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

**Year:** 2024-25 **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 <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define VECTOR\_SIZE 1000000

void initialize\_vectors(int \*vec1, int \*vec2, int size) {

    for (int i = 0; i < size; i++) {

        vec1[i] = rand() % 100;

        vec2[i] = rand() % 100;

    }

}

long long compute\_min\_scalar\_product(int \*vec1, int \*vec2, int size) {

    long long min\_scalar\_product = 0;

    #pragma omp parallel shared(vec1, vec2) reduction(+:min\_scalar\_product)

    {

        #pragma omp for collapse(1) schedule(static) nowait

        for (int i = 0; i < size; i++) {

            min\_scalar\_product += (long long)vec1[i] \* vec2[i];

        }

    }

    return min\_scalar\_product;

}

int main() {

    int \*vec1 = (int \*)malloc(VECTOR\_SIZE \* sizeof(int));

    int \*vec2 = (int \*)malloc(VECTOR\_SIZE \* sizeof(int));

    srand(0);

    initialize\_vectors(vec1, vec2, VECTOR\_SIZE);

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

    long long min\_scalar\_product = compute\_min\_scalar\_product(vec1, vec2, VECTOR\_SIZE);

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

    printf("Minimum Scalar Product: %lld\n", min\_scalar\_product);

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

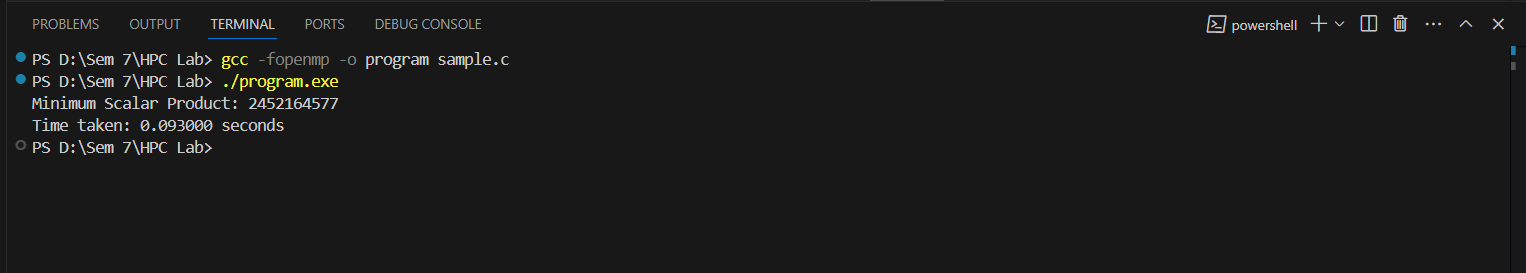
    free(vec1);

    free(vec2);

    return 0;

}

**Screenshots:**

****

**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 MAX\_SIZE 2000

void initialize\_matrix(int \*matrix, int size) {

    for (int i = 0; i < size \* size; i++) {

        matrix[i] = rand() % 100;

    }

}

void matrix\_addition\_sequential(int \*A, int \*B, int \*C, int size) {

    for (int i = 0; i < size; i++) {

        for (int j = 0; j < size; j++) {

            C[i \* size + j] = A[i \* size + j] + B[i \* size + j];

        }

    }

}

void matrix\_addition\_parallel(int \*A, int \*B, int \*C, int size, int num\_threads) {

    #pragma omp parallel for collapse(2) num\_threads(num\_threads)

    for (int i = 0; i < size; i++) {

        for (int j = 0; j < size; j++) {

            C[i \* size + j] = A[i \* size + j] + B[i \* size + j];

