Multi-Threaded Programming

Operating Systems
Multi-Threaded Programming

Lama Alghzzi | 220410092

Norah Alrubayan | 220410543

Sec251

Dr. Siwar Rekik

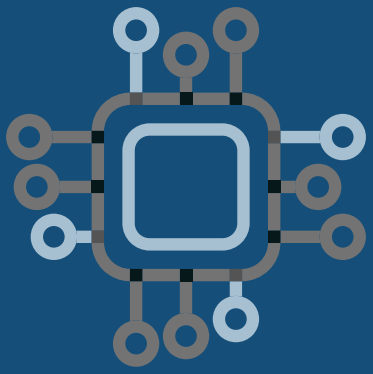


TABLE OF CONTENTS

Introduction

In this project we used java multi-threading programming to measures the runtime of summing the square of the elements of the array algorithm, in both a standard sequential version and a parallel version that uses multiple threads. There are two main advantages of multi-threading: Fist, program with multiple threads will, in general, result in better utilization of system resources, including the CPU. Second, there are several problems better solved by multiple threads. In this project we expected the result will be better when we increase the number of threads.

Design

Pseudo code of algorithm

```
method sumSquareRange(array, min, max)
    result = 0
    loop i = min to max
        result = result + (array[i] * array[i])
    endloop

    return result
end

method sum(array, threadCount)
    result = 0
    thLen = len(array) / threadCount

    parallel loop i = 0 To threadCount
        result = result + sumSquareRange(array, i * thLen, ((i + 1) *
thLen));
    endloop

    return result
end

method main()
    LENGTH = 1000
    RUNS = 20
    THREAD_COUNT = 8

    loop i = 1 to RUNS
        array = createRandomArray(LENGTH)
        total = 0
        loop j = 1 to 100
            total = sum(array, THREAD_COUNT);
        endloop

        OUTPUT total

        LENGTH = LENGTH * 2;
    endloop
end
```

Implementation

```
public static void main(String[] args) throws Throwable {
     int LENGTH = 1000; // initial length of array to sum
     int RUNS = 20; // how many times to grow by 2?
     int THREAD_COUNT = 8; // number of threads will be used

     for (int i = 1; i <= RUNS; i++) {
          int[] a = createRandomArray(LENGTH);

          // run the algorithm and time how long it takes
          long startTime1 = System.currentTimeMillis();
          int total = 0;
          for (int j = 1; j <= 100; j++) {
               total = sum3(a, THREAD_COUNT);
          }
          long endTime1 = System.currentTimeMillis();

          // int correct = sum(a);
          int correct = sumSquare(a);
          if (total != correct) {
               throw new RuntimeException("wrong sum: " + total + " vs. "
+ correct);
          }

          System.out.printf("%10d elements =>%6d ms \n", LENGTH, endTime1
- startTime1);
          LENGTH *= 2; // double size of array for next time
     }
    }
```

In the main method we had initialize the number of threads will be used and length of the array and how many times to grow by 2. In each run we create an array of random numbers then call the sum method to sum the square of the elements of the array then check if the result is correct by comparing it with the result of the standard sequential version.

Implementation

```
public static int sum3(int[] a, int threadCount) {
        int len = (int) Math.ceil(1.0 * a.length / threadCount);
        Summer[] summers = new Summer[threadCount];
        Thread[] threads = new Thread[threadCount];
        for (int i = 0; i < threadCount; i++) {
             summers[i] = new Summer(a, i * len, Math.min((i + 1) * len,
a.length));
             threads[i] = new Thread(summers[i]);
             threads[i].start();
        }
        try {
             for (Thread t : threads) {
                  t.join();
             }
        } catch (InterruptedException ie) {
        }

        int total = 0;
        for (Summer summer : summers) {
             total += summer.getSum();
        }
        return total;
    }
```

A method to computes the total sum the square of the elements of the given array.This is a parallel version that can use any number of threads. It will can make use of as many cores/CPUs as you want to give it.

```
public static int sumSquareRange(int[] a, int min, int max) {
        int result = 0;
        for (int i = min; i < max; i++) {
             result += (a[i] * a[i]);
        }
        return result;
    }
```

A helper method to compute sum square of the elements of array a, indexes [min... max]

Results

Core.java | ArraySum.java | Summer.java

```
1
2  public class Core {
3
4      public static void main(String[] args) {
5          int cores = Runtime.getRuntime().availableProcessors();
6          System.out.println(cores);
7      }
8
9  }
10
```

Problems | Javadoc | Declaration | Console

```
<terminated> Core [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Apr 28, 2023, 8:31:16 P
8
```

[Number of processor cores]

