Project #3

# Multicore Computing

## Problem 1

| | |
|---|---|
| Date | Jun 13, 2020 |
| Instructor | Bongsoo Sohn |
| Std. Name | Junhyuck Woo |
| Std. ID | 20145337 |

# INDEX

## ENVIRONMENT

**Hardware**

- MacBook Pro (15-inch, 2017)
- Processor: 2.8 GHz Quad-Core Intel Core i7
- Memory: 16GB 2133 MHz LPDDR3

**Operating System**

- macOS Catalina, ver: 10.15.4

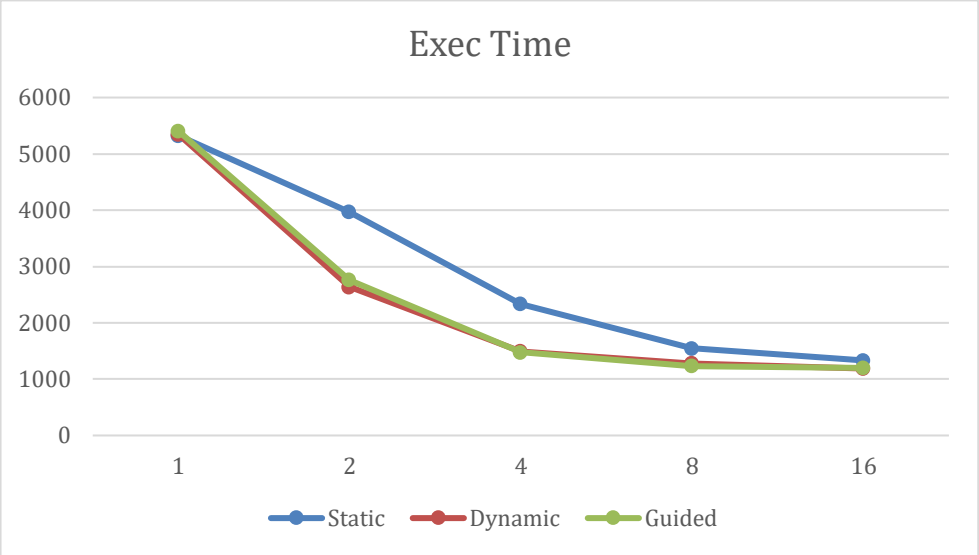
**IDE (Integrated Development Environment)**

- Visual Studio Code 1. 45.1
- gcc version 8.4.0 (Homebrew GCC 8.4.0_1)

**Testing Environment**

- iTerm2
- Build 3.3.9
- openjdk 14.0.1 2020-04-14
  OpenJDK Runtime Environment (build 14.0.1+7)
  OpenJDK 64-Bit Server VM (build 14.0.1+7, mixed mode, sharing)

## TABLE & GRAPH




▲ Fig. 1. Exec Time of multi-thread programming with static, dynamic and guided approach

| Exec Time | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| Static | 5325.252 | 3969.766 | 2332.277 | 1547.213 | 1332.913 |
| Dynamic | 5349.416 | 2634.877 | 1491.884 | 1282.083 | 1188.843 |
| Guided | 5405.115 | 2757.857 | 1478.695 | 1230.187 | 1199.511 |

▲ TABLE 1


▲ Fig. 2. Performance of multi-thread programming with static, dynamic and guided approach

| Performance (1/exec time) | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| Static | 0.00018778 | 0.0002519 | 0.00042877 | 0.00064632 | 0.00075024 |
| Dynamic | 0.00018694 | 0.00037952 | 0.00067029 | 0.00077998 | 0.00084115 |
| Guided | 0.00018501 | 0.0003626 | 0.00067627 | 0.00081288 | 0.00083367 |

