ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINERING DEPARTMENT

BLG 546E MACHINE LEARNING

CRN: 23438

Instructor: Tolga Ovatman

Report of Homework #2

April 8, 2018

Tuğrul Yatağan

504161551

Development, Build and Test Environment

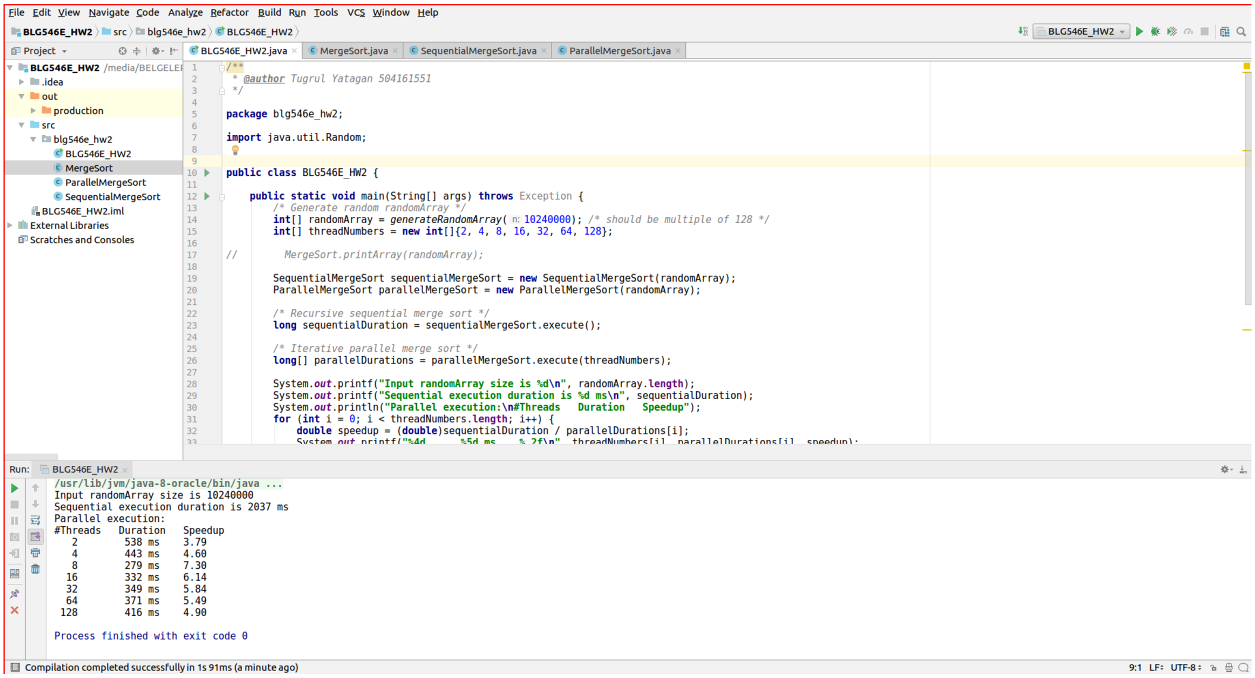
Ubuntu 16.04.4 LTS Linux kernel 4.4.0-116-generic is used for build and test environment. Test system has 6 GB of RAM and 8 core i7-3632QM 2.20 GHz CPU. Oracla Java 8 is used for Java virtual machine. Following commands is used for installing Java virtual machine and IntelliJ IDEA:

```
sudo add-apt-repository ppa:webupd8team/java

sudo apt install oracle-java8-installer

sudo snap install intellij-idea-community -classic
```

Example screen shot of development environment:



```java
/**
 * @author Tugrul Yatagan 504161551
 */

package blg546e_hw2;

import java.util.Random;


public class BLG546E_HW2 {

    public static void main(String[] args) throws Exception {
        /* Generate random randomArray */
        int[] randomArray = generateRandomArray( n: 10240000); /* should be multiple of 128 */
        int[] threadNumbers = new int[]{2, 4, 8, 16, 32, 64, 128};

//          MergeSort.printArray(randomArray);

        SequentialMergeSort sequentialMergeSort = new SequentialMergeSort(randomArray);
        ParallelMergeSort parallelMergeSort = new ParallelMergeSort(randomArray);

        /* Recursive sequential merge sort */
        long sequentialDuration = sequentialMergeSort.execute();

        /* Iterative parallel merge sort */
        long[] parallelDurations = parallelMergeSort.execute(threadNumbers);

        System.out.printf("Input randomArray size is %d\n", randomArray.length);
        System.out.printf("Sequential execution duration is %d ms\n", sequentialDuration);
        System.out.println("Parallel execution:\n#Threads   Duration   Speedup");
        for (int i = 0; i < threadNumbers.length; i++) {
            double speedup = (double)sequentialDuration / parallelDurations[i];
            System.out.printf("%4d       %5d ms     %.2f\n", threadNumbers[i], parallelDurations[i], speedup);
```

```
/usr/lib/jvm/java-8-oracle/bin/java ...
Input randomArray size is 10240000
Sequential execution duration is 2037 ms
Parallel execution:
#Threads   Duration   Speedup
   2        538 ms    3.79
   4        443 ms    4.60
   8        279 ms    7.30
  16        332 ms    6.14
  32        349 ms    5.84
  64        371 ms    5.49
 128        416 ms    4.90

Process finished with exit code 0
```

