# Multicore Computing, Project #1

| | |
|---|---|
| 과목명 | 멀티코어컴퓨팅 |
| 교수명 | 손봉수 교수님 |
| 제출일 | 2023.04.13 |
| 학　과 | 소프트웨어대학 소프트웨어학부 |
| 작성자 | 20200641 임수현 |

## Environment

11th Gen Intel(R) Core(TM) i7-1195G7 2.90GHz

| | |
|---|---|
| CPU Type | QuardCore |
| cores # | 4 |
| memory size(RAM) | 16GB |
| logical processor # | 8 |
| OS Type | Window11 (64bit) |

## How to run source Code

1. open eclipse (2021-12)

2. lunch workspace ...\proj1_multicore

3. run the source code

-prob 1. set argument

-prob 2. set matrix500.txt file at

[Run]->[Run Configurations]->[Common -> Input File]

## Problem1.

### Source Code

### 1) static load balancing(block decomposition)

```
package problem1;

import java.util.*;
public class Pc_static_block {
            private static int NUM_END = 200000;
            private static int NUM_THREADS = 4;
            private static int BLOCK_SIZE;
            private static int[] primeCounters;

            public static void main(String[] args) {
                if (args.length == 2) {
                    NUM_THREADS = Integer.parseInt(args[0]);
                    NUM_END = Integer.parseInt(args[1]);
                }

                BLOCK_SIZE = NUM_END / NUM_THREADS;
                primeCounters = new int[NUM_THREADS];

                PrimeCounterThread[] threads = new
PrimeCounterThread[NUM_THREADS];
                long startTime = System.currentTimeMillis();

                for (int i = 0; i < NUM_THREADS; i++) {
                    threads[i] = new PrimeCounterThread(i);
                    threads[i].start();
                }

                for (int i = 0; i < NUM_THREADS; i++) {
                    try {
                        threads[i].join();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }

                int totalCounter = 0;
                for (int i = 0; i < NUM_THREADS; i++) {
                    totalCounter += primeCounters[i];
                }

                long endTime = System.currentTimeMillis();
                long timeDiff = endTime - startTime;
                System.out.println("Program Execution Time: " + timeDiff +
"ms");
                System.out.println("1..." + (NUM_END - 1) + " prime# counters:
" + totalCounter);
            }

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

            private static class PrimeCounterThread extends Thread {
                private int threadIndex;

                public PrimeCounterThread(int threadIndex) {
                    this.threadIndex = threadIndex;
                }

                public void run() {
                    int startIndex = threadIndex * BLOCK_SIZE;
                    int endIndex = startIndex + BLOCK_SIZE;
                    long startThreadTime = System.currentTimeMillis();
                    for (int j = startIndex; j < endIndex; j++) {
                        if (isPrime(j)) {
                            synchronized(primeCounters) {
                                primeCounters[threadIndex]++;
                            }
                        }
                    }
                    long endThreadTime = System.currentTimeMillis();
                    System.out.println("Thread " + threadIndex + " execution
time: " + (endThreadTime - startThreadTime) + "ms");
                }
            }
}
```

## 2) static load balancing(cyclic decomposition)

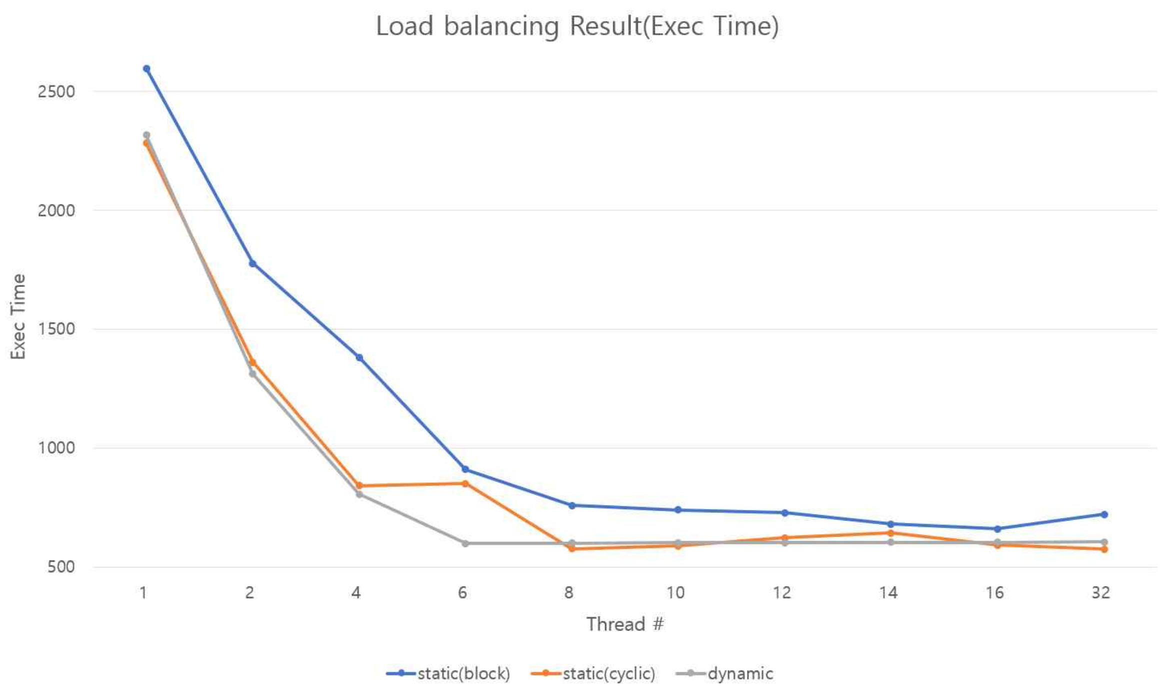
```
package problem1;

import java.util.*;

public class Pc_static_cyclic extends Thread {
    private static int NUM_END = 200000;
    private static int NUM_THREADS = 4;
    private static int[] primeCounters;

    private int threadIndex;

    public Pc_static_cyclic(int startIndex) {
        this.threadIndex = startIndex;
    }

    public void run() {
        int currentIndex = threadIndex*10;
        //System.out.println(currentIndex);
        long startThreadTime = System.currentTimeMillis();
        int step_size=10;
        while (currentIndex < NUM_END ) {
                for (int i=1;i<=step_size;i++) {
                        if (currentIndex+i >=NUM_END) {
                                break;
                        }
                        if (isPrime(currentIndex+i)) {
                                synchronized (primeCounters) {
                                        primeCounters[threadIndex]++;
                                }
                        }
                }
                currentIndex+=step_size*(NUM_THREADS);
        }
        long endThreadTime = System.currentTimeMillis();
        System.out.println("Thread " + threadIndex + " execution time: " +
(endThreadTime - startThreadTime) + "ms");
    }

    public static void main(String[] args) {
        if (args.length == 2) {
            NUM_THREADS = Integer.parseInt(args[0]);
            NUM_END = Integer.parseInt(args[1]);
        }

        primeCounters = new int[NUM_THREADS];

        Thread[] threads = new Thread[NUM_THREADS];
        long startTime = System.currentTimeMillis();

        for (int i = 0; i < NUM_THREADS; i++) {
            threads[i] = new Pc_static_cyclic(i);
            threads[i].start();
        }

        for (int i = 0; i < NUM_THREADS; i++) {
            try {
                threads[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        int totalCounter = 0;
        for (int i = 0; i < NUM_THREADS; i++) {
            totalCounter += primeCounters[i];
        }

        long endTime = System.currentTimeMillis();
        long timeDiff = endTime - startTime;
        System.out.println("Program Execution Time: " + timeDiff + "ms");
        System.out.println("1..." + (NUM_END - 1) + " prime# counters: " +
totalCounter);
    }

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

## 3) dynamic load balancing

```
package problem1;

import java.util.*;

public class Pc_dynamic {
    private static int NUM_END = 200000;
    private static int NUM_THREADS = 4;
    private static int BLOCK_SIZE;
    private static int[] primeCounters;
    private static int currentIndex = 0;

    public static void main(String[] args) {
        if (args.length == 2) {
            NUM_THREADS = Integer.parseInt(args[0]);
            NUM_END = Integer.parseInt(args[1]);
        }

        BLOCK_SIZE = 10;
        primeCounters = new int[NUM_THREADS];

        PrimeCounterThread[] threads = new
PrimeCounterThread[NUM_THREADS];
        long startTime = System.currentTimeMillis();

        for (int i = 0; i < NUM_THREADS; i++) {
            threads[i] = new PrimeCounterThread(i);
            threads[i].start();
        }

        for (int i = 0; i < NUM_THREADS; i++) {
            try {
                threads[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        int totalCounter = 0;
        for (int i = 0; i < NUM_THREADS; i++) {
            totalCounter += primeCounters[i];
        }

        long endTime = System.currentTimeMillis();
        long timeDiff = endTime - startTime;
        System.out.println("Program Execution Time: " + timeDiff + "ms");
        System.out.println("1..." + (NUM_END - 1) + " prime# counters: " +
totalCounter);
    }

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

    private static class PrimeCounterThread extends Thread {
        private int threadIndex;

        public PrimeCounterThread(int threadIndex) {
            this.threadIndex = threadIndex;
        }

        public void run() {
            long startThreadTime = System.currentTimeMillis();
            while (true) {
                int currentIndexCopy;
                synchronized (primeCounters) {
                    currentIndexCopy = currentIndex;
                    currentIndex += BLOCK_SIZE;
                }
                if (currentIndexCopy >= NUM_END) {
                    break;
                }
                int endIndex = currentIndexCopy + BLOCK_SIZE;
                for (int j = currentIndexCopy; j < endIndex; j++) {
                    if (j >= NUM_END) {
                        break;
                    }
                    if (isPrime(j)) {
                        synchronized (primeCounters) {
                            primeCounters[threadIndex]++;
                        }
                    }
                }
            }
            long endThreadTime = System.currentTimeMillis();
            System.out.println("Thread " + threadIndex + " execution time: " +
(endThreadTime - startThreadTime) + "ms");
        }

