REPORT

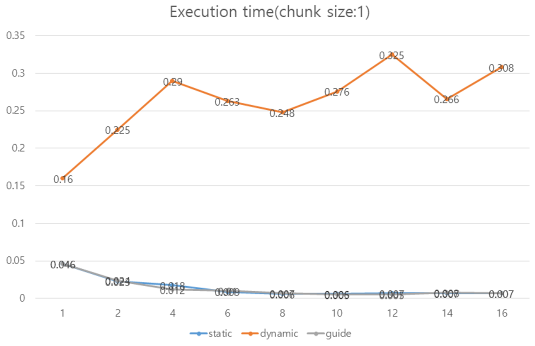
#problem2_document

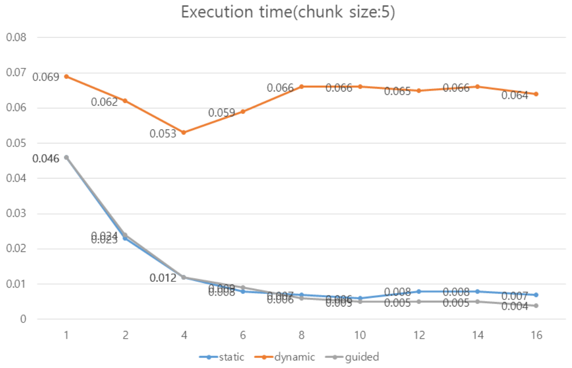
과목명	멀티코어컴퓨팅 01분반
교수명	손봉수
학번	20206319
학과	소프트웨어학부 소프트웨어전공
이름	김가연

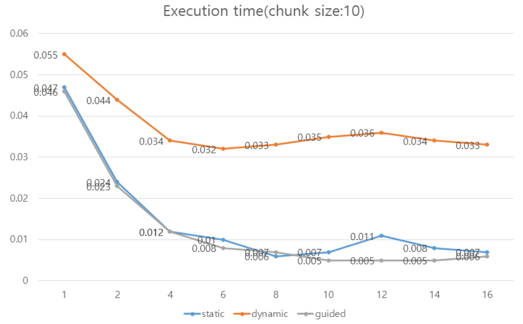
(a) Tables and graphs that show the execution time (unit: ms) for the number of entire threads = {1,2,4,6,8,10,12,14,16}.

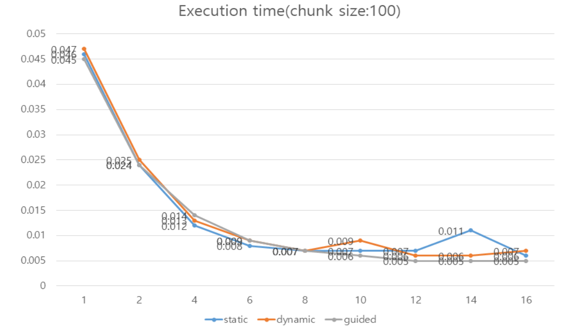
exec time (unit:ms)	chunk size	1	2	4	6	8	10	12	14	16
static	1	0.046	0.023	0.018	0.009	0.006	0.006	0.007	0.007	0.007
dynamic		0.16	0.225	0.29	0.263	0.248	0.276	0.325	0.266	0.308
guide		0.046	0.024	0.012	0.01	0.007	0.005	0.005	0.008	0.007
static	5	0.046	0.023	0.012	0.008	0.007	0.006	0.008	0.008	0.007
dynamic		0.069	0.062	0.053	0.059	0.066	0.066	0.065	0.066	0.064
guided		0.046	0.024	0.012	0.009	0.006	0.005	0.005	0.005	0.004
static	10	0.047	0.024	0.012	0.01	0.006	0.007	0.011	0.008	0.007
dynamic		0.055	0.044	0.034	0.032	0.033	0.035	0.036	0.034	0.033
guided		0.046	0.023	0.012	0.008	0.007	0.005	0.005	0.005	0.006
static	100	0.046	0.024	0.012	0.008	0.007	0.007	0.007	0.011	0.006
dynamic		0.047	0.025	0.013	0.009	0.007	0.009	0.006	0.006	0.007
guided		0.045	0.024	0.014	0.009	0.007	0.006	0.005	0.005	0.005

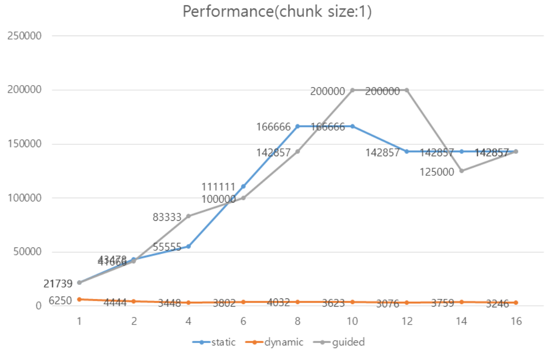
performance (1/exec time)	chunk size	1	2	4	6	8	10	12	14	16
static	1	21739	43478	55555	111111	166666	166666	142857	142857	142857
dynamic		6250	4444	3448	3802	4032	3623	3076	3759	3246
guide		21739	41666	83333	100000	142857	200000	200000	125000	142857
static	5	21739	43478	83333	125000	142857	166666	125000	125000	142857
dynamic		14492	16129	18867	16949	15151	15151	15384	15151	15625
guided		21739	41666	83333	111111	166666	200000	200000	200000	250000
static	10	21276	41666	83333	100000	166666	142857	90909	125000	142857
dynamic		18181	22727	29411	31250	30303	28571	27777	29411	30303
guided		21739	43478	83333	125000	142857	200000	200000	200000	166666
static	100	21739	41666	83333	125000	142857	142857	142857	90909	166666
dynamic		21276	40000	76923	111111	142857	111111	166666	166666	142857
guided		22222	41666	71428	111111	142857	166666	200000	200000	200000

