

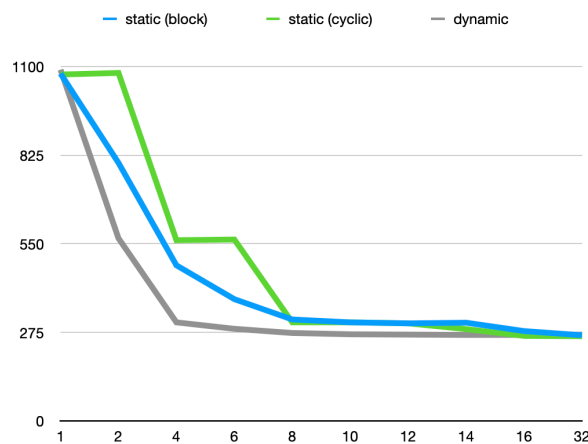
problem 1. Compute the number of 'prime numbers' between 1 and 200000.

(a) experiment environment

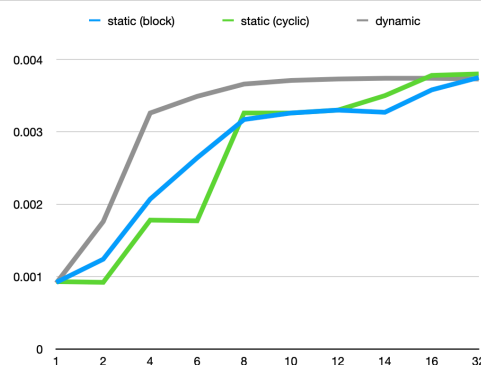
- CPU type : Apple M1
- number of cores : 8 core CPU
- memory size : 16 GB
- OS : macOS Monterey

(b) execution time (ms) per the number of entire threads = {1,2,4,6,8,10,12,14,16,32}.

Execution Time										
exec time (ms)	1	2	4	6	8	10	12	14	16	32
static (block)	1078	801	483	378	315	306	303	305	279	266
static (cyclic)	1075	1080	561	563	306	306	303	285	264	263
dynamic	1090	567	306	286	273	269	268	267	267	268



performance (1/exec time)										
performance (1/exec time)	1	2	4	6	8	10	12	14	16	32
static (block)	0.00092	0.00124	0.00207	0.00264	0.00317	0.00326	0.00330	0.00327	0.00358	0.00375
static (cyclic)	0.00093	0.00092	0.00178	0.00177	0.00326	0.00326	0.00330	0.00350	0.00378	0.00380
dynamic	0.00091	0.00176	0.00326	0.00349	0.00366	0.00371	0.00373	0.00374	0.00374	0.00373



Testing programs were done at 8-cores cpu environment. So even though you increase the number of threads more than 8, there's no more performance improvement.

But until 8-threads, the more threads you add, the higher performance it shows

Dynamic load balancing shows better performance than the others. The reason why it shows better load balancing is explained below.

