Multicore Computing

-project #1 : problem1–

과목명	멀티코어 컴퓨팅
교수명	손봉수 교수님
제출일	2023.04.16
학　번	20200283
학　과	소프트웨어학부
이　름	조범희

INDEX(2023 1st Semester Multicore computing)

1. Project Environment/Overview

2. Analysis
- Table
- Graph
- Static load balancing(BLOCK)
- Stack load balancing(CYCLIC)
- Dynamic load balancing

3. Conclusion
- Summary
- Full Results Screenshot

1. Project Environment/Overview

- Environment
- OS : Window11
- Processor : 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
- Memory : Ram 8GB
- IDE : IntelliJ IDEA 2022.1.1

- Overview
- Detecting the Number of Prime Numbers from 1 to 200,000 Using Static Load Balancing (Block, Cyclic), Dynamic Load Balancing, and Multi-Threaded Approach
- The attached code can be run on Java IDE such as IntelliJ or Eclipse.This is the java file.

2. Analysis

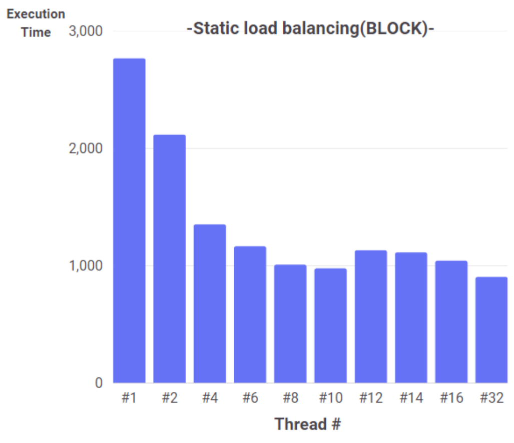
- Result Table

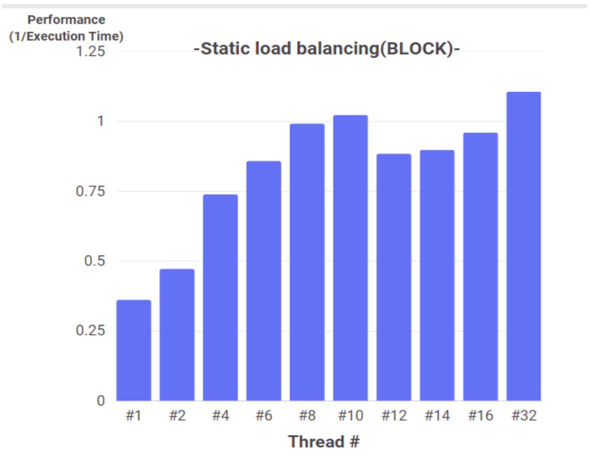
Exec time	1	2	4	6	8
Static(Block)	2770	2118	1353	1165	1008
Static(Cyclic)	2735	1752	1566	1320	1031
Dynamic	3040	2513	1267	1079	983

Exec time	10	12	14	16	32
Static(Block)	977	1131	1114	1042	904
Static(Cyclic)	953	848	1008	914	911
Dynamic	992	901	884	835	975

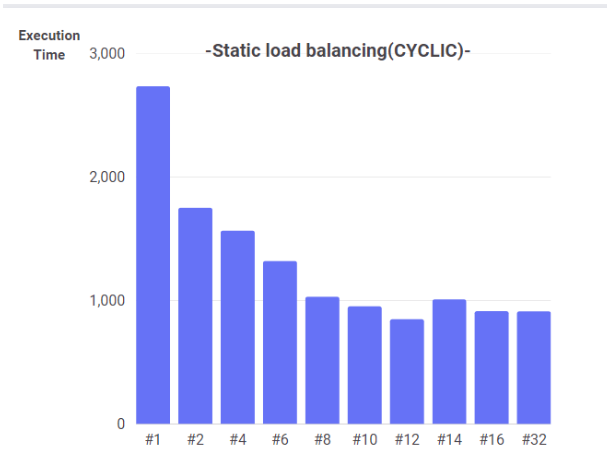
Task size = 10, Unit: ms

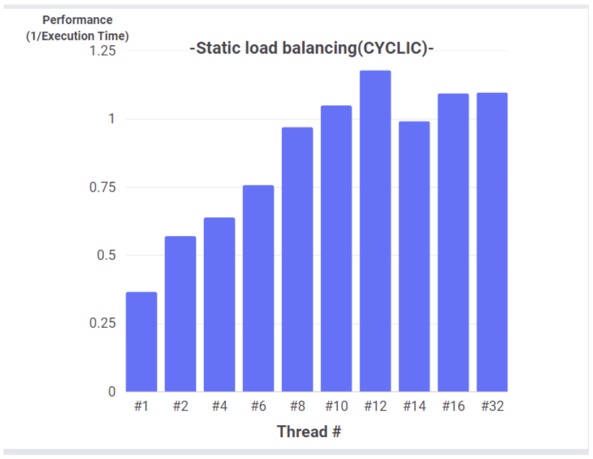
- Result Graph



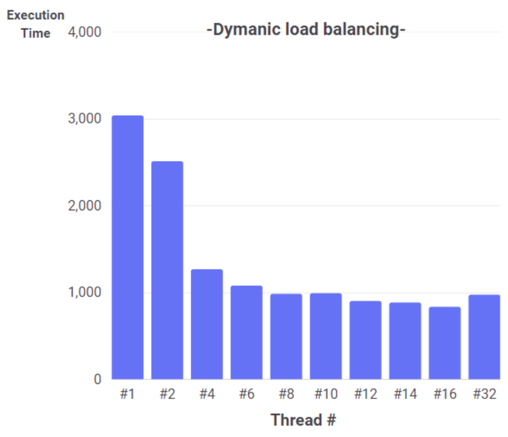


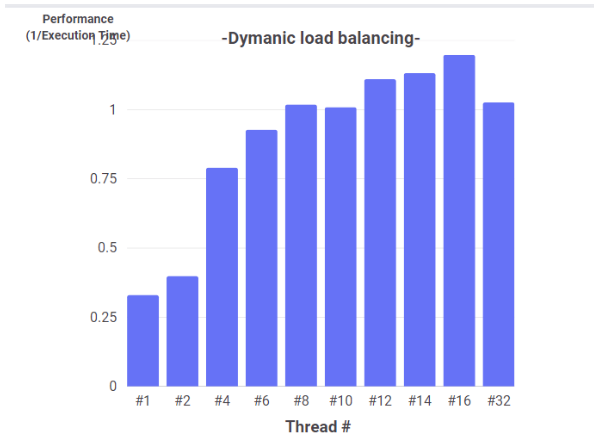
<Static load balancing (BLOCK) Execution Time, Performance>


Execution Time
-Static load balancing(CYCLIC)-
3,000
2,000
1,000
0
#1
#2
#4
#6
#8
#10
#12
#14
#16
#32


Performance (1/Execution Time)
-Static load balancing(CYCLIC)-
1.25
1
0.75
0.5
0.25
0
#1
#2
#4
#6
#8
#10
#12
#14
#16
#32
Thread #

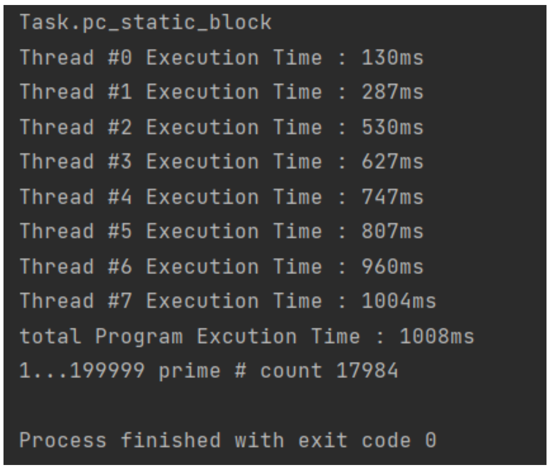
<Static load balancing (CYCLIC) Execution Time, Performance>


Execution Time
-Dymanic load balancing-
4,000
3,000
2,000
1,000
0
#1
#2
#4
#6
#8
#10
#12
#14
#16
#32
Thread #


Performance (1/Execution Time)
-Dymanic load balancing-
1.25
1
0.75
0.5
0.25
0
#1
#2
#4
#6
#8
#10
#12
#14
#16
#32
Thread #

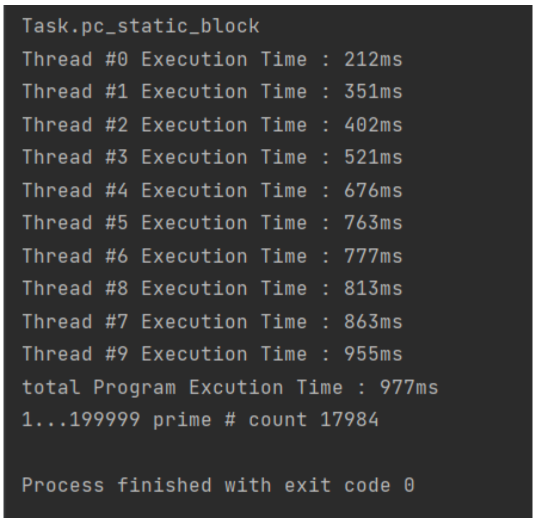
<Dynamic load balancing Execution Time, Performance>

- Static load balancing (BLOCK)



```
Task.pc_static_block
Thread #0 Execution Time : 130ms
Thread #1 Execution Time : 287ms
Thread #2 Execution Time : 530ms
Thread #3 Execution Time : 627ms
Thread #4 Execution Time : 747ms
Thread #5 Execution Time : 807ms
Thread #6 Execution Time : 960ms
Thread #7 Execution Time : 1004ms
total Program Excution Time : 1008ms
1...199999 prime # count 17984

Process finished with exit code 0
```