Core.java | ArraySum.java | Summer.java

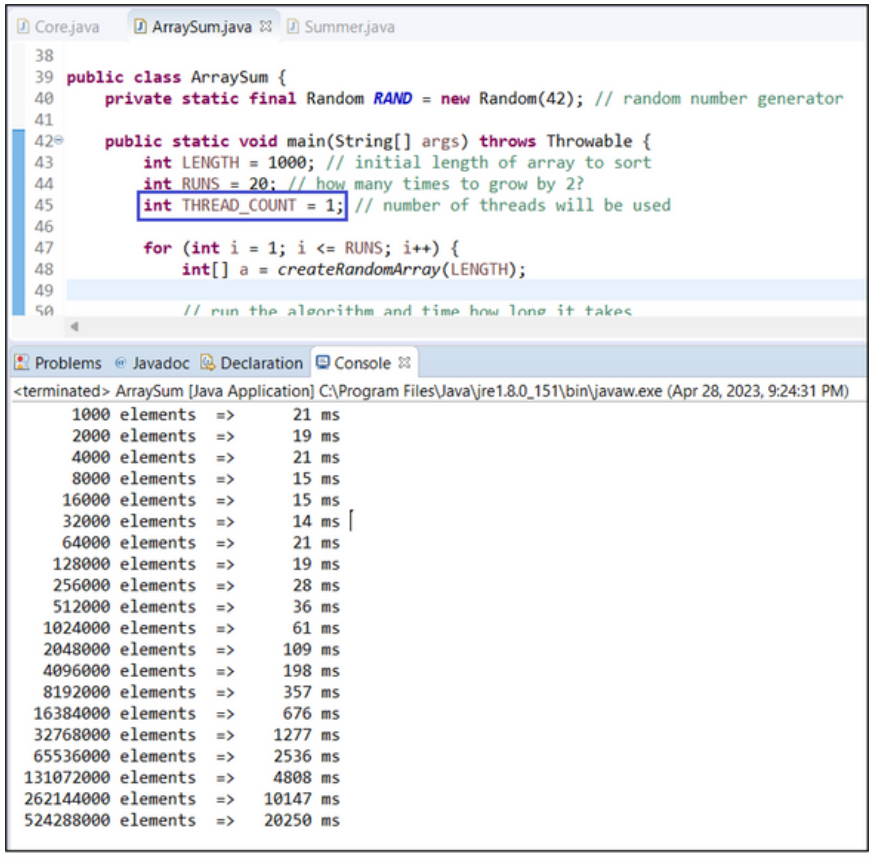
```
38
39 public class ArraySum {
40     private static final Random RAND = new Random(42); // random number generator
41
42     public static void main(String[] args) throws Throwable {
43         int LENGTH = 1000; // initial length of array to sort
44         int RUNS = 20; // how many times to grow by 2?
45         int THREAD_COUNT = 1; // number of threads will be used
46
47         for (int i = 1; i <= RUNS; i++) {
48             int[] a = createRandomArray(LENGTH);
49
50             // run the algorithm and time how long it takes
```

Problems | Javadoc | Declaration | Console

```
<terminated> ArraySum [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Apr 28, 2023, 9:24:31 PM)
     1000 elements  =>      21 ms
     2000 elements  =>      19 ms
     4000 elements  =>      21 ms
     8000 elements  =>      15 ms
    16000 elements  =>      15 ms
    32000 elements  =>      14 ms
    64000 elements  =>      21 ms
   128000 elements  =>      19 ms
   256000 elements  =>      28 ms
   512000 elements  =>      36 ms
  1024000 elements  =>      61 ms
  2048000 elements  =>     109 ms
  4096000 elements  =>     198 ms
  8192000 elements  =>     357 ms
 16384000 elements  =>     676 ms
 32768000 elements  =>    1277 ms
 65536000 elements  =>    2536 ms
131072000 elements  =>    4808 ms
262144000 elements  =>   10147 ms
524288000 elements  =>   20250 ms
```

[First run: Using 1 thread]