(c) Analysis

(1) pc_static_block.java

```
3 class PrimeThread extends Thread {
4     int to;
5     int from;
6     int count = 0;
7     long timeDiff = 0;
8
9     PrimeThread(int _from, int _to) {
10         from = _from;
11         to = _to;
12     }
13
14     public void run() { // overriding, must have this type
15         long s = System.currentTimeMillis();
16         for(int i = from; i < to; i++)
17             if (isPrime(i))
18                 count++;
19         long e = System.currentTimeMillis();
20         timeDiff = e - s;
21     }
22     long getTimeDiff() {
23         return timeDiff;
24     }
25     int getCount() {
26         return count;
27     }
28     int getFrom() {
29         return from;
30     }
31     int getTo() {
32         return to;
33     }
34
35     private boolean isPrime(int x) {
36         int i;
37         if (x<=1) return false;
38
39         for (i=2;i<x;i++)
40             if (x%i == 0)
41                 return false;
42         return true;
43     }
44 }
45
46 public class pc_static_block {
47     private static int NUM_END = 200000; // default input
48     private static int NUM_THREADS = 32; // default number of threads
49
50     public static void main (String[] args) {
51         if (args.length == 2) {
52             NUM_THREADS = Integer.parseInt(args[0]);
53             NUM_END = Integer.parseInt(args[1]);
54         }
55
56         PrimeThread[] pt = new PrimeThread[NUM_THREADS];
57
58         int counter=0;
59
60         // find prime numbers from 2 to 200000
61         long startTime = System.currentTimeMillis();
62
63         int d = NUM_END/NUM_THREADS;
64
65         for (int i = 0; i < NUM_THREADS; i++) {
66             if (i == NUM_THREADS - 1) {
67                 pt[i] = new PrimeThread(i*d, NUM_END);
68             } else {
69                 pt[i] = new PrimeThread(i*d + 1, (i+1)*d);
70             }
71             pt[i].start();
72         }
73
74         // wait the other threads
75         try {
76             for (int i = 0 ; i < NUM_THREADS; i++) {
77                 pt[i].join();
78                 counter += pt[i].getCount();
79             }
80         } catch (InterruptedException e) { System.out.println("Error occurred!"); }
81         long endTime = System.currentTimeMillis();
82
83
84         // calculate and print the result and the time taken
85         long timeDiff = endTime - startTime;
86         System.out.println("Program Execution Time: " + timeDiff + "ms");
87         System.out.println("1..." + (NUM_END-1) + " prime# counter=" + counter + "\n");
88
89         for (int i = 0 ; i < NUM_THREADS; i++) {
90             System.out.println("#" + i + " thread : range " + pt[i].getFrom() + " ... " + pt[i].getTo());
91             System.out.println("    execution time : " + pt[i].getTimeDiff() + "ms");
92         }
93     }
94 }
95
96 }
```

N number of threads find the number of prime numbers at different range and gather them into one variable at last.

Main thread divides the whole range, 0...200000, into n sub range blocks and allocate them to n threads separately.

```
Program Execution Time: 483ms
1...199999 prime# counter=17984
```

```
#0 thread : range 1 ... 50000
  execution time : 86ms
#1 thread : range 50001 ... 100000
  execution time : 234ms
#2 thread : range 100001 ... 150000
  execution time : 357ms
#3 thread : range 150000 ... 200000
  execution time : 483ms
```

If an integer gets bigger, it takes more time to check whether it's prime number or not. So Thread#3(blue) takes more time to count the number of prime numbers at its range than other Threads(red, yellow, green).

Therefore, the loads allocated to each thread are not equal.

