Implementation of big integer arithmatic

Sarbajit Ghosh

January 9, 2021

1 Calculation of optimum limb size

Consider polynomial representation of large integer. Let, limb size $\theta = 2^m$. Now let P be an integer. We have,

$$P := p_0 + p_1\theta + p_2\theta^2 + \cdots + p_{n-1}\theta^{n-1}$$

where $0 \le p_0, \ldots, p_{n-1} < 2^m$ and $0 \le p_{n-1} < 2^\delta$. We have to find proper value of limb m, [here integer size 127] such that memory waste is minimum.

Constraints from addition: Consider polynomial representation of two integers P and Q,

$$P := p_0 + p_1\theta + p_2\theta^2 + \cdots + p_{n-1}\theta^{n-1},$$
$$Q := q_0 + q_1\theta + q_2\theta^2 + \cdots + q_{n-1}\theta^{n-1}$$

Their addition is given by,

$$P + Q := (p_0 + q_0) + (p_1 + q_1)\theta + \cdots + (p_{n-1} + q_{n-1})\theta^{n-1}$$

If we add two m bit numbers then their addition will be at most $2m$ bit numbers. Here we are using box size 64-bit. To prevent overflow box size limb size can not more than 63. This constraint we have from addition.

Constraints from multiplication: Multiplication of two integers in polynomial representation is given by, $M = P * Q$ where,

$$M := m_0 + m_1\theta + m_2\theta^2 + \cdots + m_{2n-2}\theta^{2n-2}$$

and each m_i is given by

$$m_i := \sum_{0 \le j < n,\, 0 \le k < n,\, i = j + k} p_j q_k$$

.

Now when we multiplication of two m bit integer will yield a $2m$ bit integer. Again in the some these $2m$ bit integer will be add upto n times. So the resulting integer will be of size $\lceil \log n \rceil + 2m$ bit. Therefore,

$$\lceil \log n \rceil + 2m \le 64$$

. Or in other notation,

$$\lceil \log n \rceil + 2m < 64$$

But again after the multiplication we have to perform carry operation. Which consists of one addition so it might increase the size by at most one bit, hence we should have,

$$\lceil \log n \rceil + 2m < 63$$

Now here we are performing 127-bit integer arithmetic. So the number of required boxes will be given by, $n := \lceil \frac{127}{m} \rceil$. To determine the optimal value of m we will perform trail and error with the help of above equation. Thus let us try for $m = 31$, [thus $n := \lceil \frac{127}{m} \rceil = 5$] then we find that,

$$\lceil \log 5 \rceil + 2 * 31 = 64 \not< 63$$

But here we get atleast one clue that number of boxes will be 5, so let us try for limb size $m := \lfloor \frac{127}{5} \rfloor = 26$. We see that,

$$\lceil \log 5 \rceil + 2 * 26 = 55 < 63$$

In general it is true for all values of all,$26 \leq \theta \leq 30$ the above equation is satisfied.
For our implementation purpose we are taking limb size to be $\theta = 2^{28}$ and and we are using 64-bit integer for our implementation purpose.

2 Sample Run

Our software specification: Our program is written for handling handling 127-bit integer arithmetic. We have written it such a way for addition of two $2^{127} - 1$ it will able to handle 1-bit overflow. It can multiply integers of 127-bit.

Now here we are giving two sample run. Second is for to showing the output of difference when 2nd integer is large. Output of the code is as follows. Also we have shown our output in the following screen shot also the verification part is in the Figure below.

```
Enter number one:7ffffffffffffffffffffffffffffff
Enter number two:6fff45ffffffeffffffbbf873ffffff
Compact representation of sum
efff45ffffffeff          ffffbbf873fffffe
Compact representation of mult
037ffa2fffffff7f          ffffddfc39ffffff          1000ba0000000100
    000044078c000001
Compact representation of diff
1000ba0000000100          000044078c000000


Enter number one:6fff45ffffffeffffffbbf873ffffff
Enter number two:7ffffffffffffffffffffffffffffff
Compact representation of sum
efff45ffffffeff          ffffbbf873fffffe
Compact representation of mult
037ffa2fffffff7f          ffffddfc39ffffff          1000ba0000000100
    000044078c000001
2nd integer is larger
Compact representation of diff
1000ba0000000100          000044078c000000
```