        }

    }

}

int main() {

    int sizes[] = {250, 500, 750, 1000, 2000};

    int num\_sizes = sizeof(sizes) / sizeof(sizes[0]);

    int num\_threads[] = {2, 4, 8};

    int num\_thread\_counts = sizeof(num\_threads) / sizeof(num\_threads[0]);

    srand(time(NULL));

    for (int s = 0; s < num\_sizes; s++) {

        int size = sizes[s];

        int \*A = (int \*)malloc(size \* size \* sizeof(int));

        int \*B = (int \*)malloc(size \* size \* sizeof(int));

        int \*C = (int \*)malloc(size \* size \* sizeof(int));

        initialize\_matrix(A, size);

        initialize\_matrix(B, size);

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

        matrix\_addition\_sequential(A, B, C, size);

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

        printf("Sequential Time for size %d: %f seconds\n", size, end\_time - start\_time);

        for (int t = 0; t < num\_thread\_counts; t++) {

            int threads = num\_threads[t];

            omp\_set\_num\_threads(threads);

            start\_time = omp\_get\_wtime();

            matrix\_addition\_parallel(A, B, C, size, threads);

            end\_time = omp\_get\_wtime();

            printf("Parallel Time for size %d with %d threads: %f seconds\n", size, threads, end\_time - start\_time);

        }

        free(A);

        free(B);

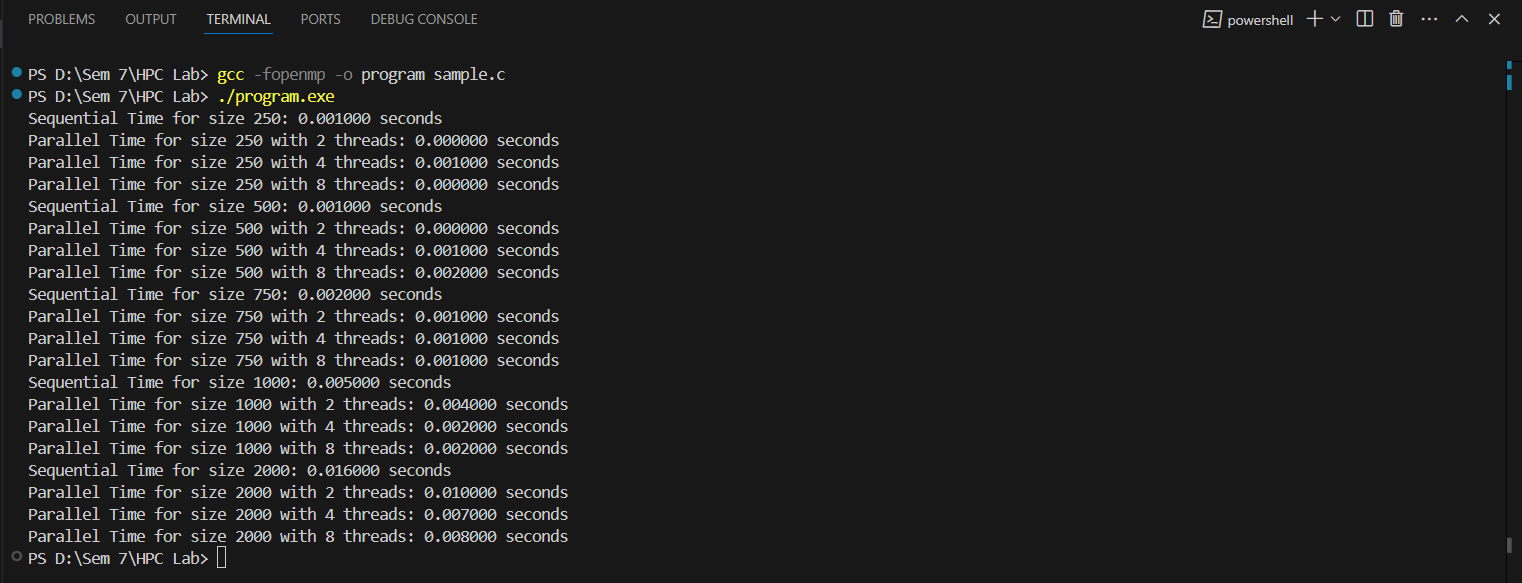
        free(C);

