**Class:** Final Year (Computer Science and Engineering)

**Year:** 2023-24 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 3**

**Exam Seat No:**

**Title of practical:**

Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

**Problem Statement 1:**

Analyse and implement a Parallel code for below program using OpenMP.

// C Program to find the minimum scalar product of two vectors (dot product)

#include<stdio.h>

#include<time.h>

#define n 100000

Void sort (int nums[])

{

    Int i, j;

    for (i = 0; i<n - 1; i++)

        for (j = 0; j< n - i - 1; j++)

            if (nums[j] >nums[j + 1])

            {

                inttemp = nums[j];

                nums[j] = nums[j + 1];

                nums[j + 1] = temp;

            }

}

voidsortDesc(intnums[])

{

    inti, j;

    for (i = 0; i<n; i++)

    {

        for (j = i + 1; j<n; j++)

        {

            if (nums[i] <nums[j])

            {

                inta = nums[i];

                nums[i] = nums[j];

                nums[j] = a;

            }

        }

    }

}

Int main()

{

    Int nums1[n], nums2[n];

    for (inti = 0; i<n; i++)

    {

        nums1[i] = 10;

    }

    for (inti = 0; i<n; i++)

    {

        nums2[i] = 20;

    }

    clock\_tt = clock();

    sort(nums1);

    sortDesc(nums2);

    t = clock() - t;

    doubletime = ((double)t) / CLOCKS\_PER\_SEC;

    printf("Time taken (seq): %f\n", time);

    intsum = 0;

    for (inti = 0; i<n; i++)

    {

        sum = sum + (nums1[i] \* nums2[i]);

    }

    printf("%d\n", sum);

    return0;

}

#include<omp.h>

#include<stdio.h>

#include<time.h>

#define n 100000

voidsort(intnums[])

{

    inti, j;

    for (i = 0; i<n; i++)

    {

        intturn = i % 2;

#pragmaompparallelfor

        for (j = turn; j<n - 1; j += 2)

            if (nums[j] >nums[j + 1])

            {

                inttemp = nums[j];

                nums[j] = nums[j + 1];

                nums[j + 1] = temp;

            }

    }

}

voidsort\_des(intnums[])

{

    inti, j;

    for (i = 0; i<n; ++i)

    {

        intturn = i % 2;

#pragmaompparallelfor

        for (j = turn; j<n - 1; j += 2)

        {

            if (nums[j] <nums[j + 1])

            {

                inttemp = nums[j];

                nums[j] = nums[j + 1];

                nums[j + 1] = temp;

            }

        }

    }

}

intmain()

{

    intnums1[n], nums2[n];

    // int i;

    for (inti = 0; i<n; i++)

    {

        nums1[i] = 10;

    }

    for (inti = 0; i<n; i++)

    {

        nums2[i] = 20;

    }

    clock\_tt;

    t = clock();

    sort(nums1);

    sort\_des(nums2);

    t = clock() - t;

    doubletime\_taken = ((double)t) / CLOCKS\_PER\_SEC;

    printf("Time taken (seq): %f\n", time\_taken);

    intsum = 0;

    for (inti = 0; i<n; i++)

    {

        sum = sum + (nums1[i] \* nums2[i]);

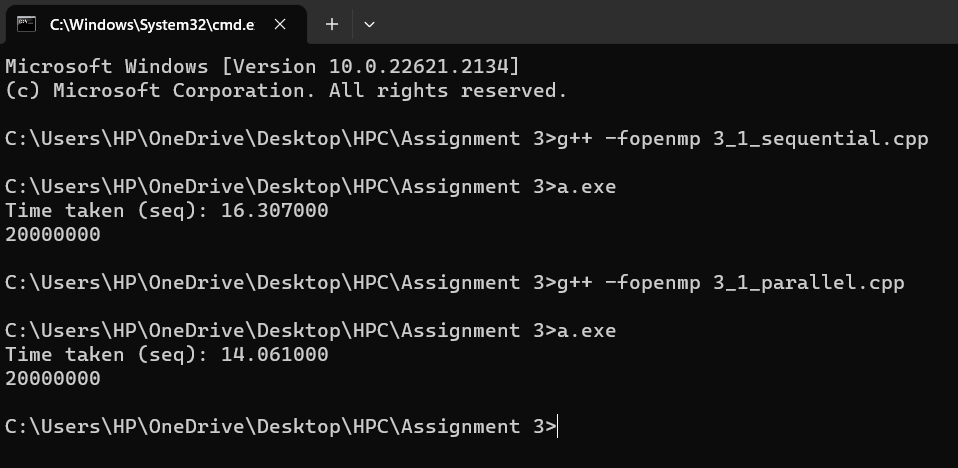
    }

    printf("%d\n", sum);

    return0;

}

**Screenshots:**

****

**Information and analysis:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Threads | Data Size | Sequential Time | Parallel Time | Speedup(Ts/Tp) |
| **8** | **10** | **0.00300** | **0.00400** | **0.75** |
| **8** | **500** | **0.20500** | **0.07600** | **2.69736834** |
| **8** | **1000** | **0.15200** | **0.15100** | **1.006622517** |
| **10** | **10** | **0.30000** | **0.00500** | **0.6** |
| **10** | **500** | **0.25000** | **0.07100** | **2.887323944** |
| **10** | **1000** | **0.15200** | **0.22500** | **0.675555556** |

**Problem Statement 2:**

Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)

i. For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads.

ii. Explain whether or not the scaling behaviour is as expected.