Verification of output using python

```
python3 -c 'print(hex( 0x7ffffffffffffffffffffffffffffff + 0
    x6fff45ffffffeffffffbbf873ffffff ))'

0xefff45ffffffeffffffbbf873fffffe

python3 -c 'print(hex( 0x7ffffffffffffffffffffffffffffff * 0
    x6fff45ffffffeffffffbbf873ffffff ))'

0x37ffa2fffffff7fffffddfc39ffffff1000ba0000000100000044078c000001

```

```
python3 -c 'print(hex( 0x7fffffffffffffffffffffffffffffff - 0x6fff45fffffffeffffffbbf873ffffff ))'

0x1000ba000000010000000044078c000000
```

Screen shot:

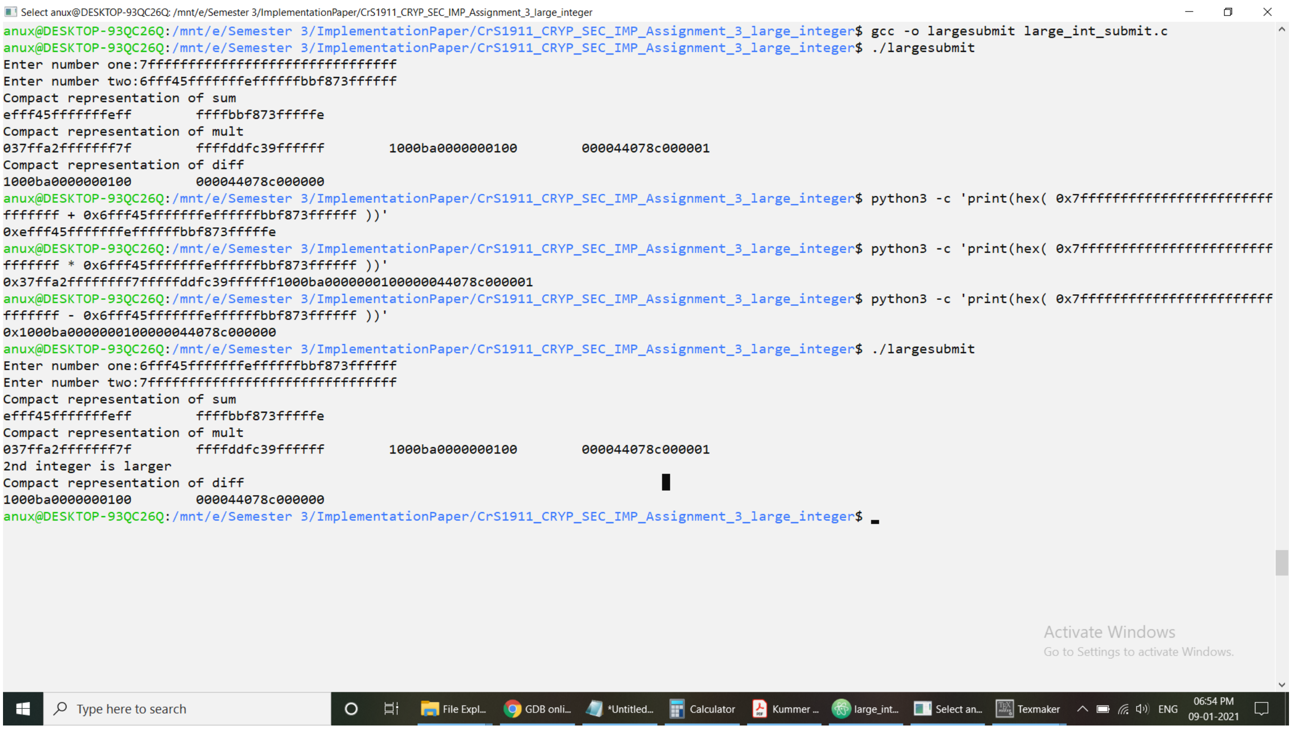

Figure 1: Demo of program & verification of outputs using python3