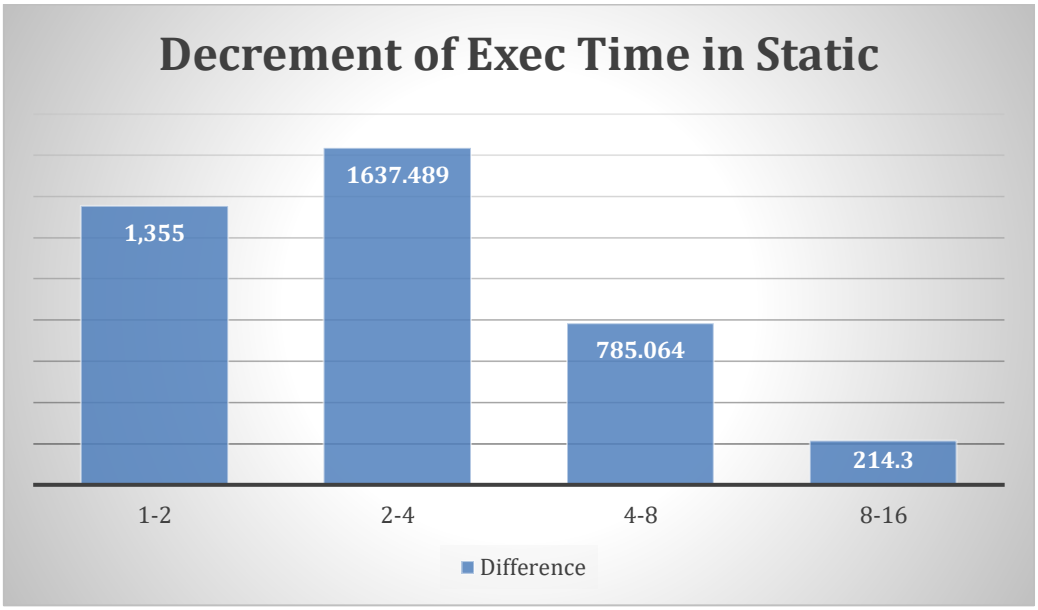
▲ TABLE 2

## EXPLANATION ON RESULT

In this problem, the given work is counting the prime number. The way checking whether the number is prime or not is to divide every number under the given one.

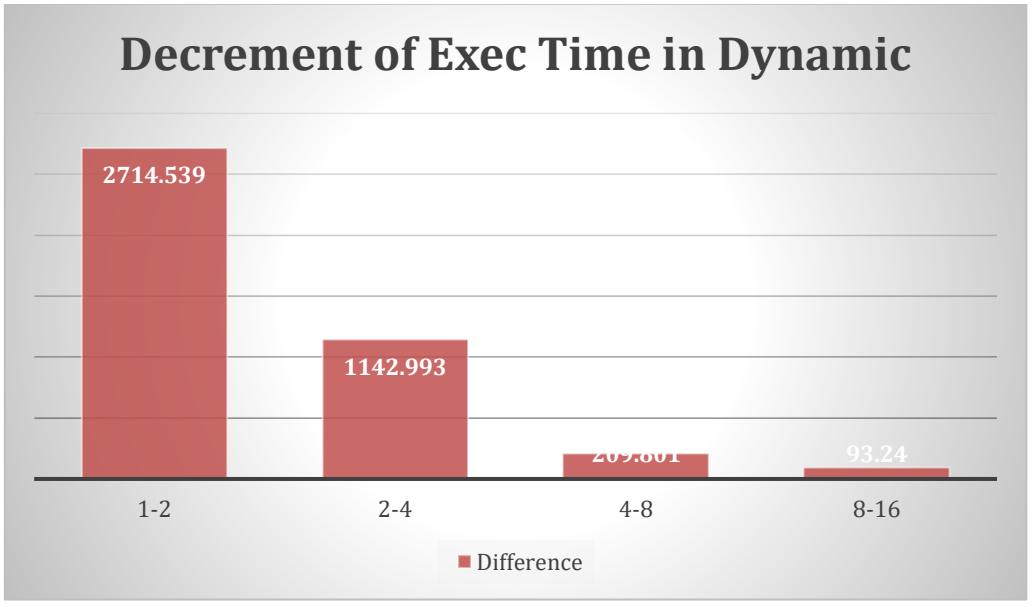
### 1. Exec Time

Fig. 1 shows the decrement tendency. The guided and dynamic approach tends to decrease larger than the static approach. To see the detail, I make 3 more figures.

