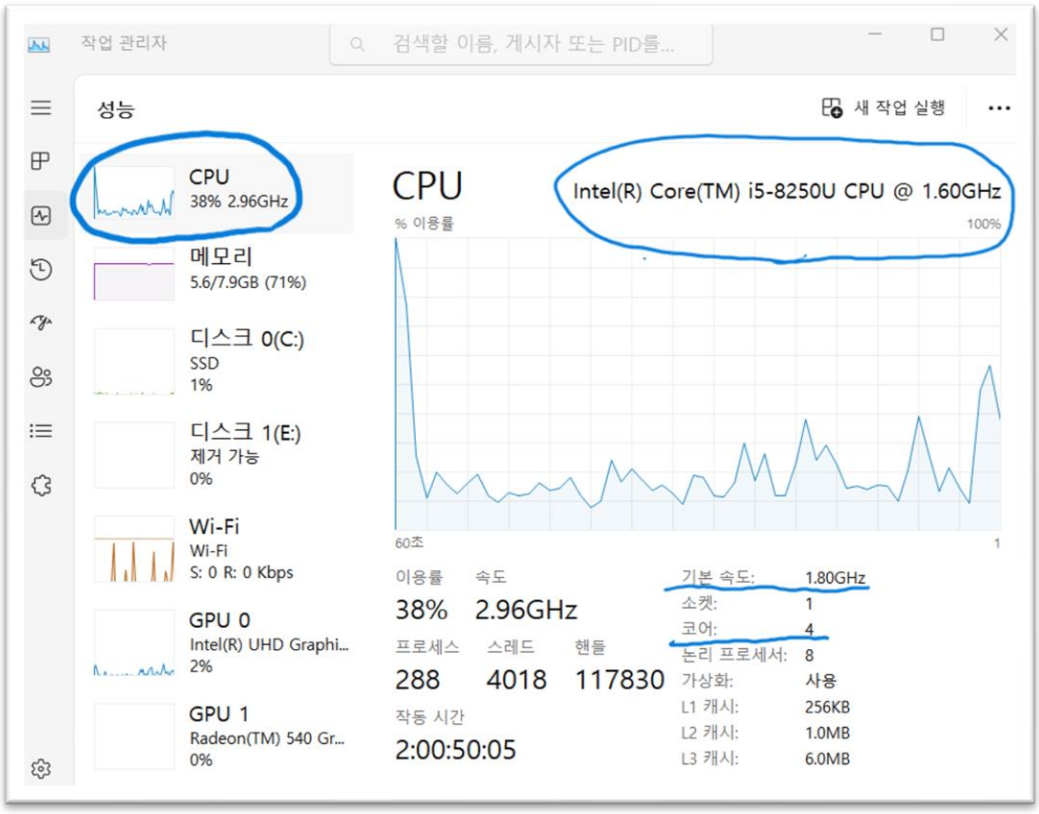
Problem2
Report

Student NO: 20183784
Student Name: 노현진

[Environment]

