**PARALLEL AND DISTRIBUTED COMPUTING LAB**

**REPORT**

**NAME:** S Shyam Sundaram

**REG NO:** 19BCE1560

**PROGRAMMING ENVIRONMENT:** OpenMP

**PROBLEM:** Finding the prefix sum and calculating pi (with and without reduction) with guided scheduling

**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 |

**PREFIX SUM**

**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};

    long \*x=(long\*)malloc(N\*sizeof(long));

    for(int i=0;i<N;++i)

    {

        x[i]=i;

    }

    for(int t=0;t<10;++t)

    {

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

        long \*y=(long\*)malloc(N\*sizeof(long));

        long \*z=(long\*)malloc(N\*sizeof(long));

        #pragma omp parallel for

        for(int i=0;i<N;++i)

        y[i]=x[i];

        float start=omp\_get\_wtime();

        int d=1;

        while(d<N)

        {

            int i;

            #pragma omp parallel for

            for(i=d;i<N;++i)

            z[i]=y[i-d];

            #pragma omp parallel for

            for(i=d;i<N;++i)

            y[i]+=z[i];

            d\*=2;

        }

        float end=omp\_get\_wtime();

        float exec=end-start;

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

        // for(int i=0;i<N;++i)

        // printf("%ld ",y[i]);

        free(y);

        free(z);

    }

    free(x);

}

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

**COMPILATION AND EXECUTION**

gcc -fopenmp prefixsum2.c

./a.out

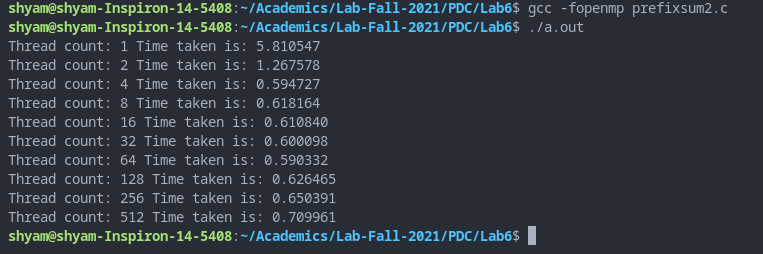
**OBSERVATIONS**

|  |  |  |
| --- | --- | --- |
| **N** | **NUMBER OF THREADS** | **GUIDED SCHEDULING EXECUTION TIME** |
| 10000000 | 1 | 5.810547 |
| 2 | 1.267578 |
| 4 | 0.594727 |
| 8 | 0.618164 |
| 16 | 0.610840 |
| 32 | 0.600098 |
| 64 | 0.590332 |
| 128 | 0.626465 |
| 256 | 0.650391 |
| 512 | 0.709961 |
| 100000000 | 1 | 43.244629 |
| 2 | 13.688965 |
| 4 | 7.655762 |
| 8 | 7.410645 |
| 16 | 7.040527 |
| 32 | 6.863281 |
| 64 | 6.920410 |
| 128 | 7.220703 |
| 256 | 7.261719 |
| 512 | 7.357910 |

**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). Guided scheduling behaves similarly to dynamic scheduling but seems to handle load imbalance better.

**SCREENSHOTS**

****

**PLOT**

**INFERENCE**

As more threads are allocated, the workload is distributed according to the respective scheduling algorithms, thus the overall execution time decreases. Guided scheduling also handles load imbalance better by starting with a larger chunk size and decreasing it as time goes on.

**CALCULATING PI WITH CRITICAL CONSTRUCT**

**CODE**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include <math.h>

#define terms 10000000

int main()

{

    int chunk = 10;

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

    for(int t=0;t<10;++t)

    {

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

        int i=3;

        double sum=4;

        double ps;

        float start=omp\_get\_wtime();

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

        for(i=3;i<2\*terms;i+=2)

        {

            ps=((pow(-1,i/2)\*4)/i);

            #pragma omp critical

            sum=sum+ps;

        }

        printf("sum= %f ",sum);

        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 picritic.c

./a.out

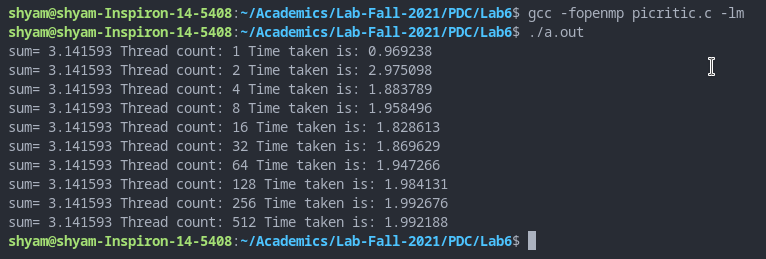
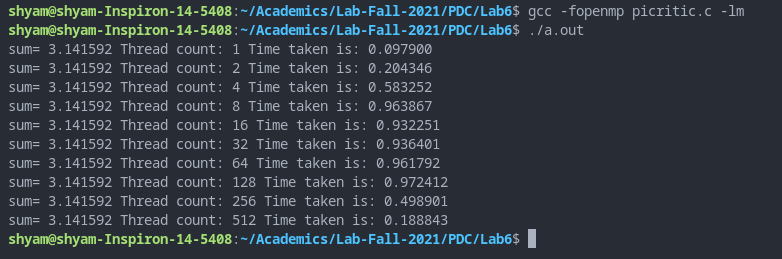
**OBSERVATIONS**