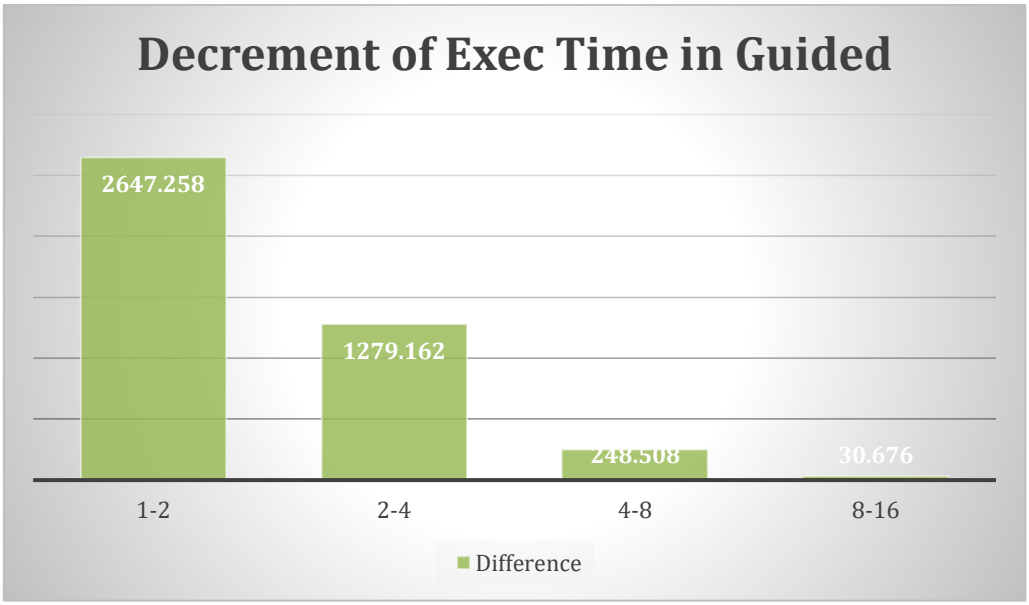



▲ Fig. 3. Decrement of execution time in the static approach

Fig. 3 shows how the execution time has decreased as the thread increases at the static load balancing approach. The task size of checking prime is greater for a big number than for a small number. In the static approach, the threads are assigned numbers in order, so the thread which receives work late is assigned more work. Thus, as the number of threads increases, the task is allocated relatively appropriately, resulting in a relatively constant reduction in execution time.