    }
}
```

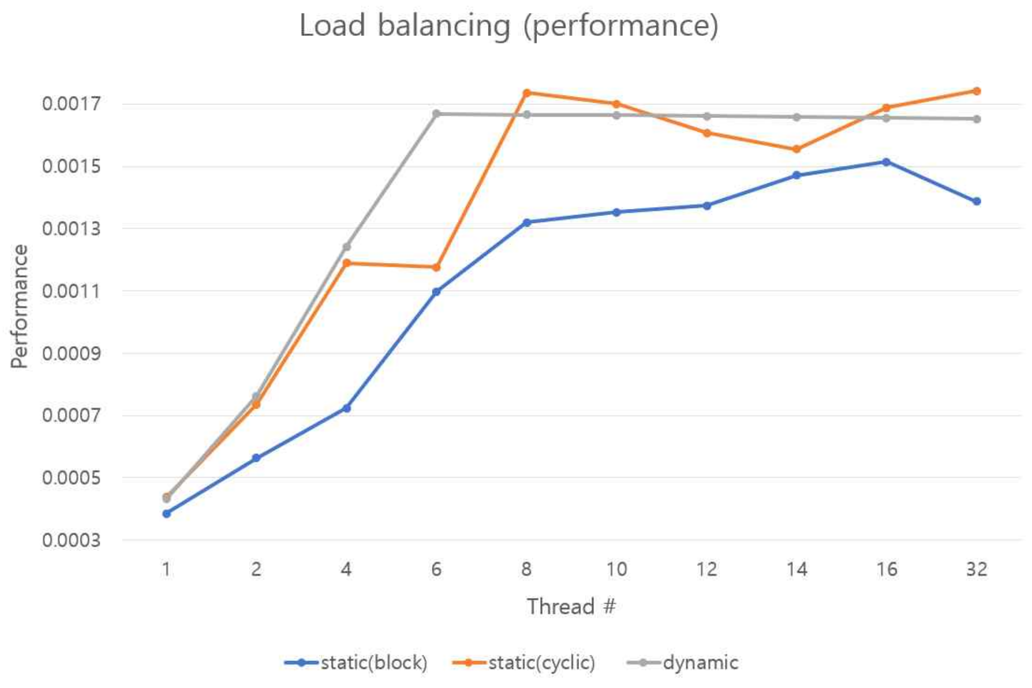
## Result (exec time)

| thread # | static (block) | static (cyclic) [task size : 10 numbers] | dynamic [task size : 10 numbers] |
|---|---|---|---|
| 1 | 2597ms | 2282ms | 2315ms |
| 2 | 1777ms | 1360ms | 1311ms |
| 4 | 1382ms | 841ms | 805ms |
| 6 | 910ms | 850ms | 599ms |
| 8 | 758ms | 576ms | 579ms |
| 10 | 739ms | 588ms | 612ms |
| 12 | 728ms | 622ms | 589ms |
| 14 | 680ms | 643ms | 613ms |
| 16 | 660ms | 592ms | 602ms |
| 32 | 721ms | 574ms | 593ms |



## (Performance)

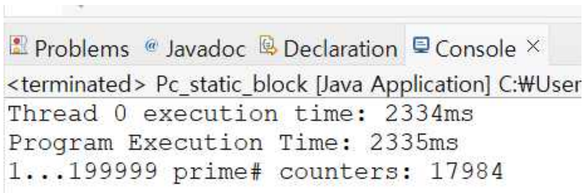
| Thread # | static (block) | static (cyclic) [task size : 10 numbers] | dynamic [task size : 10 numbers] |
|---|---|---|---|
| 1 | 0.00038506 | 0.000438 | 0.000432 |
| 2 | 0.00056275 | 0.000735 | 0.000763 |
| 4 | 0.00072359 | 0.001189 | 0.001242 |
| 6 | 0.0010989 | 0.001176 | 0.001669 |
| 8 | 0.00131926 | 0.001736 | 0.001667 |
| 10 | 0.00135318 | 0.001701 | 0.001664 |
| 12 | 0.00137363 | 0.001608 | 0.001661 |
| 14 | 0.00147059 | 0.001555 | 0.001658 |
| 16 | 0.00151515 | 0.001689 | 0.001656 |
| 32 | 0.00138696 | 0.001742 | 0.001653 |



## Interpretation of parallel results

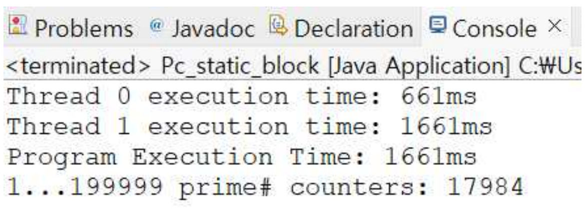
### -static(block) load balancing

I can check that performance is going higher until the number of thread is 8. But when the number of thread is 6, performance is not better than when the number of thread is 4.
And when the number of processor exceeded 8, the performance is hardly improved. I think the reason is because of the number of cores and logical processors in my laptop. There are 2 logical processors per core, but it is better to use all 8 logical processors to run the program than when only one logical processor is working or only two or four are working and the rest are not.
The smaller the thread number, that is, the smaller the prime number, the faster the thread processing ends. This is because the amount to be calculated is small. And as the number of threads increases, the exec time also decreases because each thread does less work.
Num_Thread = 1



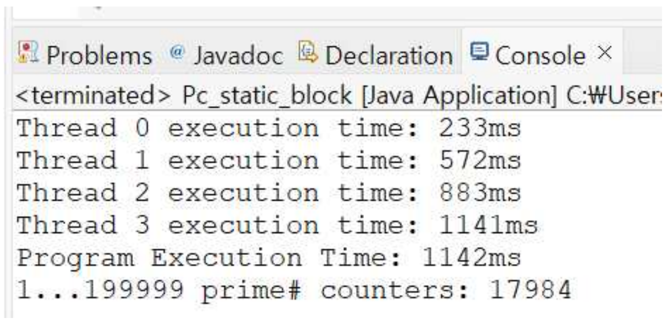
```
<terminated> Pc_static_block [Java Application] C:₩User
Thread 0 execution time: 2334ms
Program Execution Time: 2335ms
1...199999 prime# counters: 17984
```

Num_Thread = 2



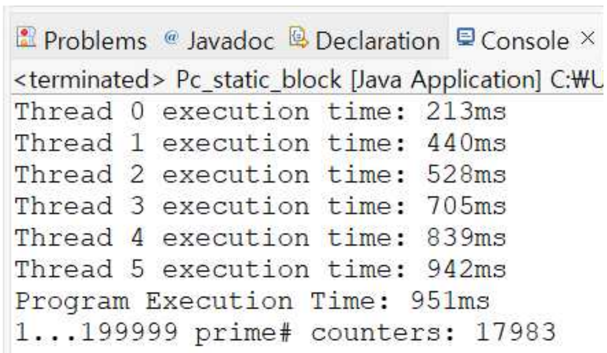
```
<terminated> Pc_static_block [Java Application] C:₩Us
Thread 0 execution time: 661ms
Thread 1 execution time: 1661ms
Program Execution Time: 1661ms
1...199999 prime# counters: 17984
```

Num_Thread = 4



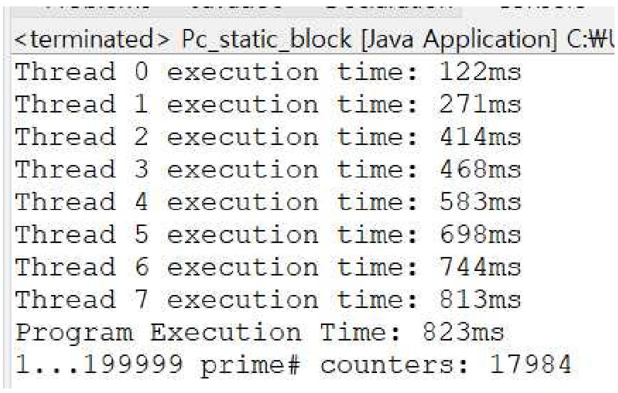
```
<terminated> Pc_static_block [Java Application] C:₩User
Thread 0 execution time: 233ms
Thread 1 execution time: 572ms
Thread 2 execution time: 883ms
Thread 3 execution time: 1141ms
Program Execution Time: 1142ms
1...199999 prime# counters: 17984
```

Num_Thread = 6



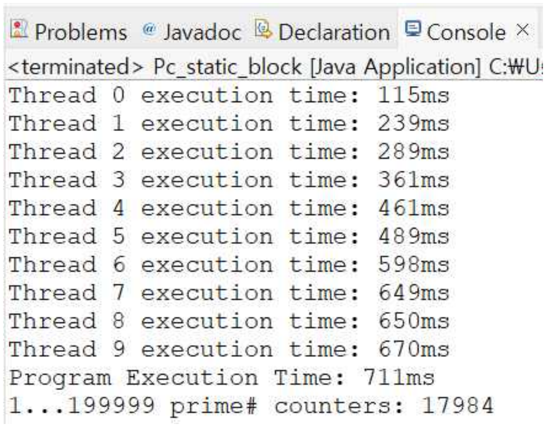
```
<terminated> Pc_static_block [Java Application] C:₩U
Thread 0 execution time: 213ms
Thread 1 execution time: 440ms
Thread 2 execution time: 528ms
Thread 3 execution time: 705ms
Thread 4 execution time: 839ms
Thread 5 execution time: 942ms
Program Execution Time: 951ms
1...199999 prime# counters: 17983
```

Num_Thread = 8



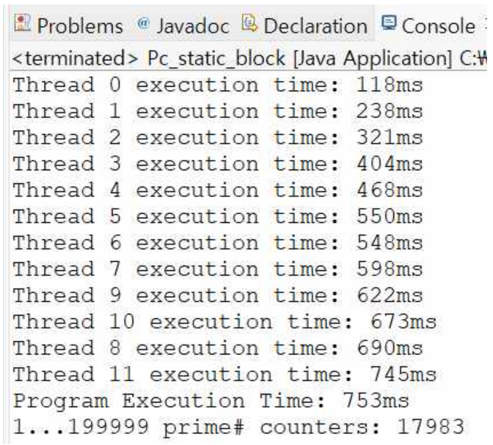
```
<terminated> Pc_static_block [Java Application] C:₩U
Thread 0 execution time: 122ms
Thread 1 execution time: 271ms
Thread 2 execution time: 414ms
Thread 3 execution time: 468ms
Thread 4 execution time: 583ms
Thread 5 execution time: 698ms
Thread 6 execution time: 744ms
Thread 7 execution time: 813ms
Program Execution Time: 823ms
1...199999 prime# counters: 17984
```

Num_Thread=10



```
<terminated> Pc_static_block [Java Application] C:₩U
Thread 0 execution time: 115ms
Thread 1 execution time: 239ms
Thread 2 execution time: 289ms
Thread 3 execution time: 361ms
Thread 4 execution time: 461ms
Thread 5 execution time: 489ms
Thread 6 execution time: 598ms
Thread 7 execution time: 649ms
Thread 8 execution time: 650ms
Thread 9 execution time: 670ms
Program Execution Time: 711ms
1...199999 prime# counters: 17984
```

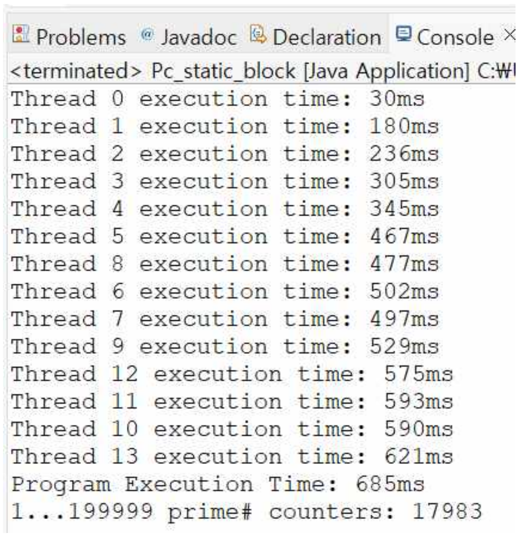
Num_Thread = 12



Problems Javadoc Declaration Console

