PDP Lab 5 – Rad George-Rares 936/2

Problem:

Goal

The goal of this lab is to implement a simple but non-trivial parallel algorithm.

Requirement

Perform the multiplication of 2 polynomials. Use both the regular $O(n^2)$ algorithm and the Karatsuba algorithm, and each in both the sequential form and a parallelized form. Compare the 4 variants.

The documentation will describe:

- the algorithms,
- the synchronization used in the parallelized variants,
- the performance measurements

1.Simple sequential algorithm:

Normal multiplication of polynomials, we multiply all coefficients of one polynomial with all coefficients of the other one and sum the coeffs with the same degree.

```
for (int i = 0; i < a.getCoefficients().size(); i++) {
    // Multiply the current term of first polynomial
    // with every term of second polynomial.
    for (int j = 0; j < b.getCoefficients().size(); j++) {
        //if (a.getCoefficients().get(i) != 0 && b.getCoefficients().get(j) != 0)
            resultCoefficients.set(i + j,  resultCoefficients.get(i+j) + a.getCoefficients().get(i) * b.getCoefficients().get(j));
    }
}
```

2.Parallel simple sequential algorithm:

This time we try to multiply the same way as above, but this time we do it on parts of one polynomial with the whole second one. We split the first one in degree/number of threads parts and multiply them accordingly.

```
@Override
public void run() {

    synchronized (result) {

        for (int i = startPos; i < stopPos; i++) {
            for (int j = 0; j < a.getDegree(); j++) {
                //if (a.getCoefficients().get(i) != 0 && b.getCoefficients().get(j) != 0)
                    result.set(i + j, result.get(i +j) + a.getCoefficients().get(i) * b.getCoefficients().get(j));
            }
        }
        System.out.println(result);
    }

}
```

3.Simple Karatsuba algorithm:

The Karatsuba method is used when the cost ratio of one multiplication and one addition is greater than 3, because then it is more efficient than the normal way. It can be used to multiply polynomials or big numbers. It splits the polynomial in three parts, which result from simple computations of the coefficients. I use the recursive approach of this, when I keep splitting until the size is 1.(source: https://eprint.iacr.org/2006/224.pdf)

Algorithm 1 ***Recursive KA, C = KA(A, B)***
INPUT: *Polynomials* $A(x)$ *and* $B(x)$
OUTPUT: $C(x) = A(x)\ B(x)$
$N = max(degree(A),\ degree(B)) + 1$
if $N == 1$ return $A \cdot B$
Let $A(x) = A_u(x)\ x^{N/2} + A_l(x)$
and $B(x) = B_u(x)\ x^{N/2} + B_l(x)$
$D_0 = KA(A_l, B_l)$
$D_1 = KA(A_u, B_u)$
$D_{0,1} = KA(A_l + A_u, B_l + B_u)$
return $D_1 x^N + (D_{0,1} - D_0 - D_1)x^{N/2} + D_0$

I follow this algorithm:

```
for (int i = 0; i < halfSize; i++) {
    aLow.set(i, A.get(i));
    aHigh.set(i, A.get(halfSize + i));
    aLowHigh.set(i, aHigh.get(i) + aLow.get(i));

    bLow.set(i, B.get(i));
    bHigh.set(i, B.get(halfSize + i));
    bLowHigh.set(i, bHigh.get(i) + bLow.get(i));
}

var productLow = karatsubaSequential(aLow, bLow);
var productHigh = karatsubaSequential(aHigh, bHigh);
var productLowHigh = karatsubaSequential(aLowHigh, bLowHigh);
```

3.Parallel Karatsuba algorithm:

The method is the same, but instead of calling the functions sequentially, I use the java implementation of the thread pool. source(https://www.baeldung.com/thread-pool-java-and-guava)

```
ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.newCachedThreadPool();
Callable<ArrayList<Integer>> task1 = () -> karatsubaSequential(aLow, bLow);
Callable<ArrayList<Integer>> task2 = () -> karatsubaSequential(aHigh, bHigh);
Callable<ArrayList<Integer>> task3 = () -> karatsubaSequential(aLowHigh, bLowHigh);

Future<ArrayList<Integer>> futureProductLow = executor.submit(task1);
Future<ArrayList<Integer>> futureProductHigh = executor.submit(task2);
Future<ArrayList<Integer>> futureProductLowHigh = executor.submit(task3);

var productLow = futureProductLow.get();
var productHigh = futureProductHigh.get();
var productLowHigh = futureProductLowHigh.get();
```

5.Performance

Degree	Seq Normal	Parallel Normal	Seq Karatsuba	Parallel Karatsuba
4	**0.08899**	**0.00399**	**0.01100**	**0.01799**
8	**0.08599**	**0.00499**	**0.01200**	**0.02200**
16	**0.08899**	**0.00600**	**0.01800**	**0.02099**