**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 critical and single constructs) 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 schedule(guided)

            for(i=d;i<N;++i)

            z[i]=y[i-d];

            #pragma omp parallel for schedule(guided)

            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);

}

**COMPILATION AND EXECUTION**

gcc -fopenmp prefixsum2.c

./a.out

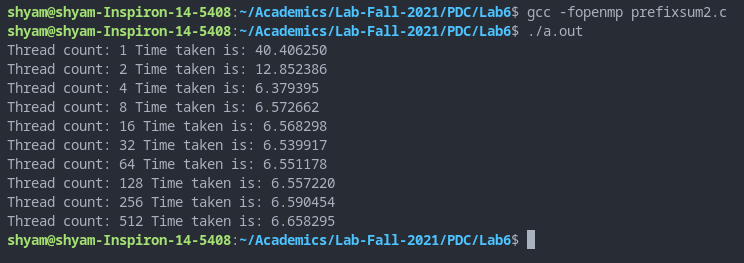
**OBSERVATIONS**

|  |  |  |
| --- | --- | --- |
| **N** | **NUMBER OF THREADS** | **GUIDED SCHEDULING EXECUTION TIME** |
| 10000000 | 1 | 5.608856 |
| 2 | 0.855698 |
| 4 | 0.704330 |
| 8 | 0.584625 |
| 16 | 0.588684 |
| 32 | 0.586212 |
| 64 | 0.590820 |
| 128 | 0.589951 |
| 256 | 0.622177 |
| 512 | 0.663330 |
| 100000000 | 1 | 40.406250 |
| 2 | 12.852386 |
| 4 | 6.379395 |
| 8 | 6.572662 |
| 16 | 6.568298 |
| 32 | 6.539917 |
| 64 | 6.551178 |
| 128 | 6.557220 |
| 256 | 6.590454 |
| 512 | 6.658295 |

**ASSUMPTION**

As the number of threads increase, the work done by each thread is reduced, thus we see an overall decline in the execution (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, 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;

}

**COMPILATION AND EXECUTION**

gcc -fopenmp picritic.c -lm

./a.out

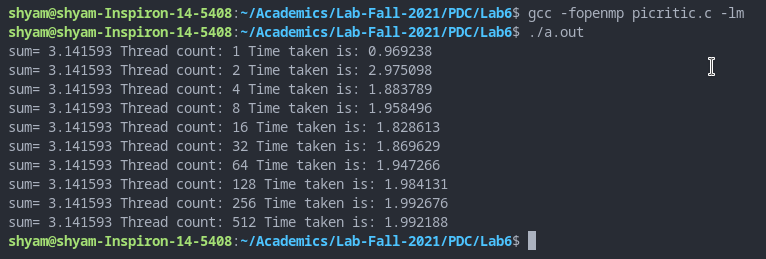
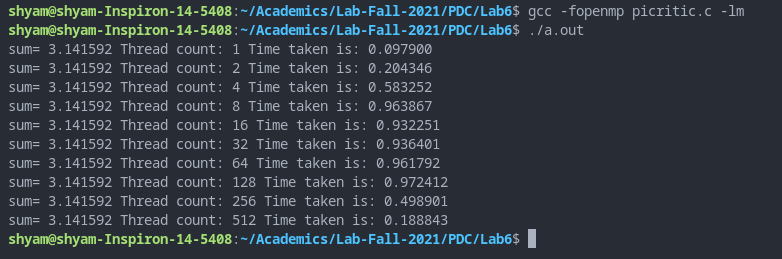
**OBSERVATIONS**

|  |  |  |
| --- | --- | --- |
| **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 (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, 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.

**CALCULATING PI WITH SINGLE 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=malloc(sizeof(double)\*thread[t]);

        for(int k=0;k<thread[t];++k)

        sum[k]=0;

        double pi=4;

        double ps;

        float start=omp\_get\_wtime();

        #pragma omp parallel private(i,ps) shared(sum)

        {

            #pragma omp for schedule(guided,chunk)

            for(i=3;i<2\*terms;i+=2)

            {

                ps=((pow(-1,i/2)\*4)/i);

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

            }

            #pragma omp single

            {

                for(int k=0;k<thread[t];++k)

                pi+=sum[k];

            }

        }

        printf("sum= %f ",pi);

        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;

}

**COMPILATION AND EXECUTION**

gcc -fopenmp pisingle.c -lm

./a.out

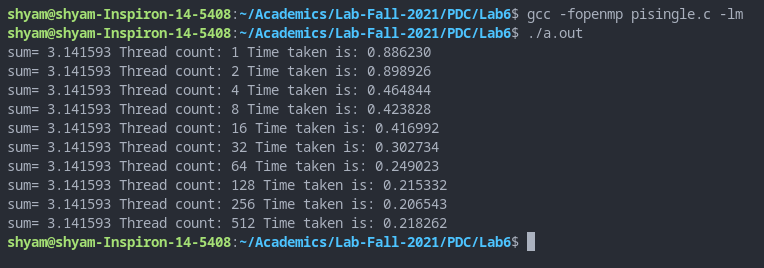
**OBSERVATIONS**

|  |  |  |
| --- | --- | --- |
| **N** | **NUMBER OF THREADS** | **GUIDED SCHEDULING EXECUTION TIME** |
| 1000000 | 1 | 0.090088 |
| 2 | 0.088135 |
| 4 | 0.052734 |
| 8 | 0.046387 |
| 16 | 0.060059 |
| 32 | 0.033691 |
| 64 | 0.034180 |
| 128 | 0.034912 |
| 256 | 0.039795 |
| 512 | 0.048340 |
| 10000000 | 1 | 0.886230 |
| 2 | 0.898926 |
| 4 | 0.464844 |
| 8 | 0.423828 |
| 16 | 0.416992 |
| 32 | 0.302734 |
| 64 | 0.249023 |
| 128 | 0.215332 |
| 256 | 0.206543 |
| 512 | 0.218262 |

**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.

**SCREENSHOTS**



**PLOTS**

**INFERENCE**

As more threads are allocated, the workload is distributed, thus the overall execution time decreases. Guided scheduling behaves similarly to dynamic scheduling but seems to handle load imbalance better.