# Multicore proj3-prob2


| | |
|---|---|
| subject | multicore |
| professor | 손봉수 교수님 |
| date | 2023.05.20 |
| department | 소프트웨어대학 소프트웨어학부 |
| name | 20200641 임수현 |

## Environment

CPU : 11th Gen Intel Core(TM) i7-1195G7 //Quard Core
Core : 4
RAM : 16GB
System type : 64 bits
OS : Window 11
Run in WSL

## Source Code

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

long num_steps = 10000000;
double step;

void main (int argc, char *argv[]){
    long i; double x, pi, sum = 0.0;
    double start_time, end_time;

    int schedul_type = atoi(argv[1]);
    int chunk_size=atoi(argv[2]);
    int thread_num=atoi(argv[3]);


    step = 1.0/(double) num_steps;

    switch (schedul_type){
    case 1:
        start_time = omp_get_wtime();
        omp_set_num_threads(thread_num);
        {
            #pragma omp parallel for reduction(+:sum) private(x)
schedule(static,chunk_size)
                for (i=0;i< num_steps; i++){
                        x = (i+0.5)*step;
                        sum = sum + 4.0/(1.0+x*x);
            }

        }
        break;
    case 2:
        start_time = omp_get_wtime();
        omp_set_num_threads(thread_num);
        {
            #pragma omp parallel for reduction(+:sum) private(x)
schedule(dynamic,chunk_size)
                for (i=0;i< num_steps; i++){
                        x = (i+0.5)*step;
                        sum = sum + 4.0/(1.0+x*x);
            }

        }
        break;
```

```
    case 3:
        start_time = omp_get_wtime();
        omp_set_num_threads(thread_num);
        {
            #pragma omp parallel for reduction(+:sum) private(x)
schedule(guided,chunk_size)
                for (i=0;i< num_steps; i++){
                    x = (i+0.5)*step;
                    sum = sum + 4.0/(1.0+x*x);
            }
        }
        break;
    }
        pi = step * sum;
        end_time = omp_get_wtime();

        double timeDiff = end_time - start_time;
    printf("Execution Time : %lfms\n", timeDiff*1000);