- **CPU**

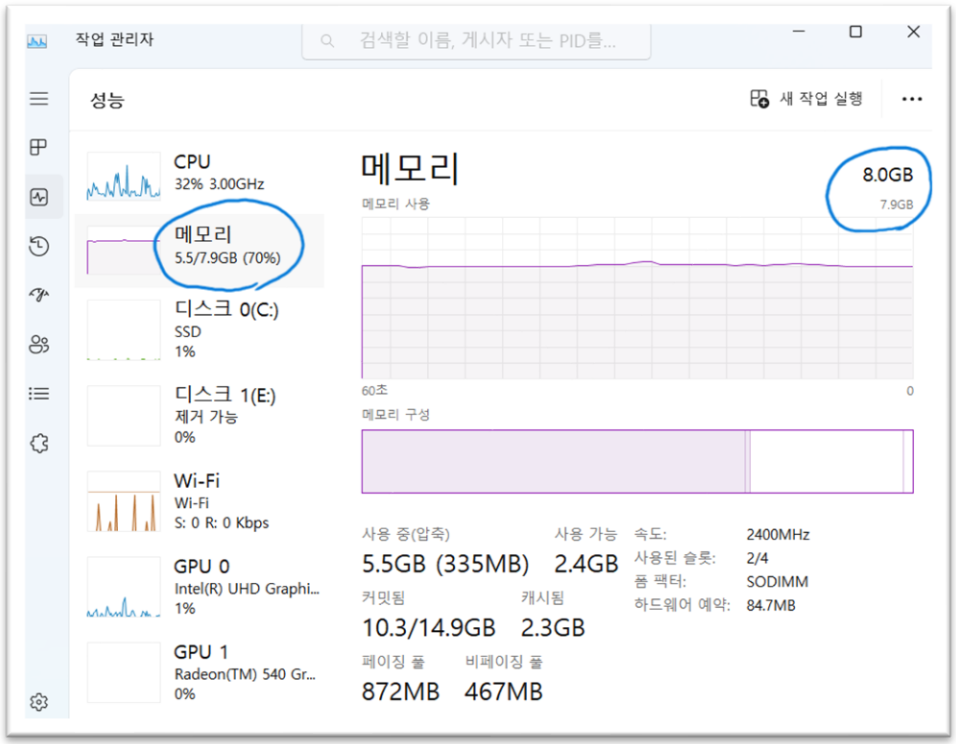


CPU type: Intel® Core™ i5-8250U CPU

Clock Speed: 1.80GHz

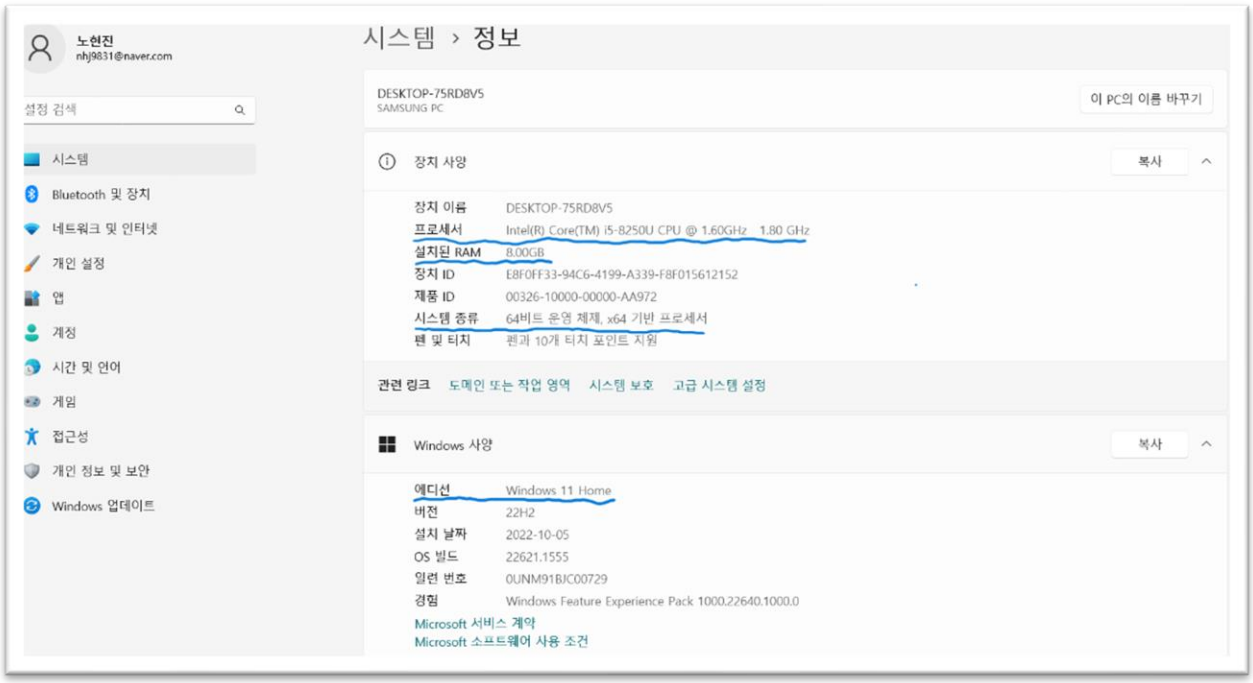
Number of cores: 4

- **Memory**



Memory size: 8.0GB

- **OS**



OS type: Windows 11

[Source Code]

- **prob2.c**

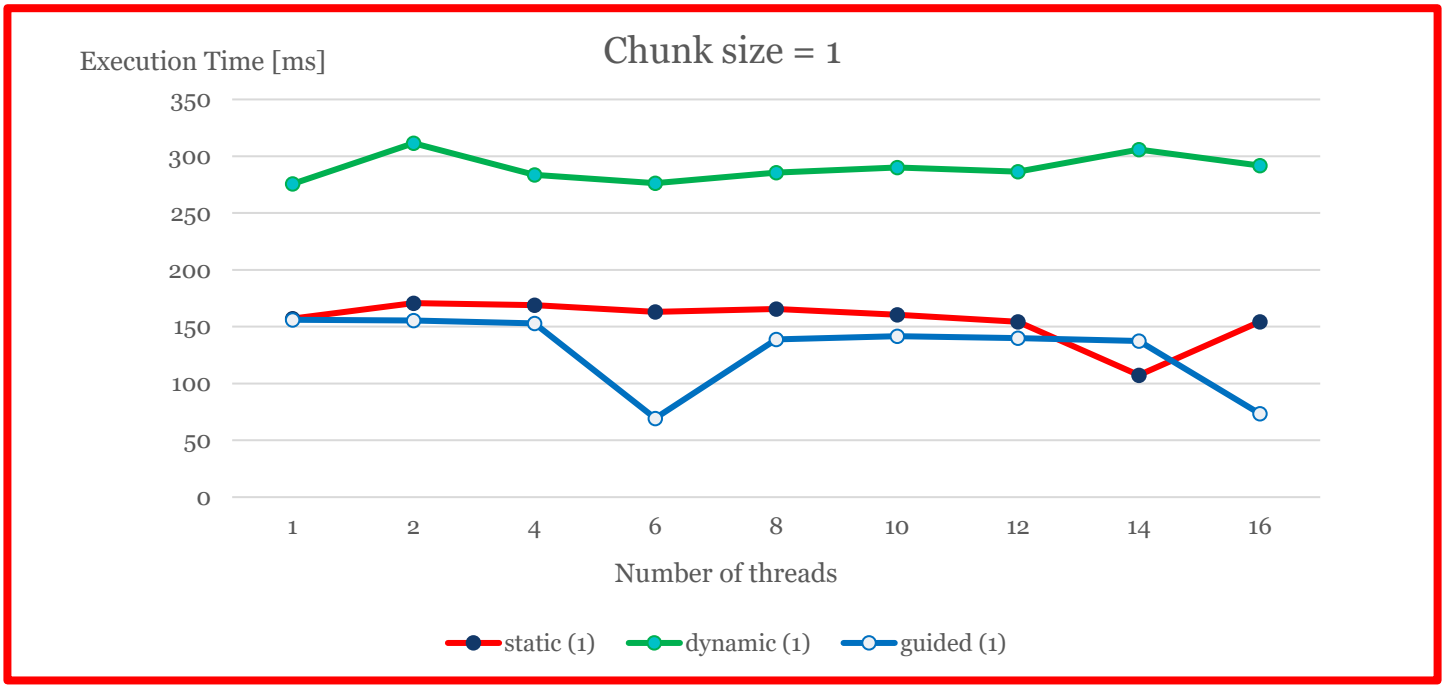
```c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_STEPS 10000000

void main(int argc, char *argv[]) {
	if (argc == 4) {
		/*
		* argv[1]: scheduling type number
		* 1: static
		* 2: dynamic
		* 3: guided
		*
		* argv[2]: chunk size
		* => 1, 5, 10, 100
		*
		* argv[3]: number of threads
		* => 1, 2, 4, 6, 8, 10, 12, 14, 16
		*/

		int scheduling_type_number = atoi(argv[1]);
		int chunk_size = atoi(argv[2]);
		int num_threads = atoi(argv[3]);
		long i;
		double step = 1.0 / (double) NUM_STEPS;
		double x, pi, sum = 0.0;
		double start_time, end_time;
		omp_set_num_threads(num_threads);
```