<Thread #8>



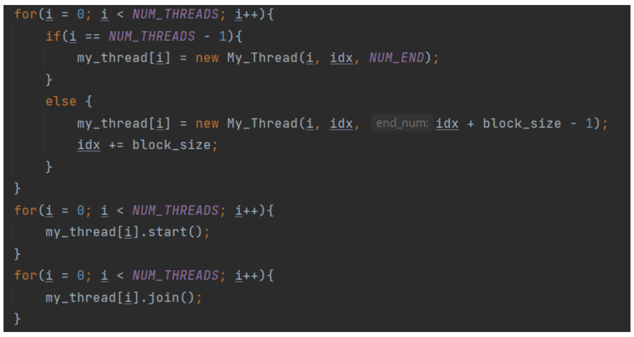
```
Task.pc_static_block
Thread #0 Execution Time : 212ms
Thread #1 Execution Time : 351ms
Thread #2 Execution Time : 402ms
Thread #3 Execution Time : 521ms
Thread #4 Execution Time : 676ms
Thread #5 Execution Time : 763ms
Thread #6 Execution Time : 777ms
Thread #8 Execution Time : 813ms
Thread #7 Execution Time : 863ms
Thread #9 Execution Time : 955ms
total Program Excution Time : 977ms
1...199999 prime # count 17984

Process finished with exit code 0
```

< Thread #10>

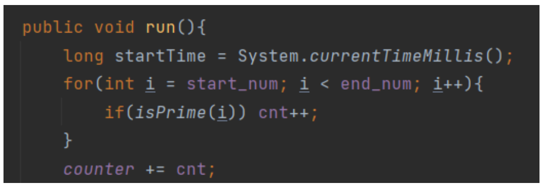
- Static load balancing is a technique where the workload is assigned at compile time. In other words, the programmer allocates the work in the code using a decomposition approach. Block-based load balancing may not be the best choice for this technique, but it is a low-overhead approach. On the other hand, cyclic-based load balancing may result in significant overhead, but it is a good choice for load balancing.

- Block-based load balancing is a technique where the given workload is divided into blocks based on the number of threads available for processing. In this problem, we allocated 200,000 numbers to each thread. This approach is relatively easy to implement and uses a multi-threaded approach, but it is not an efficient load balancing technique. As shown in the results above, we can see that the execution time among threads is significantly different, indicating poor load balancing.

- Looking at the table above, the longer the number of threads, the shorter the program execution time, but more than 10 times, it was similar or rather longer. The computer's process has four cores and eight logical threads, which are thought to be due to the excessive number of threads, resulting in excessive switching and overhead of threads.



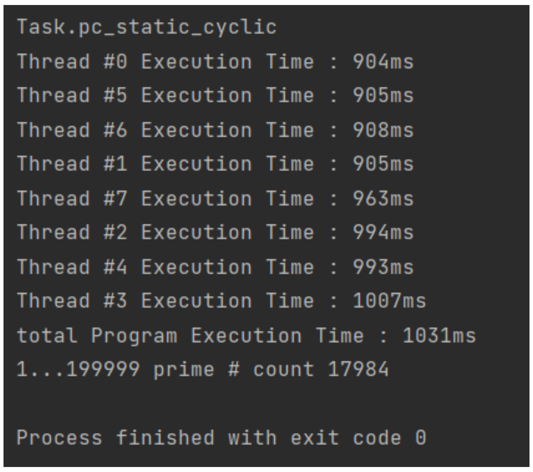
```
for(i = 0; i < NUM_THREADS; i++){
    if(i == NUM_THREADS - 1){
        my_thread[i] = new My_Thread(i, idx, NUM_END);
    }
    else {
        my_thread[i] = new My_Thread(i, idx,  end_num: idx + block_size - 1);
        idx += block_size;
    }
}
for(i = 0; i < NUM_THREADS; i++){
    my_thread[i].start();
}
for(i = 0; i < NUM_THREADS; i++){
    my_thread[i].join();
}
```



```
public void run(){
    long startTime = System.currentTimeMillis();
    for(int i = start_num; i < end_num; i++){
        if(isPrime(i)) cnt++;
    }
    counter += cnt;
```

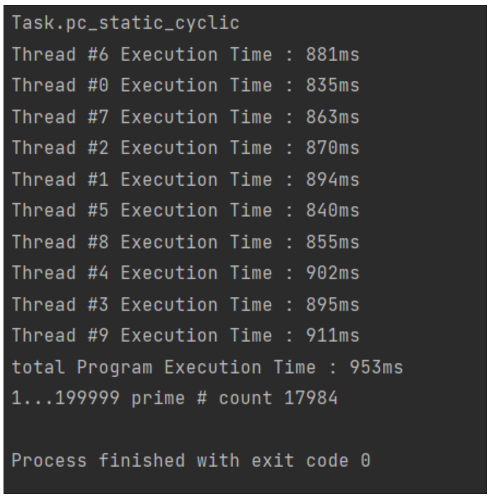
- The main function declares a thread and allocates a block. It then determines the prime number from the "run" of the thread and adds a value to the global variable "counter".

- Static load balancing (CYCLIC)



```
Task.pc_static_cyclic
Thread #0 Execution Time : 904ms
Thread #5 Execution Time : 905ms
Thread #6 Execution Time : 908ms
Thread #1 Execution Time : 905ms
Thread #7 Execution Time : 963ms
Thread #2 Execution Time : 994ms
Thread #4 Execution Time : 993ms
Thread #3 Execution Time : 1007ms
total Program Execution Time : 1031ms
1...199999 prime # count 17984

Process finished with exit code 0
```

< Thread #8>

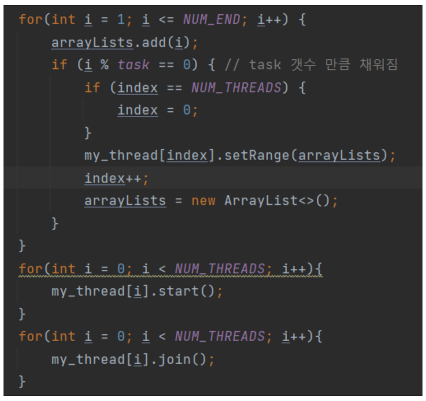


```
Task.pc_static_cyclic
Thread #6 Execution Time : 881ms
Thread #0 Execution Time : 835ms
Thread #7 Execution Time : 863ms
Thread #2 Execution Time : 870ms
Thread #1 Execution Time : 894ms
Thread #5 Execution Time : 840ms
Thread #8 Execution Time : 855ms
Thread #4 Execution Time : 902ms
Thread #3 Execution Time : 895ms
Thread #9 Execution Time : 911ms
total Program Execution Time : 953ms
1...199999 prime # count 17984

Process finished with exit code 0
```

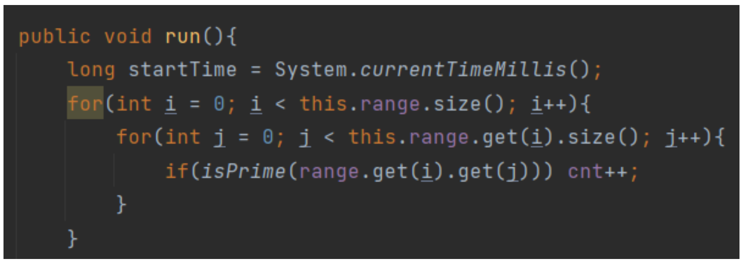
< Thread #10>

- Although it is a static load balancing method as above, the cyclic method is also affected by the number of threads and the number of tasks. Since there are 10 tasks here, 1 to 10, 11 to 20.. Repeat and assign to each thread. Looking at the execution time of each thread above, it can be seen that the load balancing is significantly improved compared to the block method. In addition, the overall time also showed better performance in cycles than in blocks.

- Looking at the table, we can see that the performance improves up to 12 threads, but doesn't improve much beyond that. This is likely due to the same reason as the block method. The difference is that while the block method didn't see much improvement beyond 10 threads, this time we see improvement up to 12 threads. This is likely due to better load balancing with the cyclic method.



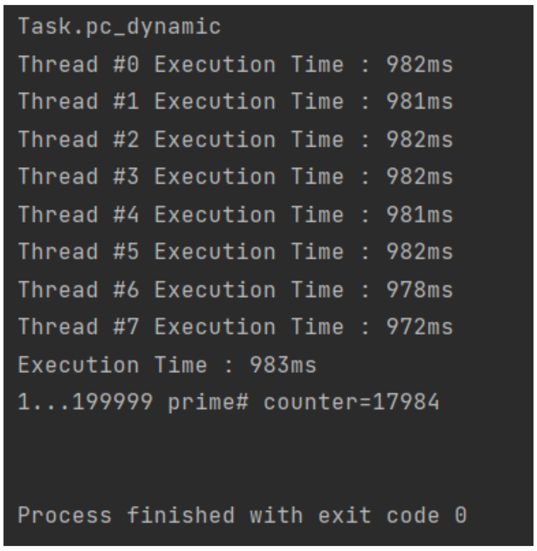
```
for(int i = 1; i <= NUM_END; i++) {
    arrayLists.add(i);
    if (i % task == 0) { // task 갯수 만큼 채워짐
        if (index == NUM_THREADS) {
            index = 0;
        }
        my_thread[index].setRange(arrayLists);
        index++;
        arrayLists = new ArrayList<>();
    }
}
for(int i = 0; i < NUM_THREADS; i++){
    my_thread[i].start();
}
for(int i = 0; i < NUM_THREADS; i++){
    my_thread[i].join();
}
```



```
public void run(){
    long startTime = System.currentTimeMillis();
    for(int i = 0; i < this.range.size(); i++){
        for(int j = 0; j < this.range.get(i).size(); j++){
            if(isPrime(range.get(i).get(j))) cnt++;
        }
    }
```

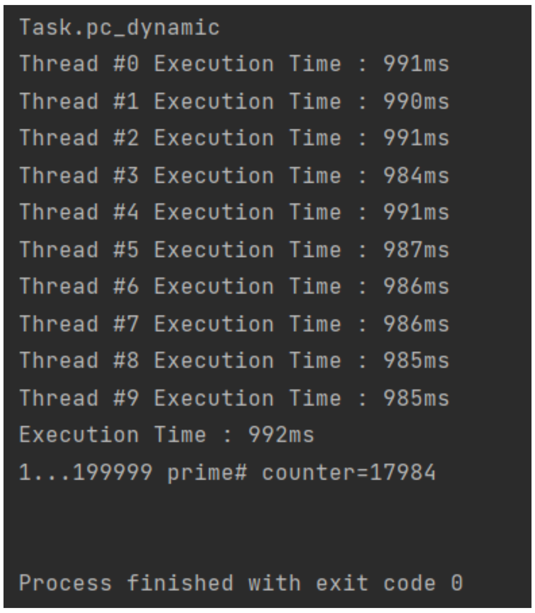
- The task allocation code has become more complex. In the case of block allocation, we only needed to divide END_NUM by the number of threads, but here we need to divide the number of tasks and assign them in order. In this code, we gave each thread an arrayList<arrayList<int>> to allocate the work items. I was worried that using ArrayList would slow down the performance due to access time, but it didn't have as much impact as I thought.

