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

**Year:** 2024-25 **Semester:** 1

**Course:** High Performance Computing Lab

**Practical No. 3**

**Exam Seat No : 21510013**

**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)

Sequential :

**Screenshots:**

**#include** <stdio.h>

**#include** <time.h>

**#define** n 100000

**int** 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])

            {

**int** temp **=** *nums*[j];

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

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

            }

}

**int** sortDesc(**int** *nums*[])

{

**int** i, j;

**for** (i **=** 0; i **<** n; i**++**)

    {

**for** (j **=** i **+** 1; j **<** n; j**++**)

        {

**if** (*nums*[i] **<** *nums*[j])

            {

**int** a **=** *nums*[i];

*nums*[i] **=** *nums*[j];

*nums*[j] **=** a;

            }

        }

    }

}

**int** main()

{

**int** nums1[n], nums2[n];

**for** (**int** i **=** 0; i **<** n; i**++**)

    {

        nums1[i] **=** 10;

    }

**for** (**int** i **=** 0; i **<** n; i**++**)

    {

        nums2[i] **=** 20;

    }

**clock\_t** t **=** clock();

    sort(nums1);

    sortDesc(nums2);

    t **=** clock() **-** t;

**double** time **=** ((**double**)t) **/** CLOCKS\_PER\_SEC;

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

**int** sum **=** 0;

**for** (**int** i **=** 0; i **<** n; i**++**)

    {

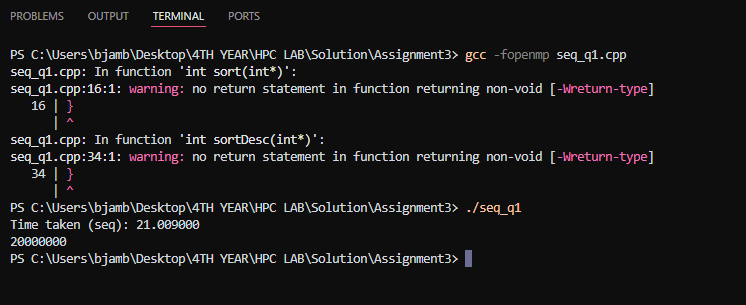
        sum **=** sum **+** (nums1[i] **\*** nums2[i]);

    }

    printf("%d\n", sum);

**return** 0;

}

****

**Parallel**

*// C Program to find the minimum scalar product of two*

**#include** <omp.h>

**#include** <stdio.h>

**#include** <time.h>

**#define** n 100000

**int** sort(**int** *nums*[])

{

**int** i, j;

**for** (i **=** 0; i **<** n; i**++**)

    {

**int** turn **=** i **%** 2;

**#pragma** **omp** **parallel** **for**

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

**if** (*nums*[j] **>** *nums*[j **+** 1])

            {

**int** temp **=** *nums*[j];

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

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

            }

    }

}

**int** sort\_des(**int** *nums*[])

{

**int** i, j;

**for** (i **=** 0; i **<** n; **++**i)

    {

**int** turn **=** i **%** 2;

**#pragma** **omp** **parallel** **for**

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

        {

**if** (*nums*[j] **<** *nums*[j **+** 1])

            {

**int** temp **=** *nums*[j];

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

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

            }

        }

    }

}

**int** main()

{

**int** nums1[n], nums2[n];

**for** (**int** i **=** 0; i **<** n; i**++**)

    {

        nums1[i] **=** 10;

    }

**for** (**int** i **=** 0; i **<** n; i**++**)

    {

        nums2[i] **=** 20;

    }

*// Set the number of threads*

    omp\_set\_num\_threads(4); *// Example: Set to 4 threads*

*// Print the number of threads used*

**int** num\_threads **=** omp\_get\_max\_threads();

    printf("Number of threads: %d\n", num\_threads);

**clock\_t** t;

    t **=** clock();

    sort(nums1);

    sort\_des(nums2);

    t **=** clock() **-** t;