Results

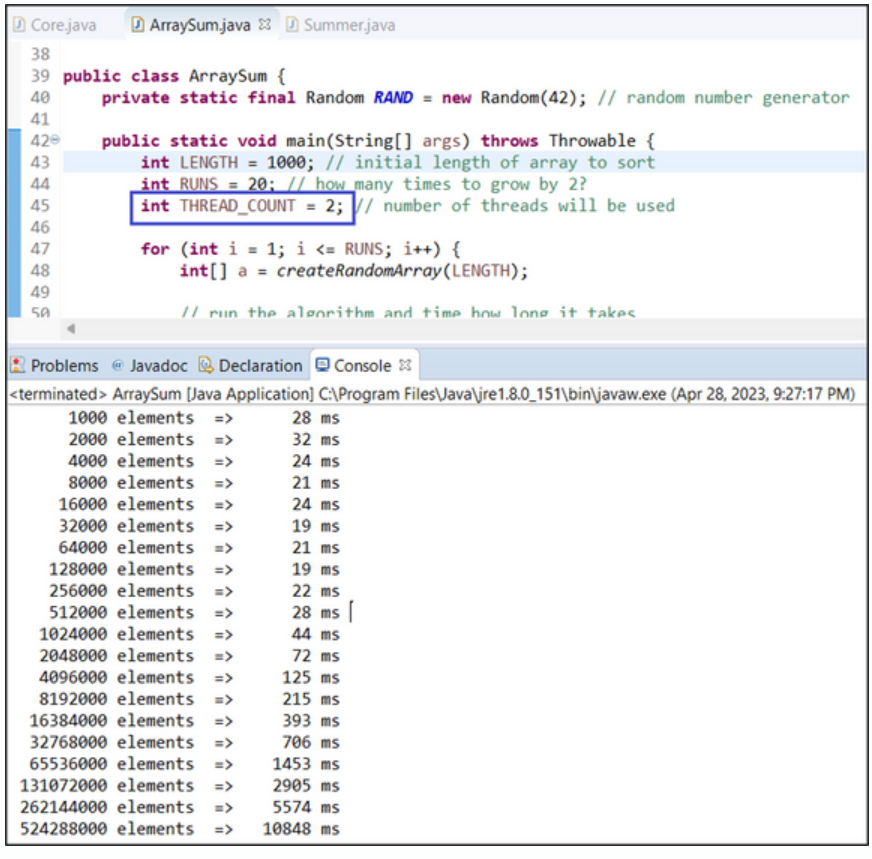


```
Core.java  ArraySum.java  Summer.java

38
39 public class ArraySum {
40     private static final Random RAND = new Random(42); // random number generator
41
42     public static void main(String[] args) throws Throwable {
43         int LENGTH = 1000; // initial length of array to sort
44         int RUNS = 20; // how many times to grow by 2?
45         int THREAD_COUNT = 1; // number of threads will be used
46
47         for (int i = 1; i <= RUNS; i++) {
48             int[] a = createRandomArray(LENGTH);
49
50             // run the algorithm and time how long it takes

Problems  Javadoc  Declaration  Console
<terminated> ArraySum [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Apr 28, 2023, 9:24:31 PM)
      1000 elements  =>      21 ms
      2000 elements  =>      19 ms
      4000 elements  =>      21 ms
      8000 elements  =>      15 ms
     16000 elements  =>      15 ms
     32000 elements  =>      14 ms
     64000 elements  =>      21 ms
    128000 elements  =>      19 ms
    256000 elements  =>      28 ms
    512000 elements  =>      36 ms
   1024000 elements  =>      61 ms
   2048000 elements  =>     109 ms
   4096000 elements  =>     198 ms
   8192000 elements  =>     357 ms
  16384000 elements  =>     676 ms
  32768000 elements  =>    1277 ms
  65536000 elements  =>    2536 ms
 131072000 elements  =>    4808 ms
 262144000 elements  =>   10147 ms
 524288000 elements  =>   20250 ms
```

First run: Using 1 thread



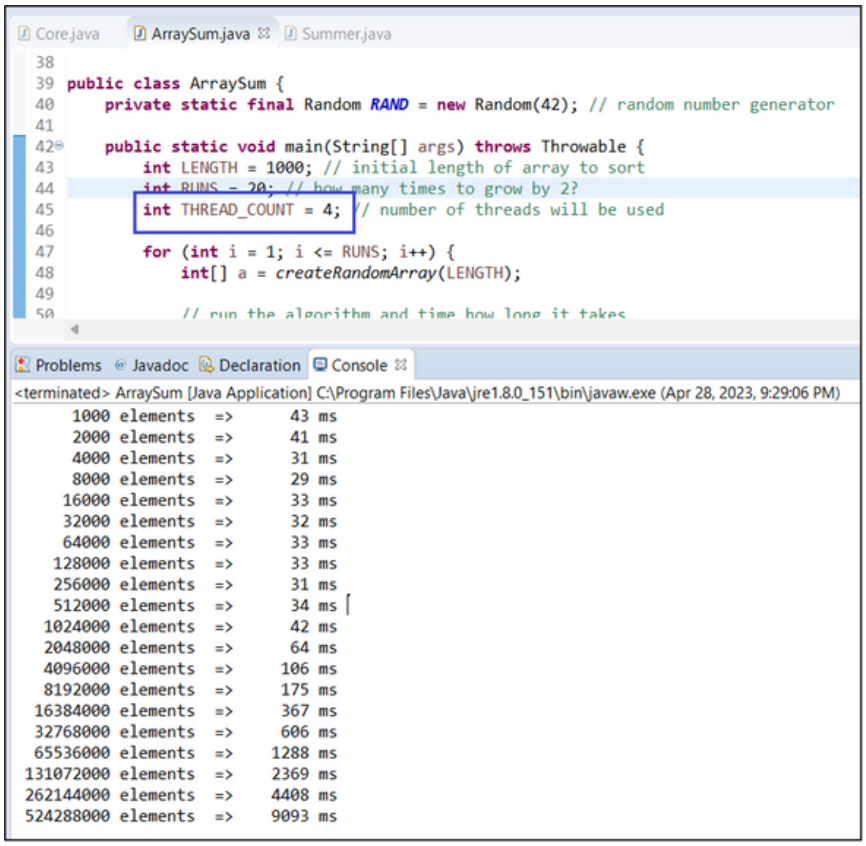
```
Core.java  ArraySum.java  Summer.java

38
39 public class ArraySum {
40     private static final Random RAND = new Random(42); // random number generator
41
42     public static void main(String[] args) throws Throwable {
43         int LENGTH = 1000; // initial length of array to sort
44         int RUNS = 20; // how many times to grow by 2?
45         int THREAD_COUNT = 2; // number of threads will be used
46
47         for (int i = 1; i <= RUNS; i++) {
48             int[] a = createRandomArray(LENGTH);
49
50             // run the algorithm and time how long it takes

Problems  Javadoc  Declaration  Console
<terminated> ArraySum [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Apr 28, 2023, 9:27:17 PM)
      1000 elements  =>      28 ms
      2000 elements  =>      32 ms
      4000 elements  =>      24 ms
      8000 elements  =>      21 ms
     16000 elements  =>      24 ms
     32000 elements  =>      19 ms
     64000 elements  =>      21 ms
    128000 elements  =>      19 ms
    256000 elements  =>      22 ms
    512000 elements  =>      28 ms
   1024000 elements  =>      44 ms
   2048000 elements  =>      72 ms
   4096000 elements  =>     125 ms
   8192000 elements  =>     215 ms
  16384000 elements  =>     393 ms
  32768000 elements  =>     706 ms
  65536000 elements  =>    1453 ms
 131072000 elements  =>    2905 ms
 262144000 elements  =>    5574 ms
 524288000 elements  =>   10848 ms
```

