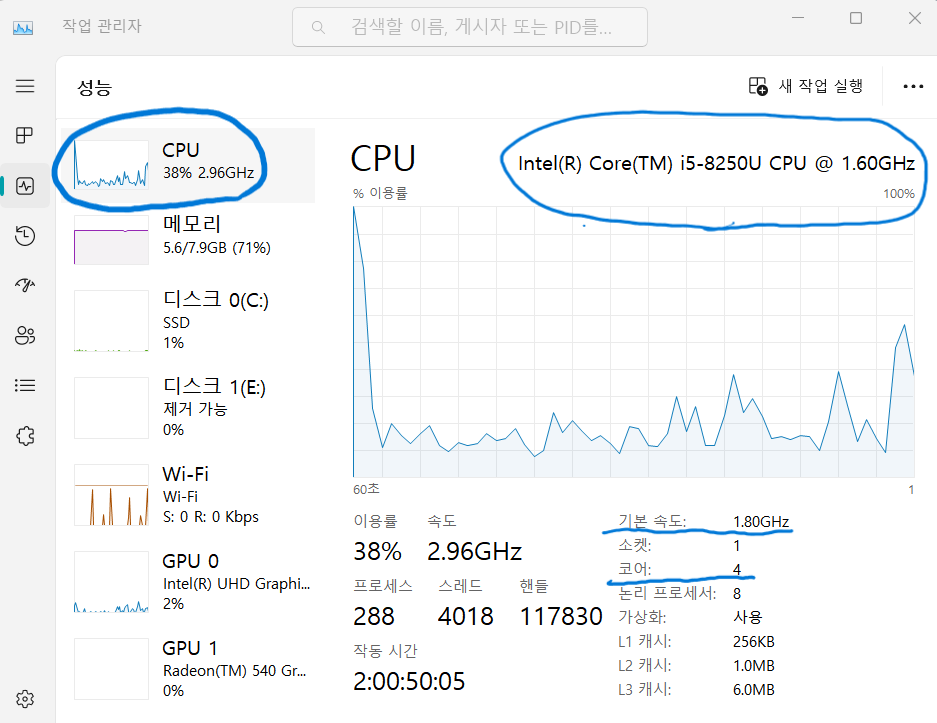
|  |  |
| --- | --- |
|  |  |
| Problem1 Report |  |
|  |  |
|  | Student NO: 20183784Student Name: 노현진 |
|  |  |

**[Environment]**

* **CPU**

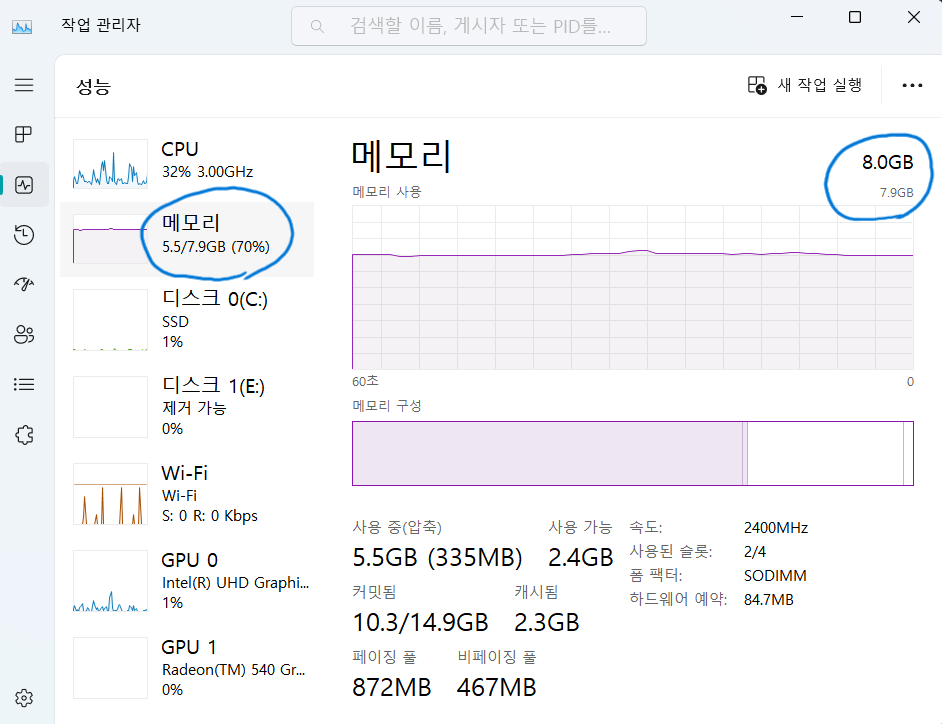


**CPU type: Intel® Core™ i5-8250U CPU**

**Clock Speed: 1.80GHz**

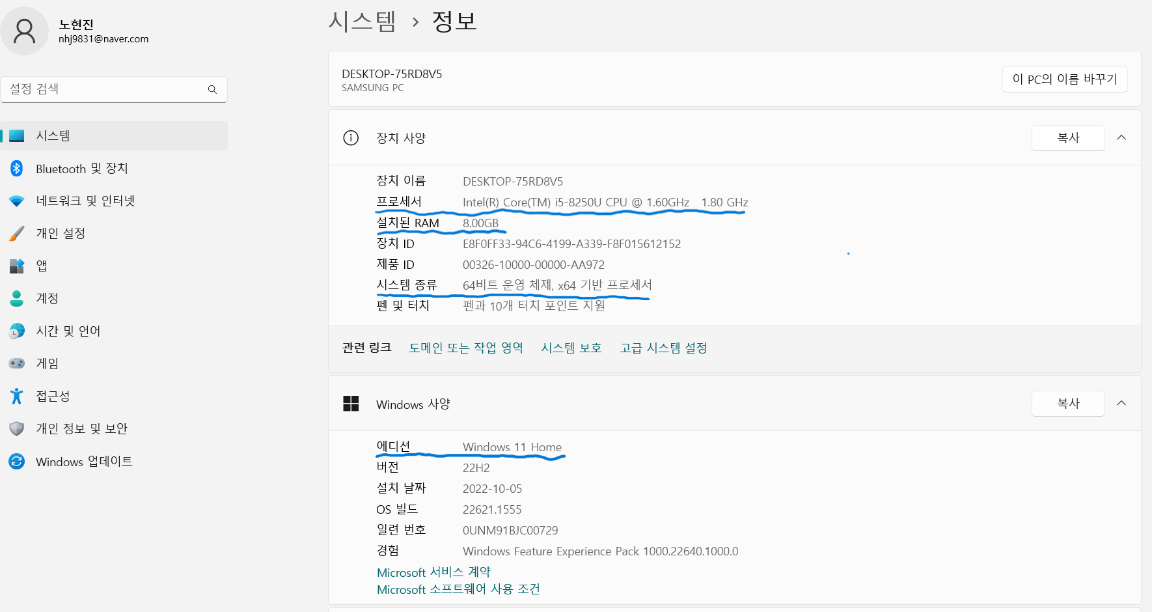
**Number of cores: 4**

* **Memory**



**Memory size: 8.0GB**

* **OS**



**OS type: Windows 11**

**[Source Code]**

* **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 |

* **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 |

**[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.