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

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

**int** sum **=** 0;

**for** (**int** i **=** 0; i **<** n; i**++**)

    {

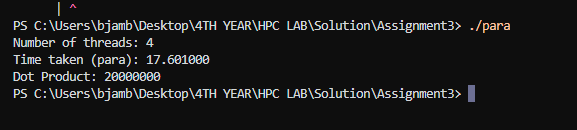
        sum **=** sum **+** (nums1[i] **\*** nums2[i]);

    }

    printf("Dot Product: %d\n", sum);

**return** 0;

}

****

**Information and analysis:**

**1. Speedup Calculation:**

**- Formula: Speedup = Sequential Time / Parallel Time**

**- Sequential Time = 21.00900 seconds**

**- Parallel Time = 17.601 seconds**

**Speedup = 21.00900 / 17.601 ≈ 1.195**

**The speedup is approximately 1.195.**

**2. Efficiency Calculation:**

**- Formula: Efficiency = Speedup / Number of Threads**

**- Speedup = 1.195**

**- Number of Threads = 4**

**Efficiency = 1.195 / 4 ≈ 0.299**

**Efficiency as a percentage = 0.299 × 100 ≈ 29.9%**

**The efficiency is approximately 29.9%.**

**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.

**Screenshots:**

**Sequential :**

**#include** <omp.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <time.h>

**#define** N 1000

**void** add(**int** **\*\****a*, **int** **\*\****b*, **int** **\*\****c*)

{

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

**for** (**int** j **=** 0; j **<** N; j**++**)

        {

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

        }

    }

}

**void** input(**int** **\*\****a*, **int** *num*)

{

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

**for** (**int** j **=** 0; j **<** N; j**++**)

        {

*a*[i][j] **=** *num*;

        }

    }

}

**void** display(**int** **\*\****a*)

{

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

**for** (**int** j **=** 0; j **<** N; j**++**)

        {

            printf("%d ", *a*[i][j]);

        }

        printf("\n");

    }

}

**int** main()

{

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

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

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

**for** (**int** i **=** 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);

**double** start **=** omp\_get\_wtime();

    add(a, b, c);

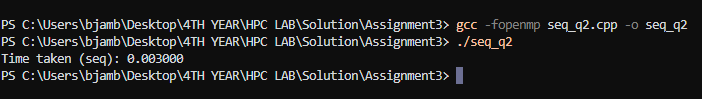
**double** end **=** omp\_get\_wtime();

*// display(c);*

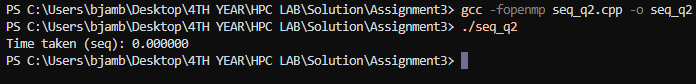
    printf("Time taken (seq): %f\n", end **-** start);

}

**N : 750**

****

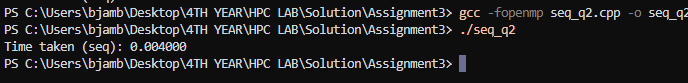
**N : 250**

****

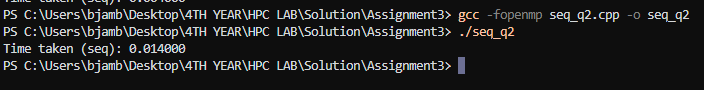
**N : 500**

****

**N : 750**

****

**N : 2000**

****

**Parallel :**

**#include** <omp.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <time.h>

**#define** N 250

**void** add(**int** **\*\****a*, **int** **\*\****b*, **int** **\*\****c*)

{

**#pragma** **omp** **parallel** **for**

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

**for** (**int** j **=** 0; j **<** N; j**++**)

        {

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

        }

    }

}

**void** input(**int** **\*\****a*, **int** *num*)

{

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

**for** (**int** j **=** 0; j **<** N; j**++**)

        {

*a*[i][j] **=** *num*;

        }

    }

}

**void** displayMatrix(**int** **\*\****a*)

{

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

**for** (**int** j **=** 0; j **<** N; j**++**)

        {

            printf("%d ", *a*[i][j]);

