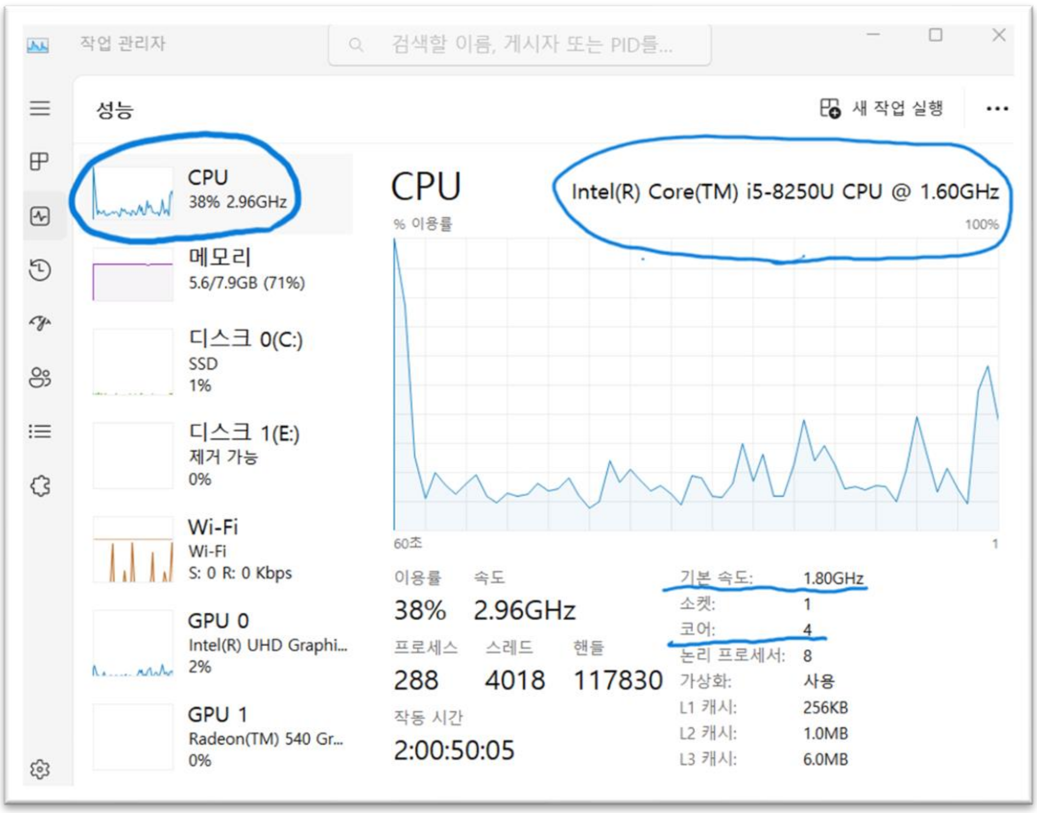
Problem1 Report

Student NO: 20183784
Student Name: 노현진

[Environment]

- CPU



작업 관리자 | 검색할 이름, 게시자 또는 PID를...

성능 | 새 작업 실행

CPU
38% 2.96GHz

메모리
5.6/7.9GB (71%)

디스크 0(C:)
SSD
1%

디스크 1(E:)
제거 가능
0%

Wi-Fi
Wi-Fi
S: 0 R: 0 Kbps

GPU 0
Intel(R) UHD Graphi...
2%

GPU 1
Radeon(TM) 540 Gr...
0%

CPU
Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz

% 이용률 100%

60초 1

이용률	속도	
38%	2.96GHz	
프로세스	스레드	핸들
288	4018	117830
작동 시간		
2:00:50:05		

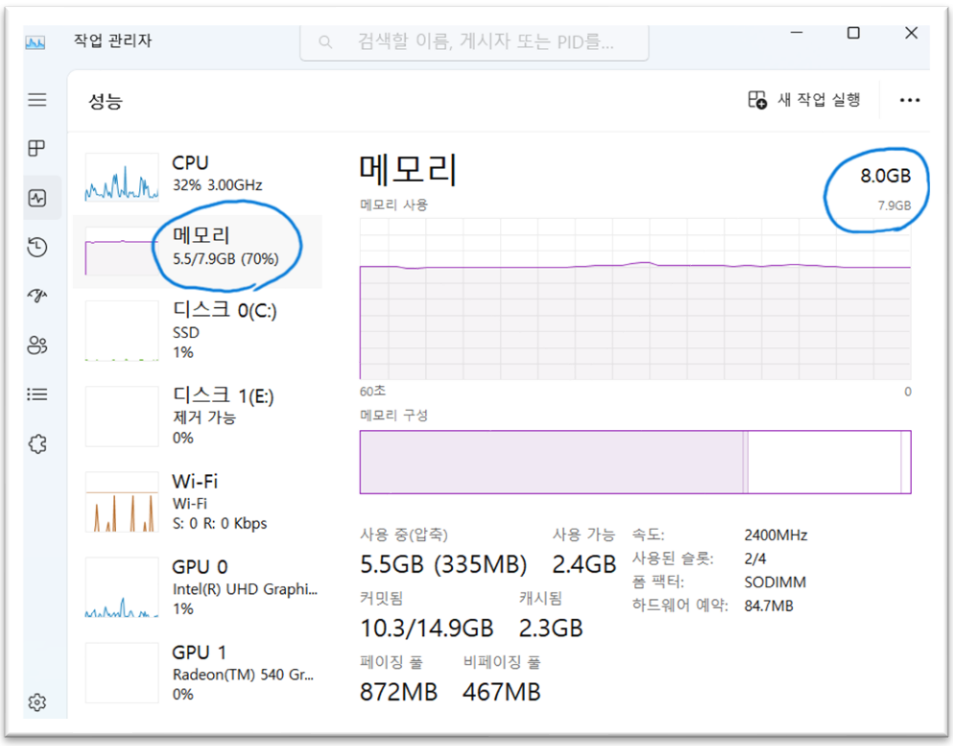
기본 속도:	1.80GHz
소켓:	1
코어:	4
논리 프로세서:	8
가상화:	사용
L1 캐시:	256KB
L2 캐시:	1.0MB
L3 캐시:	6.0MB

<u>**CPU type: Intel® Core™ i5-8250U CPU**</u>

<u>**Clock Speed: 1.80GHz**</u>

<u>**Number of cores: 4**</u>

- Memory



작업 관리자 | 검색할 이름, 게시자 또는 PID를...