Compilation completed successfully in 1s 91ms (a minute ago)

Example output:

Application is run with different array size parameters; 1 million, 10 million and 100 million.

n=100M

```
Input randomArray size is 102400000

Sequential execution duration is 20519 ms

Parallel execution:

#Threads   Duration   Speedup

   2       5178 ms    3.96
   4       4349 ms    4.72
   8       2885 ms    7.11
  16       3214 ms    6.38
  32       3416 ms    6.01
  64       3510 ms    5.85
 128       3810 ms    5.39
```

n=10M

```
Input randomArray size is 10240000
Sequential execution duration is 2052 ms
Parallel execution:
#Threads   Duration   Speedup
   2        548 ms    3.74
   4        417 ms    4.92
   8        276 ms    7.43
  16        356 ms    5.76
  32        353 ms    5.81
  64        363 ms    5.65
 128        409 ms    5.02
```

n=1M

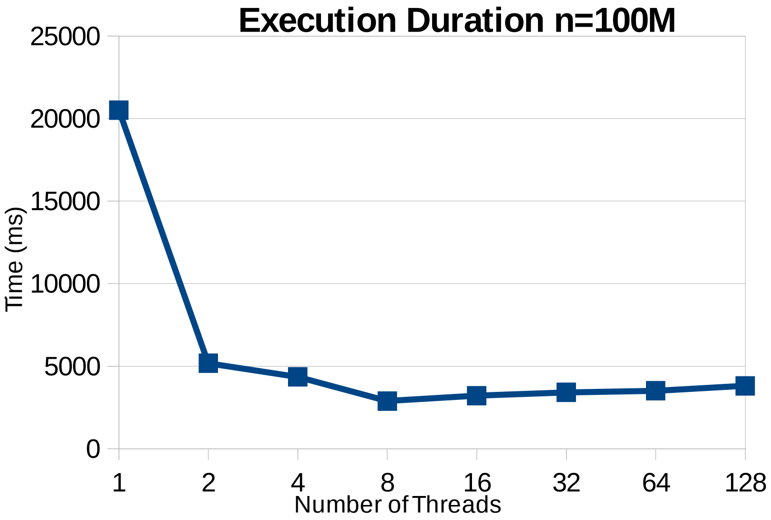
```
Input randomArray size is 1024000
Sequential execution duration is 230 ms
Parallel execution:
#Threads   Duration   Speedup
   2        136 ms    1.69
   4         50 ms    4.60
   8         40 ms    5.75
  16         40 ms    5.75
  32         43 ms    5.35
  64          56 ms    4.11
 128          83 ms    2.77
```

Test Results

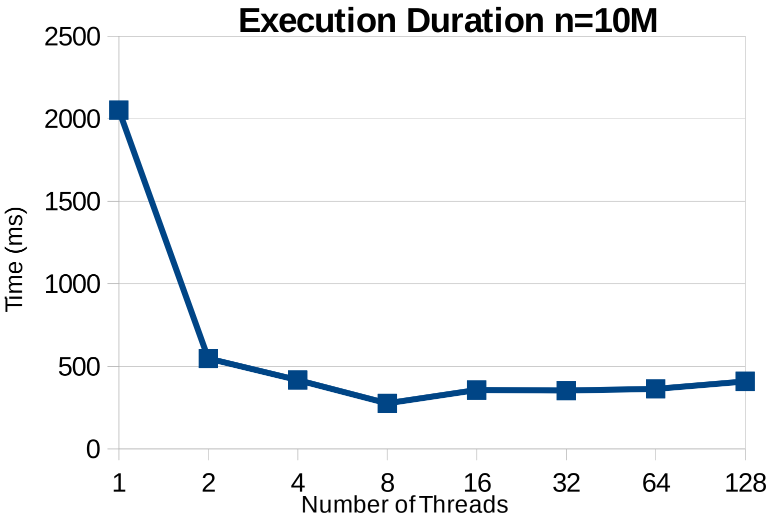
Execution time for sequential and parallel method are put in a chart. Also speedup factors for all methods are calculated in respect to sequential method.