Second Run: Using 2 threads

Results

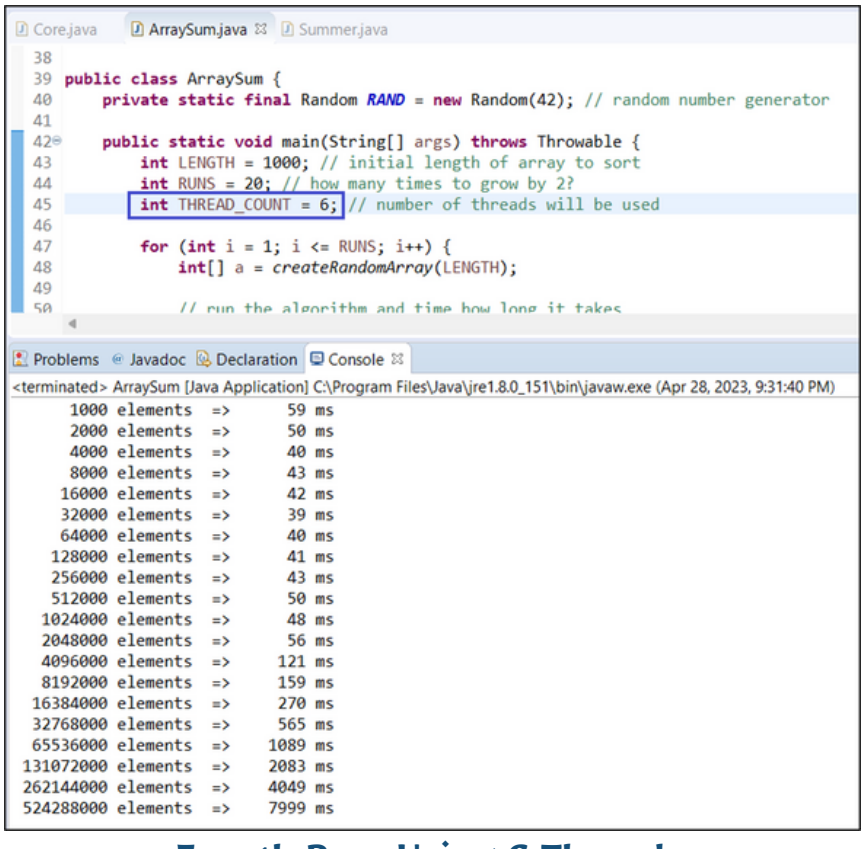


```
Core.java   ArraySum.java   Summer.java

38
39 public class ArraySum {
40     private static final Random RAND = new Random(42); // random number generator
41
42     public static void main(String[] args) throws Throwable {
43         int LENGTH = 1000; // initial length of array to sort
44         int RUNS = 20; // how many times to grow by 2?
45         int THREAD_COUNT = 4; // number of threads will be used
46
47         for (int i = 1; i <= RUNS; i++) {
48             int[] a = createRandomArray(LENGTH);
49
50             // run the algorithm and time how long it takes

Problems  Javadoc  Declaration  Console
<terminated> ArraySum [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Apr 28, 2023, 9:29:06 PM)
     1000 elements  =>      43 ms
     2000 elements  =>      41 ms
     4000 elements  =>      31 ms
     8000 elements  =>      29 ms
    16000 elements  =>      33 ms
    32000 elements  =>      32 ms
    64000 elements  =>      33 ms
   128000 elements  =>      33 ms
   256000 elements  =>      31 ms
   512000 elements  =>      34 ms
  1024000 elements  =>      42 ms
  2048000 elements  =>      64 ms
  4096000 elements  =>     106 ms
  8192000 elements  =>     175 ms
 16384000 elements  =>     367 ms
 32768000 elements  =>     606 ms
 65536000 elements  =>    1288 ms
131072000 elements  =>    2369 ms
262144000 elements  =>    4408 ms
524288000 elements  =>    9093 ms
```