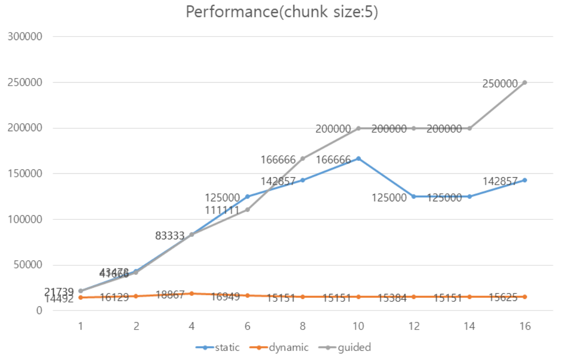


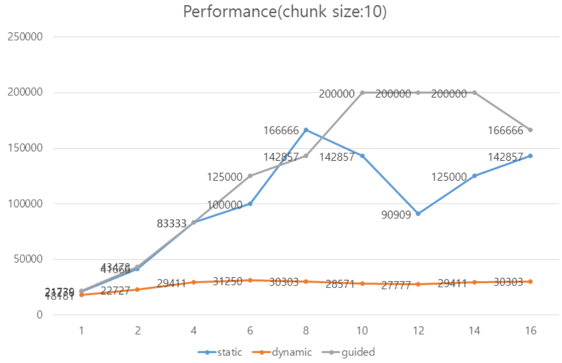


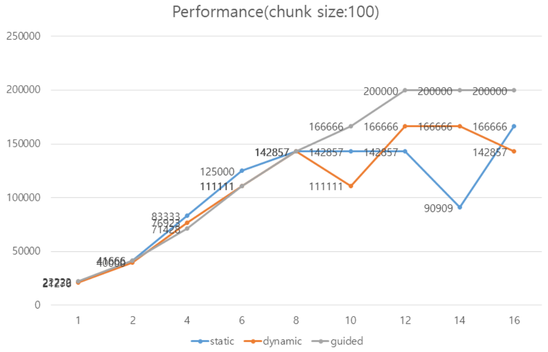












(b) Report the parallel performance of my code and explanation/analysis on the results and why such results are obtained with sufficient details.

#proj2.c

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

int main(int stn, char *arr[]) {
    if (stn != 4) {
        printf("Usage: %s scheduling_type_number chunk_size num_of_threads\n", arr[0]);
        return 1;
    }
    int num_steps = 10000000;
    int scheduling_type = atoi(arr[1]);
    int chunk_size = atoi(arr[2]);
    int num_threads = atoi(arr[3]);

    double step = 1.0 / (double)num_steps;
    double pi = 0.0;

    omp_set_num_threads(num_threads);

    double start_time = omp_get_wtime();
    double x, sum = 0.0;

    #pragma omp parallel reduction(+:sum) private(x)
    {
        if(scheduling_type == 1){
            #pragma omp for schedule(static, chunk_size)
            for(int i=0; i<num_steps; i++) {
                x = (i + 0.5) * step;
                sum += 4.0 / (1.0 + x * x);
            }
        }
        else if(scheduling_type == 2) {
            #pragma omp for schedule(dynamic, chunk_size)
            for(int i=0; i<num_steps; i++) {
                x = (i + 0.5) * step;
                sum += 4.0 / (1.0 + x * x);
            }
        }
        else if (scheduling_type == 3) {
            #pragma omp for schedule(guided, chunk_size)
            for(int i=0; i<num_steps; i++) {
                x = (i + 0.5) * step;
                sum += 4.0 / (1.0 + x * x);
            }
        }
        pi += sum * step;
    }

    double end_time = omp_get_wtime();
    double execution_time = end_time - start_time;

    printf("Result of PI calculation: %.24lf\n", pi);
    printf("Execution time: %lf ms\n", execution_time);
    return 0;
}
```

Lines 6-14: This is the code to input in the terminal according to the suggested conditions.
Lines 24-48: Execution code for receiving scheduling type and chunk size according to suggested conditions and calculating pi using openMP.
Lines 50-56: This is the code for measuring the execution time and outputting the PI calculation result and the execution time.
To measure the exact time, the omp_get_wtime() method is used.

When the chunk size was 1, 5, or 10, the execution time of dynamic scheduling was the highest, and the execution time of static scheduling and guided scheduling were similar. When the chunk size was 100, the execution times of dynamic schduling, static scheduling, and guided scheduling were similar, and when the number of threads were 10 and 12, the execution times of dynamic schduling and static scheduling were slightly higher, respectively. First, when the chunk size is 1, 5, or 10, the reason why the execution time of dynamic scheduling is the highest is that while tasks are dynamically allocated, each thread gets a task from the task queue whenever a task is requested. I suspect that each time you request a task more frequently and get a task from the task queue, the overhead will increase and thus take longer to run. On the other hand, when the chunk size is 100, the reason why the execution times of dynamic schduling, static scheduling, and guided scheduling are similar is that the larger the chunk size, the smaller the overhead required for task request and task processing, so multiple tasks can be executed at once. While processing, I think that the overhead required for processing work requests and work queues is reduced, resulting in similar execution times.
In the case of performance, since it is the reciprocal of the execution time, when the chunk size is 1, 5, or 10, the performance of dynamic scheduling is the lowest, and the performance of guided scheduling is generally the highest. On the other hand, when the chunk size is 100, the performances of dynamic schduling, static scheduling, and guided scheduling are measured similarly, and similarly, the performance of guided scheduling is measured the highest.