Following tables shows that concurrency is good until number of threads exceeds number of physical CPU cores. Maximum speedup on 8 core machine is 7.33x not 8x. Also speedup curve shape fits Amdahl’s Law.

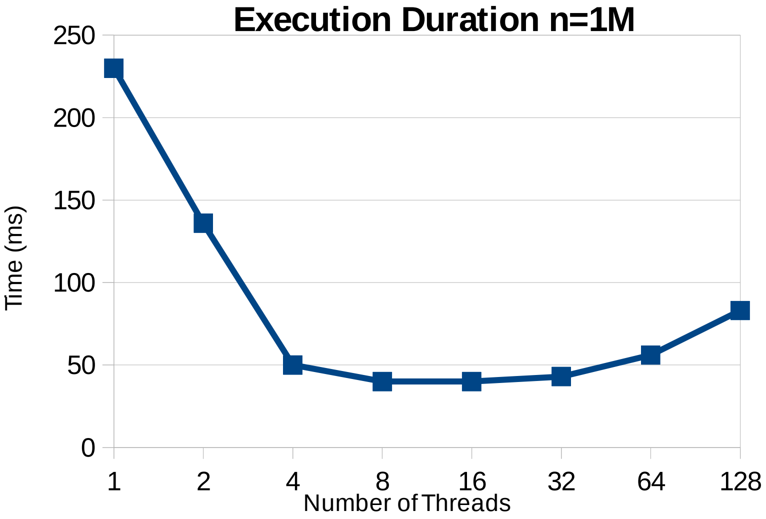
Sequential execution method uses recursive merge sort however parallel execution uses iteration since there is no way to use merge sort recursion between different threads. Recursive function calls brings extra overhead to sorting which results slower execution time. This might be the reason why parallel execution gives much better execution time.



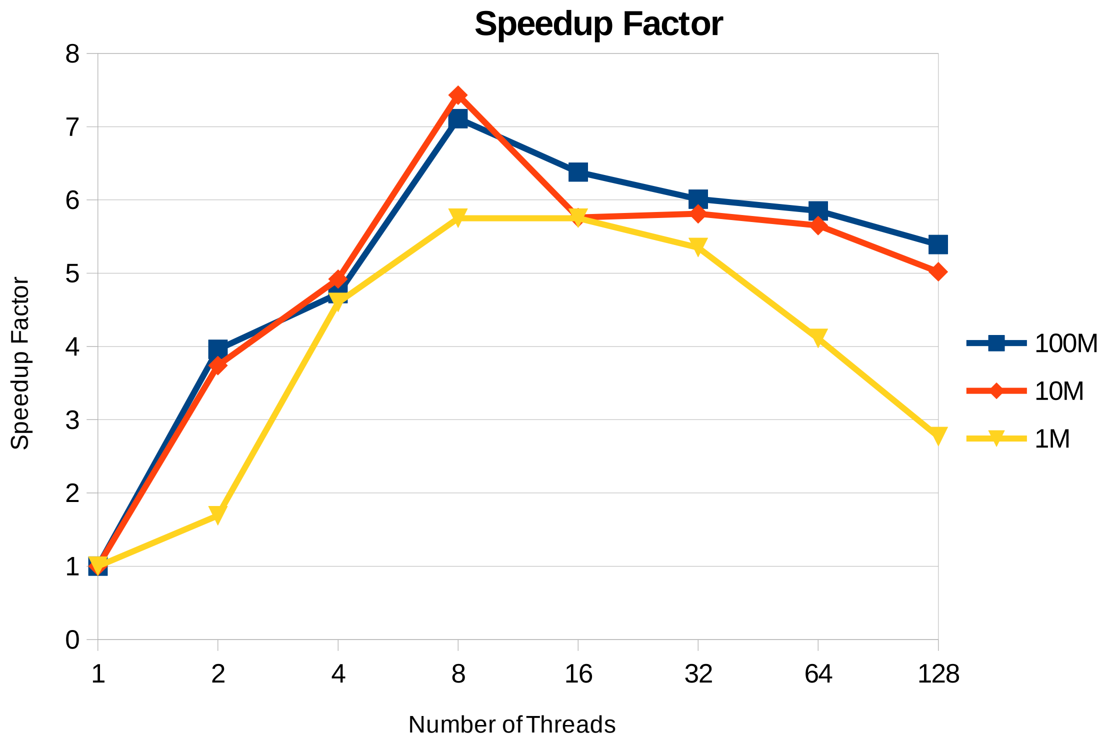
Best execution time is 2885 ms with 8 threads



Best execution time is 276 ms with 8 threads



Best execution time is 40 ms with 8 threads



Best speedup factor is 7.33x with 8 threads