#include<omp.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define N 1000

voidadd(int\*\*a, int\*\*b, int\*\*c)

{

    for (inti = 0; i<N; i++)

    {

        for (intj = 0; j<N; j++)

        {

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

        }

    }

}

voidinput(int\*\*a, intnum)

{

    for (inti = 0; i<N; i++)

    {

        for (intj = 0; j<N; j++)

        {

            a[i][j] = num;

        }

    }

}

voiddisplay(int\*\*a)

{

    for (inti = 0; i<N; i++)

    {

        for (intj = 0; j<N; j++)

        {

            printf("%d ", a[i][j]);

        }

        printf("\n");

    }

}

intmain()

{

    int \*\*a = (int \*\*)malloc(sizeof(int \*) \* N);

    int \*\*b = (int \*\*)malloc(sizeof(int \*) \* N);

    int \*\*c = (int \*\*)malloc(sizeof(int \*) \* N);

    for (inti = 0; i<N; i++)

    {

        a[i] = (int\*)malloc(sizeof(int) \* N);

        b[i] = (int\*)malloc(sizeof(int) \* N);

        c[i] = (int\*)malloc(sizeof(int) \* N);

    }

    input(a, 1);

    input(b, 1);

    doublestart = omp\_get\_wtime();

    add(a, b, c);

    doubleend = omp\_get\_wtime();

    // display(c);

    printf("Time taken (seq): %f\n", end - start);

}

#include<omp.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define N 250

voidadd(int\*\*a, int\*\*b, int\*\*c)

{

#pragmaompparallelfor

    for (inti = 0; i<N; i++)

    {

        for (intj = 0; j<N; j++)

        {

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

        }

    }

}

voidinput(int\*\*a, intnum)

{

    for (inti = 0; i<N; i++)

    {

        for (intj = 0; j<N; j++)

        {

            a[i][j] = num;

        }

    }

}

voiddisplayMatrix(int\*\*a)

{

    for (inti = 0; i<N; i++)

    {

        for (intj = 0; j<N; j++)

        {

            printf("%d ", a[i][j]);

        }

        printf("\n");

    }

}

intmain()

{

    int \*\*a = (int \*\*)malloc(sizeof(int \*) \* N);

    int \*\*b = (int \*\*)malloc(sizeof(int \*) \* N);

    int \*\*c = (int \*\*)malloc(sizeof(int \*) \* N);

    for (inti = 0; i<N; i++)

    {

        a[i] = (int\*)malloc(sizeof(int) \* N);

        b[i] = (int\*)malloc(sizeof(int) \* N);

        c[i] = (int\*)malloc(sizeof(int) \* N);

    }

    input(a, 1);

    input(b, 1);

    doublestart = omp\_get\_wtime();

    add(a, b, c);

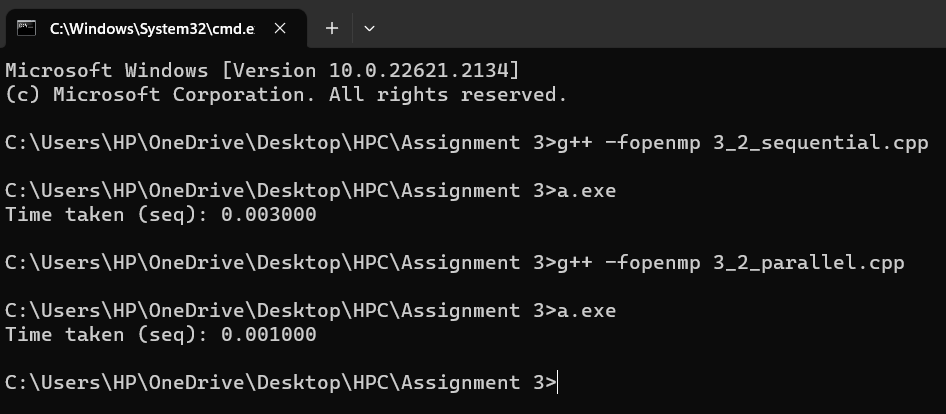
    doubleend = omp\_get\_wtime();

    // display(c);

    printf("Time taken (seq): %f\n", end - start);

}

**Screenshots:**

****

**Information and analysis:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Threads | Data Size | Sequential Time(sec)(Ts) | Parallel Time(sec)(Tp) | Speedup(Ts/Tp) |
| **2** | **250** | **0.0010000** | **0.000000** | **0** |
| **2** | **500** | **0.0010000** | **0.009000** | **0.11111111** |
| **2** | **750** | **0.0010000** | **0.015000** | **0.06666667** |
| **2** | **1000** | **0.0010000** | **0.018000** | **0.05555556** |
| **4** | **250** | **0.0010000** | **0.000000** | **0** |
| **4** | **500** | **0.0010000** | **0.005000** | **0.2** |
| **4** | **750** | **0.0010000** | **0.016000** | **0.6250** |
| **4** | **1000** | **0.0010000** | **0.017000** | **0.05882333** |
| **8** | **250** | **0.0010000** | **0.006000** | **0.16666667** |
| **8** | **500** | **0.0010000** | **0.015000** | **0.05882333** |
| **8** | **750** | **0.0010000** | **0.170000** | **0.05882333** |
| **8** | **1000** | **0.0010000** | **0.019000** | **0.05222768** |

**Github Link:** [**https://github.com/GauravP07/HPC\_Assignments**](https://github.com/GauravP07/HPC_Assignments)