        }

        printf("\n");

    }

}

**int** main()

{

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

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

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

**for** (**int** i **=** 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);

**double** start **=** omp\_get\_wtime();

    add(a, b, c);

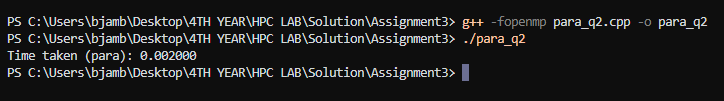
**double** end **=** omp\_get\_wtime();

*// display(c);*

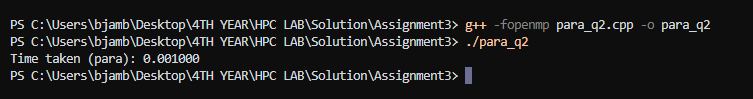
    printf("Time taken (para): %f\n", end **-** start);

}

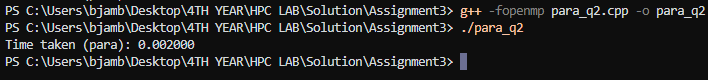
**N : 250**

****

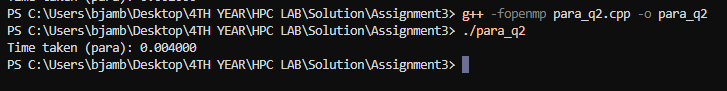
**N : 500**

****

**N : 750**

****

**N : 1000**

****

**N : 2000**

****

**Information and analysis:**

**For sequential :**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **250** | **500** | **750** | **1000** | **2000** |
| **Time** | **0.000** | **0.001** | **0.003** | **0.004** | **0.014** |

**For parallel :**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **250** | **500** | **750** | **1000** | **2000** |
| **Time** | **0.002** | **0.001** | **0.002** | **0.004** | **0.009** |

**Problem Statement 3:**

For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following: i. Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup. ii. Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup. iii. Demonstrate the use of nowait clause.

**Screenshots:**

**Static schedule :**

**#include** <omp.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#define** N 200

**int** main()

{

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

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

**int** b **=** 10;

    omp\_set\_num\_threads(6);

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

        a[i] **=** 0;

    }

**double** itime, ftime, exec\_time;

    itime **=** omp\_get\_wtime();

**#pragma** **omp** **parallel** **for** **schedule**(**static**, 2)

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

        c[i] **=** a[i] **+** b;

    }

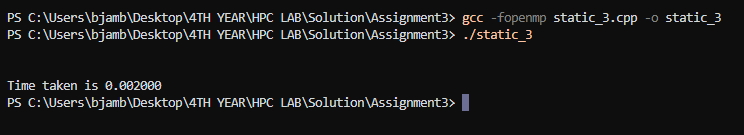
    ftime **=** omp\_get\_wtime();

    exec\_time **=** ftime **-** itime;

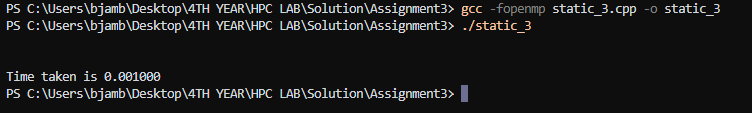
    printf("\n\nTime taken is %f\n", exec\_time);

}

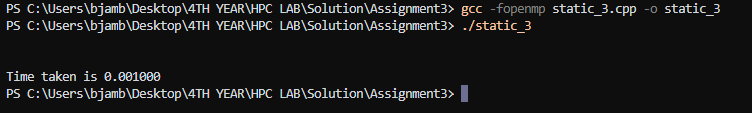
**Chunk size 2**

****

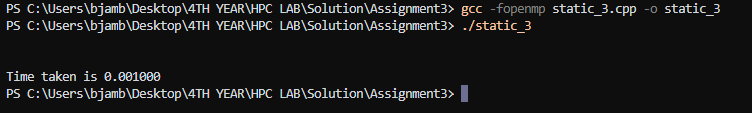
**Chunk size 4**

****

**Chunk size 6**

****

**Chunk size 8**

****

**Dynamic :**

**#include** <omp.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#define** N 200

**int** main()

{

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

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

**int** b **=** 10;

    omp\_set\_num\_threads(6);