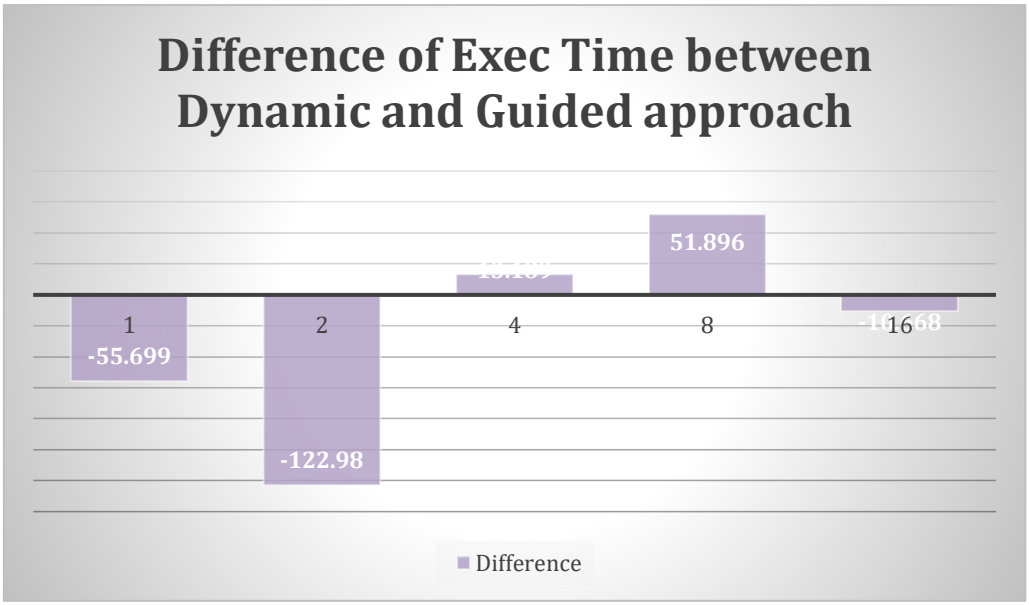



▲ Fig. 4. Decrement of execution time in the dynamic approach




▲ Fig. 5. Decrement of execution time in the guided approach

Fig. 4 and Fig. 5 shows how the execution time has decreased as the thread increases. The dynamic and guided approach gets work from the work queue. Fig4 and Fig5 show a significant reduction in run time as the initial thread increases. In addition, as the number of threads increases, the difference in execution time is reduced because of bottlenecks.




▲ Fig. 6. Difference of Exec Time between Dynamic and Guided approach

As mentioned earlier, dynamic and induced approaches use similar algorithms, so they tend to decrease execution time when increasing threads. For the guided approach, the chunk size is important. But I think the result is similar to the dynamic approach because the size is too small.

## 2. Performance

The performance evaluation criterion is 1/exe time. It means that performance and exec time have an inverse relationship. Fig 2 shows that guided and dynamic approaches produce similar performance, and the output is better than the static one. The explanation of these results is fully described in exec time.

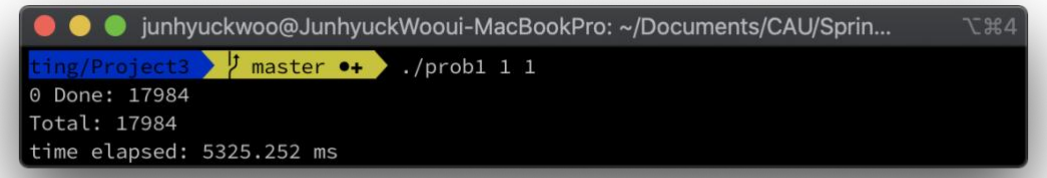
# SOURCE CODE

```
// Writer: Junhyuck Woo
// Lecture: Multicore Computing
// Organization: Chung-Ang University
// Deadline: Jun 20, 2020
// Project #3
//  - problem 1

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

int isPrime(int x);

int main(int argc, char *argv[]) {

    // Variables
    int i = 0;
    int tid = 0;
    int type = 0;
    int count = 0;
    int num_prime = 0;
    int num_thread = 0;
    double start_time, end_time;

    // Check the input argument
    if (argc != 3) {
        printf("ERROR: Wrong number of arguments\n");
        return 0;
    }

    // Check the schedule type
    type = atoi(argv[1]);
    if (type < 0 || type > 3) {
        printf("ERROR: Wrong schedule type\n");
        return 0;
    }

    // Set the thread number
    num_thread = atoi(argv[2]);
    omp_set_num_threads(num_thread);

    start_time = omp_get_wtime( );
    #pragma omp parallel private(count) private(tid)
    {
        count = 0;
        tid = omp_get_thread_num();

        // Static
        if (type == 1) {
            #pragma omp for schedule(static)
            for (i = 0; i < 200000; i++) {
                if (isPrime(i)) {
                    count++;
                }
            }
        }
        // Dynamic
        else if (type == 2) {
            #pragma omp for schedule(dynamic, 4)
            for (i = 0; i < 200000; i++) {
                if (isPrime(i)) {
                    count++;
                }
            }
        }
        // Guided
        else if (type == 3) {
            #pragma omp for schedule(guided, 4)
            for (i = 0; i < 200000; i++) {
                if (isPrime(i)) {
                    count++;
                }
            }
        }

        printf("%d Done: %d\n", tid, count);
        #pragma omp atomic
        num_prime += count;
    }
    /* implicit barrier */
    printf("Total: %d\n", num_prime);

    end_time = omp_get_wtime();
                printf("time elapsed: %.3lf ms\n", (end_time-
start_time)*1000);

    return 0;
}

int isPrime(int x) {
    int i;
    if (x<=1) {
        return 0;
    }
    for (i=2; i<x; i++) {
        if ((x%i == 0) && (i!=x)) {
            return 0;
        }
    }
    return 1;
}
```