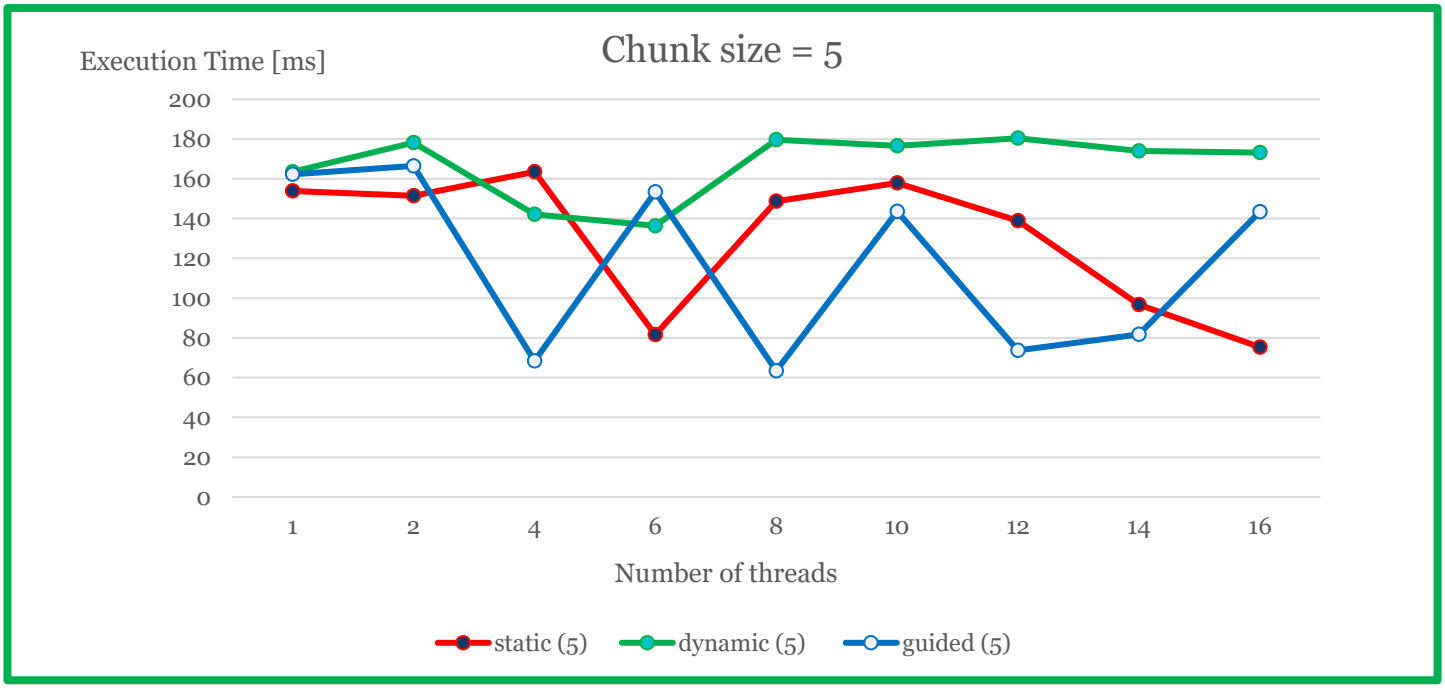
```
		start_time = omp_get_wtime();

		printf("Chunk size: %d₩n", chunk_size);
		printf("The number of threads: %d₩n", num_threads);

		switch (scheduling_type_number) {
			case 1:
				// static
				#pragma omp parallel for reduction (+:sum) schedule(static, chunk_size)
				for (i = 0; i < NUM_STEPS; i++) {
					x = (i + 0.5) * step;
					sum = sum + 4.0 / (1.0 + x * x);
				}

				pi = step * sum;

				end_time = omp_get_wtime();
				printf("pi = %.24lf₩n", pi);
				printf("The execution Time : %lfms₩n", end_time - start_time);
				break;
			case 2:
				// dynamic
				#pragma omp parallel for reduction (+:sum) schedule(dynamic, chunk_size)
				for (i = 0; i < NUM_STEPS; i++) {
					x = (i + 0.5) * step;
					sum = sum + 4.0 / (1.0 + x * x);
				}

				pi = step * sum;

				end_time = omp_get_wtime();
				printf("pi = %.24lf₩n", pi);
				printf("The execution Time : %lfms₩n", end_time - start_time);
				break;
			case 3:
				// guided
				#pragma omp parallel for reduction (+:sum) schedule(guided, chunk_size)
				for (i = 0; i < NUM_STEPS; i++) {
					x = (i + 0.5) * step;
					sum = sum + 4.0 / (1.0 + x * x);
				}

				pi = step * sum;

				end_time = omp_get_wtime();
				printf("pi = %.24lf₩n", pi);
				printf("The execution Time : %lfms₩n", end_time - start_time);
				break;
			default:
				printf("Scheduling type number should be 1, 2, or 3.₩n");
		}
	}
	else {
		printf("This program needs only three parameters.₩n");
	}
}
```