- Dynamic load balancing



```
Task.pc_dynamic
Thread #0 Execution Time : 982ms
Thread #1 Execution Time : 981ms
Thread #2 Execution Time : 982ms
Thread #3 Execution Time : 982ms
Thread #4 Execution Time : 981ms
Thread #5 Execution Time : 982ms
Thread #6 Execution Time : 978ms
Thread #7 Execution Time : 972ms
Execution Time : 983ms
1...199999 prime# counter=17984


Process finished with exit code 0
```

<Thread #8>

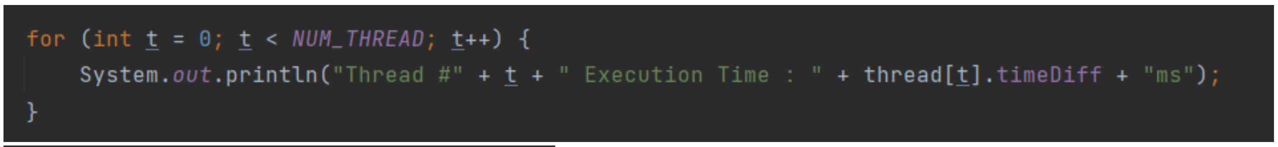


```
Task.pc_dynamic
Thread #0 Execution Time : 991ms
Thread #1 Execution Time : 990ms
Thread #2 Execution Time : 991ms
Thread #3 Execution Time : 984ms
Thread #4 Execution Time : 991ms
Thread #5 Execution Time : 987ms
Thread #6 Execution Time : 986ms
Thread #7 Execution Time : 986ms
Thread #8 Execution Time : 985ms
Thread #9 Execution Time : 985ms
Execution Time : 992ms
1...199999 prime# counter=17984


Process finished with exit code 0
```

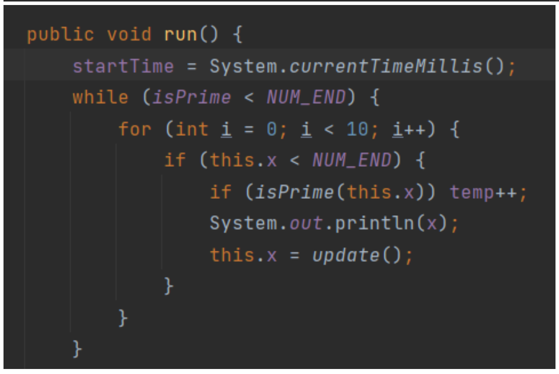
< Thread #10>

- Dynamic load balancing, unlike static load balancing, is assigned tasks at runtime. In other words, it is not clear exactly what work will be assigned to each thread because the programmer created only creates situations. In addition, the difference between dynamic load balancing and static load balancing is the synchronization problem. It is important to synchronize properly because threads continue to access the same resources.

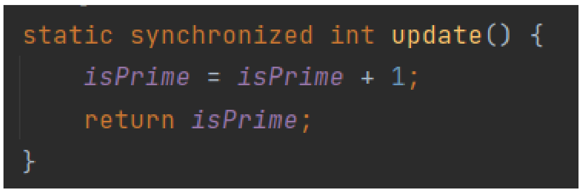
- Looking at the screenshot, we can see that load balancing is significantly better than static load balancing cyc method. The workload is evenly distributed across threads, resulting in improved performance for up to 16 threads, which is a significant improvement compared to the static method. In my opinion, dynamic load balancing may have incurred synchronization overhead, so I expected the execution time to be slower even with load balancing. However, the results were better than expected.



```
for (int t = 0; t < NUM_THREAD; t++) {
    System.out.println("Thread #" + t + " Execution Time : " + thread[t].timeDiff + "ms");
}
```



```
public void run() {
    startTime = System.currentTimeMillis();
    while (isPrime < NUM_END) {
        for (int i = 0; i < 10; i++) {
            if (this.x < NUM_END) {
                if (isPrime(this.x)) temp++;
                System.out.println(x);
                this.x = update();
            }
        }
    }
```



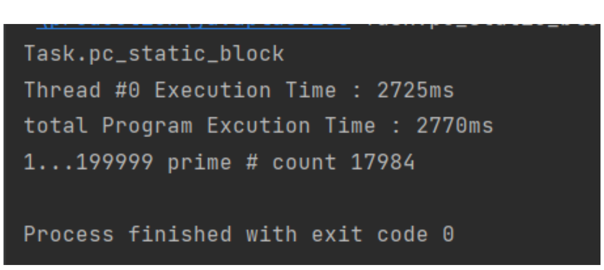
```
static synchronized int update() {
    isPrime = isPrime + 1;
    return isPrime;
}
```

- In this implementation, unlike static load balancing, there is no code assigned by the main. Because it is assigned at runtime. If you look at the code, since there are 10 Tasks, you work 10 times each, determine the prime number, and update the isPrime variable through the update function. This is because isPrime is a common variable between threads. If a synchronization problem occurs here, the result will be a fatal error.

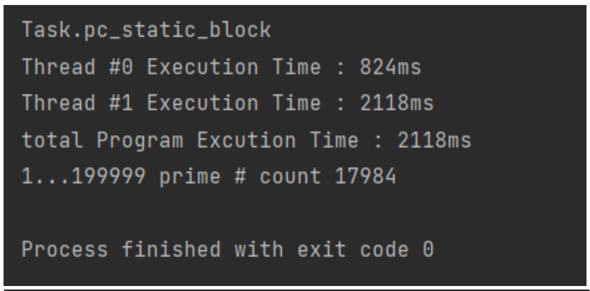
3. Conclusion

- Multi-thread programming shows faster performance compared to single threads. However, you should use a composition method for each situation. There were parts that fit my expectations and parts that didn't fit, but I think I need to study more deeply.

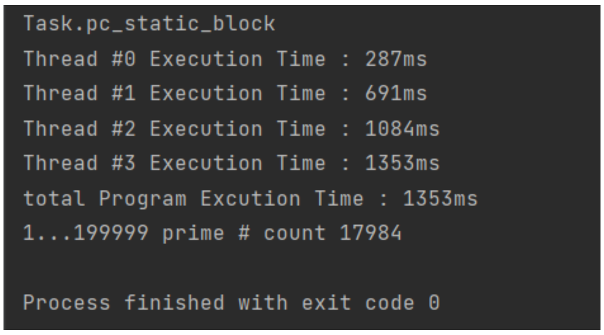
- All Results Screenshot



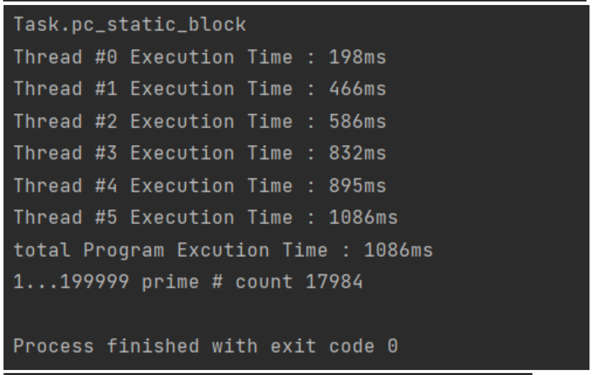
```
Task.pc_static_block
Thread #0 Execution Time : 2725ms
total Program Excution Time : 2770ms
1...199999 prime # count 17984

Process finished with exit code 0
```



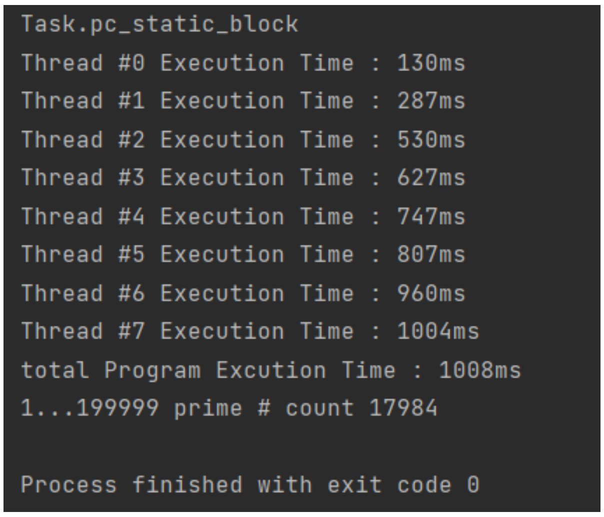
```
Task.pc_static_block
Thread #0 Execution Time : 824ms
Thread #1 Execution Time : 2118ms
total Program Excution Time : 2118ms
1...199999 prime # count 17984

Process finished with exit code 0
```