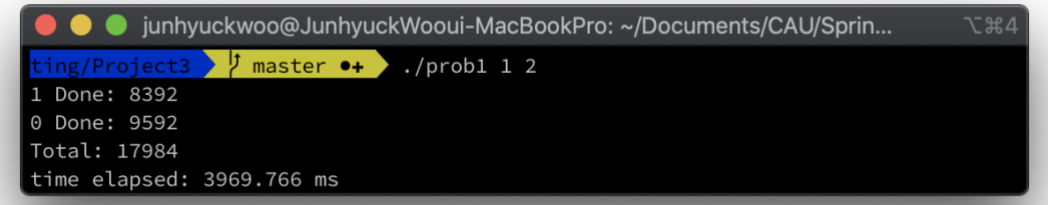
# OUTPUT

## 1. Static

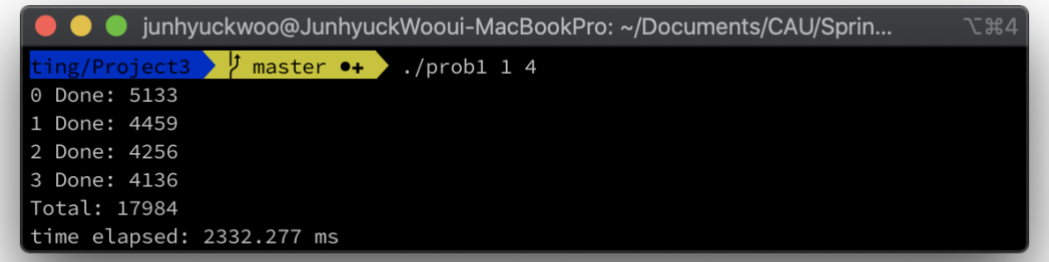
- Thread #1



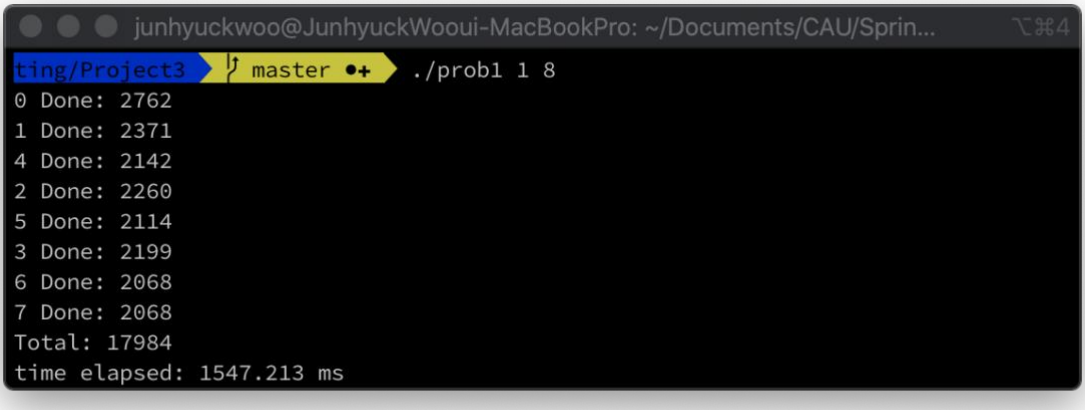

- Thread #2



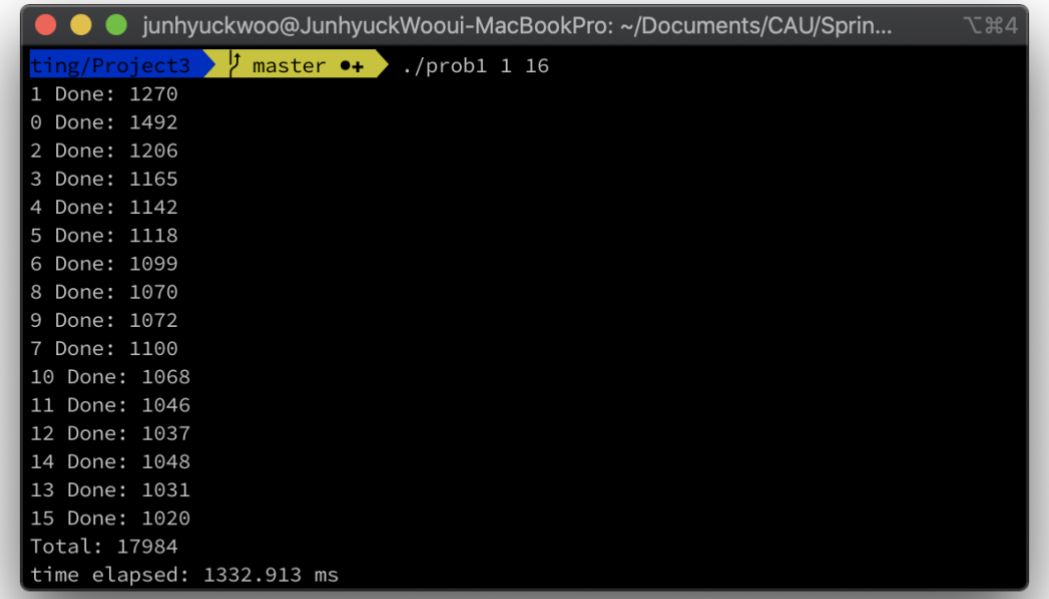