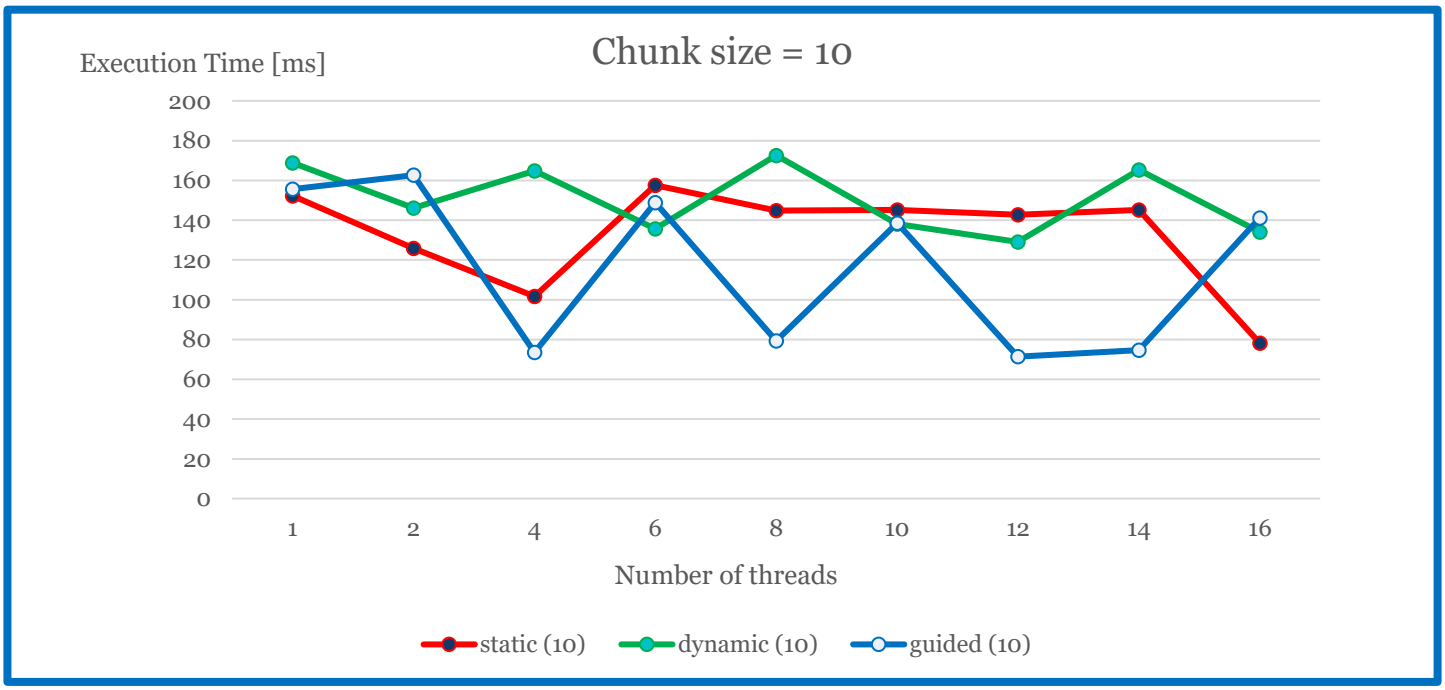
[Results]

