**PARALLEL AND DISTRIBUTED COMPUTING LAB**

**REPORT**

**NAME:** S Shyam Sundaram

**REG NO:** 19BCE1560

**PROGRAMMING ENVIRONMENT:** OpenMP

**PROBLEM: S**ummation of numbers with and without reduction and row major matrix multiplication

**DATE:** 22nd 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 |

**SUM OF N NUMBERS WITHOUT REDUCTION**

**CODE**

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#define N 100000000

int main()

{

    int chunk =20;

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

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

    printf("Size of array: %d------------\n",s);

    for(int t=0;t<10;++t)

    {

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

        long \*sum=(long\*)malloc(thread[t]\*sizeof(long));

        for(int i=0;i<thread[t];++i)

        sum[i]=0;

        long s=0;

        int i;

        float start=omp\_get\_wtime();

        #pragma omp parallel for schedule(static,chunk) shared(sum) private(i)

        for(i=0;i<N;++i)

        sum[omp\_get\_thread\_num()]+=i;

        for(int i=0;i<thread[t];++i)

        s+=sum[i];

        float end=omp\_get\_wtime();

        float exec=end-start;

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

        free(sum);

    }

    return 0;

}

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

**COMPILATION AND EXECUTION**

gcc -fopenmp sum.c

./a.out

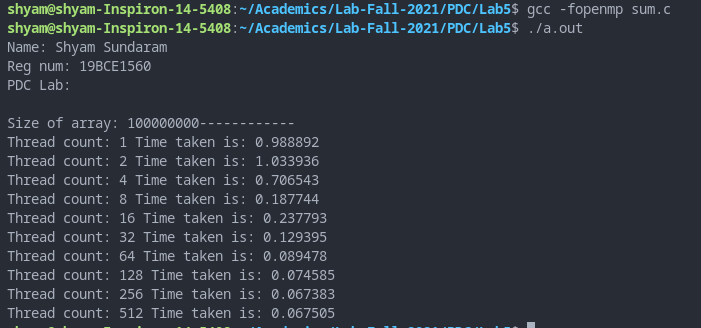
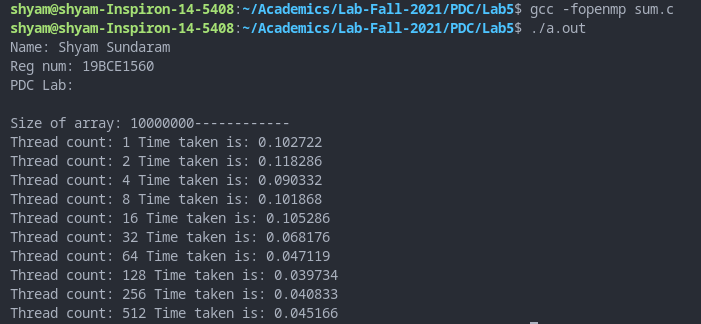
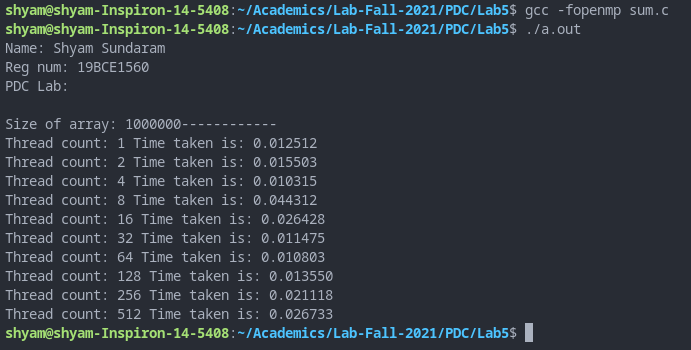
**OBSERVATIONS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **NUMBER OF THREADS** | **DEFAULT EXECUTION TIME** | **STATIC EXECUTION TIME** | **DYNAMIC EXECUTION TIME** |
| 100000 | 1 | 0.014465 | 0.014801 | 0.012512 |
| 2 | 0.014648 | 0.017029 | 0.015503 |
| 4 | 0.008362 | 0.011505 | 0.010315 |
| 8 | 0.018433 | 0.023468 | 0.044312 |
| 16 | 0.018311 | 0.017303 | 0.026428 |
| 32 | 0.008423 | 0.009186 | 0.011475 |
| 64 | 0.008301 | 0.007599 | 0.010803 |
| 128 | 0.009583 | 0.010040 | 0.013550 |
| 256 | 0.014771 | 0.015625 | 0.021118 |
| 512 | 0.022095 | 0.029694 | 0.026733 |
| 10000000 | 1 | 0.108337 | 0.102722 | 0.127991 |
| 2 | 0.130615 | 0.118286 | 0.195923 |
| 4 | 0.107605 | 0.090332 | 0.129211 |
| 8 | 0.101440 | 0.101868 | 0.132507 |
| 16 | 0.116272 | 0.105286 | 0.152222 |
| 32 | 0.066345 | 0.068176 | 0.093811 |
| 64 | 0.043274 | 0.047119 | 0.064331 |
| 128 | 0.039856 | 0.039734 | 0.061279 |
| 256 | 0.040710 | 0.040833 | 0.088501 |
| 512 | 0.035828 | 0.045166 | 0.070068 |
| 100000000 | 1 | 1.059448 | 0.988892 | 1.229126 |
| 2 | 1.137085 | 1.033936 | 2.045776 |
| 4 | 1.004272 | 0.706543 | 1.280640 |
| 8 | 0.944580 | 0.187744 | 1.284546 |
| 16 | 0.237305 | 0.237793 | 0.493774 |
| 32 | 0.121826 | 0.129395 | 0.189331 |
| 64 | 0.084229 | 0.089478 | 0.115845 |
| 128 | 0.076050 | 0.074585 | 0.105957 |
| 256 | 0.069214 | 0.067383 | 0.149048 |
| 512 | 0.064575 | 0.067505 | 0.103149 |