```
<terminated> Pc_static_block [Java Application] C:₩
Thread 0 execution time: 118ms
Thread 1 execution time: 238ms
Thread 2 execution time: 321ms
Thread 3 execution time: 404ms
Thread 4 execution time: 468ms
Thread 5 execution time: 550ms
Thread 6 execution time: 548ms
Thread 7 execution time: 598ms
Thread 9 execution time: 622ms
Thread 10 execution time: 673ms
Thread 8 execution time: 690ms
Thread 11 execution time: 745ms
Program Execution Time: 753ms
1...199999 prime# counters: 17983
```

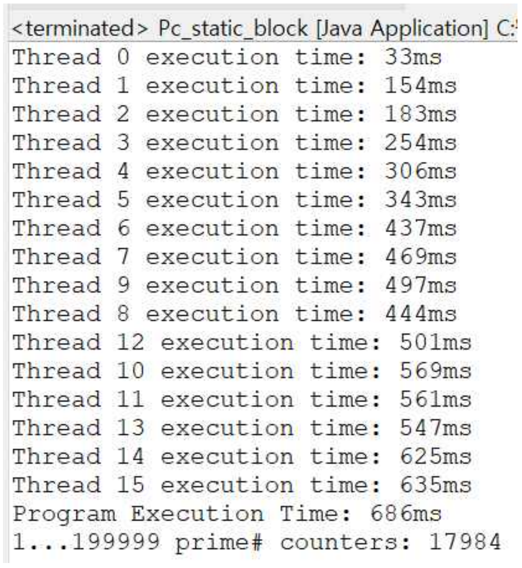
Num_Thread=14



Problems Javadoc Declaration Console

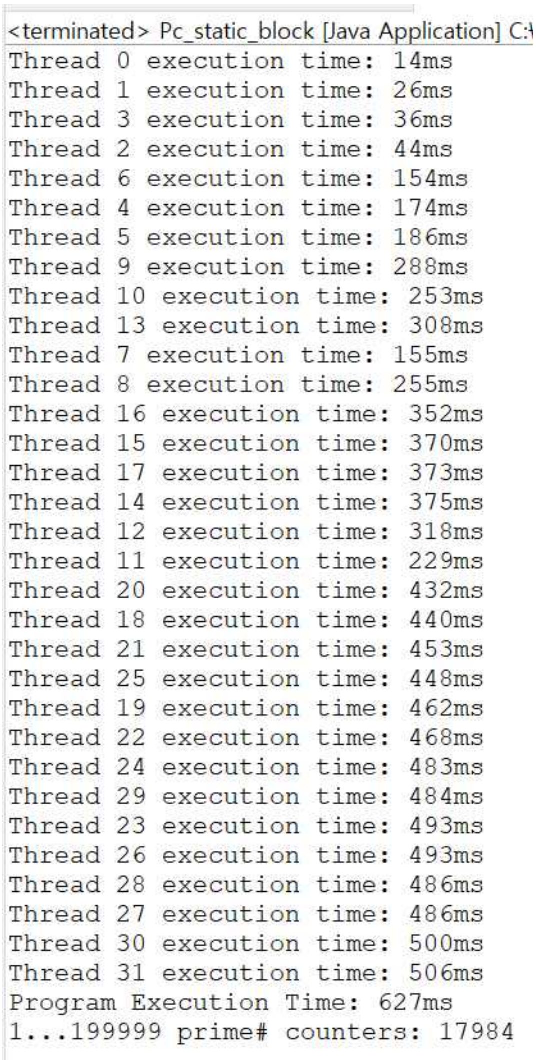
```
<terminated> Pc_static_block [Java Application] C:₩
Thread 0 execution time: 30ms
Thread 1 execution time: 180ms
Thread 2 execution time: 236ms
Thread 3 execution time: 305ms
Thread 4 execution time: 345ms
Thread 5 execution time: 467ms
Thread 8 execution time: 477ms
Thread 6 execution time: 502ms
Thread 7 execution time: 497ms
Thread 9 execution time: 529ms
Thread 12 execution time: 575ms
Thread 11 execution time: 593ms
Thread 10 execution time: 590ms
Thread 13 execution time: 621ms
Program Execution Time: 685ms
1...199999 prime# counters: 17983
```

Num_Thread=16



```
<terminated> Pc_static_block [Java Application] C:
Thread 0 execution time: 33ms
Thread 1 execution time: 154ms
Thread 2 execution time: 183ms
Thread 3 execution time: 254ms
Thread 4 execution time: 306ms
Thread 5 execution time: 343ms
Thread 6 execution time: 437ms
Thread 7 execution time: 469ms
Thread 9 execution time: 497ms
Thread 8 execution time: 444ms
Thread 12 execution time: 501ms
Thread 10 execution time: 569ms
Thread 11 execution time: 561ms
Thread 13 execution time: 547ms
Thread 14 execution time: 625ms
Thread 15 execution time: 635ms
Program Execution Time: 686ms
1...199999 prime# counters: 17984
```

Num_Thread = 32