|  |  |  |
| --- | --- | --- |
| **N** | **NUMBER OF THREADS** | **GUIDED SCHEDULING EXECUTION TIME** |
| 1000000 | 1 | 0.097900 |
| 2 | 0.204346 |
| 4 | 0.583252 |
| 8 | 0.963867 |
| 16 | 0.932251 |
| 32 | 0.936401 |
| 64 | 0.961792 |
| 128 | 0.972412 |
| 256 | 0.498901 |
| 512 | 0.188843 |
| 10000000 | 1 | 0.969238 |
| 2 | 2.975098 |
| 4 | 1.883789 |
| 8 | 1.958496 |
| 16 | 1.828613 |
| 32 | 1.869629 |
| 64 | 1.947266 |
| 128 | 1.984131 |
| 256 | 1.992676 |
| 512 | 1.992188 |

**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). Guided scheduling behaves similarly to dynamic scheduling but seems to handle load imbalance better.

**SCREENSHOTS**

****

**PLOT**

**INFERENCE**

As more threads are allocated, the workload is distributed according to the respective scheduling algorithms, thus the overall execution time decreases. Guided scheduling also handles load imbalance better by starting with a larger chunk size and decreasing it as time goes on. The critical construct ensures that only one thread updates sum at a time.

**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

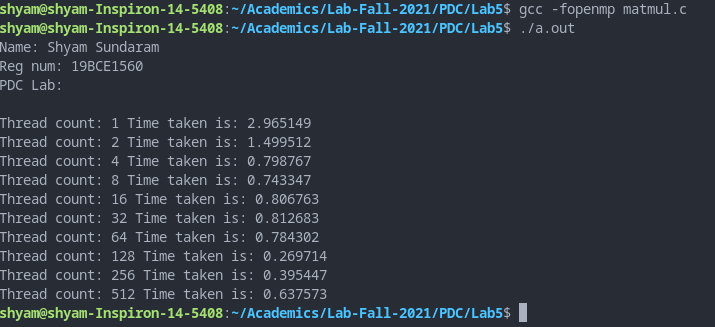
./a.out

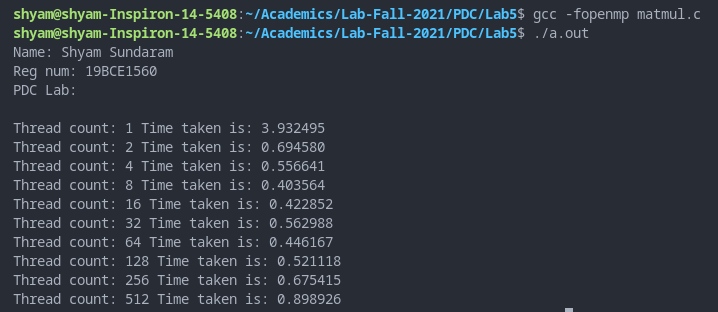
**OBSERVATIONS**

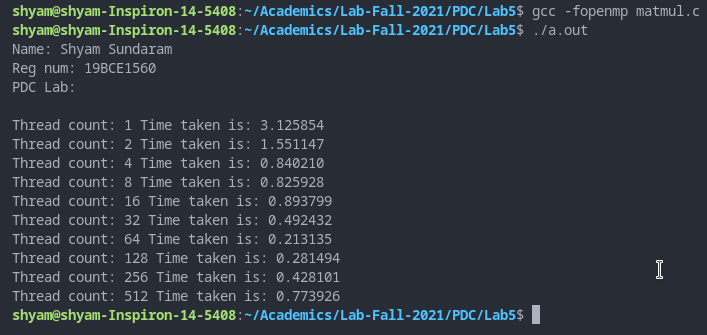
|  |  |  |  |
| --- | --- | --- | --- |
| **NUMBER OF THREADS** | **DEFAULT EXECUTION TIME** | **STATIC EXECUTION TIME** | **DYNAMIC EXECUTION TIME** |
| 1 | 2.965149 | 3.125854 | 3.932495 |
| 2 | 1.499512 | 1.551147 | 0.694580 |
| 4 | 0.798767 | 0.840210 | 0.556641 |
| 8 | 0.743347 | 0.825928 | 0.403564 |
| 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.