        printf("pi=%.24lf\n",pi);
}
```

## code explain

After receiving the schedule type, each schedule type was executed using a switch statement. The number of threads inputted in the ***omp_set_num_thread*** was put as a parameter. In addition, the pi value was obtained by calculating through a for statement using ***#pragma comp parallel for reduction (+:sum) private(x) schedule (schedule type, chunk_size)***. The sum was set to ***public*** because all threads must be able to refer to it, and the x value to be used for each thread to calculate was set to ***private***.

# Result

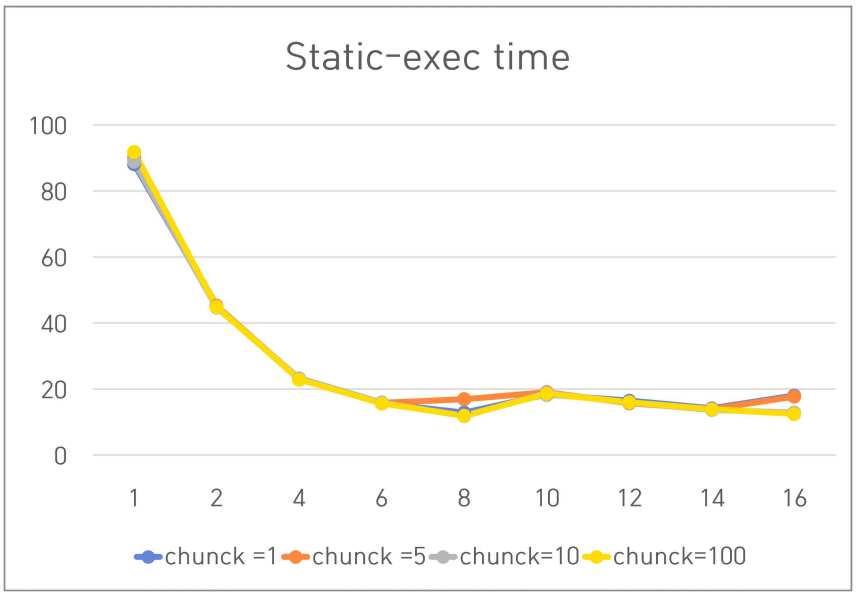
| execution time (msec) | chunk size | 1 | 2 | 4 | 6 |
|---|---|---|---|---|---|
| static | 1 | 88.07916 | 44.98113 | 23.08272 | 15.64471 |
| dynamic | | 141.6732 | 142.619 | 227.9872 | 214.9876 |
| guided | | 91.88717 | 44.55762 | 24.54581 | 16.08232 |
| static | 5 | 90.13424 | 45.18829 | 22.93662 | 15.79019 |
| dynamic | | 102.129 | 96.67987 | 62.77164 | 54.1078 |
| guided | | 88.76649 | 44.59407 | 22.79211 | 15.51427 |
| static | 10 | 88.7931 | 44.67053 | 23.25887 | 15.92089 |
| dynamic | | 94.12386 | 75.12677 | 39.7092 | 30.26232 |
| guided | | 88.52559 | 44.45776 | 22.78815 | 15.54352 |
| static | 100 | 91.75869 | 44.80331 | 22.84745 | 15.60365 |
| dynamic | | 88.43275 | 46.09615 | 24.54836 | 16.08396 |
| guided | | 88.11994 | 44.70774 | 22.89749 | 15.77961 |

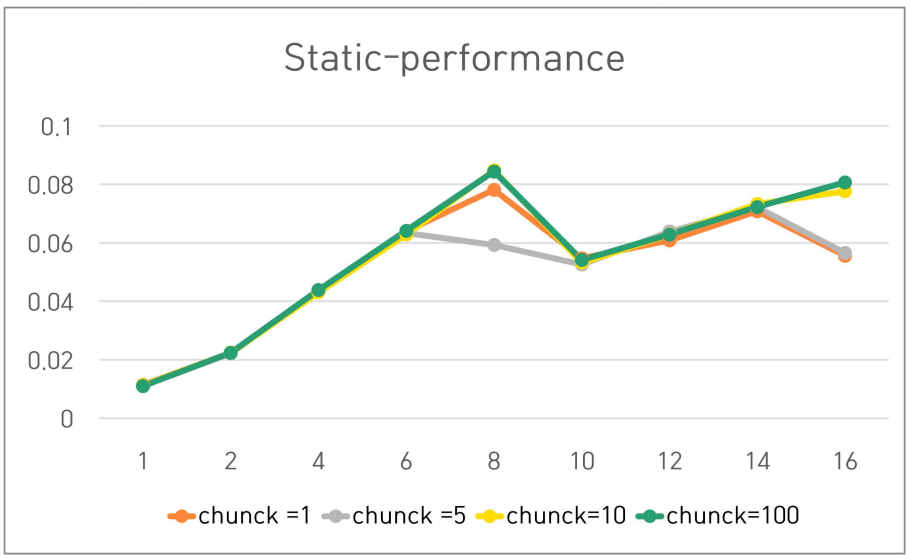
| execution time (msec) | chunk size | 8 | 10 | 12 | 14 | 16 |
|---|---|---|---|---|---|---|
| static | 1 | 12.8087 | 18.2988 | 16.4638 | 14.13903 | 18.00131 |
| dynamic | | 195.0464 | 204.5358 | 195.0464 | 194.0668 | 195.2742 |
| guided | | 12.69214 | 12.08618 | 12.15801 | 12.15558 | 12.36472 |
| static | 5 | 16.87883 | 19.01885 | 15.67211 | 13.86019 | 17.6868 |
| dynamic | | 45.75027 | 45.50876 | 45.55088 | 45.89917 | 46.59605 |
| guided | | 11.88803 | 12.02698 | 12.23279 | 11.96602 | 13.40366 |
| static | 10 | 11.79512 | 18.81129 | 15.92299 | 13.64646 | 12.88441 |