Third Run: Using 4 Threads



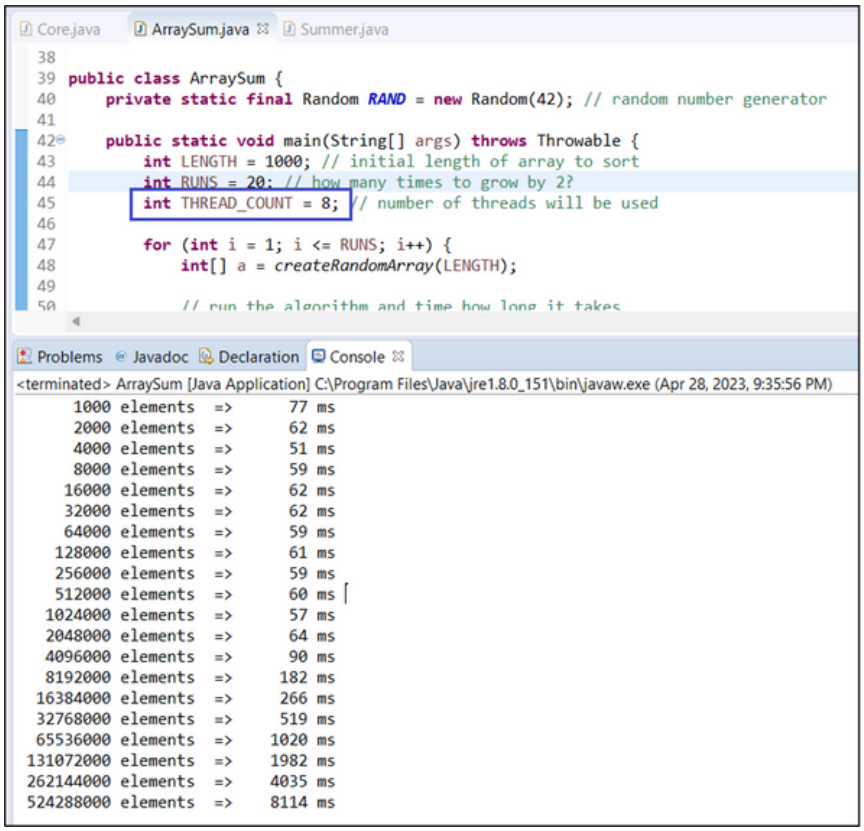
```
Core.java   ArraySum.java   Summer.java

38
39 public class ArraySum {
40     private static final Random RAND = new Random(42); // random number generator
41
42     public static void main(String[] args) throws Throwable {
43         int LENGTH = 1000; // initial length of array to sort
44         int RUNS = 20; // how many times to grow by 2?
45         int THREAD_COUNT = 6; // number of threads will be used
46
47         for (int i = 1; i <= RUNS; i++) {
48             int[] a = createRandomArray(LENGTH);
49
50             // run the algorithm and time how long it takes

Problems  Javadoc  Declaration  Console
<terminated> ArraySum [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Apr 28, 2023, 9:31:40 PM)
     1000 elements  =>      59 ms
     2000 elements  =>      50 ms
     4000 elements  =>      40 ms
     8000 elements  =>      43 ms
    16000 elements  =>      42 ms
    32000 elements  =>      39 ms
    64000 elements  =>      40 ms
   128000 elements  =>      41 ms
   256000 elements  =>      43 ms
   512000 elements  =>      50 ms
  1024000 elements  =>      48 ms
  2048000 elements  =>      56 ms
  4096000 elements  =>     121 ms
  8192000 elements  =>     159 ms
 16384000 elements  =>     270 ms
 32768000 elements  =>     565 ms
 65536000 elements  =>    1089 ms
131072000 elements  =>    2083 ms
262144000 elements  =>    4049 ms
524288000 elements  =>    7999 ms
```

