REPORT

#problem1_document

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

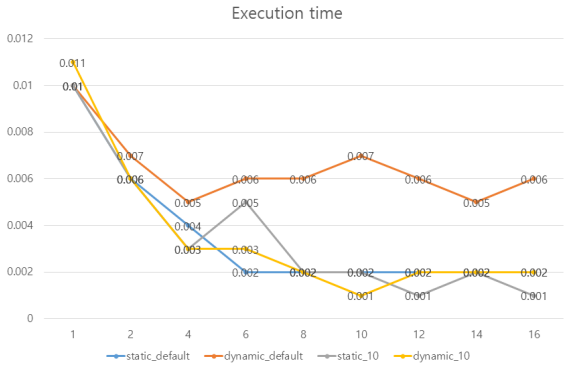
(a) What environment (e.g. CPU type, number of cores, memory size, OS type) the experimentation was performed.

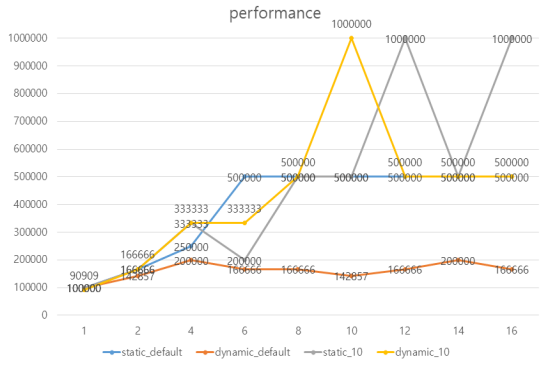
The CPU type of this machine is 12th Gen Intel(R) Core(TM) i9-12900K.
It has 16 cores, four logical processors(when hyperthreading on), and clock speed is 3.20GHz.
The memory size is 32.0GB. The speed of the memory is 4400MHz.
The OS type is 64-bit operating system, x64 based processor.
Experiments were conducted in the same environment as above.

(b) Tables and graphs that show the execution time (unit:ms) for thread number = {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	default	0.01	0.006	0.004	0.002	0.002	0.002	0.002	0.002	0.002
dynamic	default	0.01	0.007	0.005	0.006	0.006	0.007	0.006	0.005	0.006
static	10	0.01	0.006	0.003	0.005	0.002	0.002	0.001	0.002	0.001
dynamic	10	0.011	0.006	0.003	0.003	0.002	0.001	0.002	0.002	0.002

performance (1/exec time)	chunk size	1	2	4	6	8	10	12	14	16
static	default	100000	166666	250000	500000	500000	500000	500000	500000	500000
dynamic	default	100000	142857	200000	166666	166666	142857	166666	200000	166666
static	10	100000	166666	333333	200000	500000	500000	1000000	500000	1000000
dynamic	10	90909	166666	333333	333333	500000	1000000	500000	500000	500000





(c) Report the parallel performance of my code and explanation on the results and why such results can be obtained.

#prob1.c

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

bool is_prime(int num){
    if(num <= 1) return false;
```

```
    for(int i=2; i*i<= num; i++) {
        if(num%i == 0) return false;
    }
    return true;
}

int main(int stn, char *arr[]) {
    if (stn != 3) {
        printf("Usage: %s scheduling_type_number num_of_threads\n", arr[0]);
        return 1;
    }

    int scheduling_type = atoi(arr[1]);
    int num_threads = atoi(arr[2]);
    int cnt = 0;
    double start_time = omp_get_wtime();

    #pragma omp parallel num_threads(num_threads)
    {
        if(scheduling_type == 1){
            #pragma omp for schedule(static) reduction(+:cnt)
            for (int i=1; i<=200000; i++) {
                if(is_prime(i)) cnt++;
            }
        }
        else if(scheduling_type == 2){
            #pragma omp for schedule(dynamic) reduction(+:cnt)
            for(int i=1; i<=200000; i++) {
                if(is_prime(i)) cnt++;
            }
        }
        else if(scheduling_type == 3){
            #pragma omp for schedule(static, 10) reduction(+:cnt)
            for(int i=1; i<=200000; i++) {
                if(is_prime(i)) cnt++;
            }
        }
        else if(scheduling_type == 4){
            #pragma omp for schedule(dynamic, 10) reduction(+:cnt)
            for(int i=1; i<=200000; i++) {
```

```
                if(is_prime(i)) cnt++;
            }
        }
    }

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

    printf("Number of primes: %d\n", cnt);
    printf("Execution time: %lf ms\n", execution_time);

    return 0;
}
```

6-13 Line: This function is used to determine prime numbers.
16-22 Line: This is the code to input the schedule type number and thread number to receive input to the terminal according to the presented requirements.
Line 26-52: Code for parallel computing using openMP, written according to the schedule type number. 'reduction(+:cnt)' prevents one thread from calculating all primes.
Line 54-60: This is the code for outputting the execution time and the number of prime numbers.

When I entered the program result time, the time came out too tight, so I wondered if it was a code problem, so I tried running it on a 2-core laptop, but the execution time didn't come out exactly. So I thought that the reason for this was because the environment of the computer that executed the code was 16 cores and the RAM was 32.0GB. Now let's try to analyze the results. As for the execution time when the chunk size is default, dynamic scheduling took longer, and for the execution time when the chunk size was 10, static scheduling took longer when the number of threads was 6, and for the number of threads thereafter, the execution time was similar to that of dynamic scheduling. it took I thought that the execution time of dynamic scheduling would be much shorter than static scheduling, but it took similar or longer time, so I think the reason for this is the overhead of chunk size or load balancing. Dynamic scheduling dynamically allocates work units to each thread. At this time, calculations are required to determine the size of work units, which can cause overhead. While dynamically allocating work units, the work can be evenly distributed to each thread, but this result is thought to be because an imbalance between tasks can occur, which can cause overhead in dynamically allocating work units.
If the number of threads is less than 8 when the chunk size is 10, an imbalance in task distribution may occur in static scheduling in which tasks are assigned as

much as the specified chunk size due to the small chunk size. In addition, since the number of prime numbers to be calculated is large, an unbalanced workload among threads may increase execution time. If the number of threads is 8 or more, it is considered that the reason why the execution time of the two scheduling is similar is that the size of the task is sufficiently small that the imbalance of task distribution does not affect it.

In the case of performance, dynamic scheduling has the worst performance because it is the reciprocal of execution time, and when the chunk size is 10, the two scheduling shows similar performance.