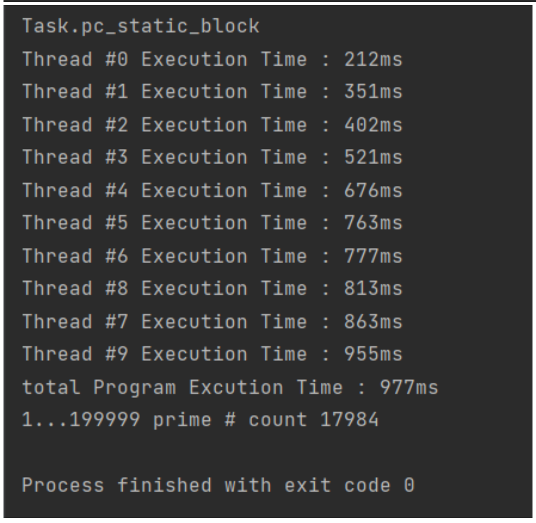
```
Task.pc_static_block
Thread #0 Execution Time : 287ms
Thread #1 Execution Time : 691ms
Thread #2 Execution Time : 1084ms
Thread #3 Execution Time : 1353ms
total Program Excution Time : 1353ms
1...199999 prime # count 17984

Process finished with exit code 0
```



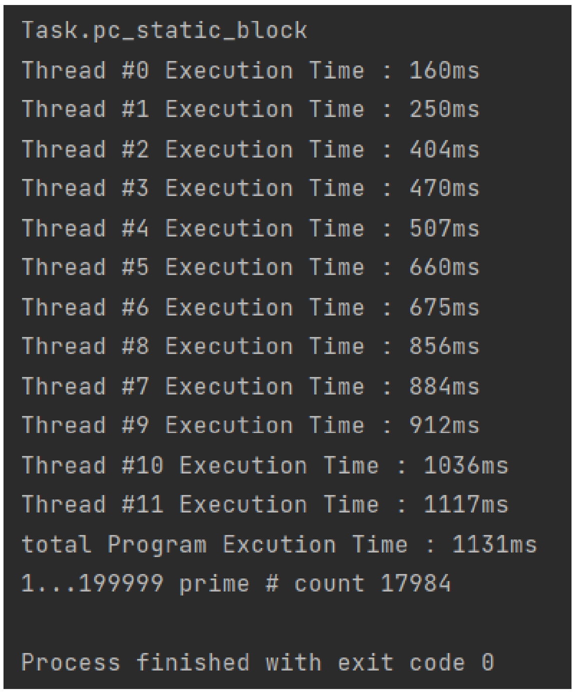
```
Task.pc_static_block
Thread #0 Execution Time : 198ms
Thread #1 Execution Time : 466ms
Thread #2 Execution Time : 586ms
Thread #3 Execution Time : 832ms
Thread #4 Execution Time : 895ms
Thread #5 Execution Time : 1086ms
total Program Excution Time : 1086ms
1...199999 prime # count 17984

Process finished with exit code 0
```



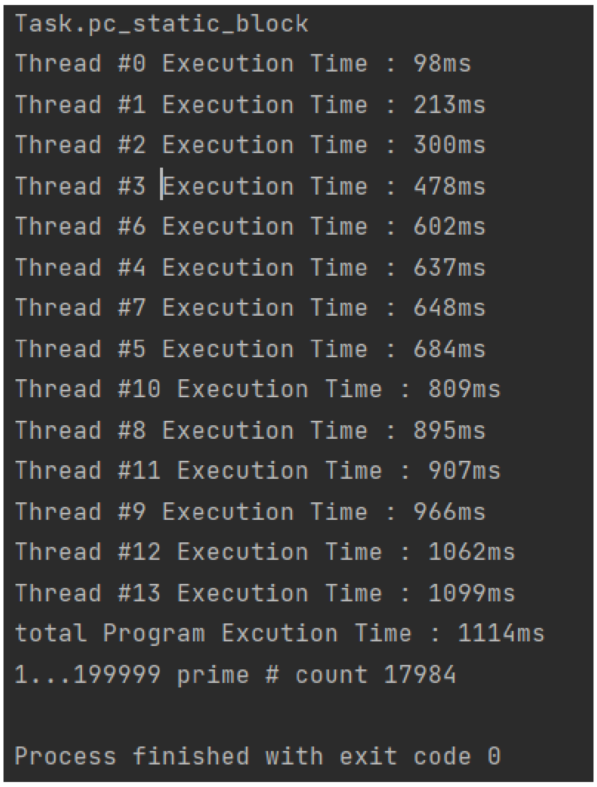
```
Task.pc_static_block
Thread #0 Execution Time : 130ms
Thread #1 Execution Time : 287ms
Thread #2 Execution Time : 530ms
Thread #3 Execution Time : 627ms
Thread #4 Execution Time : 747ms
Thread #5 Execution Time : 807ms
Thread #6 Execution Time : 960ms
Thread #7 Execution Time : 1004ms
total Program Excution Time : 1008ms
1...199999 prime # count 17984

Process finished with exit code 0
```



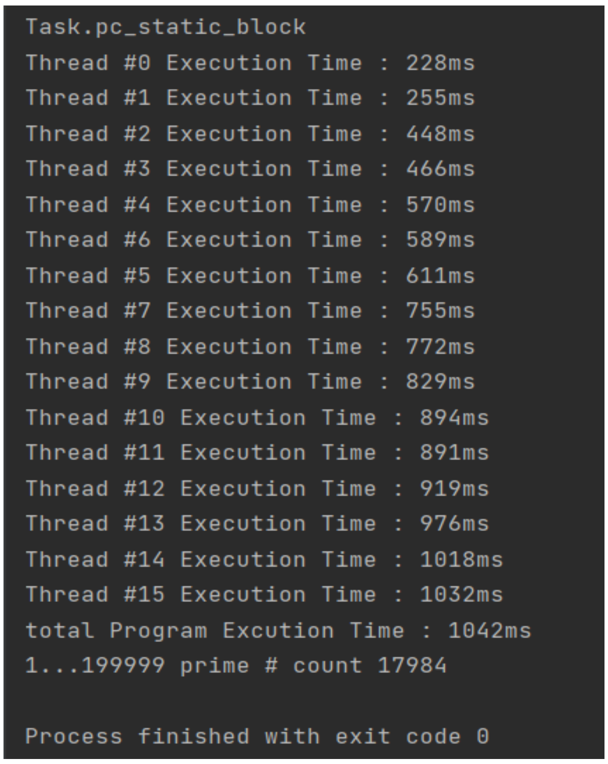
```
Task.pc_static_block
Thread #0 Execution Time : 212ms
Thread #1 Execution Time : 351ms
Thread #2 Execution Time : 402ms
Thread #3 Execution Time : 521ms
Thread #4 Execution Time : 676ms
Thread #5 Execution Time : 763ms
Thread #6 Execution Time : 777ms
Thread #8 Execution Time : 813ms
Thread #7 Execution Time : 863ms
Thread #9 Execution Time : 955ms
total Program Excution Time : 977ms
1...199999 prime # count 17984

Process finished with exit code 0
```



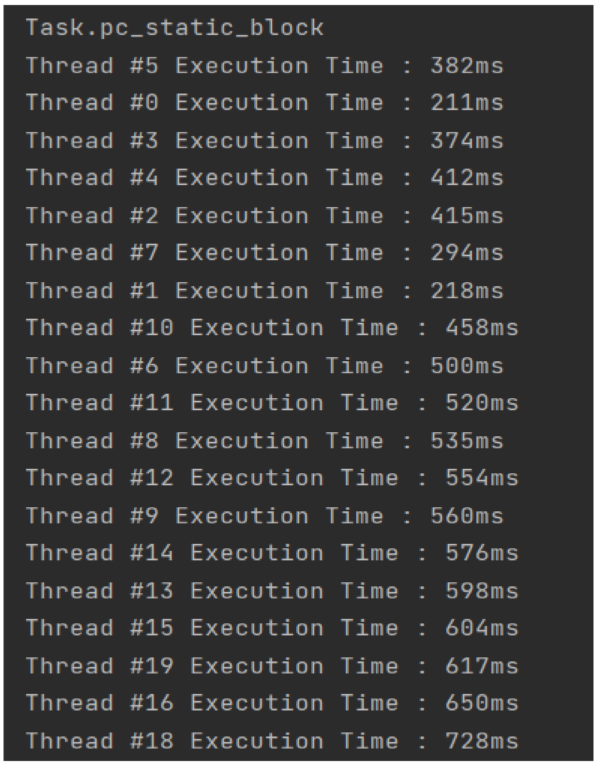
```
Task.pc_static_block
Thread #0 Execution Time : 160ms
Thread #1 Execution Time : 250ms
Thread #2 Execution Time : 404ms
Thread #3 Execution Time : 470ms
Thread #4 Execution Time : 507ms
Thread #5 Execution Time : 660ms
Thread #6 Execution Time : 675ms
Thread #8 Execution Time : 856ms
Thread #7 Execution Time : 884ms
Thread #9 Execution Time : 912ms
Thread #10 Execution Time : 1036ms
Thread #11 Execution Time : 1117ms
total Program Excution Time : 1131ms
1...199999 prime # count 17984

Process finished with exit code 0
```