Fourth Run: Using 6 Threads

Results

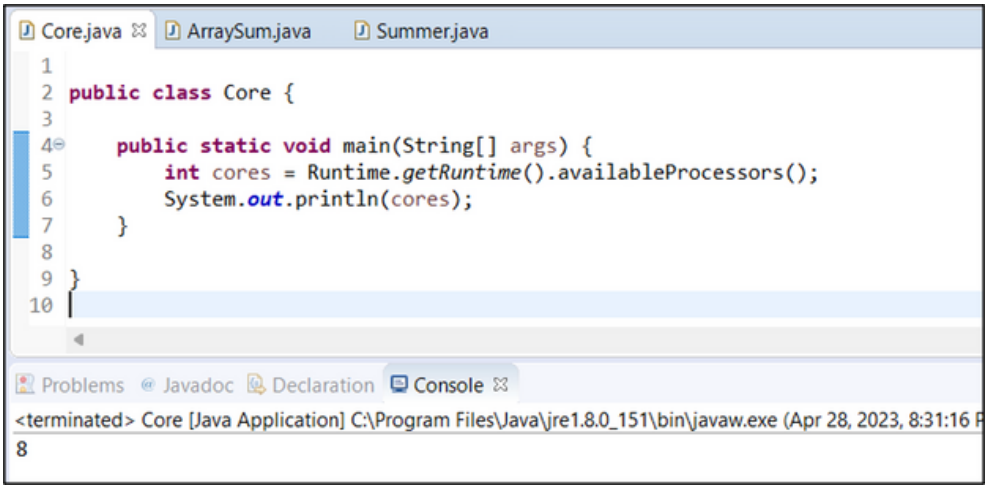


```
Core.java   ArraySum.java   Summer.java

38
39 public class ArraySum {
40     private static final Random RAND = new Random(42); // random number generator
41
42     public static void main(String[] args) throws Throwable {
43         int LENGTH = 1000; // initial length of array to sort
44         int RUNS = 20; // how many times to grow by 2?
45         int THREAD_COUNT = 8; // number of threads will be used
46
47         for (int i = 1; i <= RUNS; i++) {
48             int[] a = createRandomArray(LENGTH);
49
50             // run the algorithm and time how long it takes

Problems  Javadoc  Declaration  Console
<terminated> ArraySum [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Apr 28, 2023, 9:35:56 PM)
      1000 elements  =>      77 ms
      2000 elements  =>      62 ms
      4000 elements  =>      51 ms
      8000 elements  =>      59 ms
     16000 elements  =>      62 ms
     32000 elements  =>      62 ms
     64000 elements  =>      59 ms
    128000 elements  =>      61 ms
    256000 elements  =>      59 ms
    512000 elements  =>      60 ms
   1024000 elements  =>      57 ms
   2048000 elements  =>      64 ms
   4096000 elements  =>      90 ms
   8192000 elements  =>     182 ms
  16384000 elements  =>     266 ms
  32768000 elements  =>     519 ms
  65536000 elements  =>    1020 ms
 131072000 elements  =>    1982 ms
 262144000 elements  =>    4035 ms
 524288000 elements  =>    8114 ms
```

[Fourth Run: Using 8 Threads]



```
Core.java   ArraySum.java   Summer.java

1
2 public class Core {
3
4     public static void main(String[] args) {
5         int cores = Runtime.getRuntime().availableProcessors();
6         System.out.println(cores);
7     }
8
9 }
10

Problems  Javadoc  Declaration  Console
<terminated> Core [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Apr 28, 2023, 8:31:16 P
8
```

[Number of processor cores]

Computer Specifications

Model: Dell Inspiron 5000 Series

Processor: Intel(R) Core(TM) i7-8550U CPU Number of Cores: 8 Cores

Clock Frequency: 1.80GHz Memory: 16GB

Array Size	1Thread CPU Time	2 Threads CPU Time	% age Improvement	4 Threads CPU time	% age Improvement
256,000	28ms	22ms	21.4%	31ms	-10.7%
512,000	36ms	28ms	22.2%	34ms	5.6%
1,024,000	61ms	44ms	27.9%	42ms	31.1%
2,048,000	109ms	72ms	33.9%	64ms	41.3%
4,096,000	198ms	125ms	36.9%	106ms	46.5%
8,192,000	357ms	215ms	39.8%	175ms	51.0%
16,384,000	676ms	393ms	41.9%	367ms	45.7%
32,768,000	1,277ms	706ms	44.7%	606ms	52.5%
65,536,000	2,536ms	1,453ms	42.7%	1,288ms	49.2%
131,072,000	4,808ms	2,905ms	39.6%	2,369ms	50.7%
262,144,000	10,147ms	5,574ms	45.1%	4,408ms	56.6%
524,288,000	20,250ms	10,848ms	46.4%	9,093ms	55.1%

Array Size	6 Threads CPU Time	% age Improvement	8 Threads CPU time	% age Improvement
256,000	43ms	-53.6%	59ms	-110.7%
512,000	50ms	-38.9%	60ms	-66.7%
1,024,000	48ms	21.3%	57ms	6.6%
2,048,000	56ms	48.6%	64ms	41.3%
4,096,000	121ms	38.9%	90ms	54.5%
8,192,000	159ms	55.5%	182ms	49.0%
16,384,000	270ms	60.1%	266ms	60.7%
32,768,000	565ms	55.8%	519ms	59.4%
65,536,000	1,089ms	57.1%	1,020ms	59.8%
131,072,000	2,083ms	56.7%	1,982ms	58.8%
262,144,000	4,049ms	60.1%	4,035ms	60.2%
524,288,000	7,999ms	60.5%	8,114ms	59.9%