```
<terminated> Pc_static_block [Java Application] C:
Thread 0 execution time: 14ms
Thread 1 execution time: 26ms
Thread 3 execution time: 36ms
Thread 2 execution time: 44ms
Thread 6 execution time: 154ms
Thread 4 execution time: 174ms
Thread 5 execution time: 186ms
Thread 9 execution time: 288ms
Thread 10 execution time: 253ms
Thread 13 execution time: 308ms
Thread 7 execution time: 155ms
Thread 8 execution time: 255ms
Thread 16 execution time: 352ms
Thread 15 execution time: 370ms
Thread 17 execution time: 373ms
Thread 14 execution time: 375ms
Thread 12 execution time: 318ms
Thread 11 execution time: 229ms
Thread 20 execution time: 432ms
Thread 18 execution time: 440ms
Thread 21 execution time: 453ms
Thread 25 execution time: 448ms
Thread 19 execution time: 462ms
Thread 22 execution time: 468ms
Thread 24 execution time: 483ms
Thread 29 execution time: 484ms
Thread 23 execution time: 493ms
Thread 26 execution time: 493ms
Thread 28 execution time: 486ms
Thread 27 execution time: 486ms
Thread 30 execution time: 500ms
Thread 31 execution time: 506ms
Program Execution Time: 627ms
1...199999 prime# counters: 17984
```

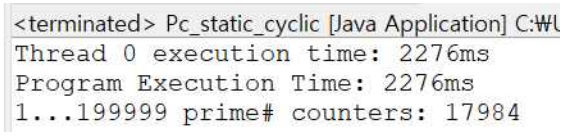
## -static(cyclic) load balancing

The Exec Times for each thread were similar because each thread performed a similar amount of computation. Cyclic decomposition Also, since the number of logical processors in the laptop is 8 and cpu is quadcore, the performance improved as the number of threads increased to 8, and then there was no significant difference in performance when the number of threads exceeded 8.

As the number of threads increases, a relatively larger number is calculated, so as the number of threads increases, the Exec Time also increases.

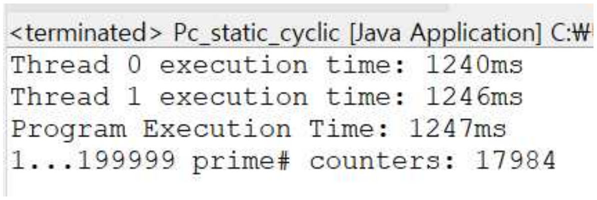
When using more than 8 threads, communication overhead occurs between threads, so there is no big difference in performance compared to block decomposition when the number of threads is 8 or less.

Num_Thread = 1



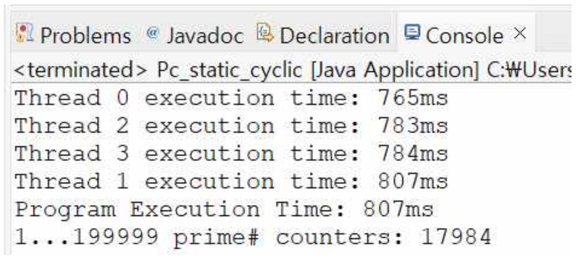
```
<terminated> Pc_static_cyclic [Java Application] C:₩
Thread 0 execution time: 2276ms
Program Execution Time: 2276ms
1...199999 prime# counters: 17984
```

Num_Thread =2



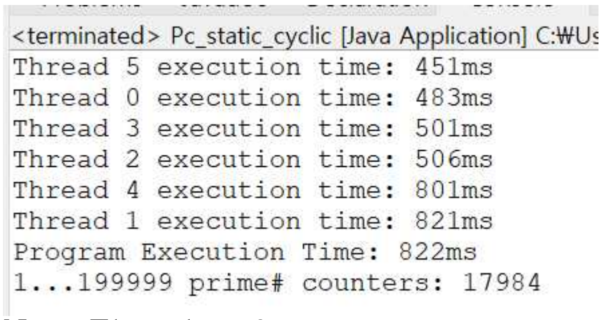
```
<terminated> Pc_static_cyclic [Java Application] C:₩
Thread 0 execution time: 1240ms
Thread 1 execution time: 1246ms
Program Execution Time: 1247ms
1...199999 prime# counters: 17984
```

Num_Thread = 4



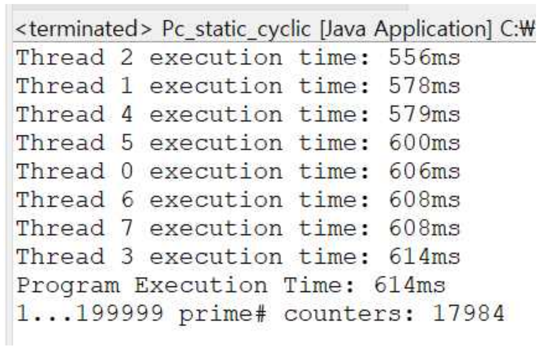
```
<terminated> Pc_static_cyclic [Java Application] C:₩Users
Thread 0 execution time: 765ms
Thread 2 execution time: 783ms
Thread 3 execution time: 784ms
Thread 1 execution time: 807ms
Program Execution Time: 807ms
1...199999 prime# counters: 17984
```

Num_Thread = 6



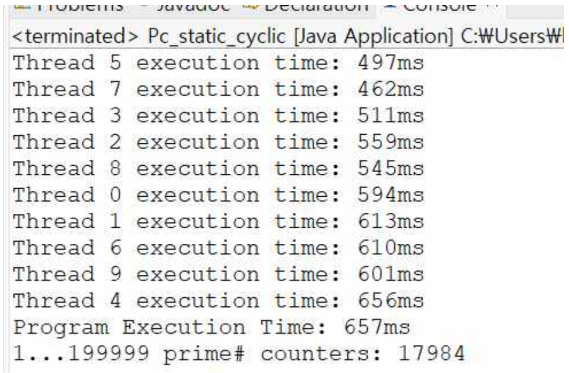
```
<terminated> Pc_static_cyclic [Java Application] C:₩Us
Thread 5 execution time: 451ms
Thread 0 execution time: 483ms
Thread 3 execution time: 501ms
Thread 2 execution time: 506ms
Thread 4 execution time: 801ms
Thread 1 execution time: 821ms
Program Execution Time: 822ms
1...199999 prime# counters: 17984
```

Num_Thread = 8



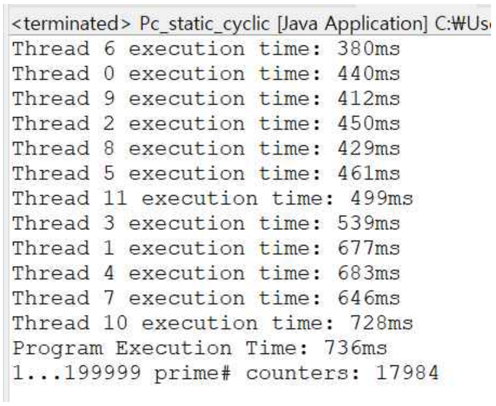
```
<terminated> Pc_static_cyclic [Java Application] C:₩
Thread 2 execution time: 556ms
Thread 1 execution time: 578ms
Thread 4 execution time: 579ms
Thread 5 execution time: 600ms
Thread 0 execution time: 606ms
Thread 6 execution time: 608ms
Thread 7 execution time: 608ms
Thread 3 execution time: 614ms
Program Execution Time: 614ms
1...199999 prime# counters: 17984
```

Num_Thread = 10



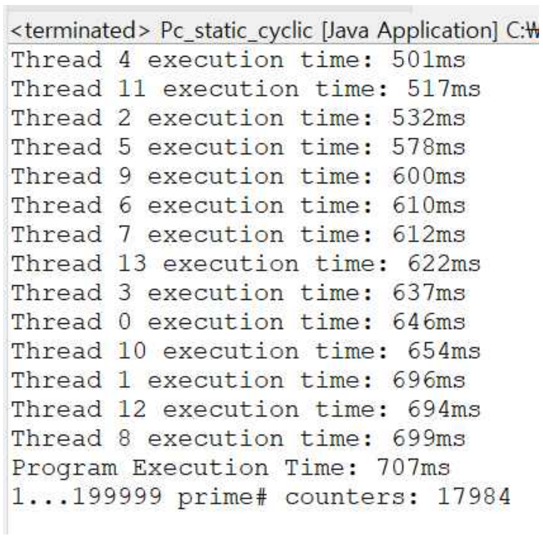
```
<terminated> Pc_static_cyclic [Java Application] C:₩Users₩
Thread 5 execution time: 497ms
Thread 7 execution time: 462ms
Thread 3 execution time: 511ms
Thread 2 execution time: 559ms
Thread 8 execution time: 545ms
Thread 0 execution time: 594ms
Thread 1 execution time: 613ms
Thread 6 execution time: 610ms
Thread 9 execution time: 601ms
Thread 4 execution time: 656ms
Program Execution Time: 657ms
1...199999 prime# counters: 17984
```

Num_Thread = 12



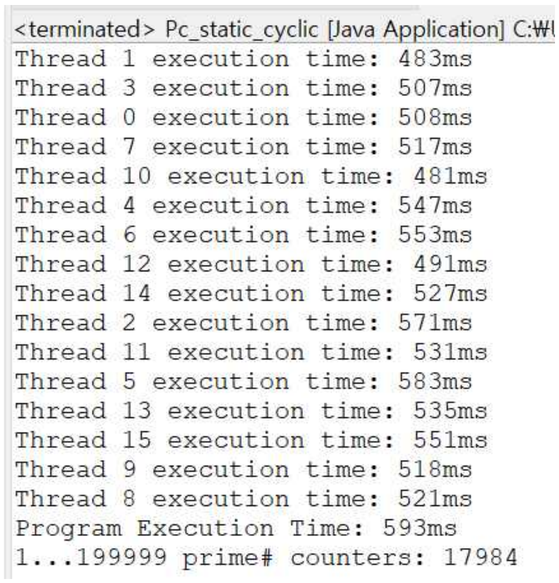
```
<terminated> Pc_static_cyclic [Java Application] C:₩Us
Thread 6 execution time: 380ms
Thread 0 execution time: 440ms
Thread 9 execution time: 412ms
Thread 2 execution time: 450ms
Thread 8 execution time: 429ms
Thread 5 execution time: 461ms
Thread 11 execution time: 499ms
Thread 3 execution time: 539ms
Thread 1 execution time: 677ms
Thread 4 execution time: 683ms
Thread 7 execution time: 646ms
Thread 10 execution time: 728ms
Program Execution Time: 736ms
1...199999 prime# counters: 17984
```

Num_Thread = 14



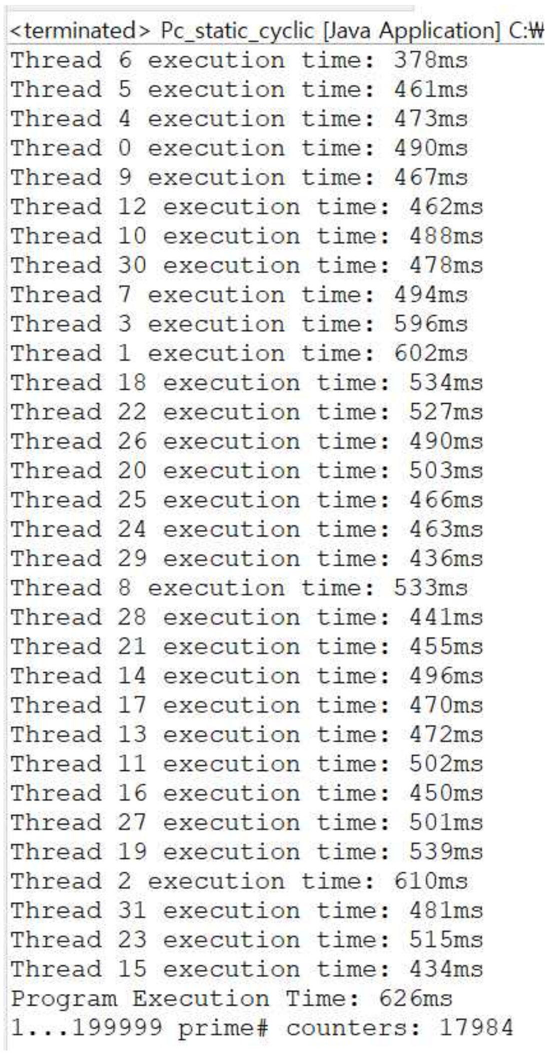
```
<terminated> Pc_static_cyclic [Java Application] C:₩
Thread 4 execution time: 501ms
Thread 11 execution time: 517ms
Thread 2 execution time: 532ms
Thread 5 execution time: 578ms
Thread 9 execution time: 600ms
Thread 6 execution time: 610ms
Thread 7 execution time: 612ms
Thread 13 execution time: 622ms
Thread 3 execution time: 637ms
Thread 0 execution time: 646ms
Thread 10 execution time: 654ms
Thread 1 execution time: 696ms
Thread 12 execution time: 694ms
Thread 8 execution time: 699ms
Program Execution Time: 707ms
1...199999 prime# counters: 17984
```

Num_Thread = 16



```
<terminated> Pc_static_cyclic [Java Application] C:₩
Thread 1 execution time: 483ms
Thread 3 execution time: 507ms
Thread 0 execution time: 508ms
Thread 7 execution time: 517ms
Thread 10 execution time: 481ms
Thread 4 execution time: 547ms
Thread 6 execution time: 553ms
Thread 12 execution time: 491ms
Thread 14 execution time: 527ms
Thread 2 execution time: 571ms
Thread 11 execution time: 531ms
Thread 5 execution time: 583ms
Thread 13 execution time: 535ms
Thread 15 execution time: 551ms
Thread 9 execution time: 518ms
Thread 8 execution time: 521ms
Program Execution Time: 593ms
1...199999 prime# counters: 17984
```

Num_Thread = 32



