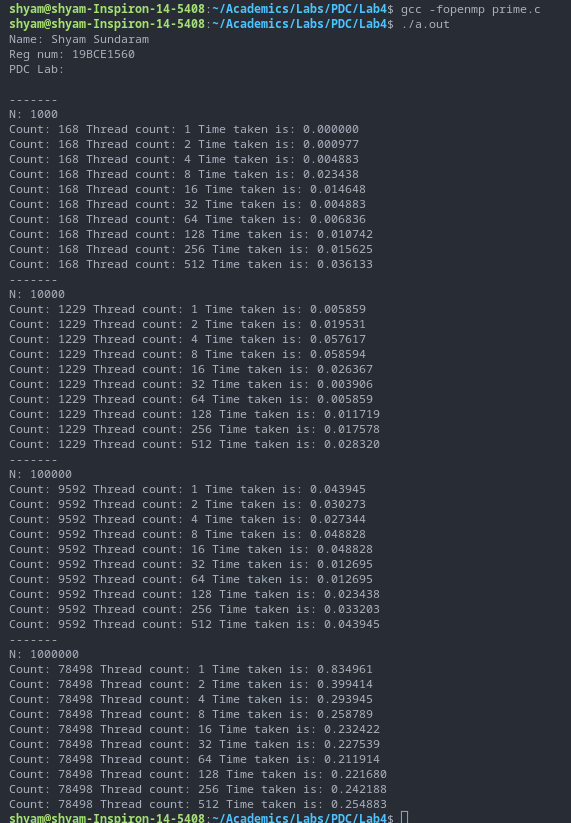
**ASSUMPTION**

As the number of threads increase, the work done by each thread is reduced, thus we see an overall decline in the execution time for all three types of scheduling.

**SCREENSHOTS**



****

****

**PLOTS**

**INFERENCE**

As more threads are allocated, the workload is distributed according to the respective scheduling algorithms, thus the overall execution time decreases.

**MATRIX MULTIPLICATION**

**CODE**

#include <stdio.h>

#include<stdlib.h>

#include<omp.h>

#define R 2500

#define C 250

int main()

{

int chunk = 10;

int thread[]={1,2,4,8,16,32,64,128,256,512};

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

float a[R][C], b[C][C], c[R][C];

for(int i=0;i<R;++i)

for(int j=0;j<C;++j)

a[i][j]=10\*j+i;

for(int i=0;i<C;++i)

for(int j=0;j<C;++j)

b[i][j]=10\*i+j;

for(int i=0;i<R;++i)

for(int j=0;j<C;++j)

c[i][j]=0;

for(int t=0;t<10;++t)

{

omp\_set\_num\_threads(thread[t]);

float start=omp\_get\_wtime();

int chunk=10;

int i,j,k;

#pragma omp parallel private(i,j,k) shared(a,b) reduction(+:c)

{

#pragma omp for collapse(3) schedule(static,chunk)

for(i=0;i<R;++i)

for(j=0;j<C;++j)

for(k=0;k<C;++k)

c[i][j]+=a[i][k]\*b[k][j];

}

float end=omp\_get\_wtime();

float exec=end-start;

printf("Thread count: %d Time taken is: %f\n",thread[t],exec);

}

return 0;

}

**NOTE:** For Static, replace schedule clause (in orange) argument from ‘dynamic’ to ‘static’. For default, remove schedule clause.

**COMPILATION AND EXECUTION**

gcc -fopenmp matmul.c

./a.out

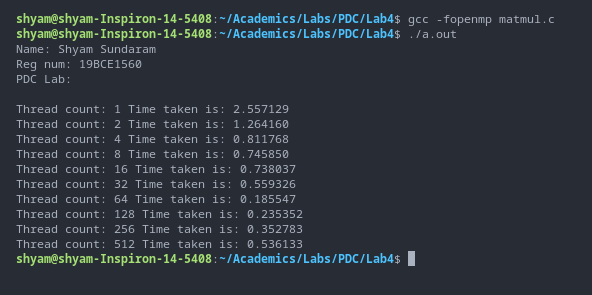
**OBSERVATIONS**

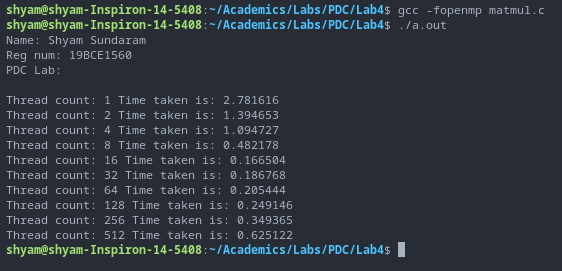
|  |  |  |  |
| --- | --- | --- | --- |
| **NUMBER OF THREADS** | **DEFAULT EXECUTION TIME** | **STATIC EXECUTION TIME** | **DYNAMIC EXECUTION TIME** |
| 1 | 2.557129 | 2.781616 | 0.828857 |
| 2 | 1.264160 | 1.394653 | 0.653564 |
| 4 | 0.811768 | 1.094727 | 0.472900 |
| 8 | 0.745850 | 0.482178 | 0.395264 |
| 16 | 0.738037 | 0.166504 | 0.406250 |
| 32 | 0.559326 | 0.186768 | 0.499268 |
| 64 | 0.185547 | 0.205444 | 0.432373 |
| 128 | 0.235352 | 0.249146 | 0.485107 |
| 256 | 0.352783 | 0.349365 | 0.540283 |
| 512 | 0.536133 | 0.625122 | 0.655273 |

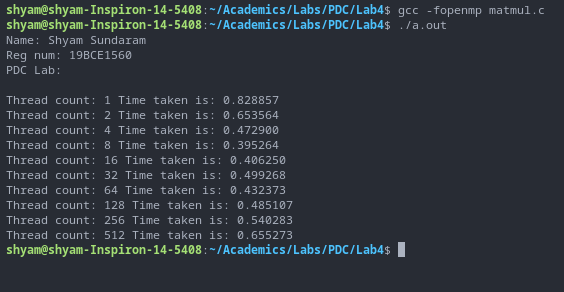
**ASSUMPTION**

As the number of threads increase, the work done by each thread is reduced, thus we see an overall decline in the execution time for all three types of scheduling.

**SCREENSHOTS**







**PLOTS**

**INFERENCE**

As more threads are allocated, the workload is distributed according to the respective scheduling algorithms, thus the overall execution time decreases.