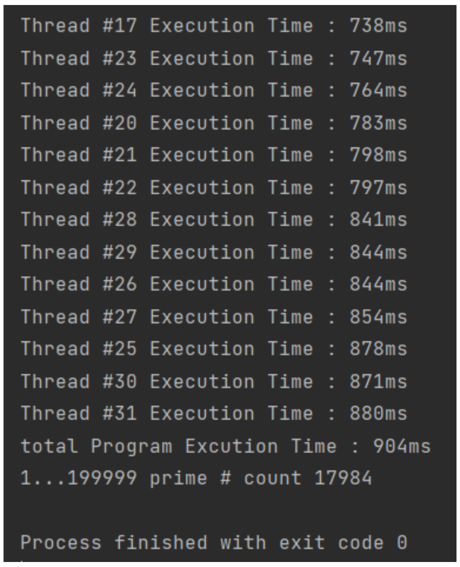
```
Task.pc_static_block
Thread #0 Execution Time : 98ms
Thread #1 Execution Time : 213ms
Thread #2 Execution Time : 300ms
Thread #3 Execution Time : 478ms
Thread #6 Execution Time : 602ms
Thread #4 Execution Time : 637ms
Thread #7 Execution Time : 648ms
Thread #5 Execution Time : 684ms
Thread #10 Execution Time : 809ms
Thread #8 Execution Time : 895ms
Thread #11 Execution Time : 907ms
Thread #9 Execution Time : 966ms
Thread #12 Execution Time : 1062ms
Thread #13 Execution Time : 1099ms
total Program Excution Time : 1114ms
1...199999 prime # count 17984

Process finished with exit code 0
```



```
Task.pc_static_block
Thread #0 Execution Time : 228ms
Thread #1 Execution Time : 255ms
Thread #2 Execution Time : 448ms
Thread #3 Execution Time : 466ms
Thread #4 Execution Time : 570ms
Thread #6 Execution Time : 589ms
Thread #5 Execution Time : 611ms
Thread #7 Execution Time : 755ms
Thread #8 Execution Time : 772ms
Thread #9 Execution Time : 829ms
Thread #10 Execution Time : 894ms
Thread #11 Execution Time : 891ms
Thread #12 Execution Time : 919ms
Thread #13 Execution Time : 976ms
Thread #14 Execution Time : 1018ms
Thread #15 Execution Time : 1032ms
total Program Excution Time : 1042ms
1...199999 prime # count 17984

Process finished with exit code 0
```

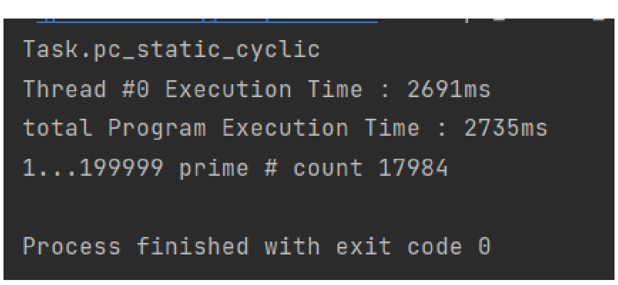


```
Task.pc_static_block
Thread #5 Execution Time : 382ms
Thread #0 Execution Time : 211ms
Thread #3 Execution Time : 374ms
Thread #4 Execution Time : 412ms
Thread #2 Execution Time : 415ms
Thread #7 Execution Time : 294ms
Thread #1 Execution Time : 218ms
Thread #10 Execution Time : 458ms
Thread #6 Execution Time : 500ms
Thread #11 Execution Time : 520ms
Thread #8 Execution Time : 535ms
Thread #12 Execution Time : 554ms
Thread #9 Execution Time : 560ms
Thread #14 Execution Time : 576ms
Thread #13 Execution Time : 598ms
Thread #15 Execution Time : 604ms
Thread #19 Execution Time : 617ms
Thread #16 Execution Time : 650ms
Thread #18 Execution Time : 728ms
```

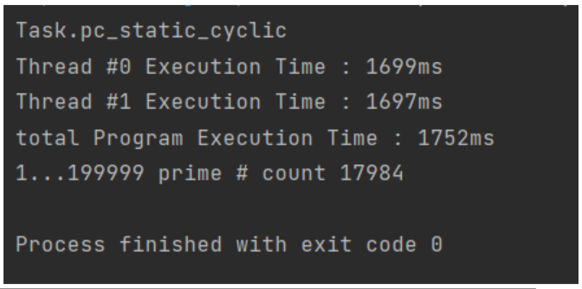


```
Thread #17 Execution Time : 738ms
Thread #23 Execution Time : 747ms
Thread #24 Execution Time : 764ms
Thread #20 Execution Time : 783ms
Thread #21 Execution Time : 798ms
Thread #22 Execution Time : 797ms
Thread #28 Execution Time : 841ms
Thread #29 Execution Time : 844ms
Thread #26 Execution Time : 844ms
Thread #27 Execution Time : 854ms
Thread #25 Execution Time : 878ms
Thread #30 Execution Time : 871ms
Thread #31 Execution Time : 880ms
total Program Excution Time : 904ms
1...199999 prime # count 17984

Process finished with exit code 0
```

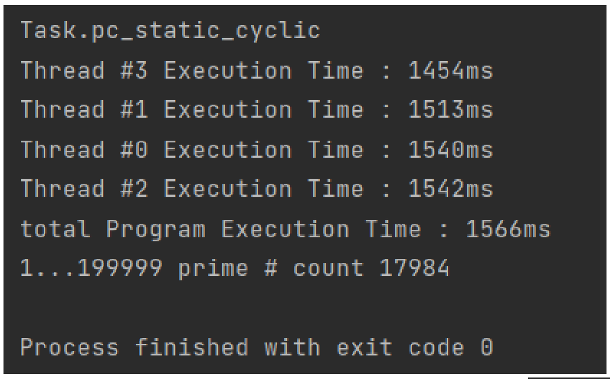
<Static load balancing(Block) Execution Time ScreanShot>



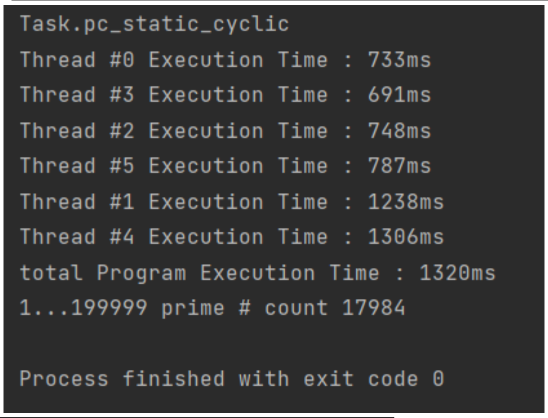
```
Task.pc_static_cyclic
Thread #0 Execution Time : 2691ms
total Program Execution Time : 2735ms
1...199999 prime # count 17984

Process finished with exit code 0
```



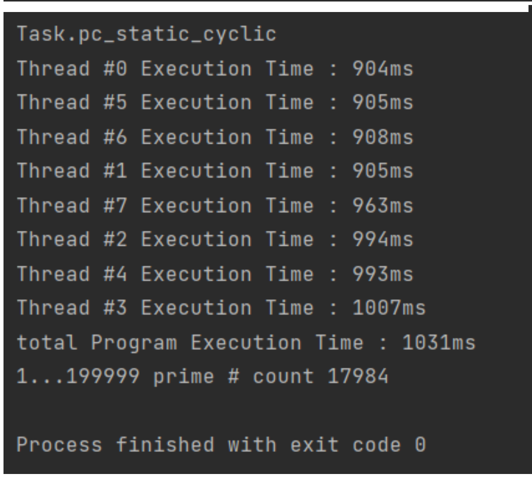
```
Task.pc_static_cyclic
Thread #0 Execution Time : 1699ms
Thread #1 Execution Time : 1697ms
total Program Execution Time : 1752ms
1...199999 prime # count 17984

Process finished with exit code 0
```



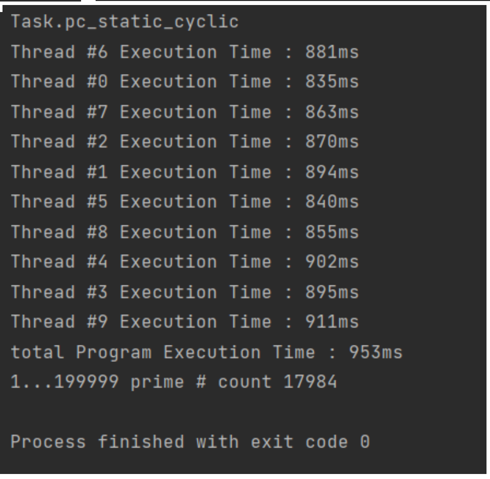
```
Task.pc_static_cyclic
Thread #3 Execution Time : 1454ms
Thread #1 Execution Time : 1513ms
Thread #0 Execution Time : 1540ms
Thread #2 Execution Time : 1542ms
total Program Execution Time : 1566ms
1...199999 prime # count 17984

Process finished with exit code 0
```



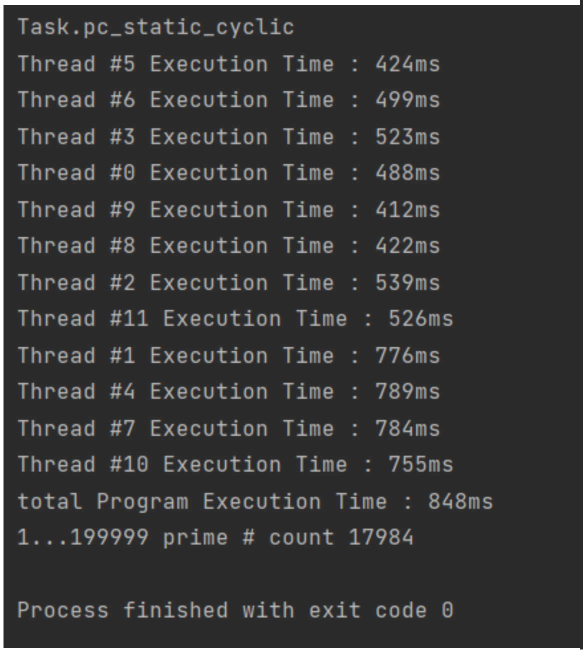
```
Task.pc_static_cyclic
Thread #0 Execution Time : 733ms
Thread #3 Execution Time : 691ms
Thread #2 Execution Time : 748ms
Thread #5 Execution Time : 787ms
Thread #1 Execution Time : 1238ms
Thread #4 Execution Time : 1306ms
total Program Execution Time : 1320ms
1...199999 prime # count 17984

Process finished with exit code 0
```