```
<terminated> Pc_static_cyclic [Java Application] C:₩
Thread 6 execution time: 378ms
Thread 5 execution time: 461ms
Thread 4 execution time: 473ms
Thread 0 execution time: 490ms
Thread 9 execution time: 467ms
Thread 12 execution time: 462ms
Thread 10 execution time: 488ms
Thread 30 execution time: 478ms
Thread 7 execution time: 494ms
Thread 3 execution time: 596ms
Thread 1 execution time: 602ms
Thread 18 execution time: 534ms
Thread 22 execution time: 527ms
Thread 26 execution time: 490ms
Thread 20 execution time: 503ms
Thread 25 execution time: 466ms
Thread 24 execution time: 463ms
Thread 29 execution time: 436ms
Thread 8 execution time: 533ms
Thread 28 execution time: 441ms
Thread 21 execution time: 455ms
Thread 14 execution time: 496ms
Thread 17 execution time: 470ms
Thread 13 execution time: 472ms
Thread 11 execution time: 502ms
Thread 16 execution time: 450ms
Thread 27 execution time: 501ms
Thread 19 execution time: 539ms
Thread 2 execution time: 610ms
Thread 31 execution time: 481ms
Thread 23 execution time: 515ms
Thread 15 execution time: 434ms
Program Execution Time: 626ms
1...199999 prime# counters: 17984
```

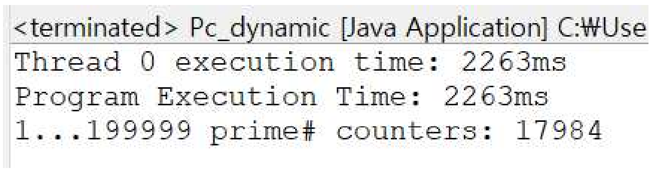
## -dynamic load balancing

As with static, since the number of logical processors is 8 and cpu is quadcore, performance improves until the number of threads reaches 8, but after that, there is little performance improvement.

Also, as the number of threads increases, run-time overhead occurs, so if the number of threads is 8 or more, run-time overhead occurs due to synchronization, so there is no difference in performance.

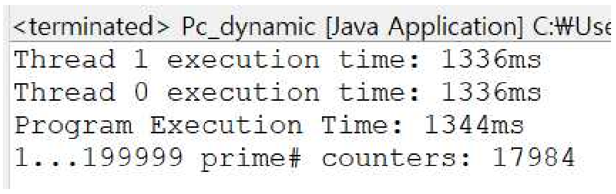
Since it is dynamic, the Exec Time for each thread changes as the number of threads increases.

Num_Thread = 1



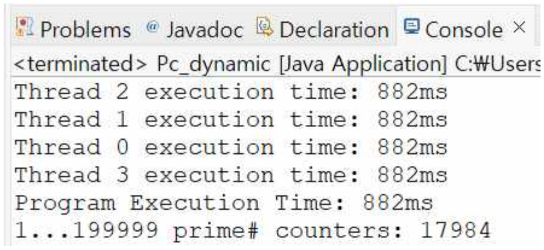
```
<terminated> Pc_dynamic [Java Application] C:₩Use
Thread 0 execution time: 2263ms
Program Execution Time: 2263ms
1...199999 prime# counters: 17984
```

Num_Thread = 2



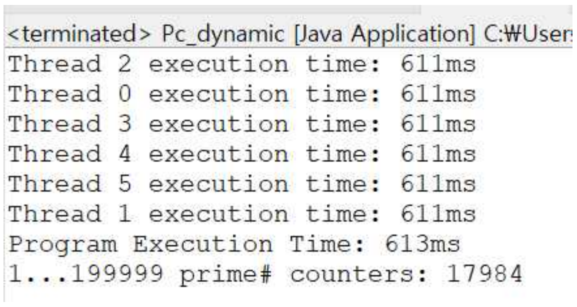
```
<terminated> Pc_dynamic [Java Application] C:₩Use
Thread 1 execution time: 1336ms
Thread 0 execution time: 1336ms
Program Execution Time: 1344ms
1...199999 prime# counters: 17984
```

Num_Thread = 4



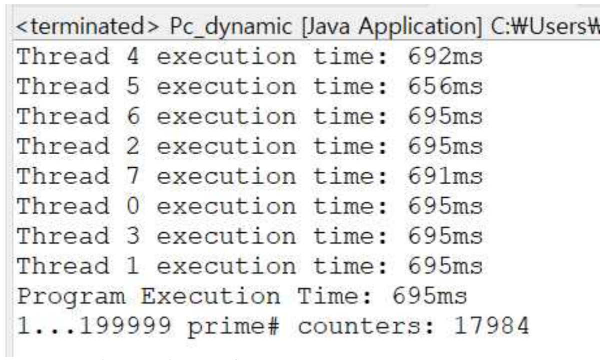
```
<terminated> Pc_dynamic [Java Application] C:₩Users
Thread 2 execution time: 882ms
Thread 1 execution time: 882ms
Thread 0 execution time: 882ms
Thread 3 execution time: 882ms
Program Execution Time: 882ms
1...199999 prime# counters: 17984
```

Num_Thread = 6



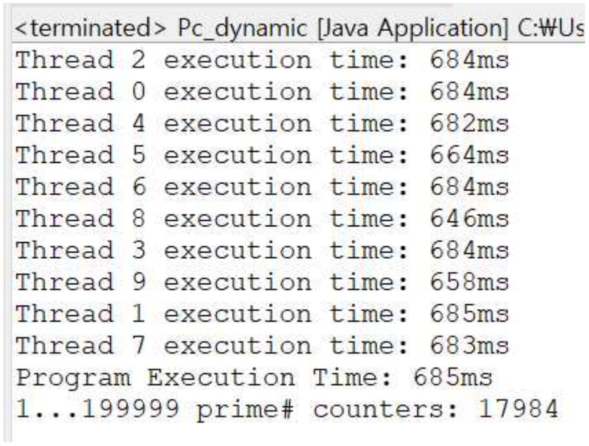
```
<terminated> Pc_dynamic [Java Application] C:₩User
Thread 2 execution time: 611ms
Thread 0 execution time: 611ms
Thread 3 execution time: 611ms
Thread 4 execution time: 611ms
Thread 5 execution time: 611ms
Thread 1 execution time: 611ms
Program Execution Time: 613ms
1...199999 prime# counters: 17984
```

Num_Thread = 8



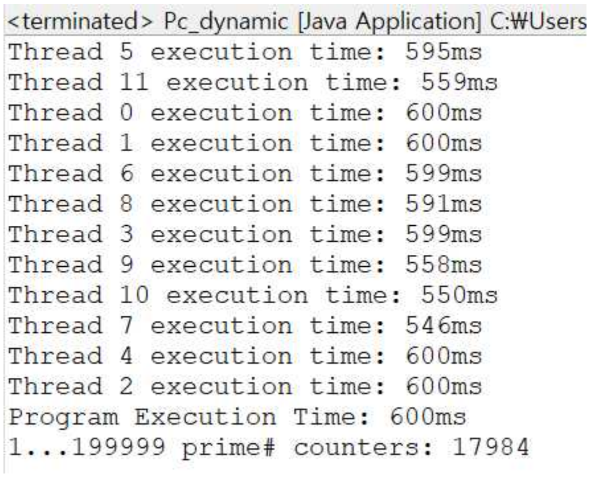
```
<terminated> Pc_dynamic [Java Application] C:₩Users₩
Thread 4 execution time: 692ms
Thread 5 execution time: 656ms
Thread 6 execution time: 695ms
Thread 2 execution time: 695ms
Thread 7 execution time: 691ms
Thread 0 execution time: 695ms
Thread 3 execution time: 695ms
Thread 1 execution time: 695ms
Program Execution Time: 695ms
1...199999 prime# counters: 17984
```

Num_Thread = 10



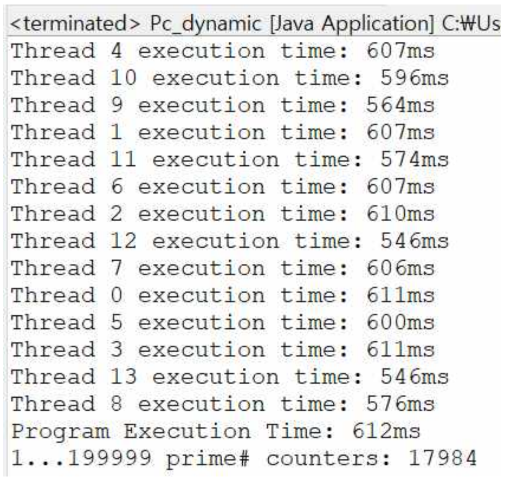
```
<terminated> Pc_dynamic [Java Application] C:₩Us
Thread 2 execution time: 684ms
Thread 0 execution time: 684ms
Thread 4 execution time: 682ms
Thread 5 execution time: 664ms
Thread 6 execution time: 684ms
Thread 8 execution time: 646ms
Thread 3 execution time: 684ms
Thread 9 execution time: 658ms
Thread 1 execution time: 685ms
Thread 7 execution time: 683ms
Program Execution Time: 685ms
1...199999 prime# counters: 17984
```

Num_Thread = 12



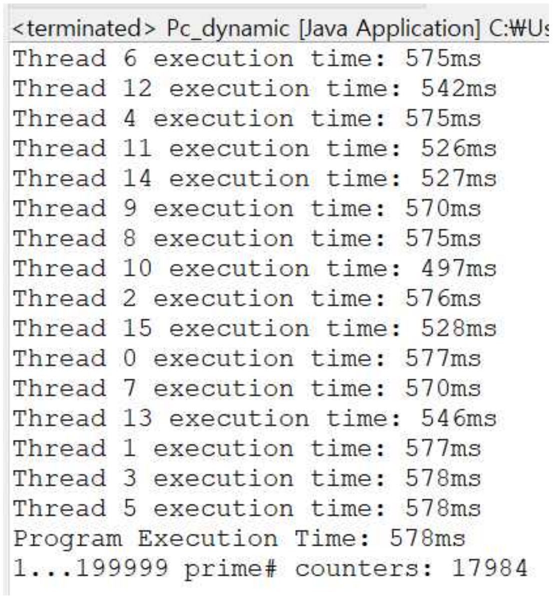
```
<terminated> Pc_dynamic [Java Application] C:₩Users
Thread 5 execution time: 595ms
Thread 11 execution time: 559ms
Thread 0 execution time: 600ms
Thread 1 execution time: 600ms
Thread 6 execution time: 599ms
Thread 8 execution time: 591ms
Thread 3 execution time: 599ms
Thread 9 execution time: 558ms
Thread 10 execution time: 550ms
Thread 7 execution time: 546ms
Thread 4 execution time: 600ms
Thread 2 execution time: 600ms
Program Execution Time: 600ms
1...199999 prime# counters: 17984
```

Num_Thread = 14



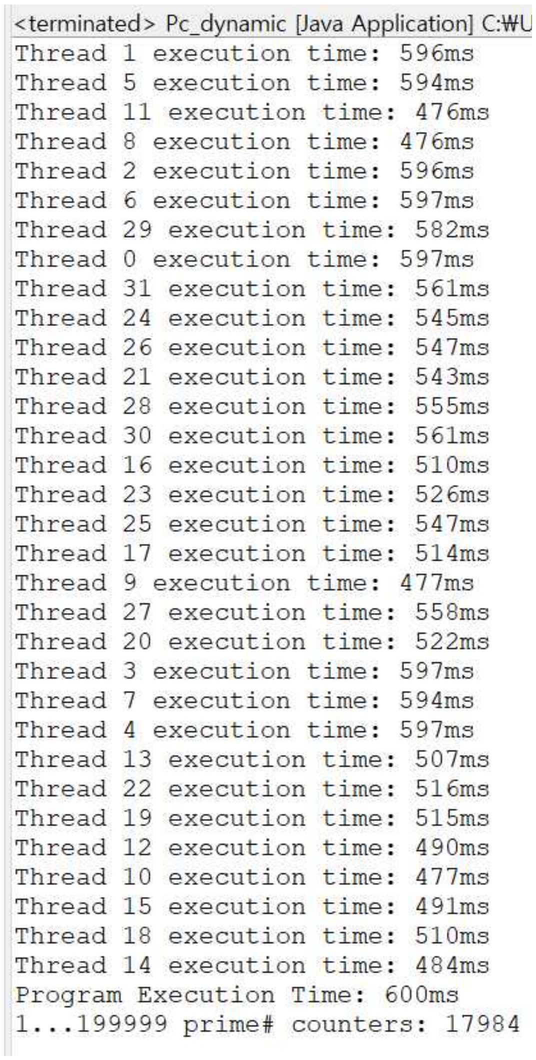
```
<terminated> Pc_dynamic [Java Application] C:₩Us
Thread 4 execution time: 607ms
Thread 10 execution time: 596ms
Thread 9 execution time: 564ms
Thread 1 execution time: 607ms
Thread 11 execution time: 574ms
Thread 6 execution time: 607ms
Thread 2 execution time: 610ms
Thread 12 execution time: 546ms
Thread 7 execution time: 606ms
Thread 0 execution time: 611ms
Thread 5 execution time: 600ms
Thread 3 execution time: 611ms
Thread 13 execution time: 546ms
Thread 8 execution time: 576ms
Program Execution Time: 612ms
1...199999 prime# counters: 17984
```

Num_Thread = 16



