Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

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

Course: High Performance Computing Lab

Practical No. 3

Exam Seat No: 21510111

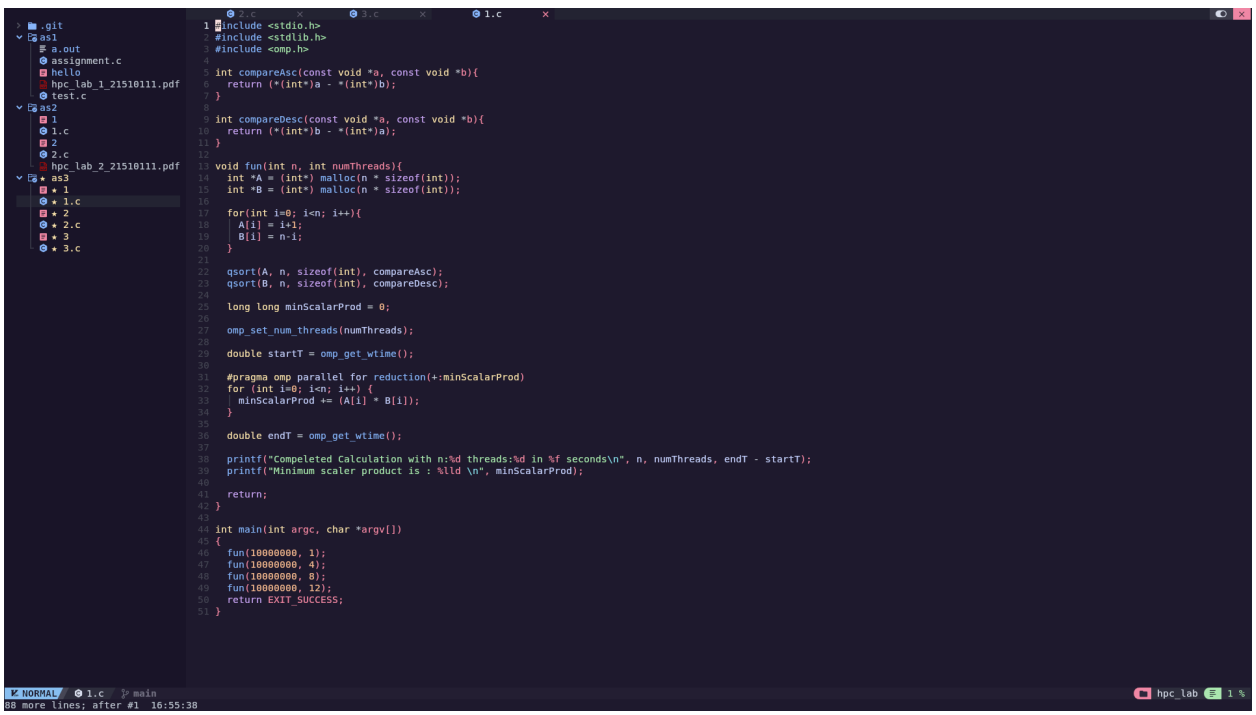
Title of practical:

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)

Screenshots:



```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int compareAsc(const void *a, const void *b){
  return (*(int*)a - *(int*)b);
}

int compareDesc(const void *a, const void *b){
  return (*(int*)b - *(int*)a);
}

void fun(int n, int numThreads){
  int *A = (int*) malloc(n * sizeof(int));
  int *B = (int*) malloc(n * sizeof(int));

  for(int i=0; i<n; i++){
    A[i] = i+1;
    B[i] = n-i;
  }

  qsort(A, n, sizeof(int), compareAsc);
  qsort(B, n, sizeof(int), compareDesc);

  long long minScalarProd = 0;

  omp_set_num_threads(numThreads);

  double startT = omp_get_wtime();

  #pragma omp parallel for reduction(+:minScalarProd)
  for (int i=0; i<n; i++) {
    minScalarProd += (A[i] * B[i]);
  }

  double endT = omp_get_wtime();

  printf("Compeleted Calculation with n:%d threads:%d in %f seconds\n", n, numThreads, endT - startT);
  printf("Minimum scaler product is : %lld \n", minScalarProd);

  return;
}

int main(int argc, char *argv[])
{
  fun(10000000, 1);
  fun(10000000, 4);
  fun(10000000, 8);
  fun(10000000, 12);
  return EXIT_SUCCESS;
}
```