| dynamic | | 23.93501 | 24.50997 | 24.60219 | 24.6474 | 24.52228 |
| guided | | 11.96801 | 12.49555 | 12.04673 | 13.14405 | 12.40213 |
| static | 100 | 11.85371 | 18.5047 | 15.92591 | 13.85421 | 12.40443 |
| dynamic | | 12.05684 | 12.54629 | 14.44072 | 12.55766 | 12.56636 |
| guided | | 12.1033 | 12.30475 | 12.25909 | 14.13145 | 12.56208 |

| performance (1/exec time) | chunk size | 1 | 2 | 4 | 6 |
|---|---|---|---|---|---|
| static | 1 | 0.011353 | 0.022232 | 0.043322 | 0.063919 |
| dynamic | | 0.007059 | 0.007012 | 0.004386 | 0.004651 |
| guided | | 0.010883 | 0.022443 | 0.04074 | 0.06218 |
| static | 5 | 0.011095 | 0.02213 | 0.043598 | 0.06333 |
| dynamic | | 0.009792 | 0.010343 | 0.015931 | 0.018482 |
| guided | | 0.011266 | 0.022425 | 0.043875 | 0.064457 |
| static | 10 | 0.011262 | 0.022386 | 0.042994 | 0.062811 |
| dynamic | | 0.010624 | 0.013311 | 0.025183 | 0.033044 |
| guided | | 0.011296 | 0.022493 | 0.043882 | 0.064335 |
| static | 100 | 0.010898 | 0.02232 | 0.043769 | 0.064088 |
| dynamic | | 0.011308 | 0.021694 | 0.040736 | 0.062174 |
| guided | | 0.011348 | 0.022367 | 0.043673 | 0.063373 |

| performance (1/exec time) | chunk size | 8 | 10 | 12 | 14 | 16 |
|---|---|---|---|---|---|---|
| static | 1 | 0.078072 | 0.054648 | 0.060739 | 0.070726 | 0.055552 |
| dynamic | | 0.005127 | 0.004889 | 0.005127 | 0.005153 | 0.005121 |
| guided | | 0.078789 | 0.082739 | 0.08225 | 0.082267 | 0.080875 |
| static | 5 | 0.059246 | 0.052579 | 0.063808 | 0.072149 | 0.056539 |
| dynamic | | 0.021858 | 0.021974 | 0.021953 | 0.021787 | 0.021461 |
| guided | | 0.084118 | 0.083146 | 0.081748 | 0.08357 | 0.074606 |
| static | 10 | 0.084781 | 0.05316 | 0.062802 | 0.073279 | 0.077613 |
| dynamic | | 0.04178 | 0.0408 | 0.040647 | 0.040572 | 0.040779 |
| guided | | 0.083556 | 0.080028 | 0.08301 | 0.07608 | 0.080631 |
| static | 100 | 0.084362 | 0.05404 | 0.062791 | 0.07218 | 0.080616 |
| dynamic | | 0.08294 | 0.079705 | 0.069249 | 0.079633 | 0.079578 |
| guided | | 0.082622 | 0.081269 | 0.081572 | 0.070764 | 0.079605 |

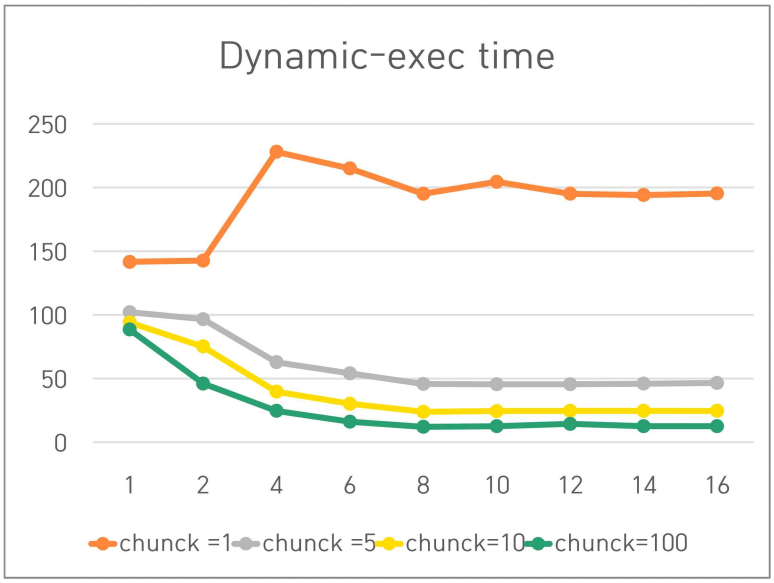