4 threads



(2) pc_static_cyclic.java

```

3  class PrimeThread extends Thread {
4      int r;
5      int n;
6      int end;
7      int count = 0;
8      long timeDiff = 0;
9
10     PrimeThread(int _n, int _end, int _r) {
11         n = _n;
12         r = _r;
13         end = _end;
14     }
15
16     public void run() { // overriding, must have this type
17         long s = System.currentTimeMillis();
18         int x = 0;
19         while (n * x + r <= end) {
20             if (isPrime(n * x + r))
21                 count++;
22             x++;
23         }
24         long e = System.currentTimeMillis();
25         timeDiff = e - s;
26     }
27     long getTimeDiff() {
28         return timeDiff;
29     }
30     int getR() {
31         return r;
32     }
33     int getCount() {
34         return count;
35     }
36
37     private boolean isPrime(int x) {
38         int i;
39         if (x <= 1) return false;
40
41         for (i = 2; i < x; i++)
42             if (x % i == 0)
43                 return false;
44         return true;
45     }
46 }

48 public class pc_static_cyclic {
49     private static int NUM_END = 200000; // default input
50     private static int NUM_THREADS = 32; // default number of threads
51
52     public static void main (String[] args) {
53         if (args.length == 2) {
54             NUM_THREADS = Integer.parseInt(args[0]);
55             NUM_END = Integer.parseInt(args[1]);
56         }
57
58         PrimeThread[] pt = new PrimeThread[NUM_THREADS];
59
60         int counter=0;
61
62         // find prime numbers from 2 to 200000
63         long startTime = System.currentTimeMillis();
64         for (int i = 0; i < NUM_THREADS; i++) {
65             pt[i] = new PrimeThread(NUM_THREADS, NUM_END, i);
66             pt[i].start();
67         }
68
69         // wait the other threads
70         try {
71             for (int i = 0; i < NUM_THREADS; i++) {
72                 pt[i].join();
73                 counter += pt[i].getCount();
74             }
75         } catch (InterruptedException e) { System.out.println("Error occurred!"); }
76         long endTime = System.currentTimeMillis();
77
78         // calculate and print the result and the time taken
79         long timeDiff = endTime - startTime;
80         System.out.println("Program Execution Time: " + timeDiff + "ms");
81         System.out.println("1..." + (NUM_END-1) + " prime# counter=" + counter + "\n");
82
83         for (int i = 0; i < NUM_THREADS; i++) {
84             System.out.println("#" + i + " thread : range " + NUM_THREADS + "k + " + pt[i].getR());
85             System.out.println("    execution time : " + pt[i].getTimeDiff() + "ms");
86         }
87     }
88 }
89
90

```

```

Program Execution Time: 308ms
1...199999 prime# counter=17984

```

```

#0 thread : range 8k + 0
  execution time : 3ms
#1 thread : range 8k + 1
  execution time : 298ms
#2 thread : range 8k + 2
  execution time : 4ms
#3 thread : range 8k + 3
  execution time : 299ms
#4 thread : range 8k + 4
  execution time : 5ms
#5 thread : range 8k + 5
  execution time : 298ms
#6 thread : range 8k + 6
  execution time : 2ms
#7 thread : range 8k + 7
  execution time : 304ms

```

8 threads



Let's look thread#0, To define whether 8k (1 ≤ k < 25000) is prime number or not, dividing 8k with 2 is enough. So even though thread#1 checks the same length of range with other threads but it works only 3ms and rest.

In contrast, thread#7 takes way more times to count because it should try to divide more times per integer than thread#0.

Therefore, the loads allocated to each thread are not equal.

(3) pc_dynamic.java

```

3 import java.util.concurrent.ArrayBlockingQueue;
4
5 class PrimeThread extends Thread {
6     ArrayBlockingQueue<Integer> q;
7     int count = 0;
8     long timeDiff = 0;
9
10    PrimeThread(ArrayBlockingQueue<Integer> queue) {
11        q = queue;
12    }
13
14    public void run() { // overriding, must have this type
15        long s = System.currentTimeMillis();
16        while (!q.isEmpty()) {
17            try {
18                if (isPrime(q.take()))
19                    count++;
20            } catch (InterruptedException e) {
21                System.out.println("error occurred while taking element out of queue ");
22            }
23            long e = System.currentTimeMillis();
24            timeDiff = e - s;
25        }
26        long getTimeDiff() {
27            return timeDiff;
28        }
29        int getCount() {
30            return count;
31        }
32    }
33
34    private boolean isPrime(int x) {
35        int i;
36        if (x <= 1) return false;
37
38        for (i = 2; i < x; i++)
39            if (x % i == 0)
40                return false;
41        return true;
42    }
43 }

```

```

Program Execution Time: 302ms
1...199999 prime# counter=17984

#0 thread execution time : 302ms
#1 thread execution time : 302ms
#2 thread execution time : 302ms
#3 thread execution time : 302ms