성능 | 새 작업 실행

CPU
32% 3.00GHz

메모리
5.5/7.9GB (70%)

디스크 0(C:)
SSD
1%

디스크 1(E:)
제거 가능
0%

Wi-Fi
Wi-Fi
S: 0 R: 0 Kbps

GPU 0
Intel(R) UHD Graphi...
1%

GPU 1
Radeon(TM) 540 Gr...
0%

메모리 8.0GB

메모리 사용 7.9GB

60초 0

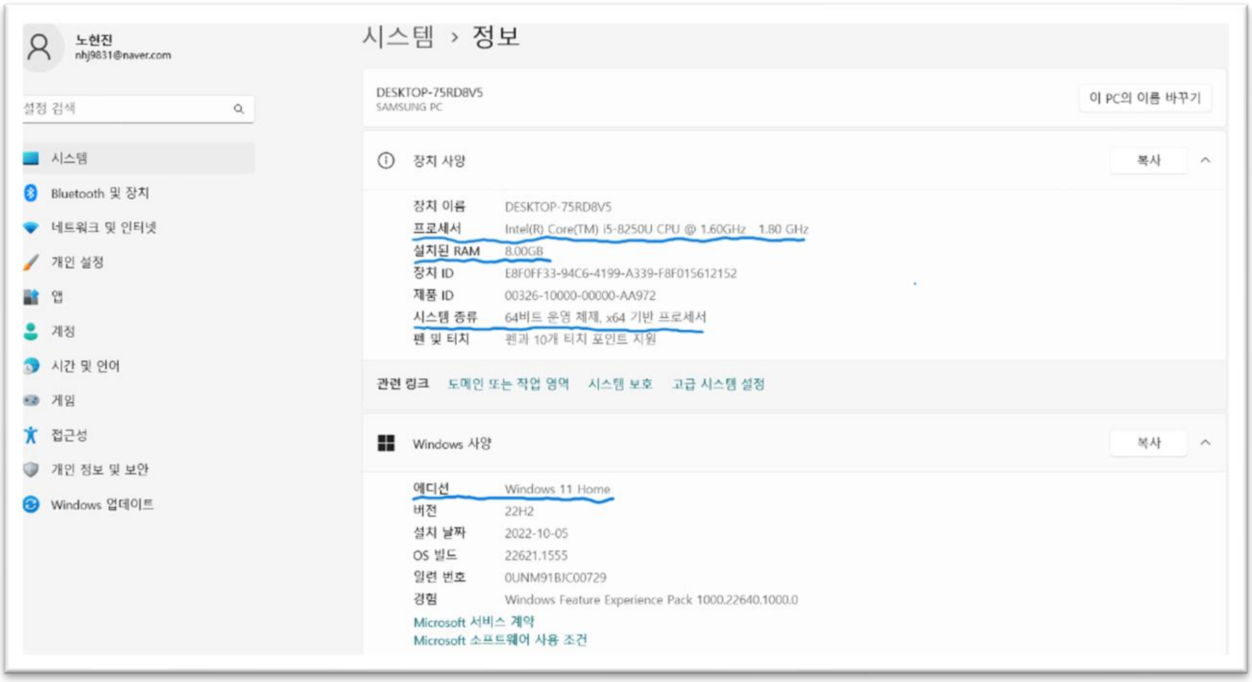
메모리 구성

사용 중(압축)	사용 가능
5.5GB (335MB)	2.4GB
커밋됨	캐시됨
10.3/14.9GB	2.3GB
페이징 풀	비페이징 풀
872MB	467MB

속도:	2400MHz
사용된 슬롯:	2/4
폼 팩터:	SODIMM
하드웨어 예약:	84.7MB

<u>**Memory size: 8.0GB**</u>

- **OS**



시스템 › 정보

DESKTOP-75RD8V5
SAMSUNG PC

장치 사양

장치 이름	DESKTOP-75RD8V5
프로세서	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
설치된 RAM	8.00GB
장치 ID	E8F0FF33-94C6-4199-A339-F8F015612152
제품 ID	00326-10000-00000-AA972
시스템 종류	64비트 운영 체제, x64 기반 프로세서
펜 및 터치	펜과 10개 터치 포인트 지원

관련 링크   도메인 또는 작업 영역   시스템 보호   고급 시스템 설정

Windows 사양

에디션	Windows 11 Home
버전	22H2
설치 날짜	2022-10-05
OS 빌드	22621.1555
일련 번호	0UNM91BJC00729
경험	Windows Feature Experience Pack 1000.22640.1000.0

Microsoft 서비스 계약
Microsoft 소프트웨어 사용 조건

OS type: Windows 11

[Source Code]