```
<terminated> Pc_dynamic [Java Application] C:₩Us
Thread 6 execution time: 575ms
Thread 12 execution time: 542ms
Thread 4 execution time: 575ms
Thread 11 execution time: 526ms
Thread 14 execution time: 527ms
Thread 9 execution time: 570ms
Thread 8 execution time: 575ms
Thread 10 execution time: 497ms
Thread 2 execution time: 576ms
Thread 15 execution time: 528ms
Thread 0 execution time: 577ms
Thread 7 execution time: 570ms
Thread 13 execution time: 546ms
Thread 1 execution time: 577ms
Thread 3 execution time: 578ms
Thread 5 execution time: 578ms
Program Execution Time: 578ms
1...199999 prime# counters: 17984
```

Num_Thread = 32



```
<terminated> Pc_dynamic [Java Application] C:₩U
Thread 1 execution time: 596ms
Thread 5 execution time: 594ms
Thread 11 execution time: 476ms
Thread 8 execution time: 476ms
Thread 2 execution time: 596ms
Thread 6 execution time: 597ms
Thread 29 execution time: 582ms
Thread 0 execution time: 597ms
Thread 31 execution time: 561ms
Thread 24 execution time: 545ms
Thread 26 execution time: 547ms
Thread 21 execution time: 543ms
Thread 28 execution time: 555ms
Thread 30 execution time: 561ms
Thread 16 execution time: 510ms
Thread 23 execution time: 526ms
Thread 25 execution time: 547ms
Thread 17 execution time: 514ms
Thread 9 execution time: 477ms
Thread 27 execution time: 558ms
Thread 20 execution time: 522ms
Thread 3 execution time: 597ms
Thread 7 execution time: 594ms
Thread 4 execution time: 597ms
Thread 13 execution time: 507ms
Thread 22 execution time: 516ms
Thread 19 execution time: 515ms
Thread 12 execution time: 490ms
Thread 10 execution time: 477ms
Thread 15 execution time: 491ms
Thread 18 execution time: 510ms
Thread 14 execution time: 484ms
Program Execution Time: 600ms
1...199999 prime# counters: 17984
```

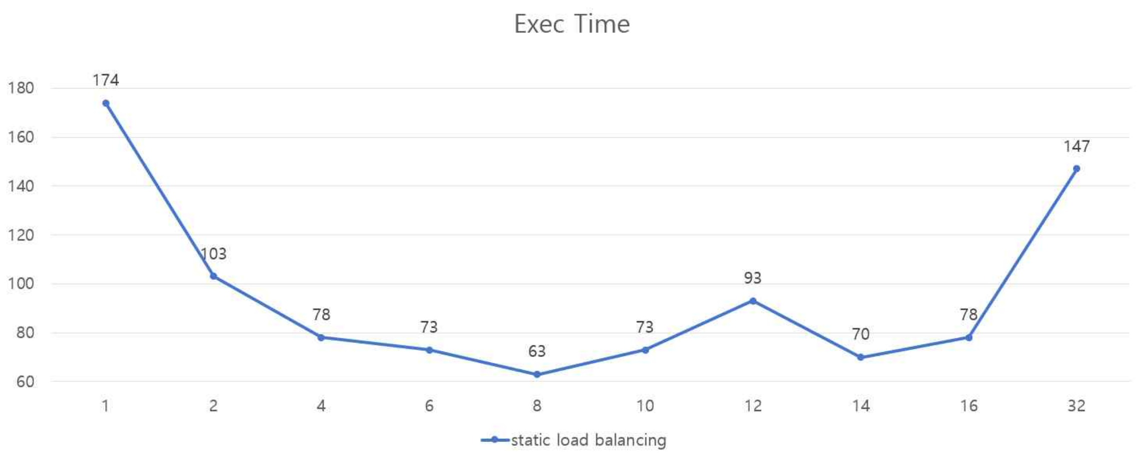
## Problem 2.

### - source code

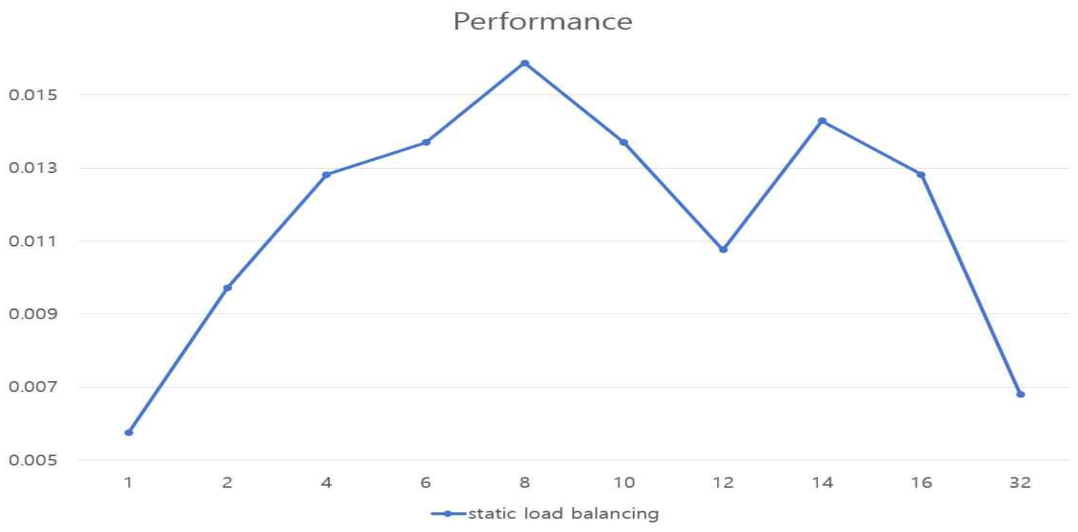
```
package problem2;

import java.util.*;
import java.lang.*;

public class MatmultD{
  private static Scanner sc = new Scanner(System.in);
 // private static int BLOCK_SIZE;
  public static void main(String [] args)  {
    int thread_no=0;
    if (args.length==1) thread_no = Integer.valueOf(args[0]);
    else thread_no = 4;


    int a[][]=readMatrix();
    int b[][]=readMatrix();

    long startTime = System.currentTimeMillis();

    MatmultThread[] thread=new MatmultThread[thread_no];

    for (int i=0;i<thread_no;i++) {
        thread[i]=new MatmultThread(i,thread_no,a,b);
    }

    for (int i=0;i<thread_no;i++) {
        thread[i].start();
    }

    for(int i=0;i<thread_no;i++) {
        try {
                thread[i].join();
                } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                }

    }
    long endTime = System.currentTimeMillis();


    for (int i=0;i<thread_no;i++) {

         System.out.println("Thread " + i+ " execution time: " + thread[i].timeDiff +
"ms");
    }

    System.out.printf("[thread_no]:%2d , [Time]:%4d ms\n", thread_no,
endTime-startTime);

  for (int t = 0; t < thread_no; t++) {
      System.out.println("Thread-" + t + " Sum : " + thread[t].temp_sum );
  }

  System.out.println("Matrix Sum = " + MatmultThread.sum + "\n");
}
   public static int[][] readMatrix() {
       int rows = sc.nextInt();
       int cols = sc.nextInt();
       int[][] result = new int[rows][cols];
       for (int i = 0; i < rows; i++) {
           for (int j = 0; j < cols; j++) {
              result[i][j] = sc.nextInt();
           }
       }
       return result;
   }

  public static void printMatrix(int[][] mat) {
  System.out.println("Matrix["+mat.length+"]["+mat[0].length+"]");
    int rows = mat.length;
    int columns = mat[0].length;
    int sum = 0;
    for (int i = 0; i < rows; i++) {
      for (int j = 0; j < columns; j++) {
        System.out.printf("%4d " , mat[i][j]);
        sum+=mat[i][j];
      }
      System.out.println();
    }
    System.out.println();
    System.out.println("Matrix Sum = " + sum + "\n");
  }

  static class MatmultThread extends Thread {
      int start;
      int next;
      int[][] a;
      int[][] b;

      static int sum;
      int temp_sum=0;
      long StartTime;
      long endTime;
      long timeDiff;

      int n;
      int m;
      int p;

      public MatmultThread(int start, int next, int[][] a, int[][] b) {
          this.start=start;
          this.next=next;
          this.a=a;
          this.b=b;

          }
      public void run() {
          StartTime=System.currentTimeMillis();

          n= a[0].length;
          m= a.length;
          p= b[0].length;

          for(int i = start;i < m;i+=next){
              for(int j = 0;j < p;j++){
                 // temp_sum=0;

                  for(int k = 0;k < n;k++){
                          temp_sum += a[i][k] * b[k][j];
                  }

              }
          }
```

```
                sum=sum+temp_sum;


                endTime=System.currentTimeMillis();
                timeDiff=endTime-StartTime;