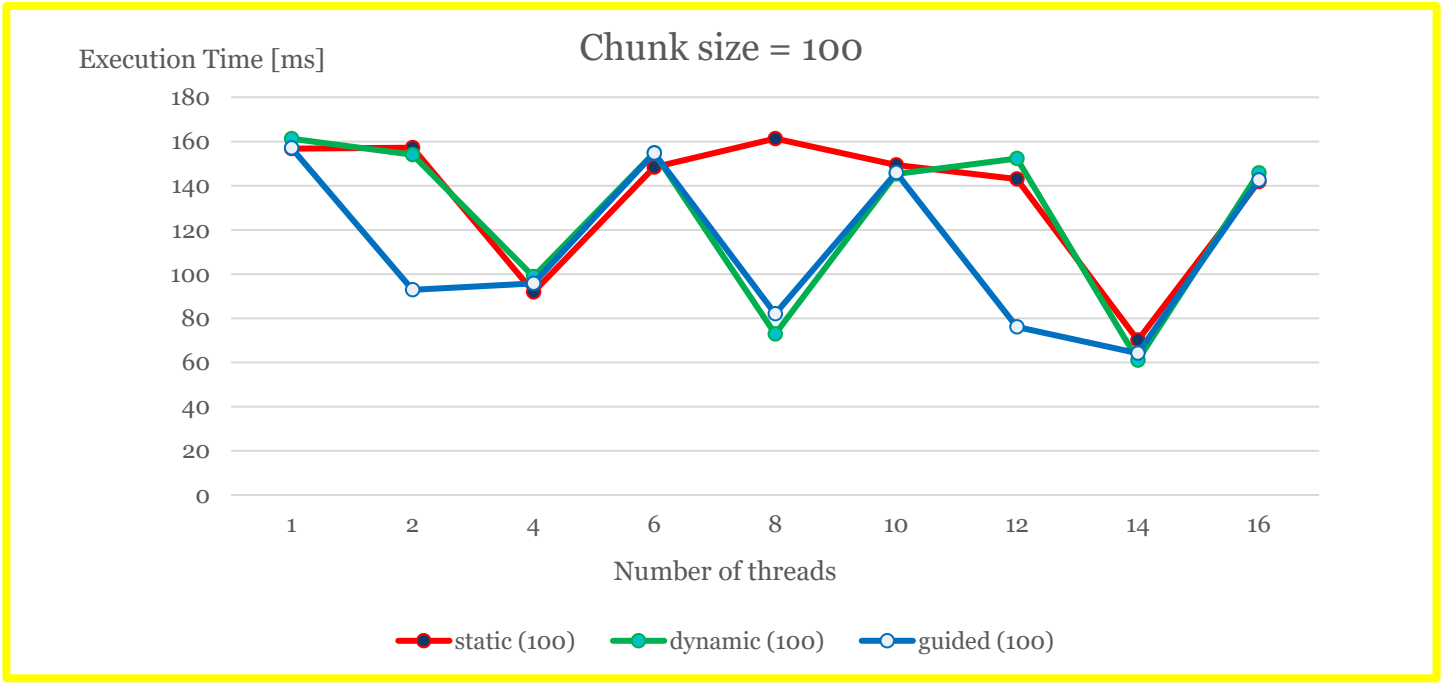
- **Execution Time**

Execution time (unit:ms)	Chunk Size	1	2	4	6	8	10	12	14	16
static	1	157.067	170.750	168.986	163.036	165.700	160.453	154.240	107.360	154.205
dynamic		275.678	311.475	283.562	276.219	285.548	290.200	286.356	305.809	291.712
guided		156.082	155.518	152.983	69.365	138.948	141.760	139.852	137.328	73.376
static	5	153.831	151.390	163.514	81.618	148.711	157.879	138.867	96.669	75.383
dynamic		163.591	178.118	142.086	136.382	179.642	176.504	180.392	173.991	173.116
guided		162.227	166.489	68.382	153.407	63.442	143.503	73.706	81.758	143.466
static	10	152.197	125.826	101.642	157.581	144.886	145.253	142.720	145.110	78.191
dynamic		168.866	146.063	164.891	135.730	172.622	138.050	129.074	165.337	134.013
guided		155.563	162.643	73.493	148.954	79.336	138.563	71.383	74.665	141.155
static	100	156.729	157.113	91.888	148.284	161.303	149.346	142.994	70.031	141.759
dynamic		161.240	153.951	98.794	154.846	72.815	145.257	152.219	61.015	145.712
guided		156.982	92.811	95.677	154.687	81.936	145.880	76.031	64.153	142.605