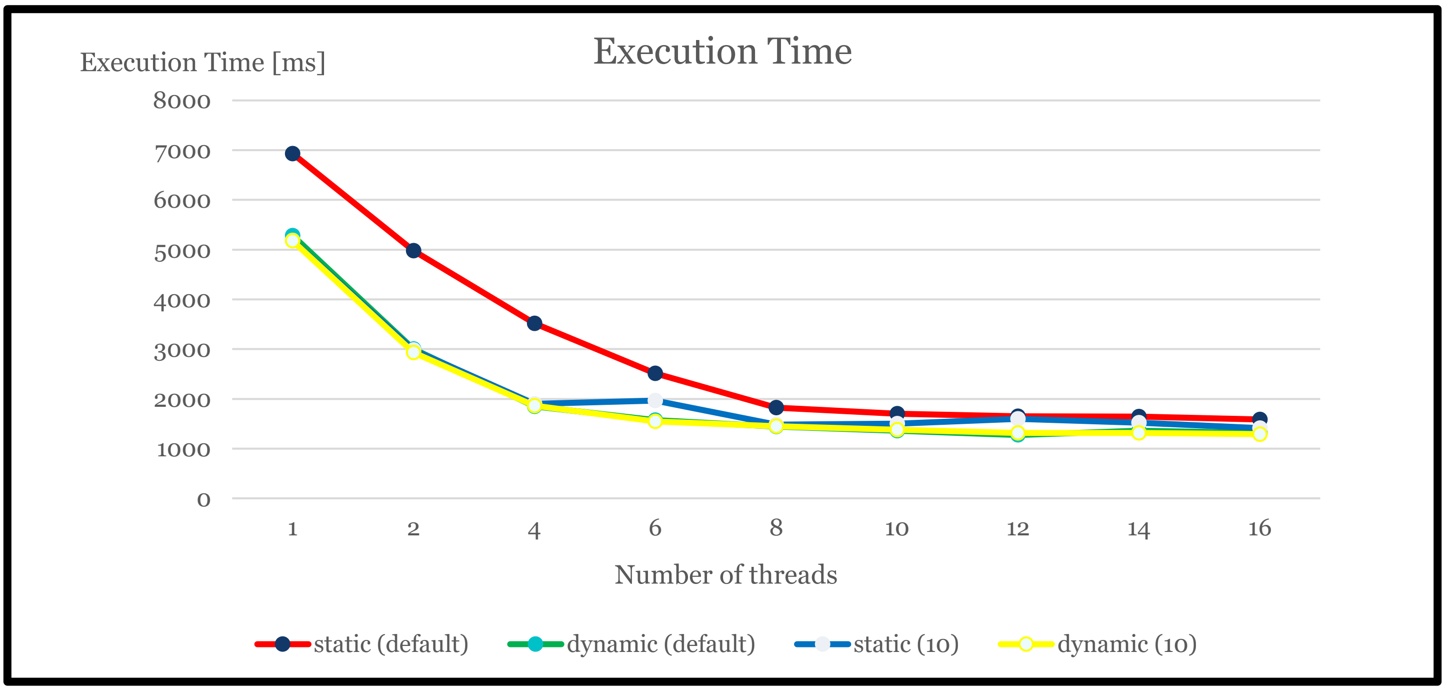
- **prob1.c**

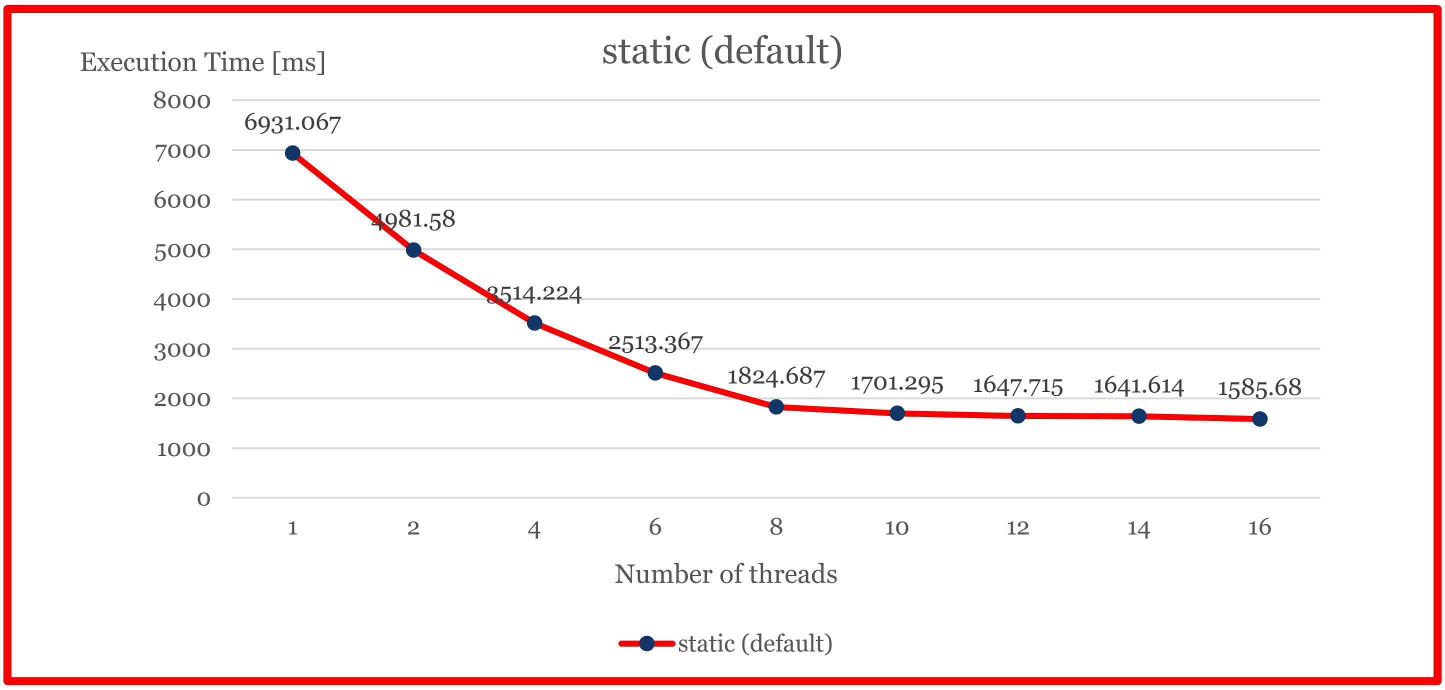
```
#include <omp.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

#define END_NUM 200000

bool isPrime(int x) {
        if (x <= 1) {
                return false;
        }
        for (int i = 2; i < x; i++) {
                if (x % i == 0) {
                        return false;
                }
        }
        return true;
}

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

                int scheduling_type_number = atoi(argv[1]);
                int num_threads = atoi(argv[2]);
                int i;
                int count = 0;
                double start_time, end_time;
                omp_set_num_threads(num_threads);
                start_time = omp_get_wtime();

                switch (scheduling_type_number) {
                        case 1:
                                // 1: static with default chunk size
                                #pragma omp parallel for reduction (+:count) schedule(static)
                                for (i = 1; i <= END_NUM; i++) {
                                        if (isPrime(i)) {
                                                count++;
                                        }
                                }
                                end_time = omp_get_wtime();

                                printf("The number of threads: %d₩n", num_threads);
                                printf("The number of 'prime numbers' from 1 to 200000: %d₩n", count);
                                printf("The execution time: %lfs₩n", end_time - start_time);
                                break;
                        case 2:
                                // 2: dynamic with default chunk
                                #pragma omp parallel for reduction (+:count) schedule(dynamic)
                                for (i = 1; i <= END_NUM; i++) {
                                        if (isPrime(i)) {
                                                count++;
                                        }
                                }
                                end_time = omp_get_wtime();

                                printf("The number of threads: %d₩n", num_threads);
                                printf("The number of 'prime numbers' from 1 to 200000: %d₩n", count);
                                printf("The execution time: %lfs₩n", end_time - start_time);
                                break;
                        case 3:
                                // 3: static with chunk size 10
                                #pragma omp parallel for reduction (+:count) schedule(static, 10)
                                for (i = 1; i <= END_NUM; i++) {
                                        if (isPrime(i)) {
                                                count++;
                                        }
                                }
                                end_time = omp_get_wtime();

                                printf("The number of threads: %d₩n", num_threads);
                                printf("The number of 'prime numbers' from 1 to 200000: %d₩n", count);
                                printf("The execution time: %lfs₩n", end_time - start_time);
                                break;
                        case 4:
                                // 4: dynamic with chunk size 10
                                #pragma omp parallel for reduction (+:count) schedule(dynamic, 10)
                                for (i = 1; i <= END_NUM; i++) {
                                        if (isPrime(i)) {
                                                count++;
                                        }
                                }
                                end_time = omp_get_wtime();

                                printf("The number of threads: %d₩n", num_threads);
                                printf("The number of 'prime numbers' from 1 to 200000: %d₩n", count);
                                printf("The execution time: %lfs₩n", end_time - start_time);
                                break;
                            default:
                                printf("Scheduling type number should be 1, 2, 3, or 4.₩n");
                        }
                    }
                    else {
                        printf("This program needs only two parameters₩n");
                    }
                }
```