            }
        }

}
```

## Result



| Thread # | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|
| exec time (ms) | 174 | 103 | 78 | 73 | 63 | 73 | 93 | 70 | 78 | 147 |



| Thread # | 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| performance | 0.005747 | 0.009708 | 0.012820 | 0.01369 | 0.015873 |

| 10 | 12 | 14 | 16 | 32 |
|---|---|---|---|---|
| 0.013698 | 0.0107526 | 0.0142857 | 0.0128205 | 0.0068027 |

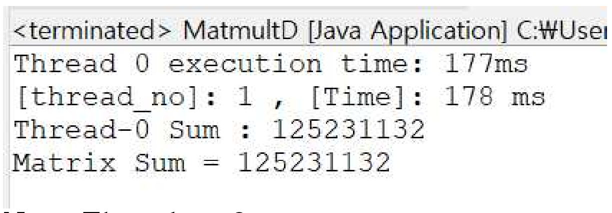
## Interpretation of parallel results

Like ‘prob1’, the number of logical processors in my notebook is 8 and quadcore , so the performance is better than the number of threads less than 8.

Although the amount of calculations for each thread is similar, the time for each thread is also different because the value to be calculated for each thread is different.

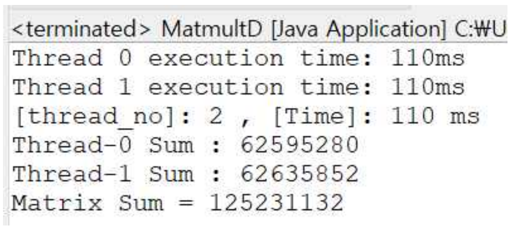
When there are 32 threads, the performance is degraded. For each thread, there are threads that do not work, and some threads do a lot of work. In addition, there are too many threads, so communication overhead occurs, so you can see that the exec time rather increases.

Num_Thread = 1



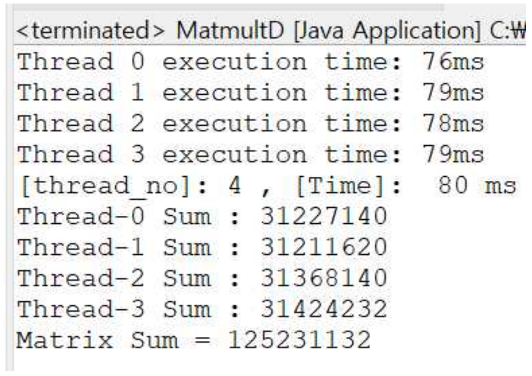
```
<terminated> MatmultD [Java Application] C:₩User
Thread 0 execution time: 177ms
[thread_no]: 1 , [Time]: 178 ms
Thread-0 Sum : 125231132
Matrix Sum = 125231132
```

Num_Thread = 2



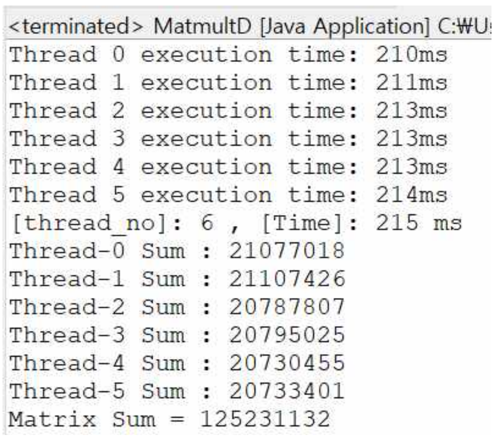
```
<terminated> MatmultD [Java Application] C:₩U
Thread 0 execution time: 110ms
Thread 1 execution time: 110ms
[thread_no]: 2 , [Time]: 110 ms
Thread-0 Sum : 62595280
Thread-1 Sum : 62635852
Matrix Sum = 125231132
```

Num_Thread = 4



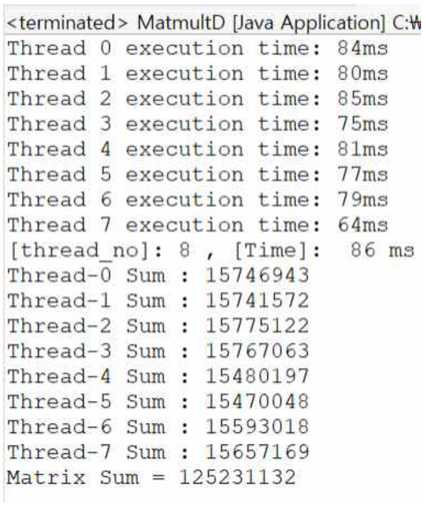
```
<terminated> MatmultD [Java Application] C:₩
Thread 0 execution time: 76ms
Thread 1 execution time: 79ms
Thread 2 execution time: 78ms
Thread 3 execution time: 79ms
[thread_no]: 4 , [Time]:  80 ms
Thread-0 Sum : 31227140
Thread-1 Sum : 31211620
Thread-2 Sum : 31368140
Thread-3 Sum : 31424232
Matrix Sum = 125231132
```

Num_Thread = 6



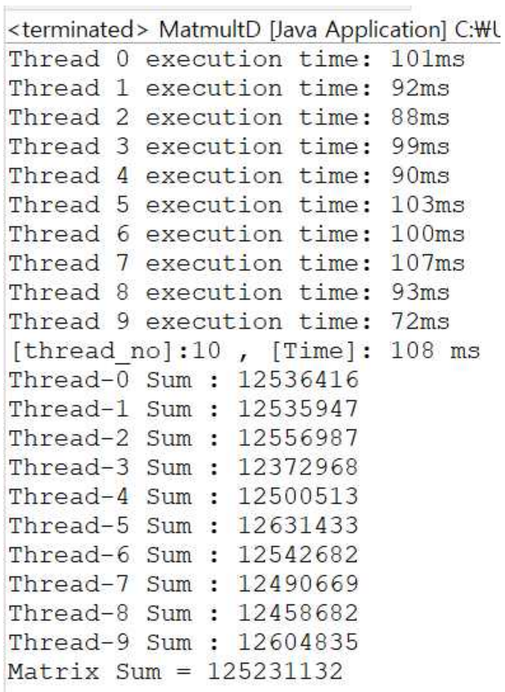
```
<terminated> MatmultD [Java Application] C:₩U
Thread 0 execution time: 210ms
Thread 1 execution time: 211ms
Thread 2 execution time: 213ms
Thread 3 execution time: 213ms
Thread 4 execution time: 213ms
Thread 5 execution time: 214ms
[thread_no]: 6 , [Time]: 215 ms
Thread-0 Sum : 21077018
Thread-1 Sum : 21107426
Thread-2 Sum : 20787807
Thread-3 Sum : 20795025
Thread-4 Sum : 20730455
Thread-5 Sum : 20733401
Matrix Sum = 125231132
```

Num_Thread = 8



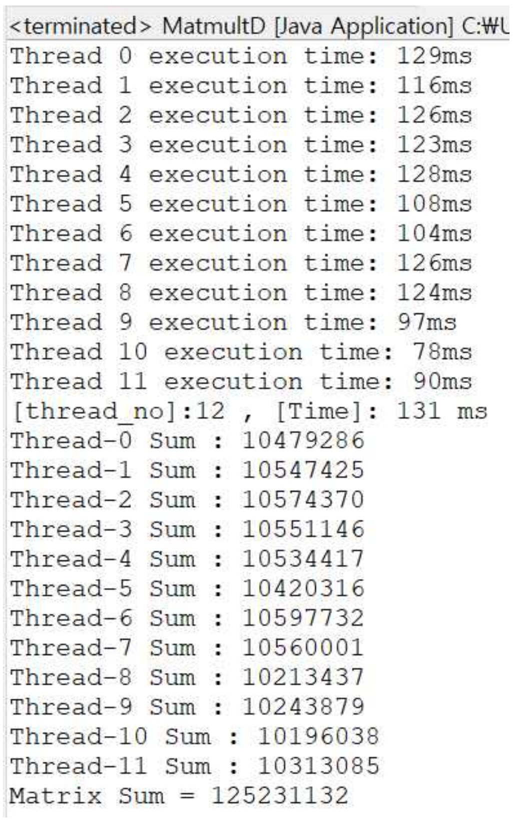
```
<terminated> MatmultD [Java Application] C:₩
Thread 0 execution time: 84ms
Thread 1 execution time: 80ms
Thread 2 execution time: 85ms
Thread 3 execution time: 75ms
Thread 4 execution time: 81ms
Thread 5 execution time: 77ms
Thread 6 execution time: 79ms
Thread 7 execution time: 64ms
[thread_no]: 8 , [Time]:  86 ms
Thread-0 Sum : 15746943
Thread-1 Sum : 15741572
Thread-2 Sum : 15775122
Thread-3 Sum : 15767063
Thread-4 Sum : 15480197
Thread-5 Sum : 15470048
Thread-6 Sum : 15593018
Thread-7 Sum : 15657169
Matrix Sum = 125231132
```

Num_Thread = 10



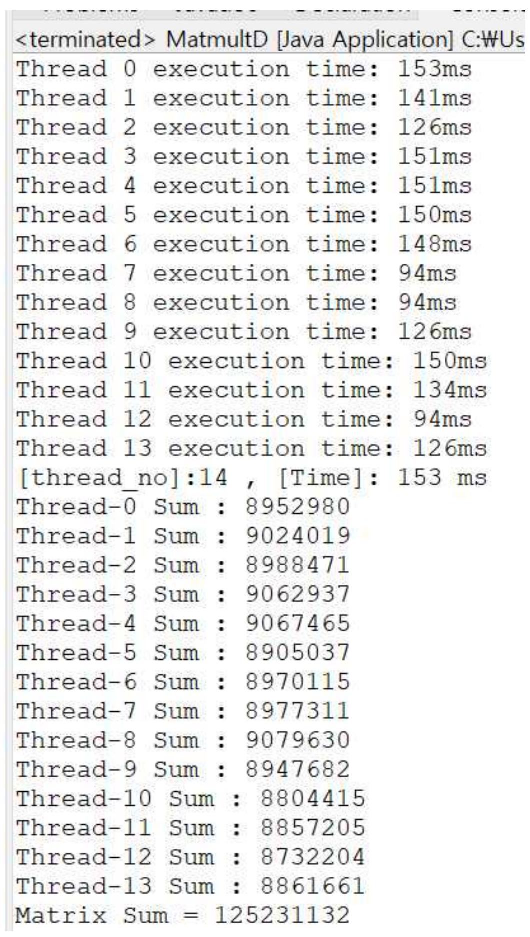
```
<terminated> MatmultD [Java Application] C:₩U
Thread 0 execution time: 101ms
Thread 1 execution time: 92ms
Thread 2 execution time: 88ms
Thread 3 execution time: 99ms
Thread 4 execution time: 90ms
Thread 5 execution time: 103ms
Thread 6 execution time: 100ms
Thread 7 execution time: 107ms
Thread 8 execution time: 93ms
Thread 9 execution time: 72ms
[thread_no]:10 , [Time]: 108 ms
Thread-0 Sum : 12536416
Thread-1 Sum : 12535947
Thread-2 Sum : 12556987
Thread-3 Sum : 12372968
Thread-4 Sum : 12500513
Thread-5 Sum : 12631433
Thread-6 Sum : 12542682
Thread-7 Sum : 12490669
Thread-8 Sum : 12458682
Thread-9 Sum : 12604835
Matrix Sum = 125231132
```

Num_Thread = 12



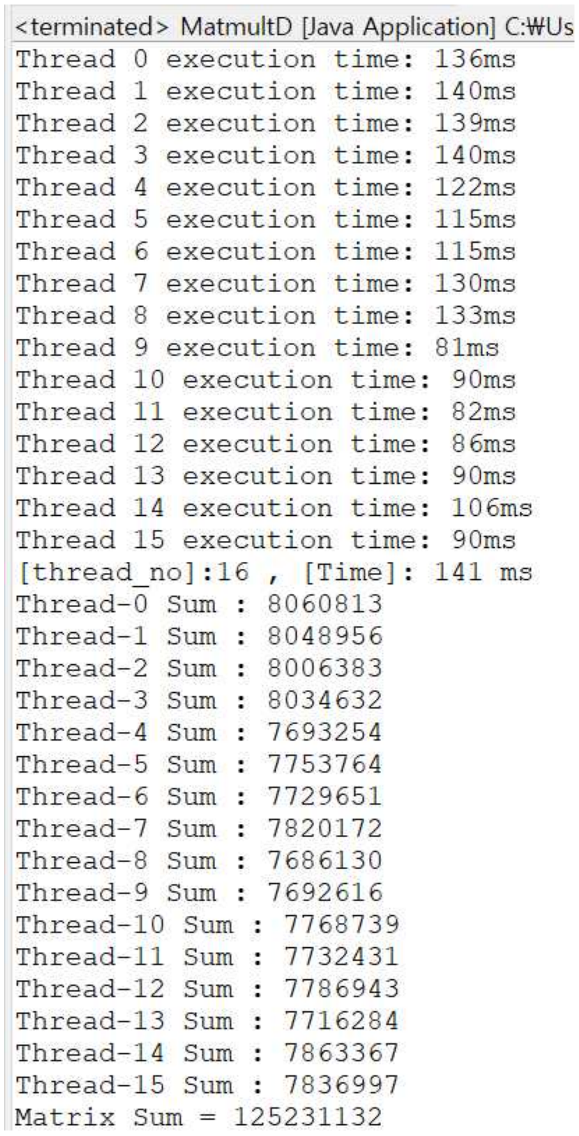
```
<terminated> MatmultD [Java Application] C:₩U
Thread 0 execution time: 129ms
Thread 1 execution time: 116ms
Thread 2 execution time: 126ms
Thread 3 execution time: 123ms
Thread 4 execution time: 128ms
Thread 5 execution time: 108ms
Thread 6 execution time: 104ms
Thread 7 execution time: 126ms
Thread 8 execution time: 124ms
Thread 9 execution time: 97ms
Thread 10 execution time: 78ms
Thread 11 execution time: 90ms
[thread_no]:12 , [Time]: 131 ms
Thread-0 Sum : 10479286
Thread-1 Sum : 10547425
Thread-2 Sum : 10574370
Thread-3 Sum : 10551146
Thread-4 Sum : 10534417
Thread-5 Sum : 10420316
Thread-6 Sum : 10597732
Thread-7 Sum : 10560001
Thread-8 Sum : 10213437
Thread-9 Sum : 10243879
Thread-10 Sum : 10196038
Thread-11 Sum : 10313085
Matrix Sum = 125231132
```

Num_Thread = 14



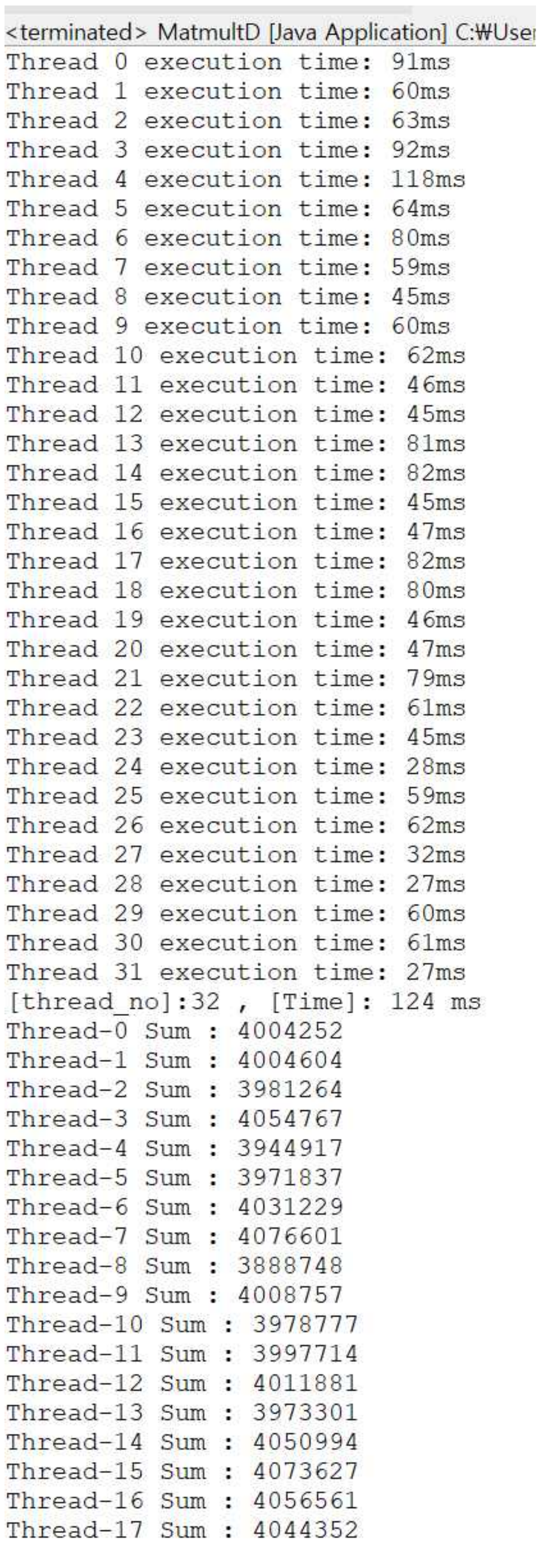
```
<terminated> MatmultD [Java Application] C:₩Us
Thread 0 execution time: 153ms
Thread 1 execution time: 141ms
Thread 2 execution time: 126ms
Thread 3 execution time: 151ms
Thread 4 execution time: 151ms
Thread 5 execution time: 150ms
Thread 6 execution time: 148ms
Thread 7 execution time: 94ms
Thread 8 execution time: 94ms
Thread 9 execution time: 126ms
Thread 10 execution time: 150ms
Thread 11 execution time: 134ms
Thread 12 execution time: 94ms
Thread 13 execution time: 126ms
[thread_no]:14 , [Time]: 153 ms
Thread-0 Sum : 8952980
Thread-1 Sum : 9024019
Thread-2 Sum : 8988471
Thread-3 Sum : 9062937
Thread-4 Sum : 9067465
Thread-5 Sum : 8905037
Thread-6 Sum : 8970115
Thread-7 Sum : 8977311
Thread-8 Sum : 9079630
Thread-9 Sum : 8947682
Thread-10 Sum : 8804415
Thread-11 Sum : 8857205
Thread-12 Sum : 8732204
Thread-13 Sum : 8861661
Matrix Sum = 125231132
```

Num_Thread = 16

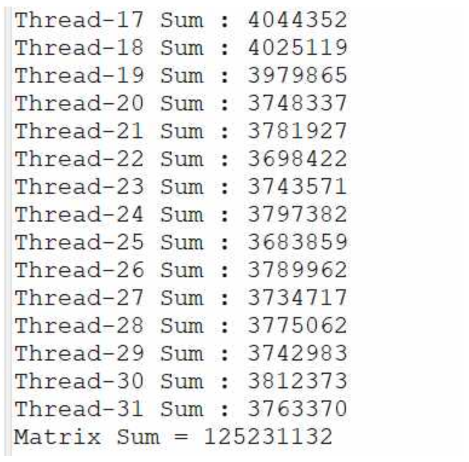