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

        a[i] **=** 0;

    }

**double** itime, ftime, exec\_time;

    itime **=** omp\_get\_wtime();

**#pragma** **omp** **parallel** **for** **schedule**(**dynamic**, 2)

**for** (**int** i **=** 0; i **<** N; i**++**)

    {

        c[i] **=** a[i] **+** b;

    }

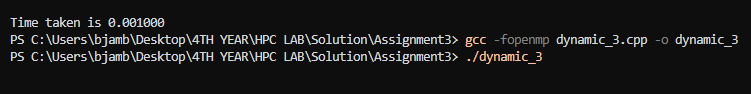
    ftime **=** omp\_get\_wtime();

    exec\_time **=** ftime **-** itime;

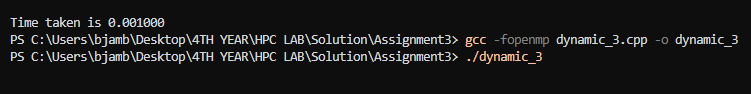
    printf("\n\nTime taken is %f\n", exec\_time);

}

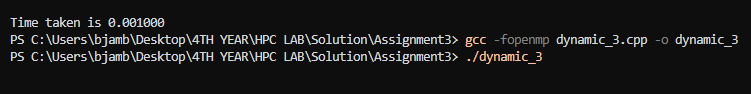
**Chunk size – 2**

****

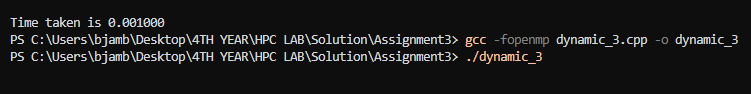
**Chunk size 4**

****

**Chunk size 6**

****

**Chunk size 8**

****

**Information and analysis:**

**Static schedule**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chunk size** | **2** | **4** | **6** | **8** |
| **Time** | **0.002** | **0.001** | **0.001** | **0.001** |

**Dynamic schedule**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chunk size** | **2** | **4** | **6** | **8** |
| **Time** | **0.001** | **0.001** | **0.001** | **0.001** |

**Nowait clause**

**#include** <omp.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#define** N 10

**void** hello\_world()

{

    printf("Hello world\n");

}

**void** print(**int** *i*)

{

    printf("Value %d\n", *i*);

}

**int** main()

{

**#pragma** **omp** **parallel**

    {

**#pragma** **omp** **for** **nowait**

**for** (**int** i **=** 0; i **<** N; i**++**)

        {

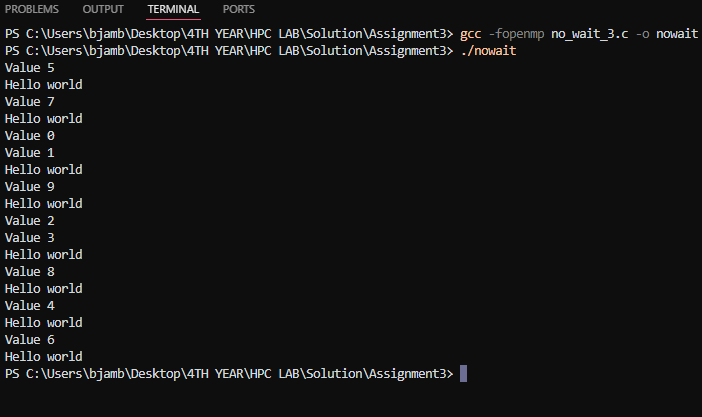
            print(i);

        }

        hello\_world();

    }

}

****

**Github Link:**