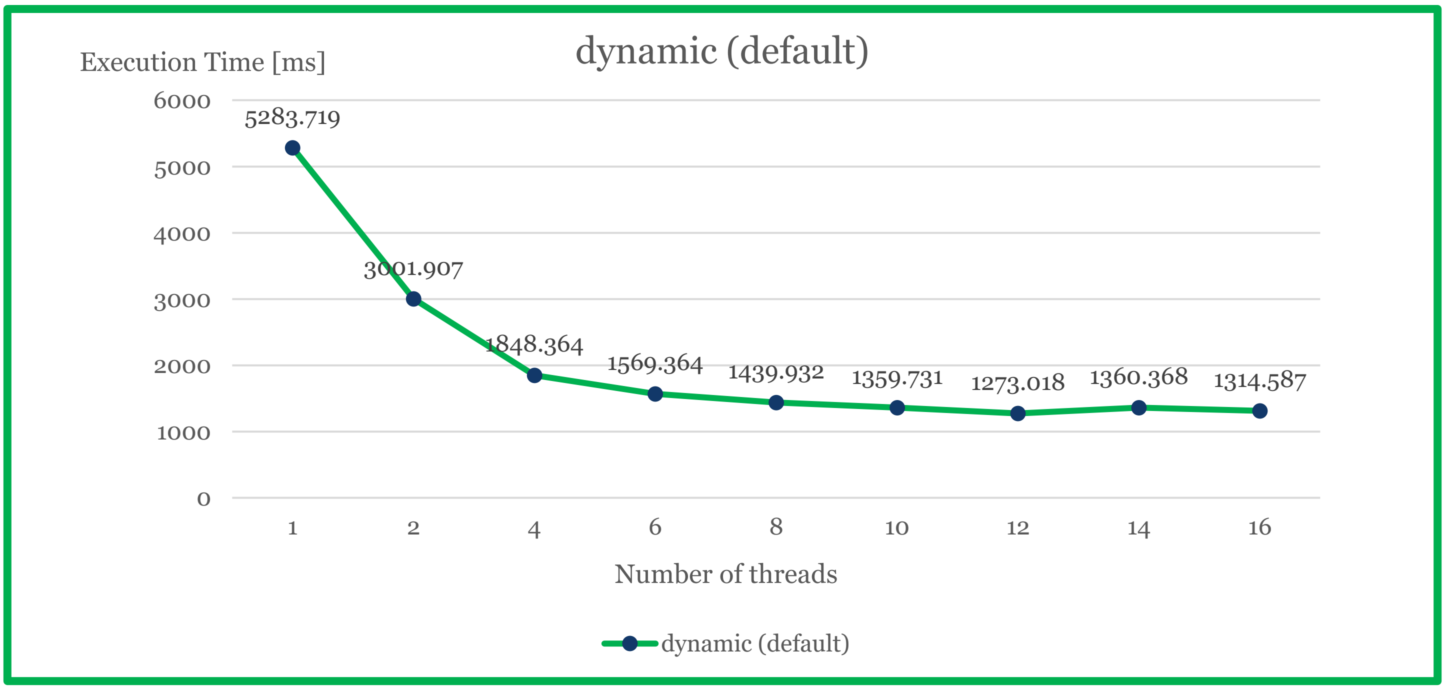
[Results]