```
Thread 0 execution time: 136ms
Thread 1 execution time: 140ms
Thread 2 execution time: 139ms
Thread 3 execution time: 140ms
Thread 4 execution time: 122ms
Thread 5 execution time: 115ms
Thread 6 execution time: 115ms
Thread 7 execution time: 130ms
Thread 8 execution time: 133ms
Thread 9 execution time: 81ms
Thread 10 execution time: 90ms
Thread 11 execution time: 82ms
Thread 12 execution time: 86ms
Thread 13 execution time: 90ms
Thread 14 execution time: 106ms
Thread 15 execution time: 90ms
[thread_no]:16 , [Time]: 141 ms
Thread-0 Sum : 8060813
Thread-1 Sum : 8048956
Thread-2 Sum : 8006383
Thread-3 Sum : 8034632
Thread-4 Sum : 7693254
Thread-5 Sum : 7753764
Thread-6 Sum : 7729651
Thread-7 Sum : 7820172
Thread-8 Sum : 7686130
Thread-9 Sum : 7692616
Thread-10 Sum : 7768739
Thread-11 Sum : 7732431
Thread-12 Sum : 7786943
Thread-13 Sum : 7716284
Thread-14 Sum : 7863367
Thread-15 Sum : 7836997
Matrix Sum = 125231132
```

Num_Thread = 32



```
<terminated> MatmultD [Java Application] C:₩User
Thread 0 execution time: 91ms
Thread 1 execution time: 60ms
Thread 2 execution time: 63ms
Thread 3 execution time: 92ms
Thread 4 execution time: 118ms
Thread 5 execution time: 64ms
Thread 6 execution time: 80ms
Thread 7 execution time: 59ms
Thread 8 execution time: 45ms
Thread 9 execution time: 60ms
Thread 10 execution time: 62ms
Thread 11 execution time: 46ms
Thread 12 execution time: 45ms
Thread 13 execution time: 81ms
Thread 14 execution time: 82ms
Thread 15 execution time: 45ms
Thread 16 execution time: 47ms
Thread 17 execution time: 82ms
Thread 18 execution time: 80ms
Thread 19 execution time: 46ms
Thread 20 execution time: 47ms
Thread 21 execution time: 79ms
Thread 22 execution time: 61ms
Thread 23 execution time: 45ms
Thread 24 execution time: 28ms
Thread 25 execution time: 59ms
Thread 26 execution time: 62ms
Thread 27 execution time: 32ms
Thread 28 execution time: 27ms
Thread 29 execution time: 60ms
Thread 30 execution time: 61ms
Thread 31 execution time: 27ms
[thread_no]:32 , [Time]: 124 ms
Thread-0 Sum : 4004252
Thread-1 Sum : 4004604
Thread-2 Sum : 3981264
Thread-3 Sum : 4054767
Thread-4 Sum : 3944917
Thread-5 Sum : 3971837
Thread-6 Sum : 4031229
Thread-7 Sum : 4076601
Thread-8 Sum : 3888748
Thread-9 Sum : 4008757
Thread-10 Sum : 3978777
Thread-11 Sum : 3997714
Thread-12 Sum : 4011881
Thread-13 Sum : 3973301
Thread-14 Sum : 4050994
Thread-15 Sum : 4073627
Thread-16 Sum : 4056561
Thread-17 Sum : 4044352
```



```
Thread-17 Sum : 4044352
Thread-18 Sum : 4025119
Thread-19 Sum : 3979865
Thread-20 Sum : 3748337
Thread-21 Sum : 3781927
Thread-22 Sum : 3698422
Thread-23 Sum : 3743571
Thread-24 Sum : 3797382
Thread-25 Sum : 3683859
Thread-26 Sum : 3789962
Thread-27 Sum : 3734717
Thread-28 Sum : 3775062
Thread-29 Sum : 3742983
Thread-30 Sum : 3812373
Thread-31 Sum : 3763370
Matrix Sum = 125231132
```