Information and analysis:

```
*[main][~/acad/hpc_lab/as3]$ gcc -fopenmp 1.c -o 1 && ./1
Compeleted Calculation with n:10000000 threads:1 in 0.033436 seconds
Minimum scaler product is : -51629037803648
Compeleted Calculation with n:10000000 threads:4 in 0.008695 seconds
Minimum scaler product is : -51629037803648
Compeleted Calculation with n:10000000 threads:8 in 0.007379 seconds
Minimum scaler product is : -51629037803648
Compeleted Calculation with n:10000000 threads:12 in 0.007908 seconds
Minimum scaler product is : -51629037803648
```

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:

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void fun(int n, int numThreads){
  int **mat1 = (int**)malloc(n * sizeof(int*));
  int **mat2 = (int**)malloc(n * sizeof(int*));
  int **ans = (int**)malloc(n * sizeof(int*));

  for (int i = 0; i < n; i++) {
    mat1[i] = (int*)malloc(n * sizeof(int));
    mat2[i] = (int*)malloc(n * sizeof(int));
    ans[i] = (int*)malloc(n * sizeof(int));
  }
  omp_set_num_threads(numThreads);

  double startT = omp_get_wtime();

  #pragma omp parallel fo
  for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++){
      ans[i][j] = mat1[i][j] + mat2[i][j];
    }
  }

  double endT = omp_get_wtime();

  printf("Time taken for n:%d threads:%d : %f \n", n, numThreads, endT - startT);

  for (int i = 0; i < n; i++) {
    free(mat1[i]);
    free(mat2[i]);
    free(ans[i]);
  }
  free(mat1);
  free(mat2);
  free(ans);

  return;
}

int main(int argc, char *argv[])
{
  int matSizes[] = {250, 500, 750, 1000, 2000};
  int numThreads[] = {2, 4, 8};
  for(int i=0; i<5; i++){
    printf("\nCalculations for matrix size %d : \n", matSizes[i]);
    for(int j=0; j<3; j++){
      fun(matSizes[i], numThreads[j]);
    }
  }
  return EXIT_SUCCESS;
}
```

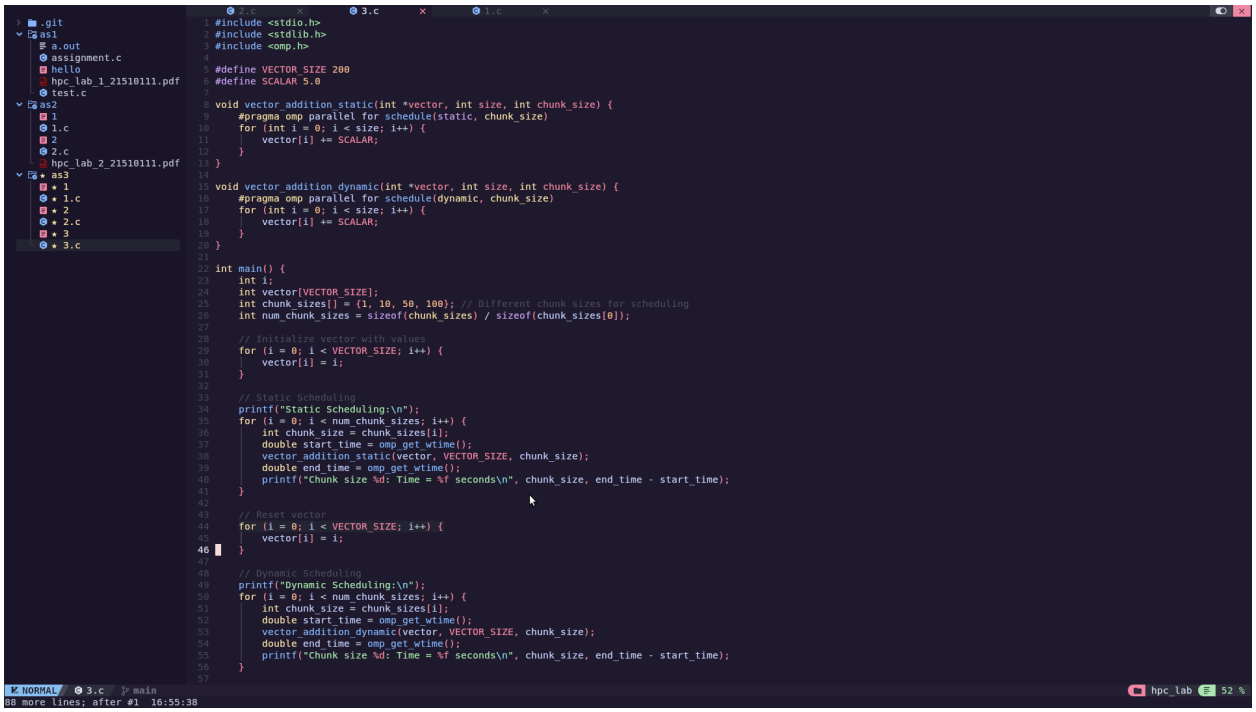
Information and analysis:

```
*[main][~/acad/hpc_lab/as3]$ gcc -fopenmp 2.c -o 2 && ./2

Calculations for matrix size 250 :
Time taken for n:250 threads:2 : 0.000618
Time taken for n:250 threads:4 : 0.000451
Time taken for n:250 threads:8 : 0.000580

Calculations for matrix size 500 :
Time taken for n:500 threads:2 : 0.001618
Time taken for n:500 threads:4 : 0.000965
Time taken for n:500 threads:8 : 0.001061

Calculations for matrix size 750 :
Time taken for n:750 threads:2 : 0.003522
Time taken for n:750 threads:4 : 0.001970
Time taken for n:750 threads:8 : 0.002727

Calculations for matrix size 1000 :
Time taken for n:1000 threads:2 : 0.007273
Time taken for n:1000 threads:4 : 0.005509
Time taken for n:1000 threads:8 : 0.005752

Calculations for matrix size 2000 :
Time taken for n:2000 threads:2 : 0.034298
Time taken for n:2000 threads:4 : 0.028746
Time taken for n:2000 threads:8 : 0.019612
```

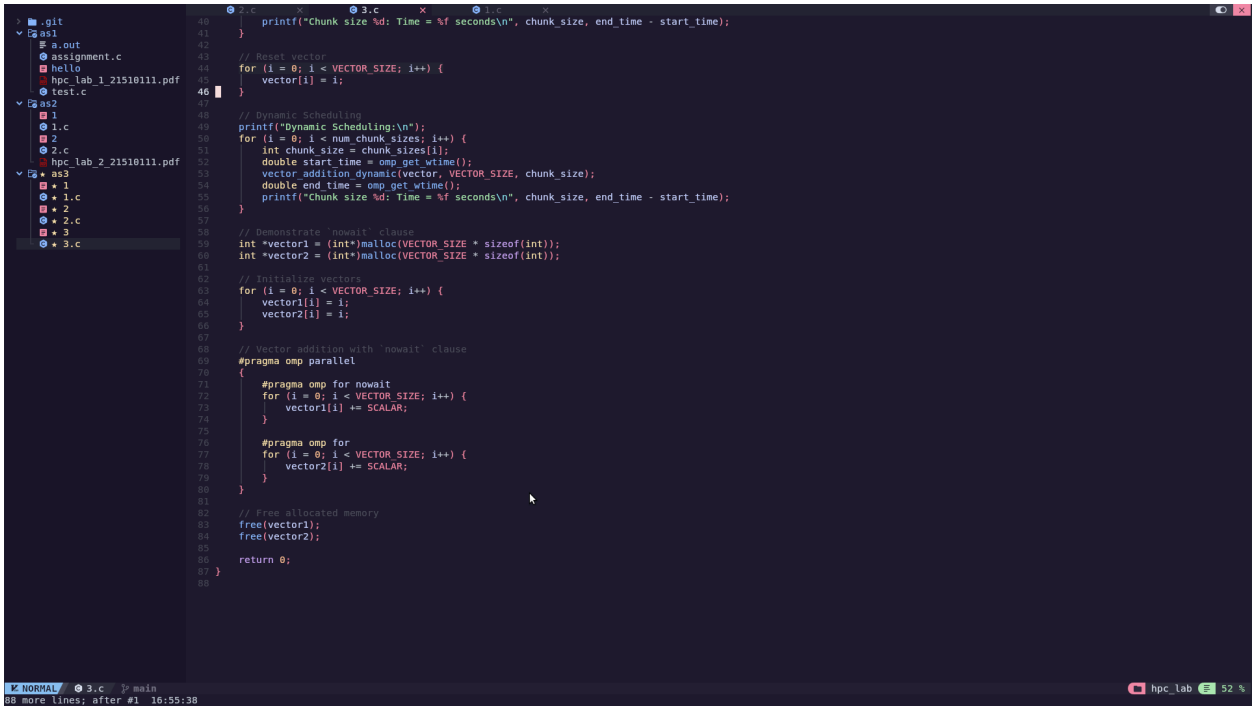
Problem Statement 3:

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:



```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define VECTOR_SIZE 200
#define SCALAR 5.0

void vector_addition_static(int *vector, int size, int chunk_size) {
    #pragma omp parallel for schedule(static, chunk_size)
    for (int i = 0; i < size; i++) {
        vector[i] += SCALAR;
    }
}

void vector_addition_dynamic(int *vector, int size, int chunk_size) {
    #pragma omp parallel for schedule(dynamic, chunk_size)
    for (int i = 0; i < size; i++) {
        vector[i] += SCALAR;
    }
}

int main() {
    int i;
    int vector[VECTOR_SIZE];
    int chunk_sizes[] = {1, 10, 50, 100}; // Different chunk sizes for scheduling
    int num_chunk_sizes = sizeof(chunk_sizes) / sizeof(chunk_sizes[0]);

    // Initialize vector with values
    for (i = 0; i < VECTOR_SIZE; i++) {
        vector[i] = i;
    }

    // Static Scheduling
    printf("Static Scheduling:\n");
    for (i = 0; i < num_chunk_sizes; i++) {
        int chunk_size = chunk_sizes[i];
        double start_time = omp_get_wtime();
        vector_addition_static(vector, VECTOR_SIZE, chunk_size);
        double end_time = omp_get_wtime();
        printf("Chunk size %d: Time = %f seconds\n", chunk_size, end_time - start_time);
    }

    // Reset vector
    for (i = 0; i < VECTOR_SIZE; i++) {
        vector[i] = i;
    }

    // Dynamic Scheduling
    printf("Dynamic Scheduling:\n");
    for (i = 0; i < num_chunk_sizes; i++) {
        int chunk_size = chunk_sizes[i];
        double start_time = omp_get_wtime();
        vector_addition_dynamic(vector, VECTOR_SIZE, chunk_size);
        double end_time = omp_get_wtime();
        printf("Chunk size %d: Time = %f seconds\n", chunk_size, end_time - start_time);
    }
```



Information and analysis:

```
*[main][~/acad/hpc_lab/as3]$ gcc -fopenmp 3.c -o 3 && ./3
Static Scheduling:
Chunk size 1: Time = 0.000587 seconds
Chunk size 10: Time = 0.000006 seconds
Chunk size 50: Time = 0.000005 seconds
Chunk size 100: Time = 0.000006 seconds
Dynamic Scheduling:
Chunk size 1: Time = 0.000026 seconds
Chunk size 10: Time = 0.000009 seconds
Chunk size 50: Time = 0.000006 seconds
Chunk size 100: Time = 0.000006 seconds
*[main][~/acad/hpc_lab/as3]$
```

Github Link: https://github.com/Sidd-77/hpc-lab/tree/main/as3