Results

```
27                if (total != correct) {
```

```
Output - ArraySum (run)
run:
      1000 elements  =>       1 ms
      2000 elements  =>       0 ms
      4000 elements  =>       1 ms
      8000 elements  =>       2 ms
     16000 elements  =>       0 ms
     32000 elements  =>       1 ms
     64000 elements  =>       2 ms
    128000 elements  =>       4 ms
    256000 elements  =>       9 ms
    512000 elements  =>      18 ms
   1024000 elements  =>      41 ms
   2048000 elements  =>      86 ms
   4096000 elements  =>     192 ms
   8192000 elements  =>     344 ms
  16384000 elements  =>     694 ms
  32768000 elements  =>    1381 ms
  65536000 elements  =>    2750 ms
BUILD SUCCESSFUL (total time: 7 seconds)
```

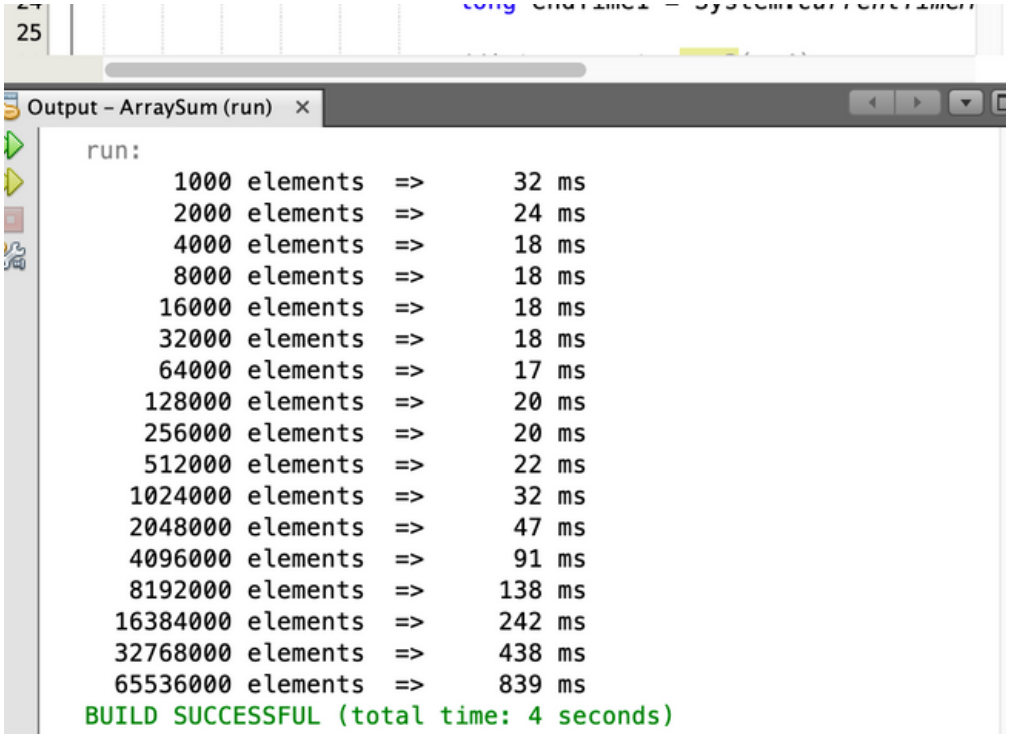
First run: Using 1 thread

```
26                int correct =sum2(a);
```

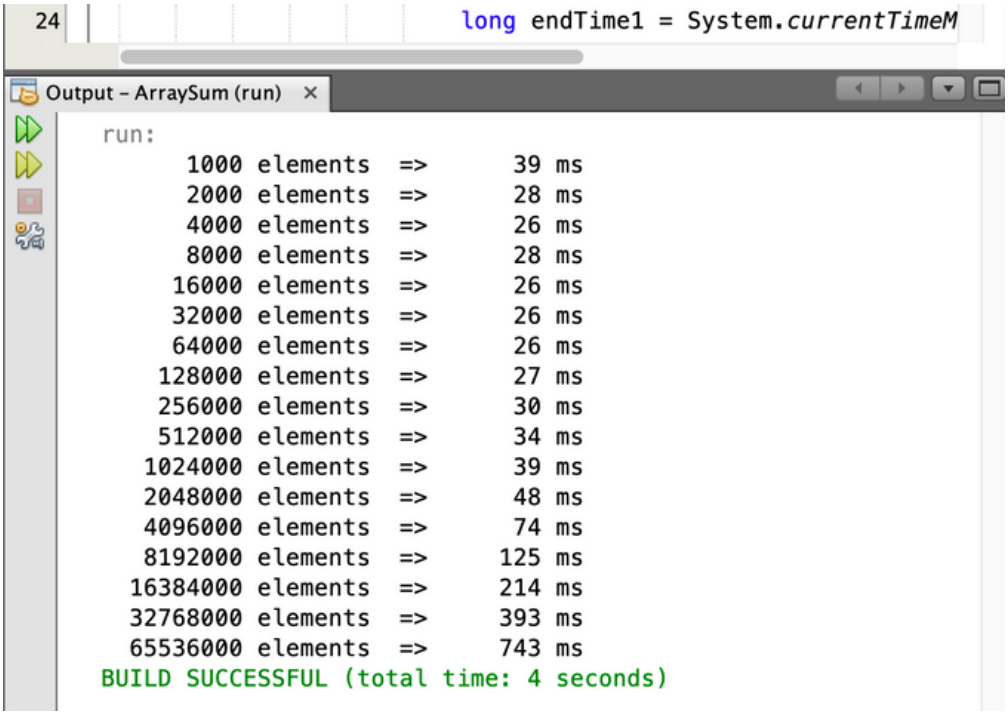
```
Output - ArraySum (run)
run:
      1000 elements  =>      18 ms
      2000 elements  =>      16 ms
      4000 elements  =>      13 ms
      8000 elements  =>      10 ms
     16000 elements  =>      10 ms
     32000 elements  =>      10 ms
     64000 elements  =>      12 ms
    128000 elements  =>      12 ms
    256000 elements  =>      14 ms
    512000 elements  =>      18 ms
   1024000 elements  =>      32 ms
   2048000 elements  =>      58 ms
   4096000 elements  =>     104 ms
   8192000 elements  =>     198 ms
  16384000 elements  =>     373 ms
  32768000 elements  =>     727 ms
  65536000 elements  =>    1434 ms
BUILD SUCCESSFUL (total time: 5 seconds)
```