```

4 threads

```

45 public class pc_static_dynamic {
46     private static int NUM_END = 200000; // default input
47     private static int NUM_THREADS = 4; // default number of threads
48
49     public static void main (String[] args) {
50         if (args.length == 2) {
51             NUM_THREADS = Integer.parseInt(args[0]);
52             NUM_END = Integer.parseInt(args[1]);
53         }
54
55         int[] numbers = new int[NUM_END+1];
56         numbers[NUM_END] = NUM_END;
57         ArrayBlockingQueue<Integer> queue = new ArrayBlockingQueue<Integer>(NUM_END);
58         for (int i = 0; i < NUM_THREADS; i++) {
59             numbers[i] = i;
60             queue.add(i+1);
61         }
62         PrimeThread[] pt = new PrimeThread[NUM_THREADS];
63
64         int counter=0;
65
66         // find prime numbers from 2 to 200000
67         long startTime = System.currentTimeMillis();
68         for (int i = 0; i < NUM_THREADS; i++) {
69             pt[i] = new PrimeThread(queue);
70             pt[i].start();
71         }
72
73         // wait the other threads
74         try {
75             for (int i = 0; i < NUM_THREADS; i++) {
76                 pt[i].join();
77                 counter += pt[i].getCount();
78             }
79         } catch (InterruptedException e) { System.out.println("Error occurred!"); }
80         long endTime = System.currentTimeMillis();
81
82         // calculate and print the result and the time taken
83         long timeDiff = endTime - startTime;
84         System.out.println("Program Execution Time: " + timeDiff + "ms");
85         System.out.println("1..." + (NUM_END-1) + " prime# counter=" + counter + "\n");
86
87         for (int i = 0; i < NUM_THREADS; i++) {
88             System.out.println("#" + i + " thread execution time : " + pt[i].getTimeDiff() + "ms");
89         }
90     }
91 }
92
93
94 }

```

```

Program Execution Time: 268ms
1...199999 prime# counter=17984

#0 thread execution time : 268ms
#1 thread execution time : 268ms
#2 thread execution time : 268ms
#3 thread execution time : 268ms
#4 thread execution time : 268ms
#5 thread execution time : 268ms
#6 thread execution time : 267ms
#7 thread execution time : 267ms

```

8 threads

In this case, there's only one queue which all the threads can get jobs from. They share the queue together. So loads are distributed to the threads randomly and as equal amount. Therefore all thread work equally.

(d) How to compile and execute the source code

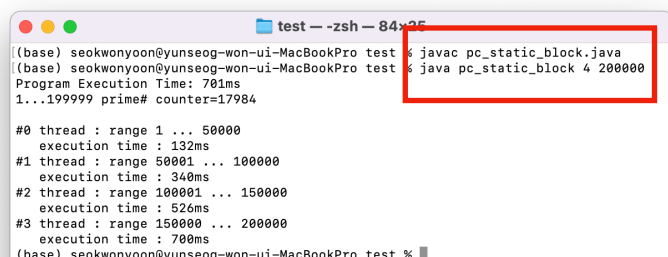
At terminal,

- (1) javac pc_static_block.java
- (2) javac pc_static_cyclic.java
- (3) javac pc_static_dynamic.java

After compilation execute the program.

<java "filename" "numberOfThreads" "numRange">

- (1) java pc_static_block 4 200000
- (2) java pc_static_cyclic 6 12000
- (3) java pc_static_dynamic 8 200000



```

(base) seekwonyoon@yunseog-won-ui-MacBookPro test % javac pc_static_block.java
(base) seekwonyoon@yunseog-won-ui-MacBookPro test % java pc_static_block 4 200000
Program Execution Time: 701ms
1...199999 prime# counter=17984

#0 thread : range 1 ... 50000
execution time : 132ms
#1 thread : range 50001 ... 100000
execution time : 340ms
#2 thread : range 100001 ... 150000
execution time : 526ms
#3 thread : range 150001 ... 200000
execution time : 700ms
(base) seekwonyoon@yunseog-won-ui-MacBookPro test %

```