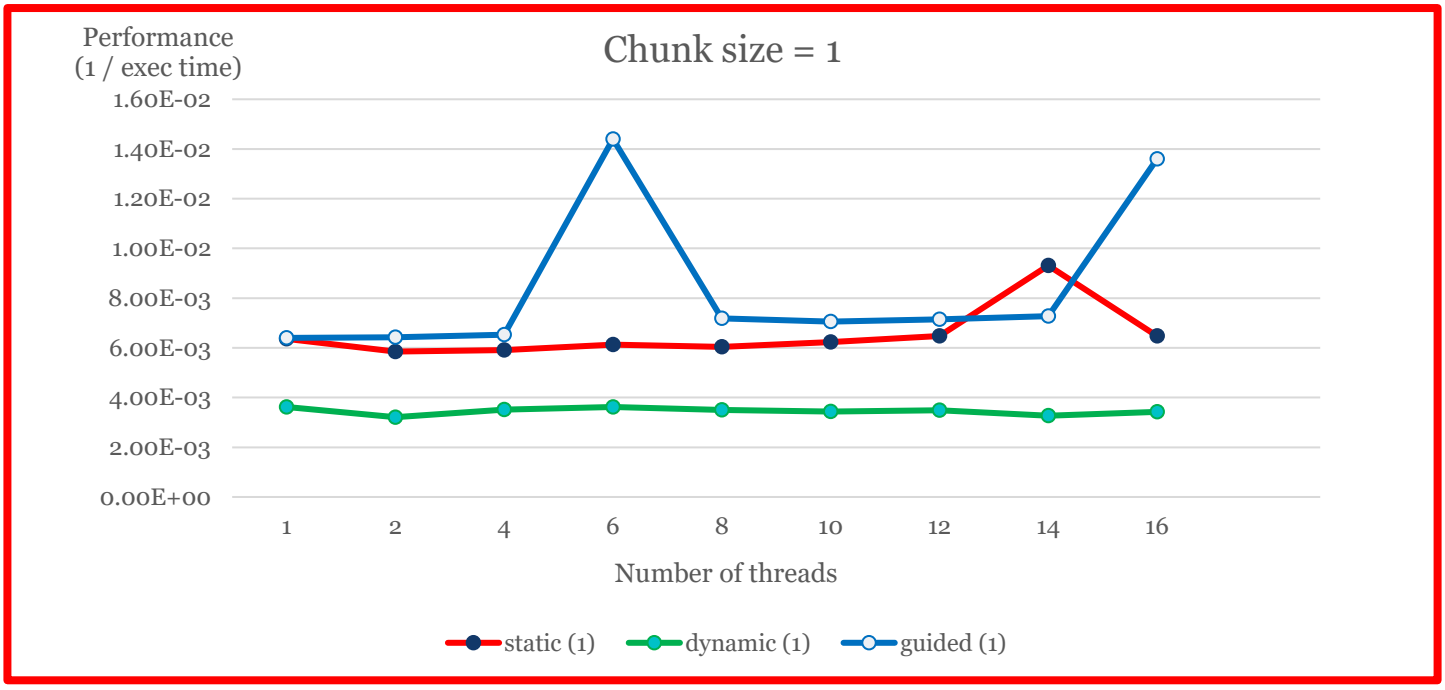

Chunk size = 5
Execution Time [ms]
200
180
160
140
120
100
80
60
40
20
0
1
2
4
6
8
10
12
14
16
Number of threads
static (5)
dynamic (5)
guided (5)