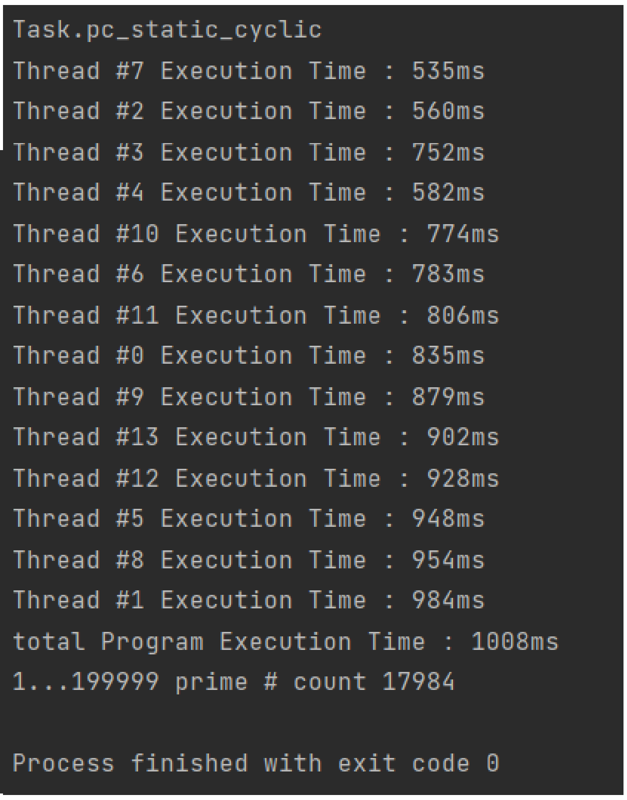
```
Task.pc_static_cyclic
Thread #0 Execution Time : 904ms
Thread #5 Execution Time : 905ms
Thread #6 Execution Time : 908ms
Thread #1 Execution Time : 905ms
Thread #7 Execution Time : 963ms
Thread #2 Execution Time : 994ms
Thread #4 Execution Time : 993ms
Thread #3 Execution Time : 1007ms
total Program Execution Time : 1031ms
1...199999 prime # count 17984

Process finished with exit code 0
```



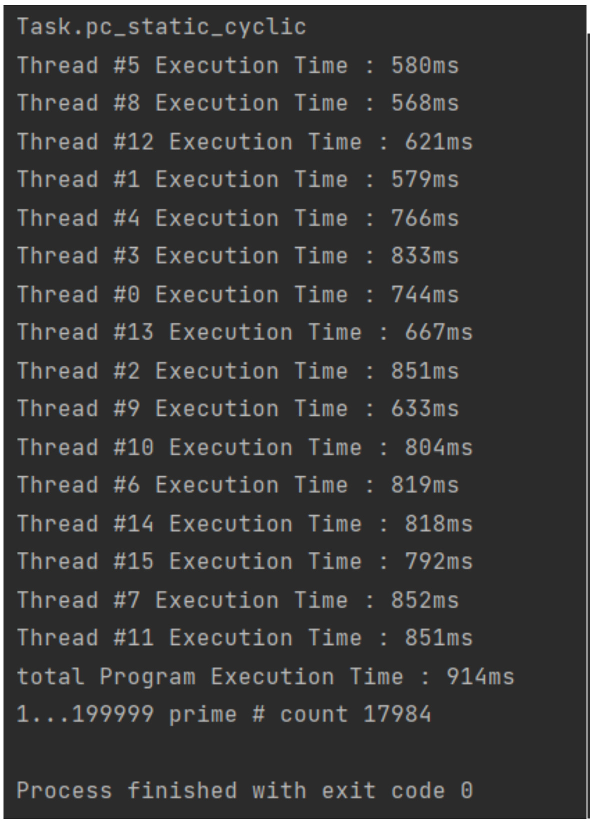
```
Task.pc_static_cyclic
Thread #6 Execution Time : 881ms
Thread #0 Execution Time : 835ms
Thread #7 Execution Time : 863ms
Thread #2 Execution Time : 870ms
Thread #1 Execution Time : 894ms
Thread #5 Execution Time : 840ms
Thread #8 Execution Time : 855ms
Thread #4 Execution Time : 902ms
Thread #3 Execution Time : 895ms
Thread #9 Execution Time : 911ms
total Program Execution Time : 953ms
1...199999 prime # count 17984

Process finished with exit code 0
```



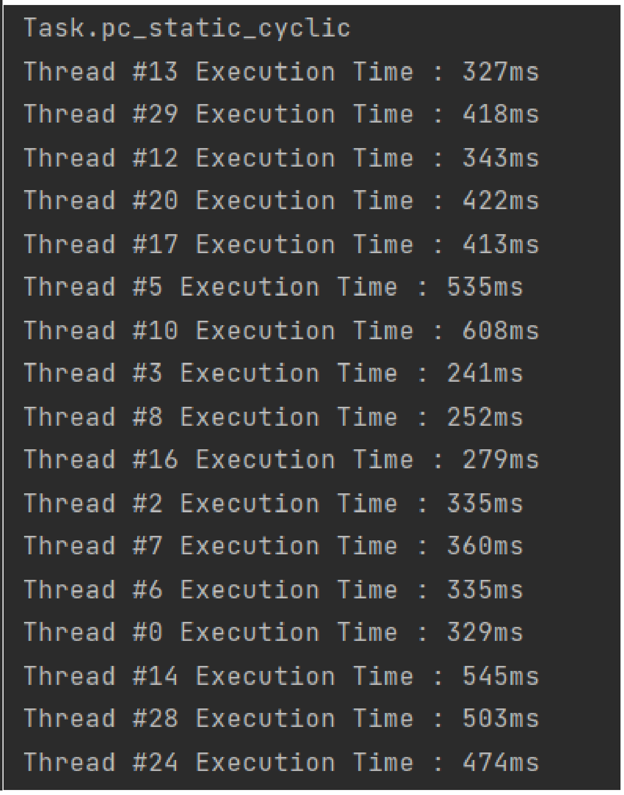
```
Task.pc_static_cyclic
Thread #5 Execution Time : 424ms
Thread #6 Execution Time : 499ms
Thread #3 Execution Time : 523ms
Thread #0 Execution Time : 488ms
Thread #9 Execution Time : 412ms
Thread #8 Execution Time : 422ms
Thread #2 Execution Time : 539ms
Thread #11 Execution Time : 526ms
Thread #1 Execution Time : 776ms
Thread #4 Execution Time : 789ms
Thread #7 Execution Time : 784ms
Thread #10 Execution Time : 755ms
total Program Execution Time : 848ms
1...199999 prime # count 17984

Process finished with exit code 0
```



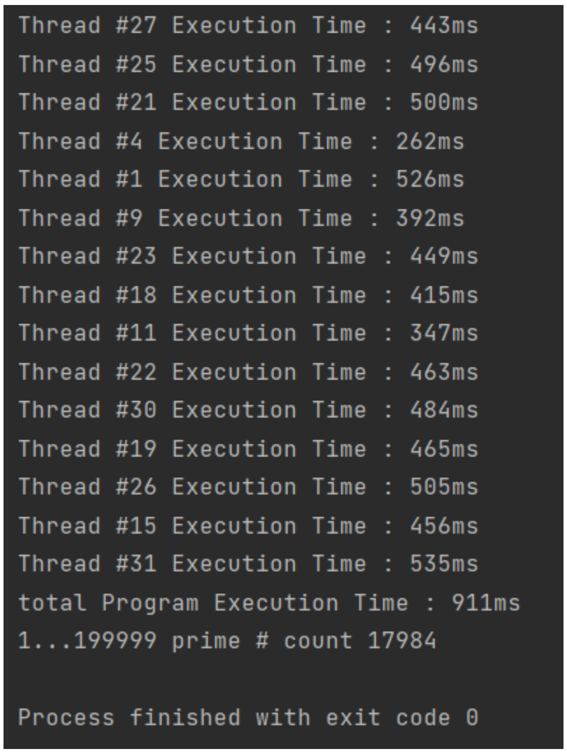
```
Task.pc_static_cyclic
Thread #7 Execution Time : 535ms
Thread #2 Execution Time : 560ms
Thread #3 Execution Time : 752ms
Thread #4 Execution Time : 582ms
Thread #10 Execution Time : 774ms
Thread #6 Execution Time : 783ms
Thread #11 Execution Time : 806ms
Thread #0 Execution Time : 835ms
Thread #9 Execution Time : 879ms
Thread #13 Execution Time : 902ms
Thread #12 Execution Time : 928ms
Thread #5 Execution Time : 948ms
Thread #8 Execution Time : 954ms
Thread #1 Execution Time : 984ms
total Program Execution Time : 1008ms
1...199999 prime # count 17984

Process finished with exit code 0
```



```
Task.pc_static_cyclic
Thread #5 Execution Time : 580ms
Thread #8 Execution Time : 568ms
Thread #12 Execution Time : 621ms
Thread #1 Execution Time : 579ms
Thread #4 Execution Time : 766ms
Thread #3 Execution Time : 833ms
Thread #0 Execution Time : 744ms
Thread #13 Execution Time : 667ms
Thread #2 Execution Time : 851ms
Thread #9 Execution Time : 633ms
Thread #10 Execution Time : 804ms
Thread #6 Execution Time : 819ms
Thread #14 Execution Time : 818ms
Thread #15 Execution Time : 792ms
Thread #7 Execution Time : 852ms
Thread #11 Execution Time : 851ms
total Program Execution Time : 914ms
1...199999 prime # count 17984

Process finished with exit code 0
```