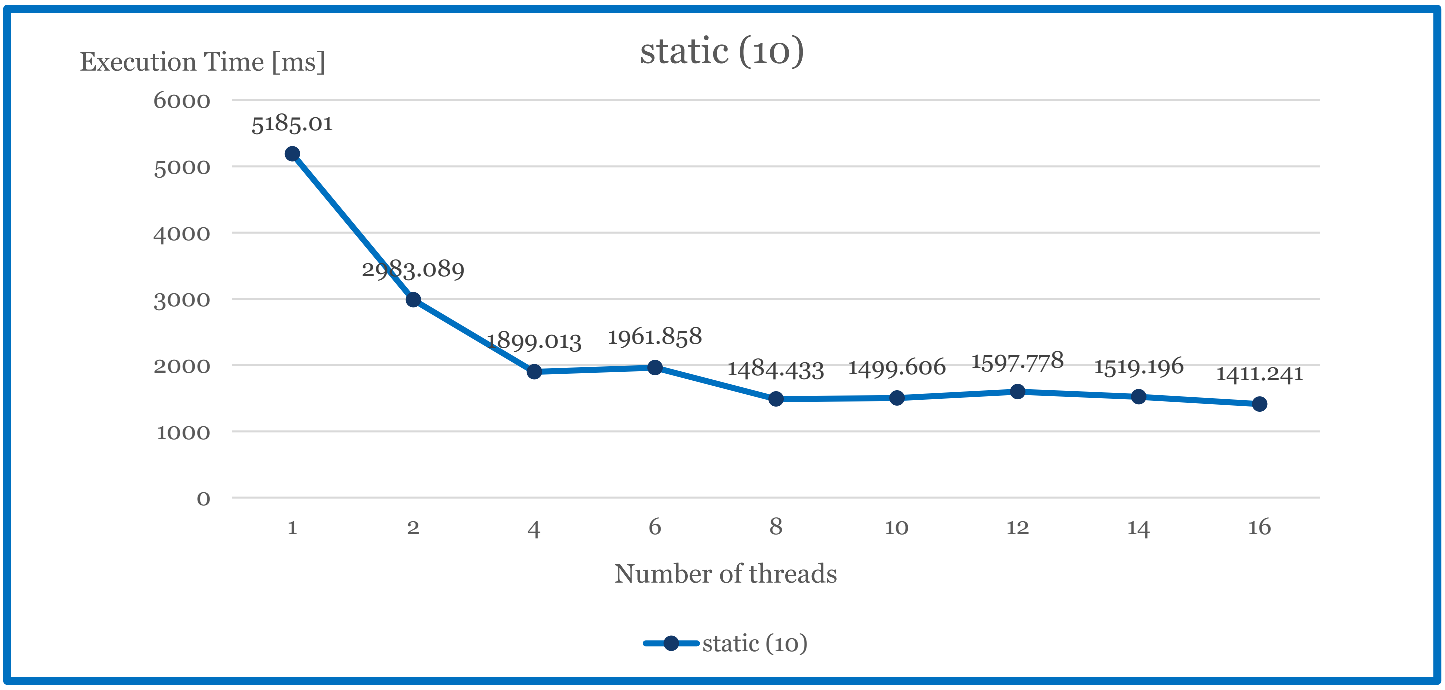
- **Execution Time**

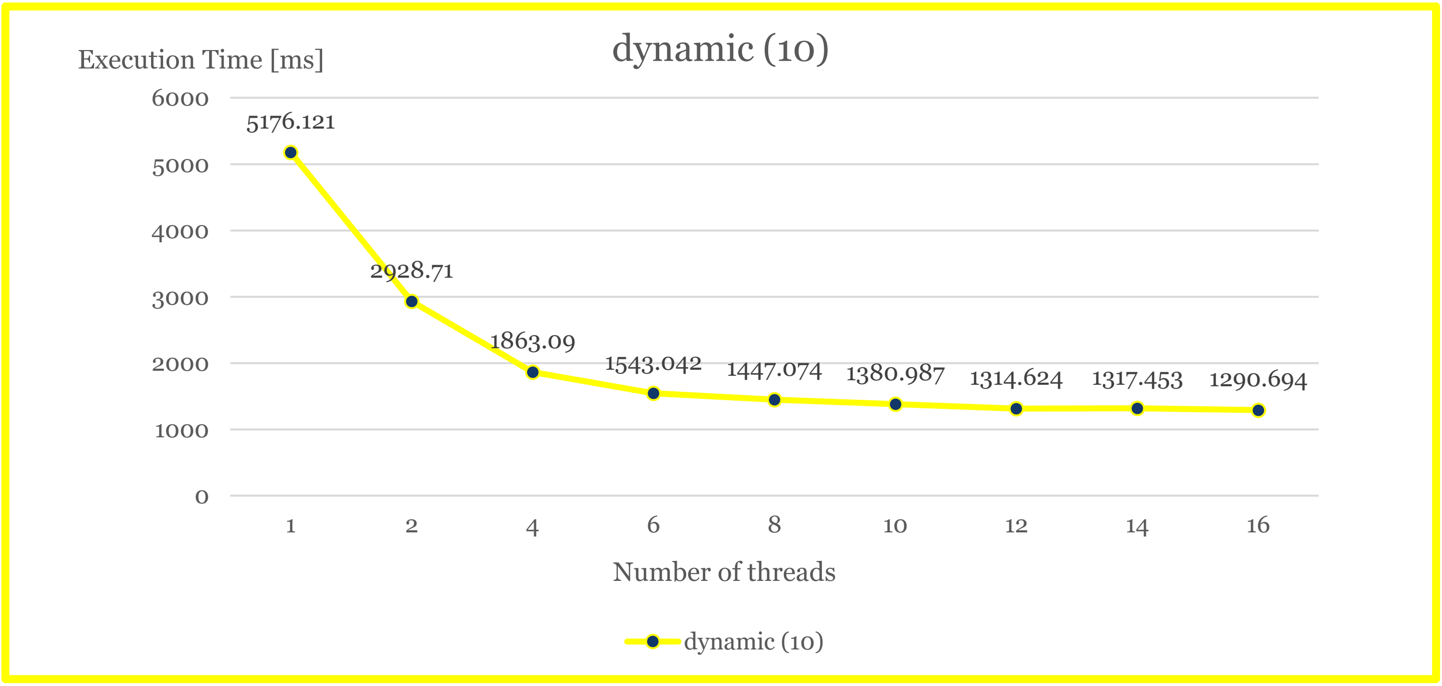
Exec time (unit: ms)	Chunk Size	1	2	4	6	8	10	12	14	16
static	default	6931.067	4981.580	3514.224	2513.367	1824.687	1701.295	1647.715	1641.614	1585.680
dynamic	default	5283.719	3001.907	1848.364	1569.837	1439.932	1359.731	1273.018	1360.368	1314.587
static	10	5185.010	2983.089	1899.013	1961.858	1484.433	1499.606	1597.778	1519.196	1411.241
dynamic	10	5176.121	2928.710	1863.090	1543.042	1447.074	1380.987	1314.624	1317.453	1290.694




static (default)
Execution Time [ms]
8000
7000
6000
5000
4000
3000
2000
1000
0
6931.067
4981.58
3514.224
2513.367
1824.687
1701.295
1647.715
1641.614
1585.68
1
2
4
6
8
10
12
14
16
Number of threads
static (default)