    }

    return 0;

}

**Screenshots:**

****

**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.

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define VECTOR\_SIZE 200

void vector\_scalar\_addition(int \*vector, int scalar, int chunk\_size, omp\_sched\_t schedule\_type) {

    omp\_set\_schedule(schedule\_type, chunk\_size);

    #pragma omp parallel for schedule(runtime)

    for (int i = 0; i < VECTOR\_SIZE; i++) {

        vector[i] += scalar;

    }

}

int main() {

    int vector[VECTOR\_SIZE];

    int scalar = 5;

    for (int i = 0; i < VECTOR\_SIZE; i++) {

        vector[i] = i;

    }

    int chunk\_sizes[] = {1, 5, 10};

    omp\_sched\_t schedules[] = {omp\_sched\_static, omp\_sched\_dynamic};

    for (int s = 0; s < 2; s++) {

        for (int c = 0; c < 3; c++) {

            omp\_set\_schedule(schedules[s], chunk\_sizes[c]);

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

            vector\_scalar\_addition(vector, scalar, chunk\_sizes[c], schedules[s]);

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

            printf("Schedule: %s, Chunk Size: %d, Time taken: %f seconds\n",

                   schedules[s] == omp\_sched\_static ? "Static" : "Dynamic",

                   chunk\_sizes[c],

                   end\_time - start\_time);

        }

    }

    #pragma omp parallel

    {

        #pragma omp for schedule(static, 5) nowait

        for (int i = 0; i < VECTOR\_SIZE; i++) {

            vector[i] += scalar;

        }

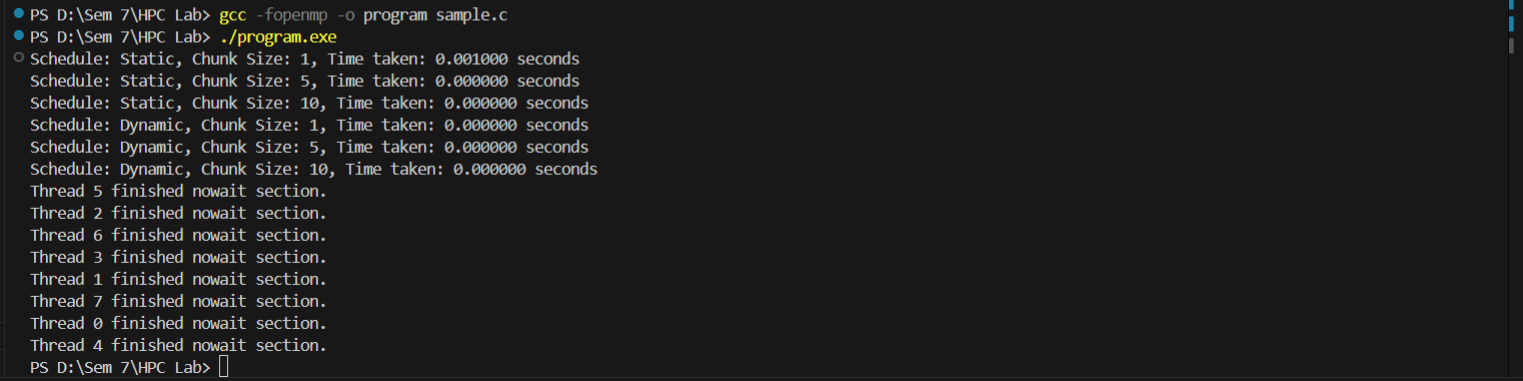
        printf("Thread %d finished nowait section.\n", omp\_get\_thread\_num());

    }

    return 0;

}

**Screenshots:**

****

**Github Link:** [**https://github.com/AbhijeetKamalekar15/HPC-Lab**](https://github.com/AbhijeetKamalekar15/HPC-Lab)