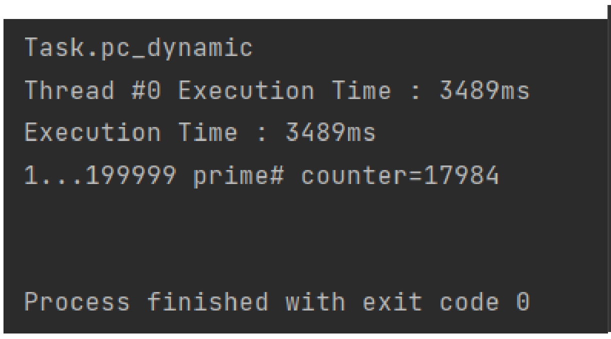


```
Task.pc_static_cyclic
Thread #13 Execution Time : 327ms
Thread #29 Execution Time : 418ms
Thread #12 Execution Time : 343ms
Thread #20 Execution Time : 422ms
Thread #17 Execution Time : 413ms
Thread #5 Execution Time : 535ms
Thread #10 Execution Time : 608ms
Thread #3 Execution Time : 241ms
Thread #8 Execution Time : 252ms
Thread #16 Execution Time : 279ms
Thread #2 Execution Time : 335ms
Thread #7 Execution Time : 360ms
Thread #6 Execution Time : 335ms
Thread #0 Execution Time : 329ms
Thread #14 Execution Time : 545ms
Thread #28 Execution Time : 503ms
Thread #24 Execution Time : 474ms
```

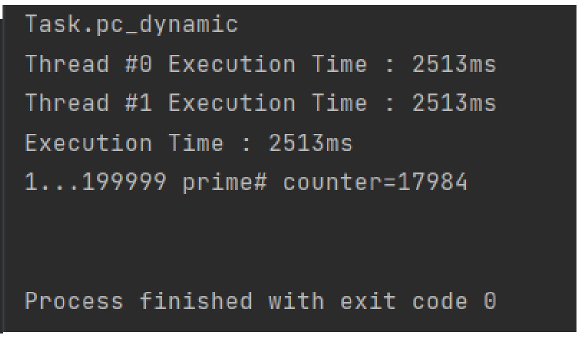


```
Thread #27 Execution Time : 443ms
Thread #25 Execution Time : 496ms
Thread #21 Execution Time : 500ms
Thread #4 Execution Time : 262ms
Thread #1 Execution Time : 526ms
Thread #9 Execution Time : 392ms
Thread #23 Execution Time : 449ms
Thread #18 Execution Time : 415ms
Thread #11 Execution Time : 347ms
Thread #22 Execution Time : 463ms
Thread #30 Execution Time : 484ms
Thread #19 Execution Time : 465ms
Thread #26 Execution Time : 505ms
Thread #15 Execution Time : 456ms
Thread #31 Execution Time : 535ms
total Program Execution Time : 911ms
1...199999 prime # count 17984

Process finished with exit code 0
```

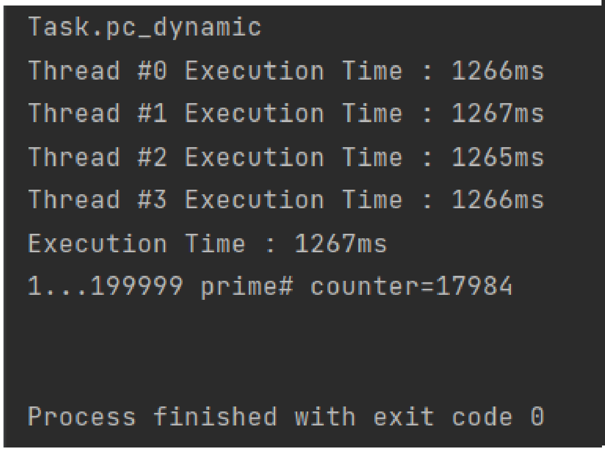
<Static load balancing(Cyclic) Execution Time ScreanShot>



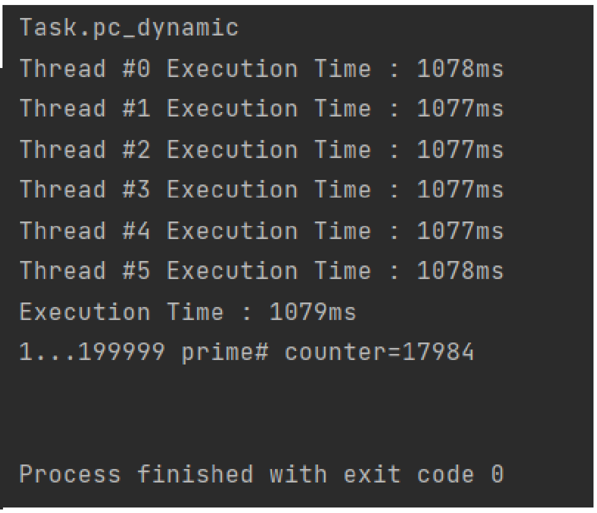
```
Task.pc_dynamic
Thread #0 Execution Time : 3489ms
Execution Time : 3489ms
1...199999 prime# counter=17984


Process finished with exit code 0
```



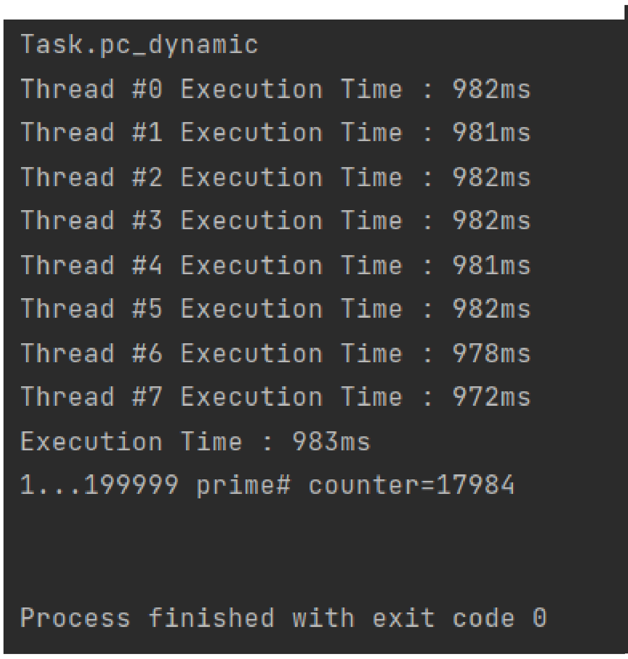
```
Task.pc_dynamic
Thread #0 Execution Time : 2513ms
Thread #1 Execution Time : 2513ms
Execution Time : 2513ms
1...199999 prime# counter=17984


Process finished with exit code 0
```



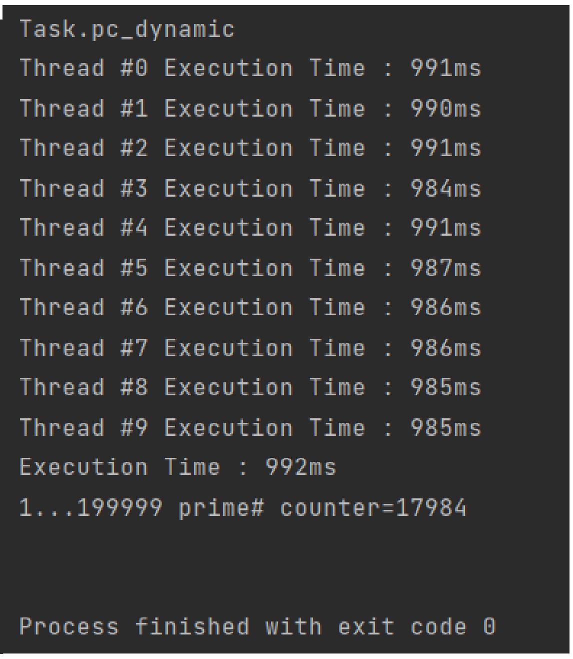
```
Task.pc_dynamic
Thread #0 Execution Time : 1266ms
Thread #1 Execution Time : 1267ms
Thread #2 Execution Time : 1265ms
Thread #3 Execution Time : 1266ms
Execution Time : 1267ms
1...199999 prime# counter=17984


Process finished with exit code 0
```



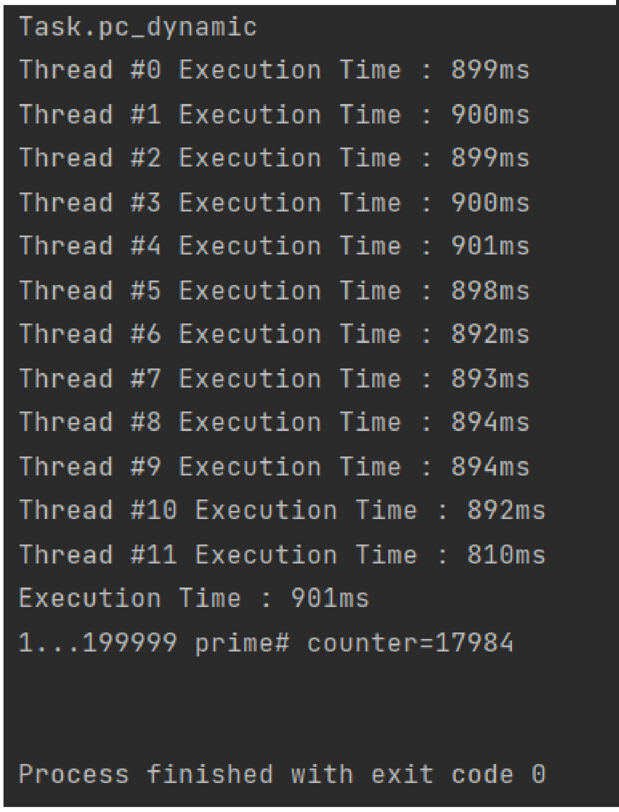
```
Task.pc_dynamic
Thread #0 Execution Time : 1078ms
Thread #1 Execution Time : 1077ms
Thread #2 Execution Time : 1077ms
Thread #3 Execution Time : 1077ms
Thread #4 Execution Time : 1077ms
Thread #5 Execution Time : 1078ms
Execution Time : 1079ms
1...199999 prime# counter=17984


Process finished with exit code 0
```