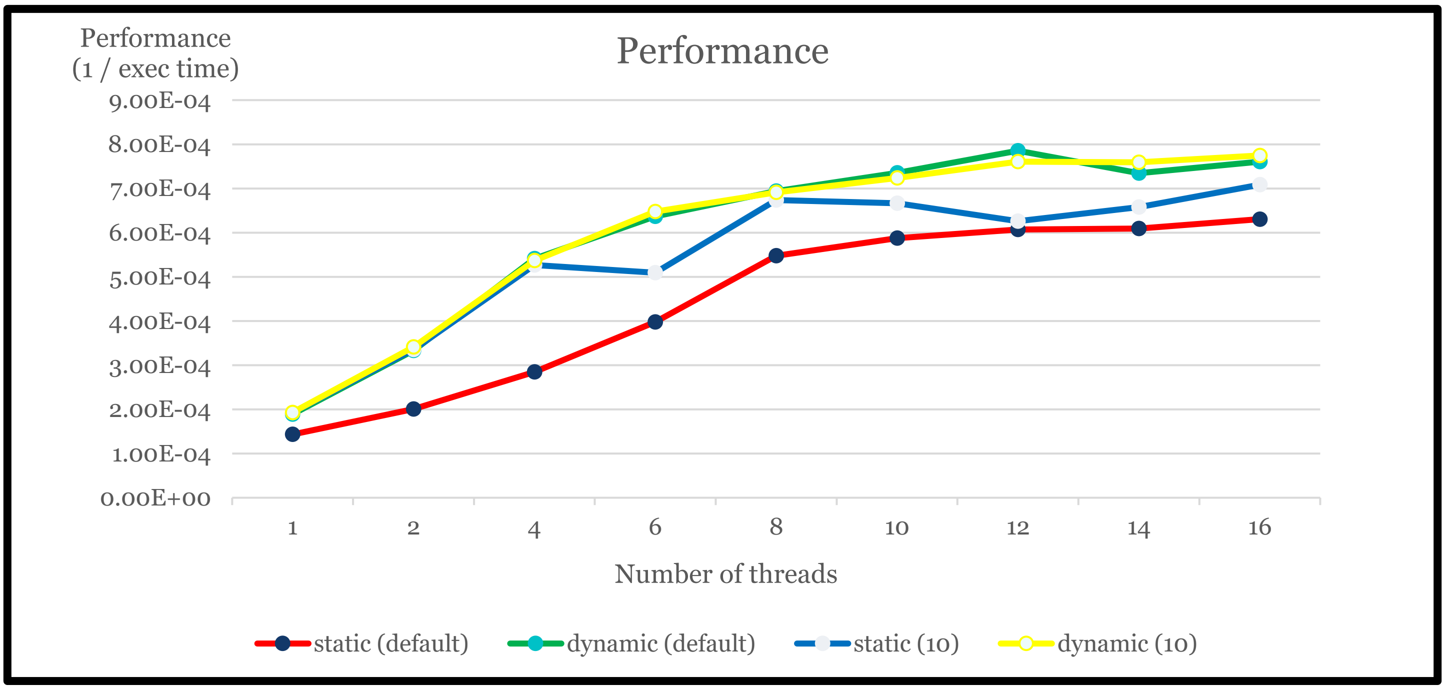

dynamic (default)
Execution Time [ms]
6000
5000
4000
3000
2000
1000
0
5283.719
3001.907
1848.364
1569.364
1439.932
1359.731
1273.018
1360.368
1314.587
1
2
4
6
8
10
12
14
16
Number of threads
dynamic (default)

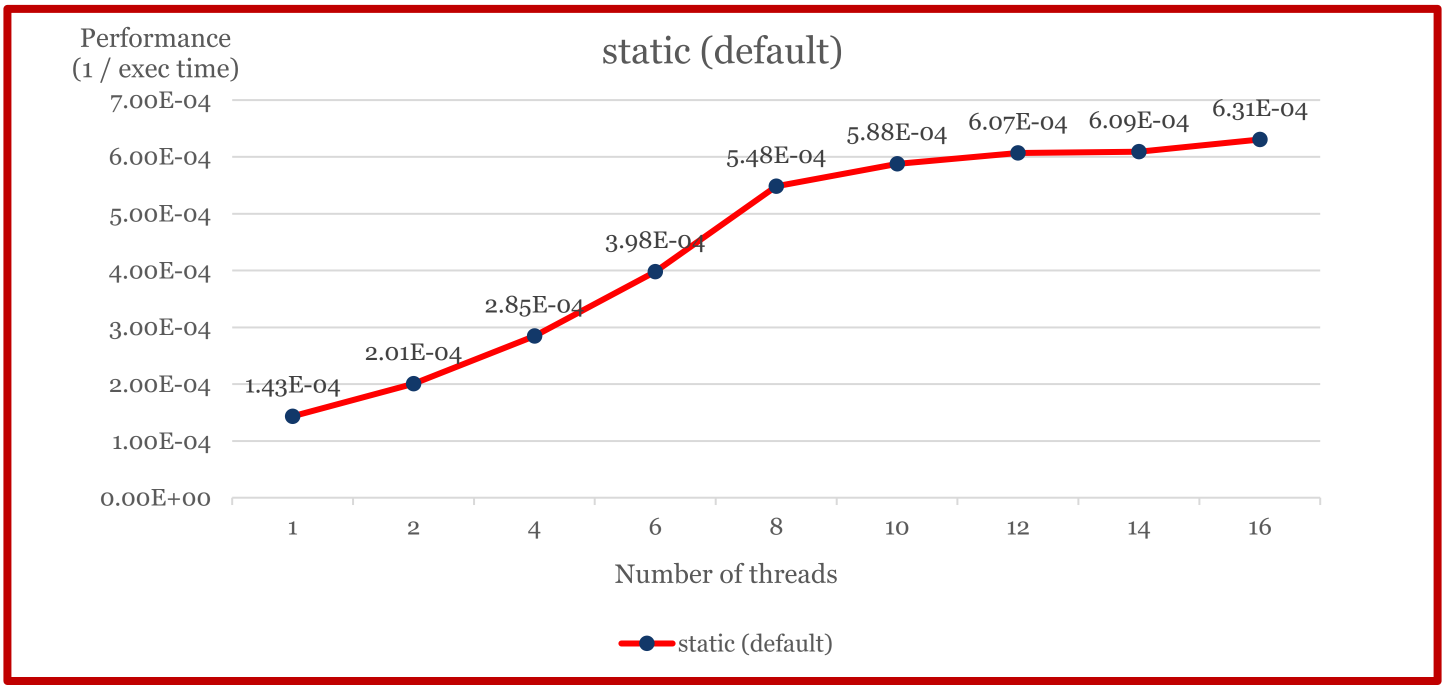

static (10)
Execution Time [ms]
6000
5000
4000
3000
2000
1000
0
5185.01
2983.089
1899.013
1961.858
1484.433
1499.606
1597.778
1519.196
1411.241
1
2
4
6
8
10
12
14
16
Number of threads
static (10)