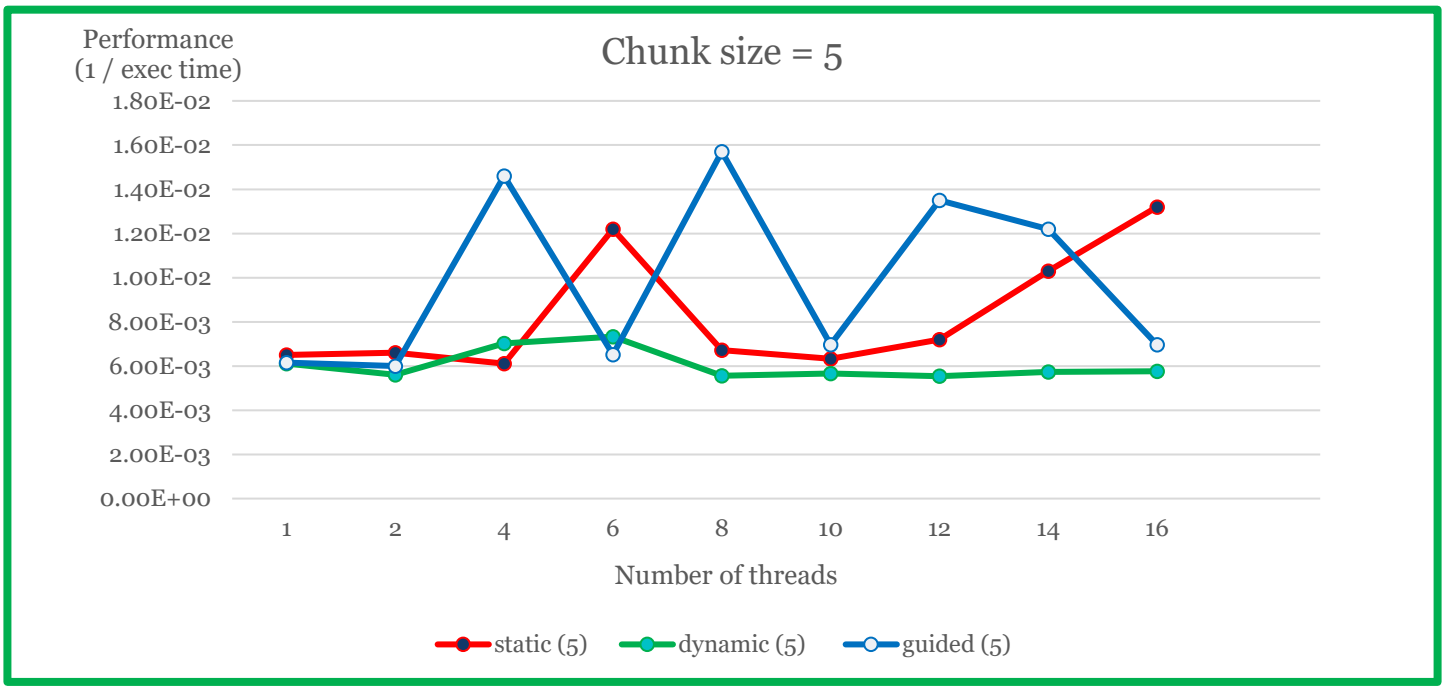

Chunk size = 10
Execution Time [ms]
200
180
160
140
120
100
80
60
40
20
0
1
2
4
6
8
10
12
14
16
Number of threads
static (10)
dynamic (10)
guided (10)

