PDP Lab 7 – Rad George-Rares 936/2

Problem:

Goal

The goal of this lab is to implement a distributed algorithm using MPI.

Requirement

Perform the multiplication of 2 polynomials, by distributing computation across several nodes using MPI. Use both the regular O(n2) algorithm and the Karatsuba algorithm. Compare the performance with the "regular" CPU implementation from lab 5.

Algorithms:

O2:

```
        MPI.COMM_WORLD.Send(aCoef,  offset: 0, aCoef.length,MPI.INT,i, tag: 0);
        MPI.COMM_WORLD.Send(bCoef,  offset: 0,bCoef.length,MPI.INT,i, tag: 0);
        MPI.COMM_WORLD.Send(bufferStart,  offset: 0, count: 1,MPI.INT,i, tag: 0);
        MPI.COMM_WORLD.Send(bufferStop,  offset: 0, count: 1,MPI.INT,i, tag: 0);
    }

    int[] result = new int[2*aCoef.length-1];
    for(int i = 1 ;i<n;i++) {
        int[] currentResult = new int[2*aCoef.length-1];
        MPI.COMM_WORLD.Recv(currentResult,  offset: 0,  count: 2*aCoef.length-1, MPI.INT, i,  tag: 0);

        for(int j = 0; j<2*aCoef.length-1;j++)
        {
            result[j] +=currentResult[j];
        }

    }
```

```
MPI.COMM_WORLD.Recv(first,  offset: 0, polySize, MPI.INT,  source: 0,  tag: 0);
MPI.COMM_WORLD.Recv(second,  offset: 0, polySize, MPI.INT,  source: 0,  tag: 0);
MPI.COMM_WORLD.Recv(start,  offset: 0,  count: 1, MPI.INT,  source: 0,  tag: 0);
MPI.COMM_WORLD.Recv(stop,  offset: 0,  count: 1, MPI.INT,  source: 0,  tag: 0);

for (int i = start[0]; i < stop[0]; i++) {

    for (int j = 0; j < polySize; j++) {
        re[i + j] += first[i] * second[j];
    }
}
var a = new Polynomial(re);
System.out.println("Partial result of rank " + MPI.COMM_WORLD.Rank() + " is: ");
System.out.println(a.toString());
MPI.COMM_WORLD.Send(re,  offset: 0, re.length, MPI.INT,  dest: 0,  tag: 0);
```

Karatsuba:

```
MPI.COMM_WORLD.Send(aLow, offset: 0,aLow.length,MPI.INT, dest: currentId+lengths[0], tag: 2);
MPI.COMM_WORLD.Send(bLow, offset: 0,bLow.length,MPI.INT, dest: currentId+lengths[0], tag: 3);
lengths[1] = aLowHigh.length;
lengths[2] = bLowHigh.length;
MPI.COMM_WORLD.Send(lengths, offset: 0,lengths.length,MPI.INT, dest: currentId+2*lengths[0], tag: 1);
MPI.COMM_WORLD.Send(aLowHigh, offset: 0,aLowHigh.length,MPI.INT, dest: currentId+2*lengths[0], tag: 2);
MPI.COMM_WORLD.Send(bLowHigh, offset: 0,bLowHigh.length,MPI.INT, dest: currentId+2*lengths[0], tag: 3);

resultHigh = karatsubaRec(aHigh,bHigh,lengths[0],currentId);

//get the results
int[] lowSize = new int[1];
int[] lowHighSize = new int[1];
MPI.COMM_WORLD.Recv(lowSize, offset: 0, count: 1,MPI.INT, source: currentId+lengths[0], tag: 4);
MPI.COMM_WORLD.Recv(lowHighSize, offset: 0, count: 1,MPI.INT, source: currentId+2*lengths[0], tag: 4);
resultLow = new int[lowSize[0]];
resultLowHigh = new int[lowHighSize[0]];

MPI.COMM_WORLD.Recv(resultLow, offset: 0,resultLow.length,MPI.INT, source: currentId+lengths[0], tag: 5);
MPI.COMM_WORLD.Recv(resultLowHigh, offset: 0,resultLowHigh.length,MPI.INT, source: currentId+2*lengths[0], tag: 5);

noProc -= 2 * ((noProc / 3));
```

```
MPI.COMM_WORLD.Recv(lenghts, offset: 0, count: 4,MPI.INT,MPI.ANY_SOURCE, tag: 1);
int[] a = new int[lenghts[1]];
int[] b = new int[lenghts[2]];

int source = lenghts[3];
MPI.COMM_WORLD.Recv(a, offset: 0,lenghts[1],MPI.INT,source, tag: 2);
MPI.COMM_WORLD.Recv(b, offset: 0,lenghts[2],MPI.INT,source, tag: 3);

int[] result = karatsubaRec(a,b,lenghts[3],currentId);
//System.out.println(source);
var poli = new Polynomial(result);
System.out.println("partial result in worker");
System.out.println(poli.toString());

int[] resultSize = new int[1];
resultSize[0] = result.length;

MPI.COMM_WORLD.Send(resultSize, offset: 0, count: 1,MPI.INT,source, tag: 4);
MPI.COMM_WORLD.Send(result, offset: 0,result.length,MPI.INT,source, tag: 5);
```

Distribution and Communication:

Since I am running on a quadcore processor I split the work between 1 main thread and 3 worker nodes.

For the O2 algorithm we split the polynomials in chunks depending on the number of processes available. We send the parts from the main node to the workers and the workers send the result back. The main node just adds all the results to the final result.

For the Karatsuba algorithm we use a recursive approach in this manner:

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 $D_1 x^N + (D_{0,1} - D_0 - D_1)x^{N/2} + D_0$

(source: https://eprint.iacr.org/2006/224.pdf)

The polynomial is split in three parts until each part has size one, and then the computation in the picture is applied. We send from the main node the low and high parts along with their sizes in order to compute D_01. The worker node just calls the karatsubaRecursive function on these low and high parts and sends the result back to the main node.

For polynomial with degree 4.

Karatsuba: 0.07s

O2: 0.09s