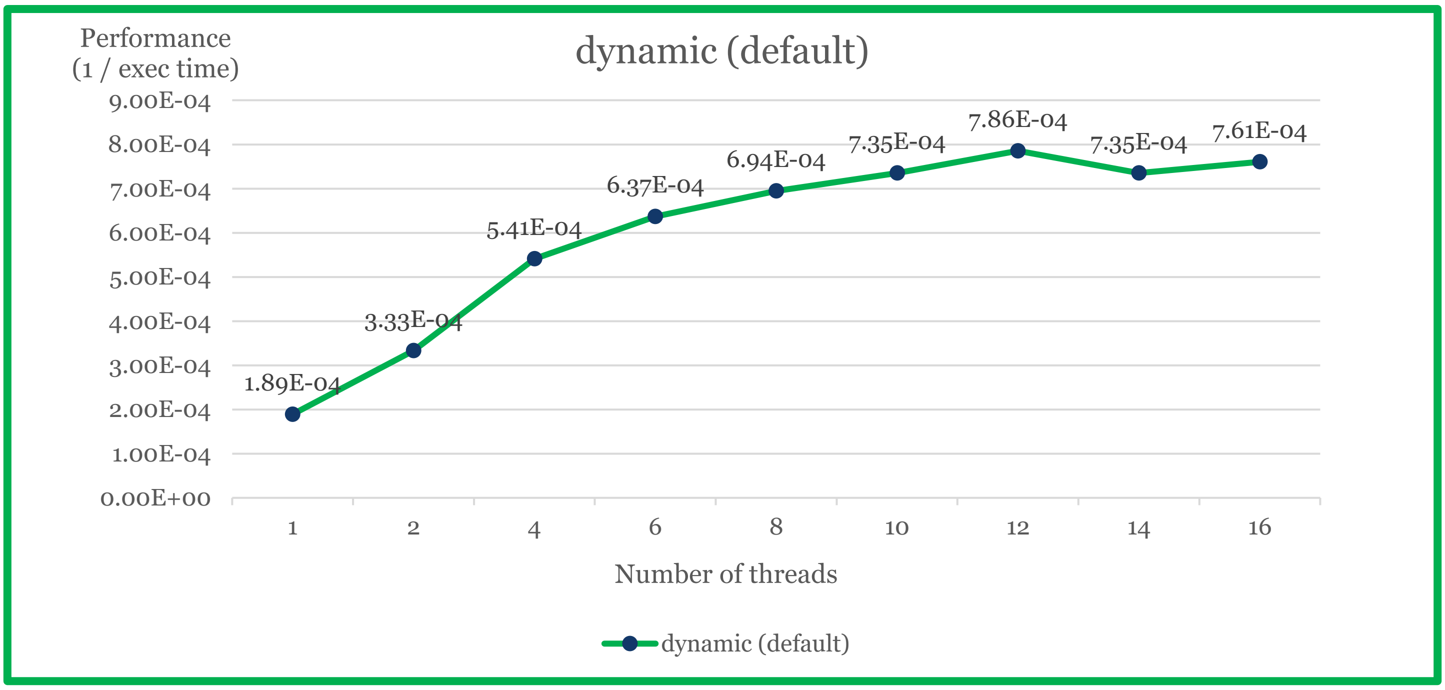

dynamic (10)
Execution Time [ms]
6000
5000
4000
3000
2000
1000
0
5176.121
2928.71
1863.09
1543.042
1447.074
1380.987
1314.624
1317.453
1290.694
1
2
4
6
8
10
12
14
16
Number of threads
dynamic (10)