**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 (up to a point in some cases).

**SCREENSHOTS**

****

**PLOT**

**INFERENCE**

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

**SUM OF N NUMBERS WITH REDUCTION**

**CODE**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define N 10000000

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");

    int s=N;

    printf("Size of array: %d------------\n",s);

    for(int t=0;t<10;++t)

    {

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

        int sum=0;

        int i;

        float start=omp\_get\_wtime();

        #pragma omp parallel for schedule(dynamic,chunk) private(i) reduction(+:sum)

        for(i=0;i<N;++i)

            sum+=i;

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

**COMPILATION AND EXECUTION**

gcc -fopenmp sumred.c

./a.out

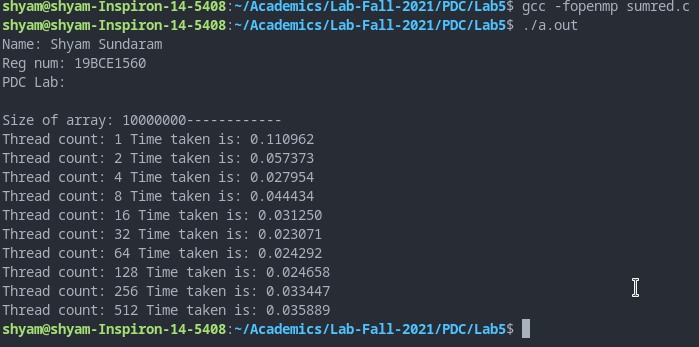
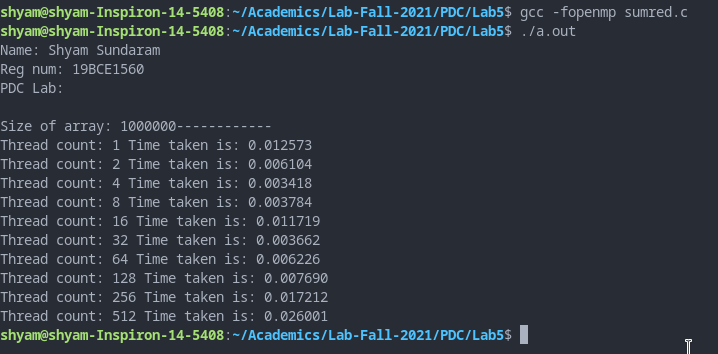
**OBSERVATIONS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **NUMBER OF THREADS** | **DEFAULT EXECUTION TIME** | **STATIC EXECUTION TIME** | **DYNAMIC EXECUTION TIME** |
| 1000000 | 1 | 0.006958 | 0.011108 | 0.014526 |
| 2 | 0.003418 | 0.005615 | 0.016113 |
| 4 | 0.001831 | 0.002930 | 0.011475 |
| 8 | 0.010742 | 0.014282 | 0.046631 |
| 16 | 0.010254 | 0.004150 | 0.016479 |
| 32 | 0.001099 | 0.003906 | 0.018188 |
| 64 | 0.001221 | 0.004761 | 0.012451 |
| 128 | 0.002197 | 0.008057 | 0.018555 |
| 256 | 0.003052 | 0.013672 | 0.021362 |
| 512 | 0.005615 | 0.024658 | 0.033447 |
| 10000000 | 1 | 0.120605 | 0.110962 | 0.140991 |
| 2 | 0.062622 | 0.057373 | 0.153809 |
| 4 | 0.031738 | 0.027954 | 0.120239 |
| 8 | 0.046631 | 0.044434 | 0.092896 |
| 16 | 0.039551 | 0.031250 | 0.094482 |
| 32 | 0.025024 | 0.023071 | 0.147339 |
| 64 | 0.025024 | 0.024292 | 0.095825 |
| 128 | 0.024780 | 0.024658 | 0.098999 |
| 256 | 0.029907 | 0.033447 | 0.109863 |
| 512 | 0.037354 | 0.035889 | 0.118896 |