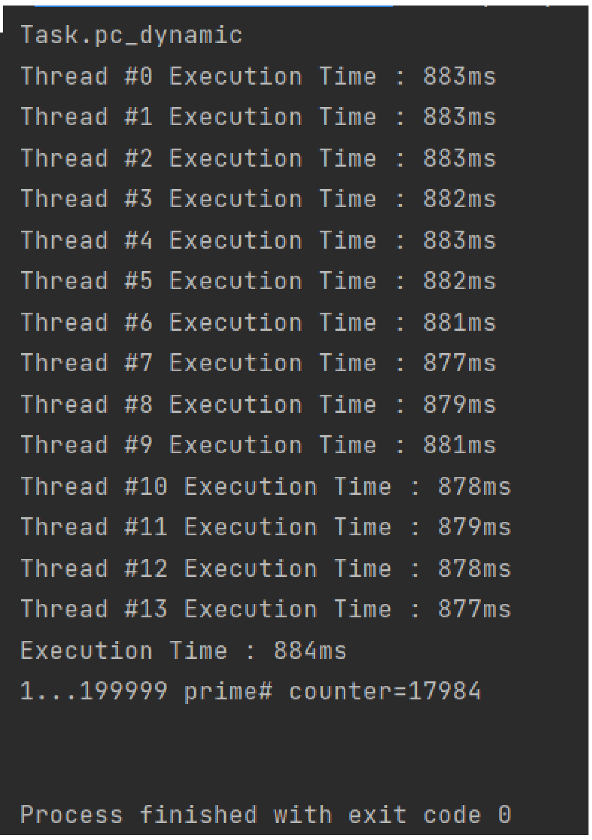
```
Task.pc_dynamic
Thread #0 Execution Time : 982ms
Thread #1 Execution Time : 981ms
Thread #2 Execution Time : 982ms
Thread #3 Execution Time : 982ms
Thread #4 Execution Time : 981ms
Thread #5 Execution Time : 982ms
Thread #6 Execution Time : 978ms
Thread #7 Execution Time : 972ms
Execution Time : 983ms
1...199999 prime# counter=17984


Process finished with exit code 0
```



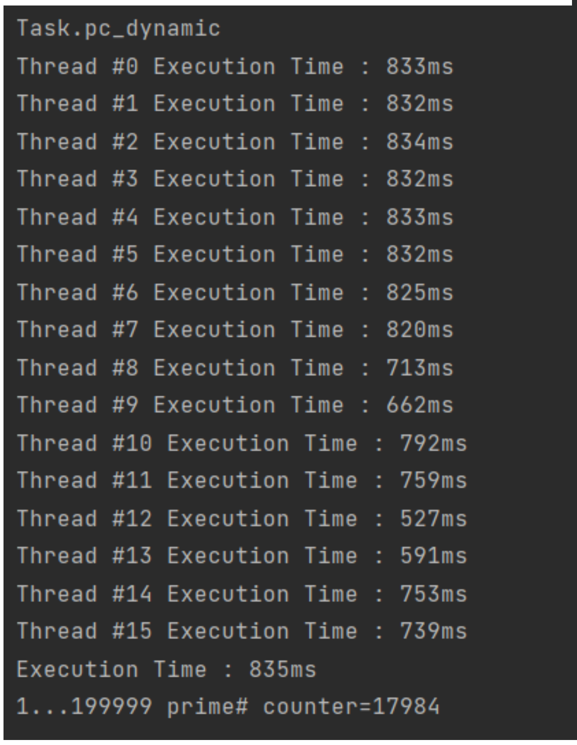
```
Task.pc_dynamic
Thread #0 Execution Time : 991ms
Thread #1 Execution Time : 990ms
Thread #2 Execution Time : 991ms
Thread #3 Execution Time : 984ms
Thread #4 Execution Time : 991ms
Thread #5 Execution Time : 987ms
Thread #6 Execution Time : 986ms
Thread #7 Execution Time : 986ms
Thread #8 Execution Time : 985ms
Thread #9 Execution Time : 985ms
Execution Time : 992ms
1...199999 prime# counter=17984


Process finished with exit code 0
```



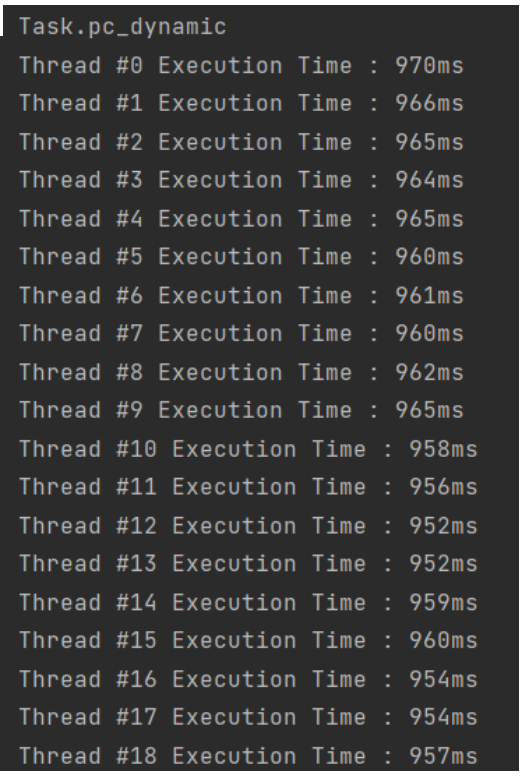
```
Task.pc_dynamic
Thread #0 Execution Time : 899ms
Thread #1 Execution Time : 900ms
Thread #2 Execution Time : 899ms
Thread #3 Execution Time : 900ms
Thread #4 Execution Time : 901ms
Thread #5 Execution Time : 898ms
Thread #6 Execution Time : 892ms
Thread #7 Execution Time : 893ms
Thread #8 Execution Time : 894ms
Thread #9 Execution Time : 894ms
Thread #10 Execution Time : 892ms
Thread #11 Execution Time : 810ms
Execution Time : 901ms
1...199999 prime# counter=17984


Process finished with exit code 0
```



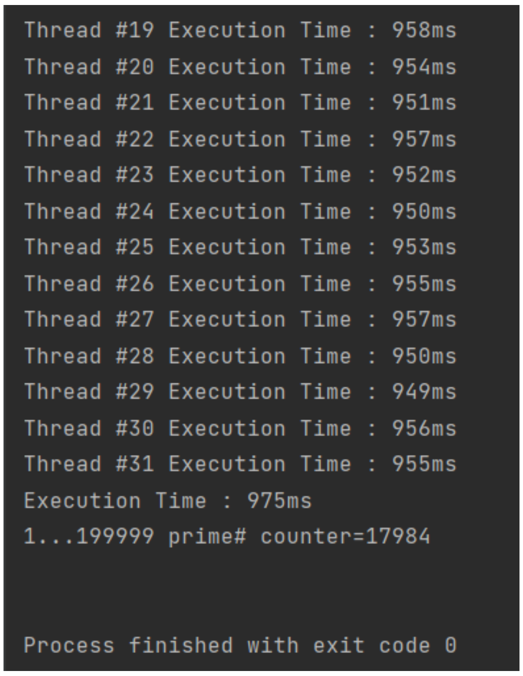
```
Task.pc_dynamic
Thread #0 Execution Time : 883ms
Thread #1 Execution Time : 883ms
Thread #2 Execution Time : 883ms
Thread #3 Execution Time : 882ms
Thread #4 Execution Time : 883ms
Thread #5 Execution Time : 882ms
Thread #6 Execution Time : 881ms
Thread #7 Execution Time : 877ms
Thread #8 Execution Time : 879ms
Thread #9 Execution Time : 881ms
Thread #10 Execution Time : 878ms
Thread #11 Execution Time : 879ms
Thread #12 Execution Time : 878ms
Thread #13 Execution Time : 877ms
Execution Time : 884ms
1...199999 prime# counter=17984


Process finished with exit code 0
```



```
Task.pc_dynamic
Thread #0 Execution Time : 833ms
Thread #1 Execution Time : 832ms
Thread #2 Execution Time : 834ms
Thread #3 Execution Time : 832ms
Thread #4 Execution Time : 833ms
Thread #5 Execution Time : 832ms
Thread #6 Execution Time : 825ms
Thread #7 Execution Time : 820ms
Thread #8 Execution Time : 713ms
Thread #9 Execution Time : 662ms
Thread #10 Execution Time : 792ms
Thread #11 Execution Time : 759ms
Thread #12 Execution Time : 527ms
Thread #13 Execution Time : 591ms
Thread #14 Execution Time : 753ms
Thread #15 Execution Time : 739ms
Execution Time : 835ms
1...199999 prime# counter=17984
```



```
Task.pc_dynamic
Thread #0 Execution Time : 970ms
Thread #1 Execution Time : 966ms
Thread #2 Execution Time : 965ms
Thread #3 Execution Time : 964ms
Thread #4 Execution Time : 965ms
Thread #5 Execution Time : 960ms
Thread #6 Execution Time : 961ms
Thread #7 Execution Time : 960ms
Thread #8 Execution Time : 962ms
Thread #9 Execution Time : 965ms
Thread #10 Execution Time : 958ms
Thread #11 Execution Time : 956ms
Thread #12 Execution Time : 952ms
Thread #13 Execution Time : 952ms
Thread #14 Execution Time : 959ms
Thread #15 Execution Time : 960ms
Thread #16 Execution Time : 954ms
Thread #17 Execution Time : 954ms
Thread #18 Execution Time : 957ms
```



```
Thread #19 Execution Time : 958ms
Thread #20 Execution Time : 954ms
Thread #21 Execution Time : 951ms
Thread #22 Execution Time : 957ms
Thread #23 Execution Time : 952ms
Thread #24 Execution Time : 950ms
Thread #25 Execution Time : 953ms
Thread #26 Execution Time : 955ms
Thread #27 Execution Time : 957ms
Thread #28 Execution Time : 950ms
Thread #29 Execution Time : 949ms
Thread #30 Execution Time : 956ms
Thread #31 Execution Time : 955ms
Execution Time : 975ms
1...199999 prime# counter=17984


Process finished with exit code 0
```

<Dynamic load balancing Execution Time ScreanShot>