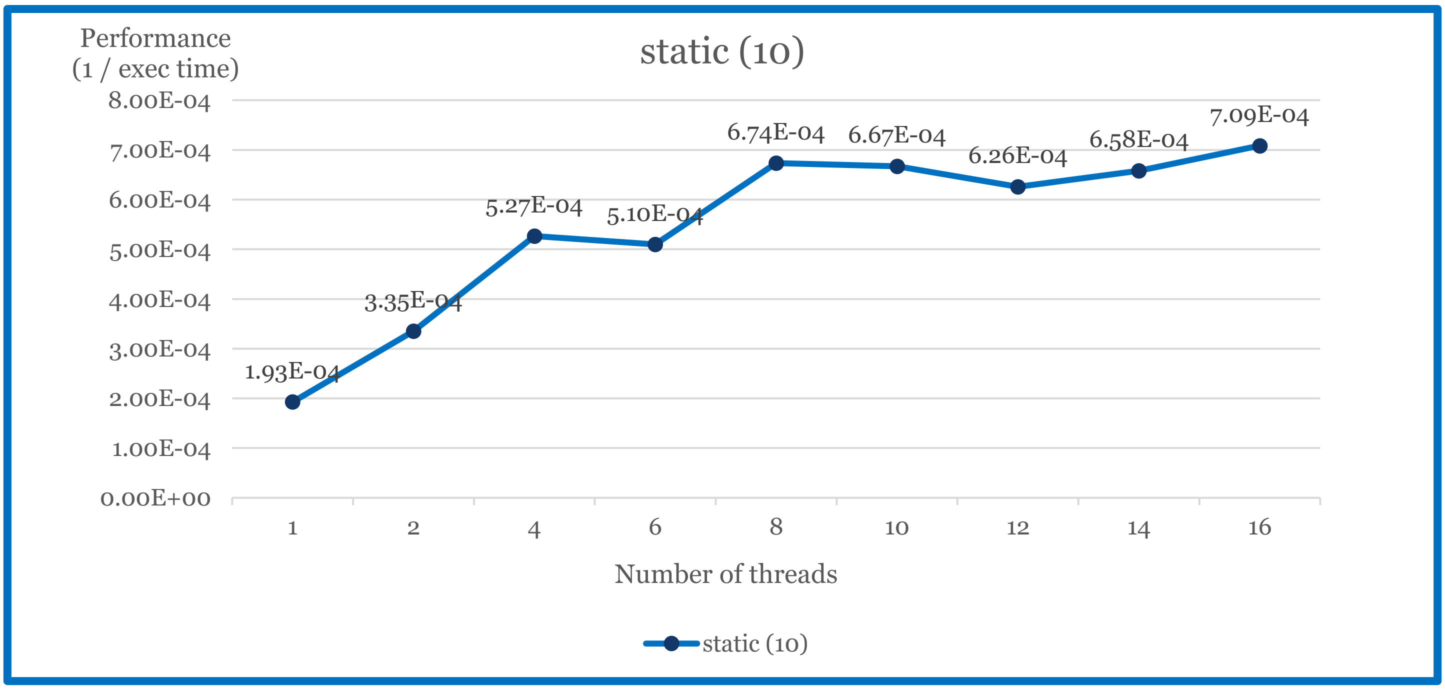
- **Performance**

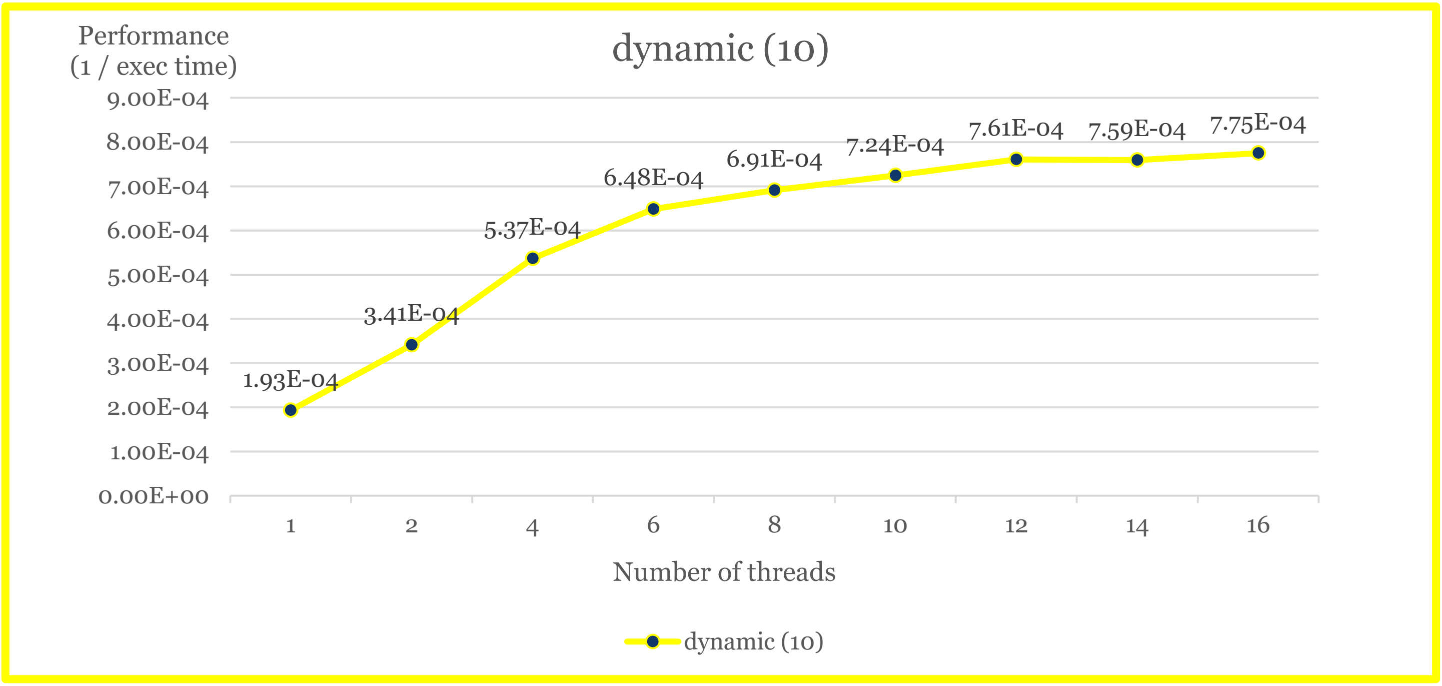
Performance (1/exec time)	Chunk Size	1	2	4	6	8	10	12	14	16
static	default	1.443e-4	2.007e-4	2.846e-4	3.979e-4	5.480e-4	5.877e-4	6.069e-4	6.091e-4	6.306e-4
dynamic	default	1.892e-4	3.331e-4	5.410e-4	6.370e-4	6.944e-4	7.354e-4	7.855e-4	7.350e-4	7.606e-4
static	10	1.928e-4	3.352e-4	5.265e-4	5.097e-4	6.736e-4	6.668e-4	6.258e-4	6.582e-4	7.085e-4
dynamic	10	1.931e-4	3.414e-4	5.367e-4	6.480e-4	6.910e-4	7.241e-4	7.606e-4	7.590e-4	7.747e-4






Performance
(1 / exec time)
dynamic (default)
9.00E-04
8.00E-04
7.00E-04
6.00E-04
5.00E-04
4.00E-04
3.00E-04
2.00E-04
1.00E-04
0.00E+00
1.89E-04
3.33E-04
5.41E-04
6.37E-04
6.94E-04
7.35E-04
7.86E-04
7.35E-04
7.61E-04
1
2
4
6
8
10
12
14
16
Number of threads
dynamic (default)


Performance
(1 / exec time)
static (10)
8.00E-04
7.00E-04
6.00E-04
5.00E-04
4.00E-04
3.00E-04
2.00E-04
1.00E-04
0.00E+00
1.93E-04
3.35E-04
5.27E-04
5.10E-04
6.74E-04
6.67E-04
6.26E-04
6.58E-04
7.09E-04
1
2
4
6
8
10
12
14
16
Number of threads
static (10)



[Explanation/Analysis on the Results]

The above results show that if the number of threads increases, the execution time decreases and performance is improved independent of scheduling method. And when the number of threads is large, then the extent of performance enhancement is very small. It's because creating a lot of threads makes large overheads. Also, dynamic approach is better than static approach. Because large numbers are harder to test whether they are prime or not than small numbers. In terms of chunk size, if chuck size becomes 10, the performance of static approach is improved but dynamic approach not.