Second Run: Using 2 threads

Results



Third Run: Using 4 Threads



Fourth Run: Using 6 Threads

Results

```
24                long endTime1 = System.currentTimeM
```

Output – ArraySum (run)

```
run:
      1000 elements  =>      52 ms
      2000 elements  =>      39 ms
      4000 elements  =>      40 ms
      8000 elements  =>      35 ms
     16000 elements  =>      35 ms
     32000 elements  =>      38 ms
     64000 elements  =>      42 ms
    128000 elements  =>      40 ms
    256000 elements  =>      43 ms
    512000 elements  =>      54 ms
   1024000 elements  =>      55 ms
   2048000 elements  =>      65 ms
   4096000 elements  =>      74 ms
   8192000 elements  =>     118 ms
  16384000 elements  =>     207 ms
  32768000 elements  =>     393 ms
  65536000 elements  =>     764 ms
BUILD SUCCESSFUL (total time: 4 seconds)
```

Fourth Run: Using 8 Threads

```
2   package core;
3
4   public class Core {
5
6       public static void main(String[] args) {
7   int cores=Runtime.getRuntime().availableProcessors();
8   System.out.print(cores);
9
10      }
11
12  }
13
```

Output – core (run)

```
run:
8BUILD SUCCESSFUL (total time: 0 seconds)
```

Number of processor cores

Computer Specifications

Model: MacBookPro16,2

Processor: Quad-Core Intel Core i5 Number of Cores: 8

Clock Frequency: 2GHz Memory: 16GB

Array Size	1Thread CPU Time	2 Threads CPU Time	% age Improvement	4 Threads CPU time	% age Improvement
128000	4ms	12ms	-200%	20ms	-400%
256000	9ms	14ms	-55.55%	20ms	-122.22%
512000	18ms	18ms	0%	22ms	-22.22%
1024000	41ms	32ms	21.95%	32ms	21.95%
2048000	86ms	58ms	32.55%	47ms	45.35%
4096000	192ms	104ms	45.83%	91ms	52.60%
8192000	344ms	198ms	42.44%	138ms	59.88%
16384000	694ms	373ms	46.25%	242ms	65.13%
32768000	1381ms	727ms	47.36%	438ms	68.28%
65536000	2750ms	1434ms	47.85%	839ms	69.49%

Array Size	6 Threads CPU Time	% age Improvement	8 Threads CPU time	% age Improvement
128000	27ms	-575%	40ms	-900%
256000	30ms	-233.33%	43ms	-377.78%
512000	34ms	-88.89%	54ms	-200%
1024000	39ms	4.88%	55ms	-34.15%
2048000	48ms	44.19%	65ms	24.42%
4096000	74ms	61.46%	74ms	61.46%
8192000	125ms	63.66%	118ms	65.70%
16384000	214ms	69.16%	207ms	70.17%
32768000	393ms	70.18%	393ms	70.18%
65536000	743ms	72.98%	764ms	72.22%

Conclusion and future improvements

In this project we found that multi-threading programming result in better utilization of system resources, including the CPU and several problems better solved by multiple threads. In this project we expected the result will be better when we increase the number of threads.

To improve the code, we can check first the number of number of cores/CPUs in the machine and use and idle number of threads according to the number of cores. And as we see if the array size is small, it's better to use few threads.

references and tools

- https://stackoverflow.com/questions/4759570/finding-number-of-cores-in-java

- https://www.calculatorsoup.com/calculators/algebra/percentage-increase-calculator.php