- Thread #4



- Thread #8

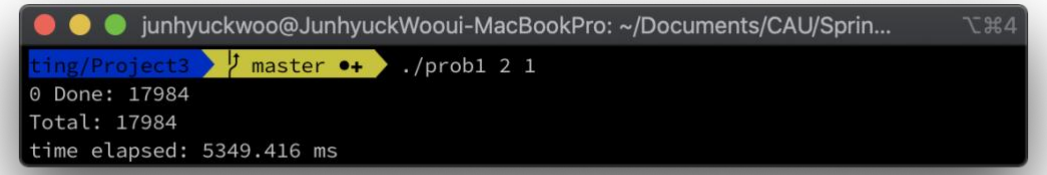



- Thread #16

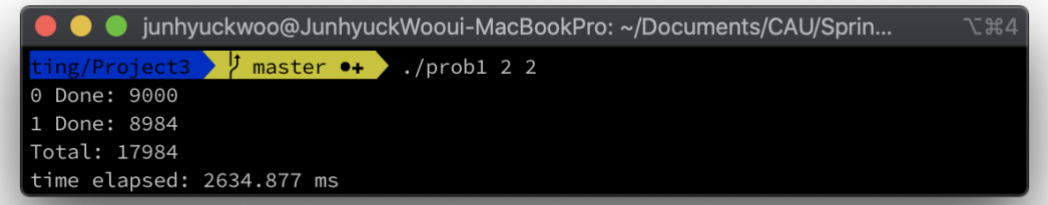


## 2. Dynamic

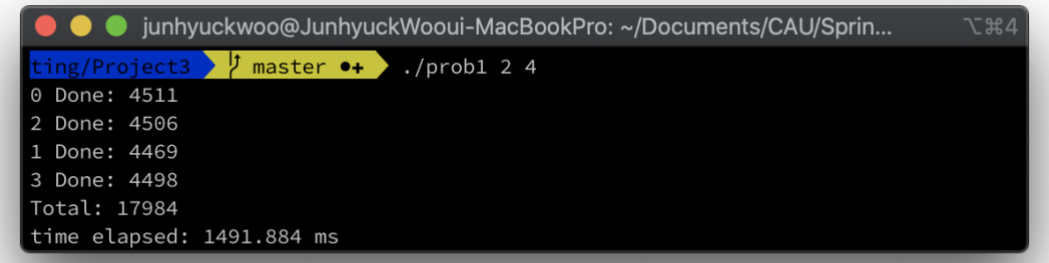