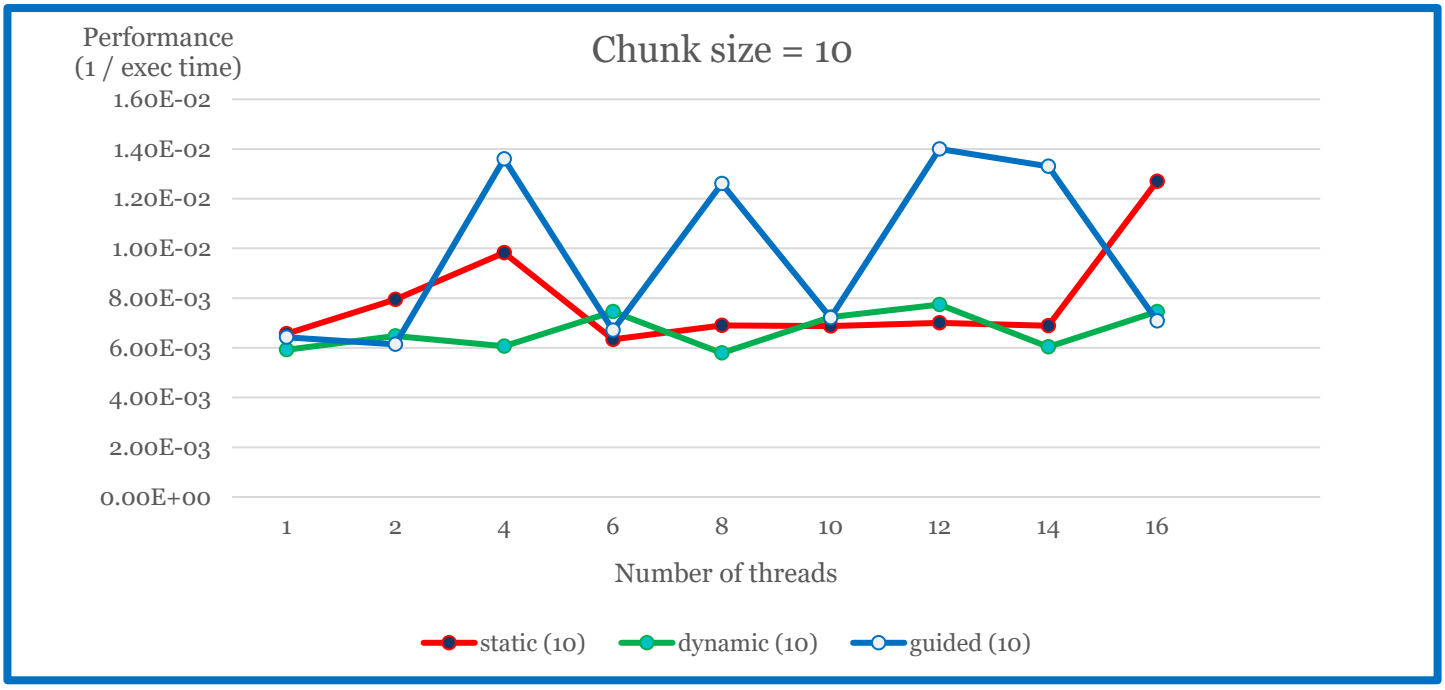


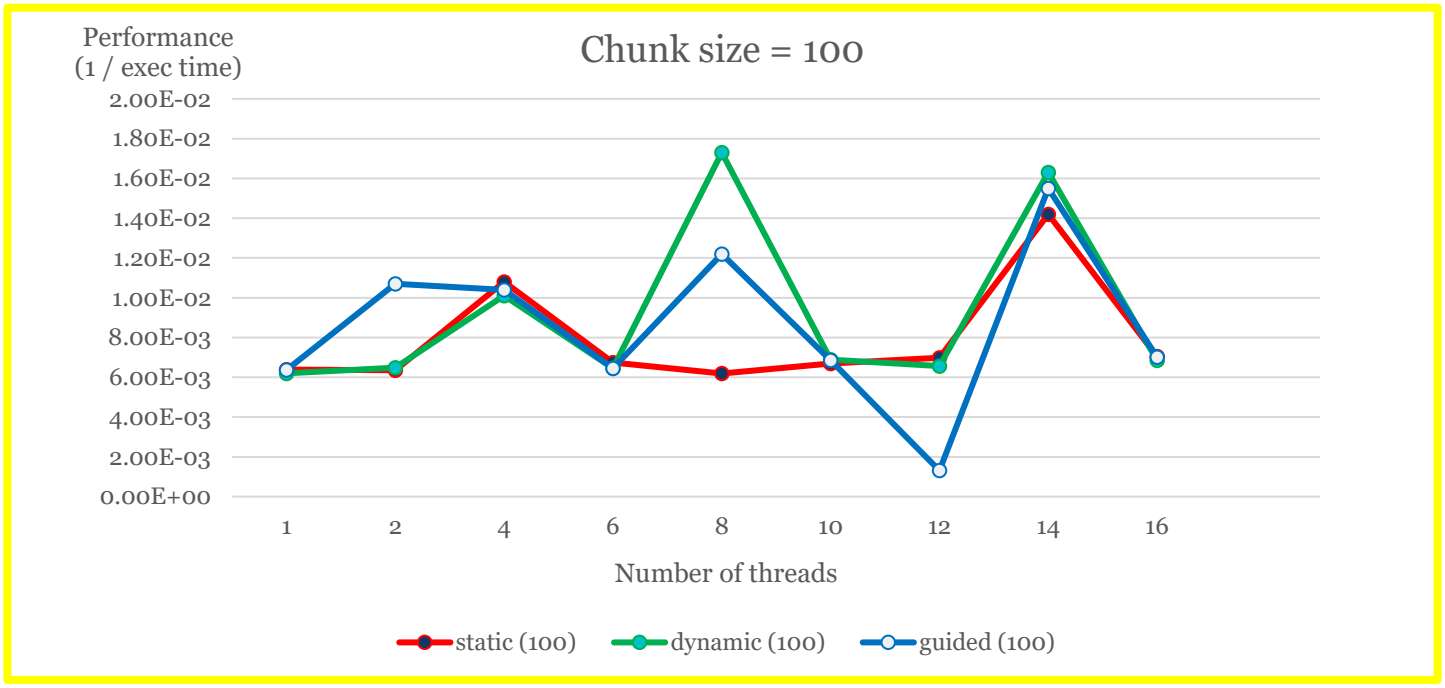
- **Performance**

performance (1/exec time)	Chunk Size	1	2	4	6	8	10	12	14	16
static	1	6.36e-3	5.85e-3	5.91e-3	6.13e-3	6.04e-3	6.23e-3	6.48e-3	9.31e-3	6.48e-3
dynamic		3.62e-3	3.21e-3	3.52e-3	3.62e-3	3.50e-3	3.44e-3	3.49e-3	3.27e-3	3.42e-3
guided		6.40e-3	6.43e-3	6.53e-3	1.44e-2	7.19e-3	7.05e-3	7.15e-3	7.28e-3	1.36e-2
static	5	6.50e-3	6.60e-3	6.11e-3	1.22e-2	6.72e-3	6.33e-3	7.20e-3	1.03e-2	1.32e-2
dynamic		6.11e-3	5.61e-3	7.03e-3	7.33e-3	5.56e-3	5.66e-3	5.54e-3	5.74e-3	5.77e-3
guided		6.16e-3	6.00e-3	1.46e-2	6.51e-3	1.57e-2	6.96e-3	1.35e-2	1.22e-2	6.97e-3
static	10	6.57e-3	7.94e-3	9.83e-3	6.34e-3	6.90e-3	6.88e-3	7.00e-3	6.89e-3	1.27e-2
dynamic		5.92e-3	6.84e-3	6.06e-3	7.46e-3	5.79e-3	7.24e-3	7.74e-3	6.04e-3	7.46e-3
guided		6.42e-3	6.14e-3	1.36e-2	6.71e-3	1.26e-2	7.21e-3	1.40e-2	1.33e-2	7.08e-3
static	100	6.38e-3	6.36e-3	1.08e-2	6.74e-3	6.19e-3	6.69e-3	6.99e-3	1.42e-2	7.05e-3
dynamic		6.20e-3	6.49e-3	1.01e-2	6.45e-3	1.37e-2	6.88e-3	6.56e-3	1.63e-2	6.86e-3
guided		6.37e-3	1.07e-2	1.04e-2	6.46e-3	1.22e-2	6.85e-3	1.31e-2	1.55e-2	7.01e-3


Chunk size = 1
Performance
(1 / exec time)
1.60E-02
1.40E-02
1.20E-02
1.00E-02
8.00E-03
6.00E-03
4.00E-03
2.00E-03
0.00E+00
1
2
4
6
8
10
12
14
16
Number of threads
static (1)
dynamic (1)
guided (1)


Chunk size = 5
Performance
(1 / exec time)
1.80E-02
1.60E-02
1.40E-02
1.20E-02
1.00E-02
8.00E-03
6.00E-03
4.00E-03
2.00E-03
0.00E+00
1
2
4
6
8
10
12
14
16
Number of threads
static (5)
dynamic (5)
guided (5)





[Explanation/Analysis on the Results]

The above results show that the guided approach is the best approach among all the methods. The execution of time and performance of guided approach is the best. Also, in case of small chunk size, dynamic approach is worst. If chunk size get larger, dynamic approach may be best approach.