Name: Muhammad Soban Rasheed FA20-BCS-020

Khubaib ur Rehman FA20-BCS-141

Danish Nasar FA20-BCS-006

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

#define SIZE 250
#define EVEN_COUNT 100
#define REPEAT 1000

void initializeArray(int **array) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            array[i][j] = rand() % 1000; // Random values between 0 and 999
        }
    }
}

void findEvenNumbersParallel(int **array1, int **array2, int **array3, int *evenArray,
int num_threads, omp_sched_t schedule) {
    int count = 0;
    omp_set_num_threads(num_threads);
```

```
omp_set_schedule(schedule, 0);

#pragma omp parallel
{
    int local_count = 0;
    int local_even[EVEN_COUNT];

    #pragma omp for collapse(2)
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (local_count < EVEN_COUNT && array1[i][j] % 2 == 0) {
                local_even[local_count++] = array1[i][j];
            }
            if (local_count < EVEN_COUNT && array2[i][j] % 2 == 0) {
                local_even[local_count++] = array2[i][j];
            }
            if (local_count < EVEN_COUNT && array3[i][j] % 2 == 0) {
                local_even[local_count++] = array3[i][j];
            }
        }
    }

    #pragma omp critical
    {
```

```
            for (int k = 0; k < local_count && count < EVEN_COUNT; k++) {
                evenArray[count++] = local_even[k];
            }
        }
    }
}

void findEvenNumbersSerial(int **array1, int **array2, int **array3, int *evenArray) {
    int count = 0;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (count < EVEN_COUNT && array1[i][j] % 2 == 0) {
                evenArray[count++] = array1[i][j];
            }
            if (count < EVEN_COUNT && array2[i][j] % 2 == 0) {
                evenArray[count++] = array2[i][j];
            }
            if (count < EVEN_COUNT && array3[i][j] % 2 == 0) {
                evenArray[count++] = array3[i][j];
            }
        }
    }
}
```

```
int main() {
    int **array1 = (int **)malloc(SIZE * sizeof(int *));
    int **array2 = (int **)malloc(SIZE * sizeof(int *));
    int **array3 = (int **)malloc(SIZE * sizeof(int *));
    for (int i = 0; i < SIZE; i++) {
        array1[i] = (int *)malloc(SIZE * sizeof(int));
        array2[i] = (int *)malloc(SIZE * sizeof(int));
        array3[i] = (int *)malloc(SIZE * sizeof(int));
    }
    int evenArray[EVEN_COUNT];
    double start_time, end_time;
    double serial_time;

    srand(time(NULL));

    initializeArray(array1);
    initializeArray(array2);
    initializeArray(array3);

    // Serial Execution
    start_time = omp_get_wtime();
    for (int r = 0; r < REPEAT; r++) {
        findEvenNumbersSerial(array1, array2, array3, evenArray);
    }
```

```
    end_time = omp_get_wtime();
    serial_time = (end_time - start_time) / REPEAT;
    printf("Serial Execution Time: %f seconds\n", serial_time);

    // Parallel Execution with scheduling
    int threads[] = {2, 4, 8, 12, 16, 24};
    omp_sched_t schedules[] = {omp_sched_static, omp_sched_dynamic,
omp_sched_guided};
    const char* schedule_names[] = {"Static", "Dynamic", "Guided"};

    double exec_times[6];
    double speedups[6];

    for (int t = 0; t < 6; t++) {
        start_time = omp_get_wtime();
        for (int r = 0; r < REPEAT; r++) {
            findEvenNumbersParallel(array1, array2, array3, evenArray, threads[t],
omp_sched_static);
        }
        end_time = omp_get_wtime();
        exec_times[t] = (end_time - start_time) / REPEAT;
        speedups[t] = serial_time / exec_times[t];
        printf("Execution Time with %d threads: %f seconds, Speedup: %f\n", threads[t],
exec_times[t], speedups[t]);
    }
```

```
    double schedule_times[3];
    for (int s = 0; s < 3; s++) {
        start_time = omp_get_wtime();
        for (int r = 0; r < REPEAT; r++) {
            findEvenNumbersParallel(array1, array2, array3, evenArray, 24, schedules[s]);
        }
        end_time = omp_get_wtime();
        schedule_times[s] = (end_time - start_time) / REPEAT;
        printf("Execution Time with %s scheduling: %f seconds\n", schedule_names[s],
schedule_times[s]);
    }

    printf("Fourth array (even numbers):\n");
    for (int i = 0; i < EVEN_COUNT; i++) {
        printf("%d ", evenArray[i]);
    }
    printf("\n");

    for (int i = 0; i < SIZE; i++) {
        free(array1[i]);
        free(array2[i]);
        free(array3[i]);
    }
    free(array1);
```

```
    free(array2);
    free(array3);

    // Export results to a CSV file (for ease of importing to Excel)
    FILE *f = fopen("execution_times.csv", "w");
    fprintf(f, "Threads,Execution Time (s),Speedup\n");
    for (int t = 0; t < 6; t++) {
        fprintf(f, "%d,%f,%f\n", threads[t], exec_times[t], speedups[t]);
    }
    fprintf(f, "Scheduling Method,Execution Time (s)\n");
    for (int s = 0; s < 3; s++) {
        fprintf(f, "%s,%f\n", schedule_names[s], schedule_times[s]);
    }
    fclose(f);

    return 0;
}
```