- Thread #1



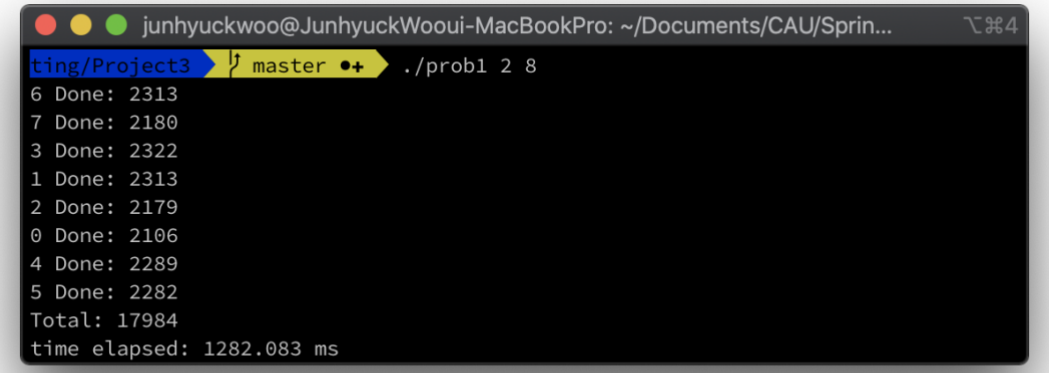

- Thread #2



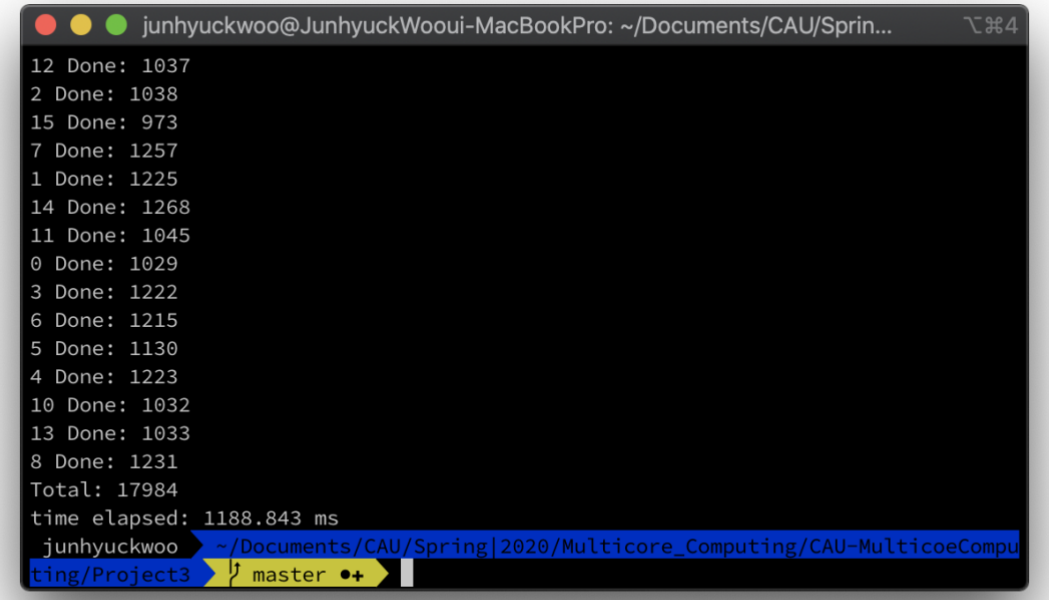

- Thread #4



- Thread #8

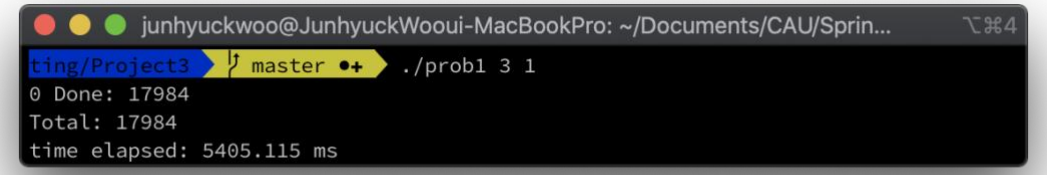



- Thread #16