## Static scheduling

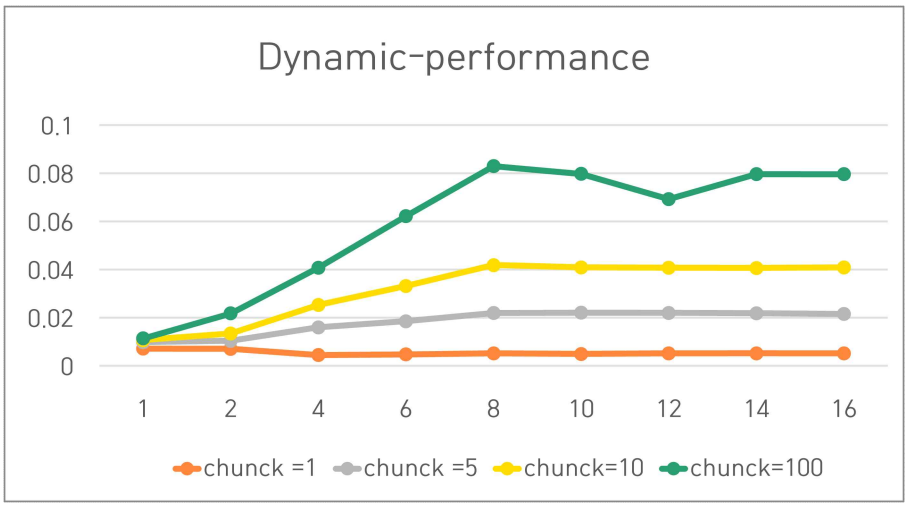



The performance is similar regardless of the chunk size because the calculation inside the for statement calculates a similar amount for each thread chunk. It can be seen that the workload is evenly distributed, so that load balancing between threads is maintained to some extent. And as the number of threads increases, the performance improves and then becomes constant. I think this is due to the occurrence of communication overhead. Therefore, increasing the amount of threads does not necessarily improve performance.

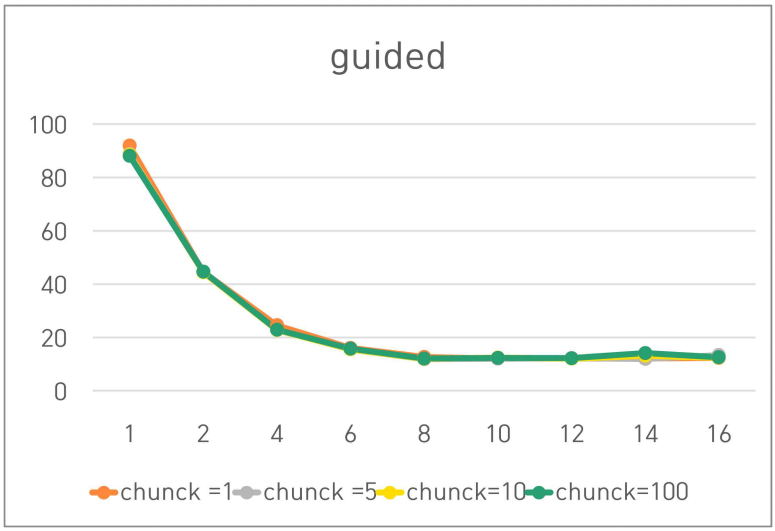
## Dynamic scheduling

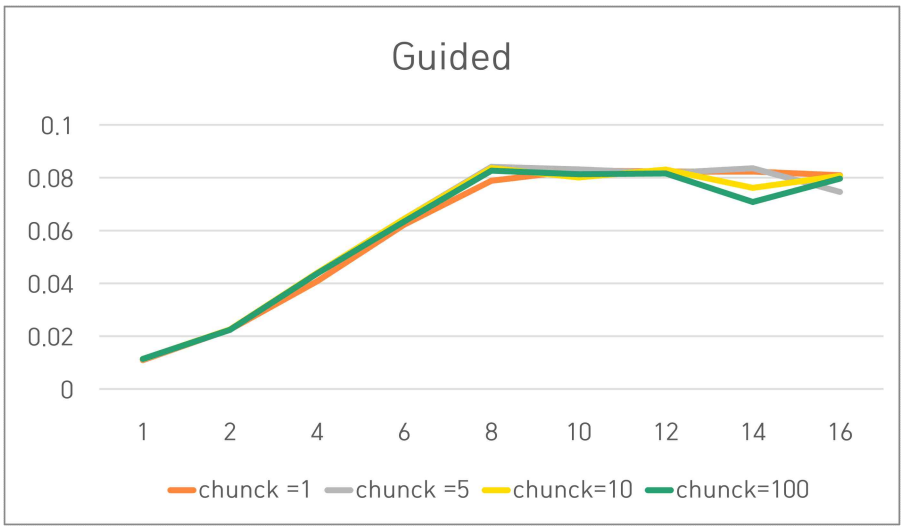



It can be seen that the performance is different depending on the chunk size. Smaller chunk size can cause overhead in allocating and adjusting tasks. So it can be seen that the smaller the chunk size, the lower the performance. And as the number of threads increases, the performance improves, and if there are more than 8, the performance does not increase. Like static, it can be seen that communication overhead occurs and performance is no longer increased.

## Guided scheduling




The guided method first starts with a large chunk size and then decreases the chunk size as it goes back, helping the thread that finished the calculation quickly to make the next calculation. Therefore, regardless of the chunk size, the exec time and performance result are the same. Even if the chunk size is small, all threads are busy at the beginning of the run, so the chunk size doesn't matter. However, the chunk size becomes smaller as it goes back, and it can be seen that the performance improves as the number of threads processed by the calculated thread increases. However, performance does not continue to increase. When the number of threads increases and the number of threads exceeds 8, the exec time is no longer reduced. This means that there is no difference in performance even if the number of threads increases due to communication overhead between threads.