**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 (up to a point in some cases).

**SCREENSHOTS**

****

**PLOT**

**INFERENCE**

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

**ROW MAJOR MATRIX MULTIPLICATION**

**CODE**

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#define M 2500

#define N 250

#define L 300

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[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;

    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 schedule(dynamic,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",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

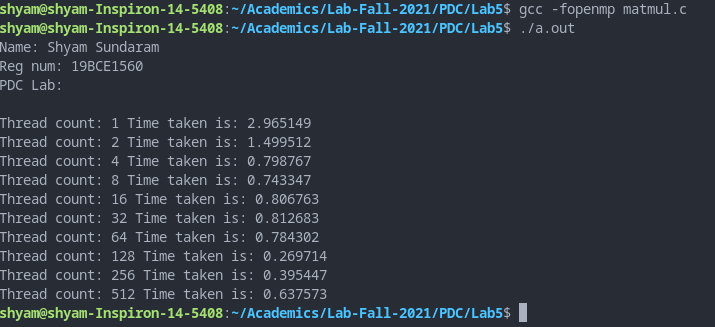
./a.out

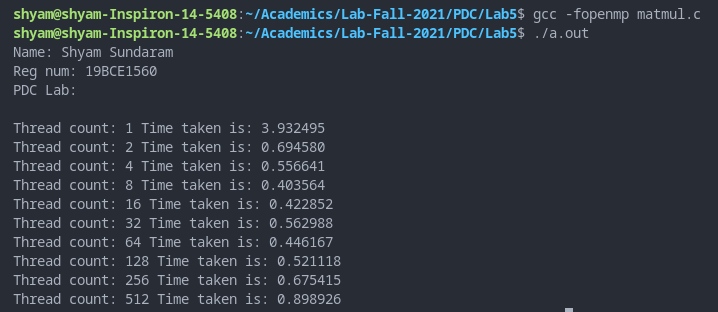
**OBSERVATIONS**

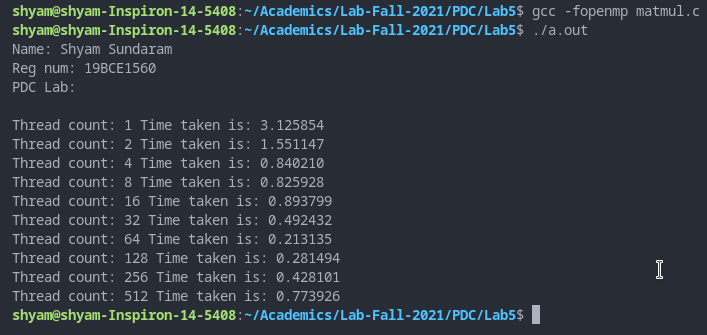
|  |  |  |  |
| --- | --- | --- | --- |
| **NUMBER OF THREADS** | **DEFAULT EXECUTION TIME** | **STATIC EXECUTION TIME** | **DYNAMIC EXECUTION TIME** |
| 1 | 2.965149 | 3.125854 | 3.932495 |
| 2 | 1.499512 | 1.551147 | 0.694580 |
| 4 | 0.798767 | 0.840210 | 0.556641 |
| 8 | 0.743347 | 0.825928 | 0.403564 |
| 16 | 0.806763 | 0.893799 | 0.422852 |
| 32 | 0.812683 | 0.492432 | 0.562988 |
| 64 | 0.784302 | 0.213135 | 0.446167 |
| 128 | 0.269714 | 0.281494 | 0.521118 |
| 256 | 0.395447 | 0.428101 | 0.675415 |
| 512 | 0.637573 | 0.773926 | 0.898926 |

**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.