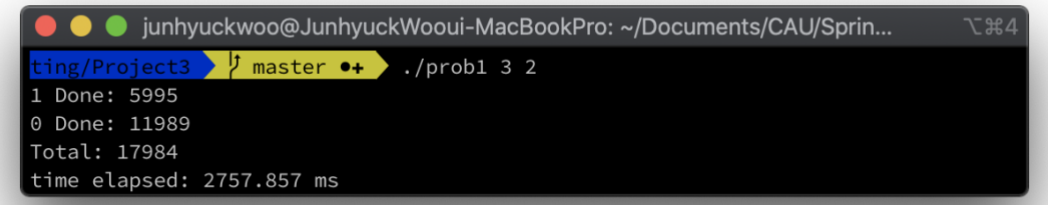


## 3. Guided

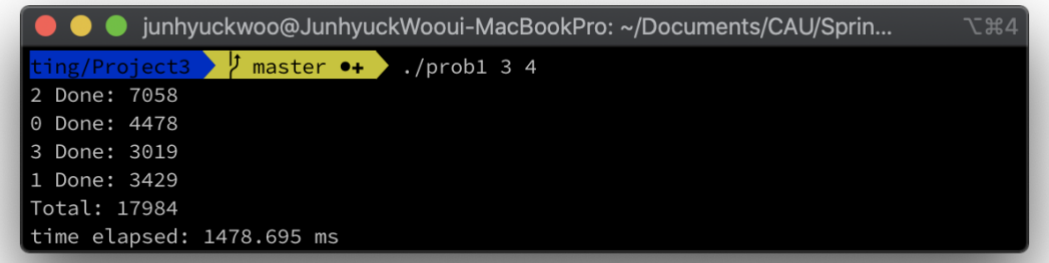
- Thread #1



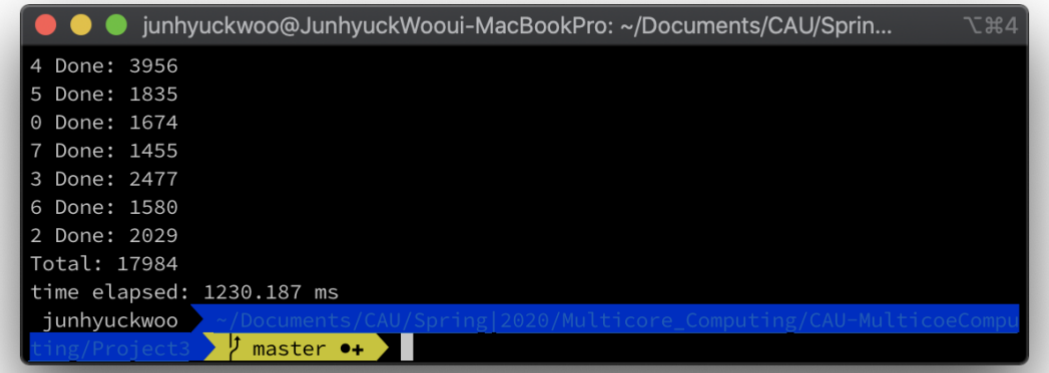

- Thread #2




- Thread #4



- Thread #8




- Thread #16