Q2:

Environment settings:

ring dimension: 2^12

Batchsize: 8

rescaleTech = FLEXIBLEAUTO

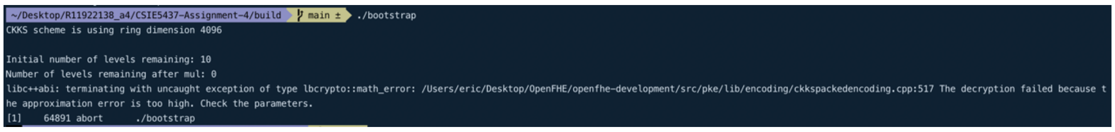
total level with bootstrap: 30

x1 = vector<int>{2, 2, 2, 2}, start at level 20

x2 = vector<int>{2, 2, 2, 2}, start at level 0

I multiply x1 with x2 by 11 times (ie. 2^12 in each element of vector)

Before bootstrap:

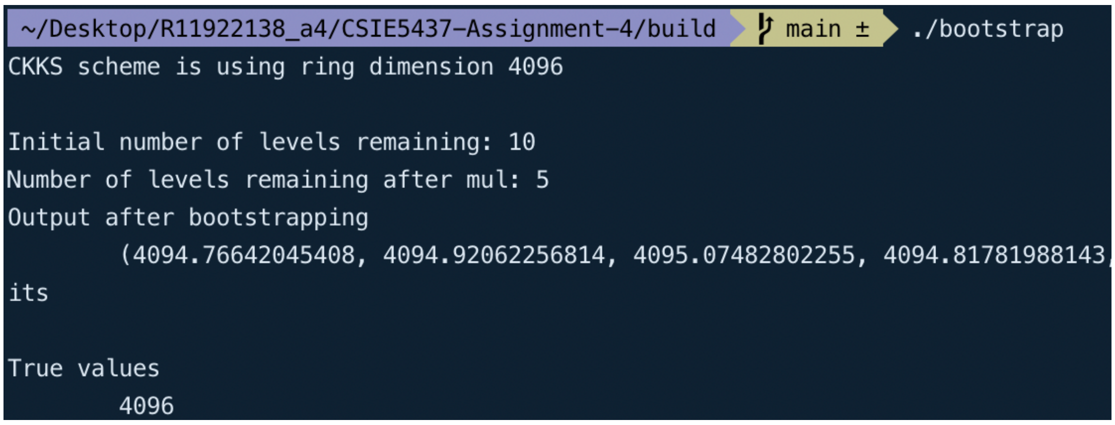


```
~/Desktop/R11922138_a4/CSIE5437-Assignment-4/build  main ±  ./bootstrap
CKKS scheme is using ring dimension 4096

Initial number of levels remaining: 10
Number of levels remaining after mul: 0
libc++abi: terminating with uncaught exception of type lbcrypto::math_error: /Users/eric/Desktop/OpenFHE/openfhe-development/src/pke/lib/encoding/ckkspackedencoding.cpp:517 The decryption failed because t
he approximation error is too high. Check the parameters.
[1]    64891 abort      ./bootstrap
```

In the picture, it shows that the remaining levels of x1 has turn to 0. Since x1 starts at level 20, it only left 10 levels to use. A 11 times multiplication exhausts its levels and even exceed. The result can't be decrypted due to the error is extremely big.

After bootstrap:



```
~/Desktop/R11922138_a4/CSIE5437-Assignment-4/build  main ±  ./bootstrap
CKKS scheme is using ring dimension 4096

Initial number of levels remaining: 10
Number of levels remaining after mul: 5
Output after bootstrapping
        (4094.76642045408, 4094.92062256814, 4095.07482802255, 4094.81781988143
its

True values
        4096
```

In the picture, we can see the errors significantly reduce, from can't be decrypted to about 0.03% of errors. The levels of x1 also increase, since the goal of bootstrap is to increase the number of available levels we have, meaning we can do more multiplications.

Q3:

The main challenge multiplication of matrix and vector is we need to use ***Halevi and Shoup*** diagonal method.

First step:

Construct diagonal matrix

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    if (i >= (int)M.size() || j >= (int)M[0].size())
      continue;
    // TODO: construct the diagonal matrix according to M
    diagonal[j][i] = M[i][(i + j) % (int)M[0].size()];
  }
}
```

Second step:

Initialize sum, since the original sum is just a ciphertext without any initialization. I initialize it as a vector of n zeros.

```
vector<double> sum_vec(n);
ptxtsum = context->MakeCKKSPackedPlaintext(sum_vec);
sum = context->Encrypt(key_pair.publicKey, ptxtsum);
```

Third step:

I think this is the most challenging step, since the providing tutorial of diagonal method doesn't mention it. The rotation of CKKS will rotate the whole numslots, but the vector can be any sizes. It will pad numslots - vector.size zeros at the end of ciphertext. When rotation, the zeros will intervene our computation. The solution is to add a copy of vector at the end of origin vector, so the new ciphertext will look like this: [v, v, 0 ,0, …..0]. Therefore, the rotation of this new vector will work as we want.

```
// make v = [v,v] because of rotation
Ciphertext<DCRTPoly> temp = ctx;
for(int j = 0; j < (int)M[0].size(); j++){
  temp = context->EvalRotate(temp, -1);
}
v = context->EvalAdd(v, temp);
```

Last step:

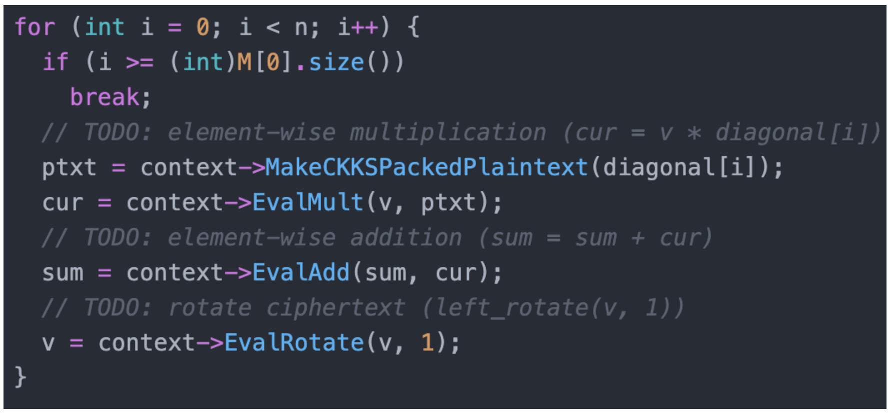
Two points to note. First, we need to pack the row of diagonal matrix as CKKS plaintext. Second, the origin for loop didn't have the break condition, so it will run until i==n. But, the diagonal matrix only have M[0].size * M.size valid data although its size is n*n. So, I break out the loop when i is bigger than M[0].size.

Test environment: 2020 M1 macbook air (8GB ram)

Without break statement: 3:17 min (197sec)

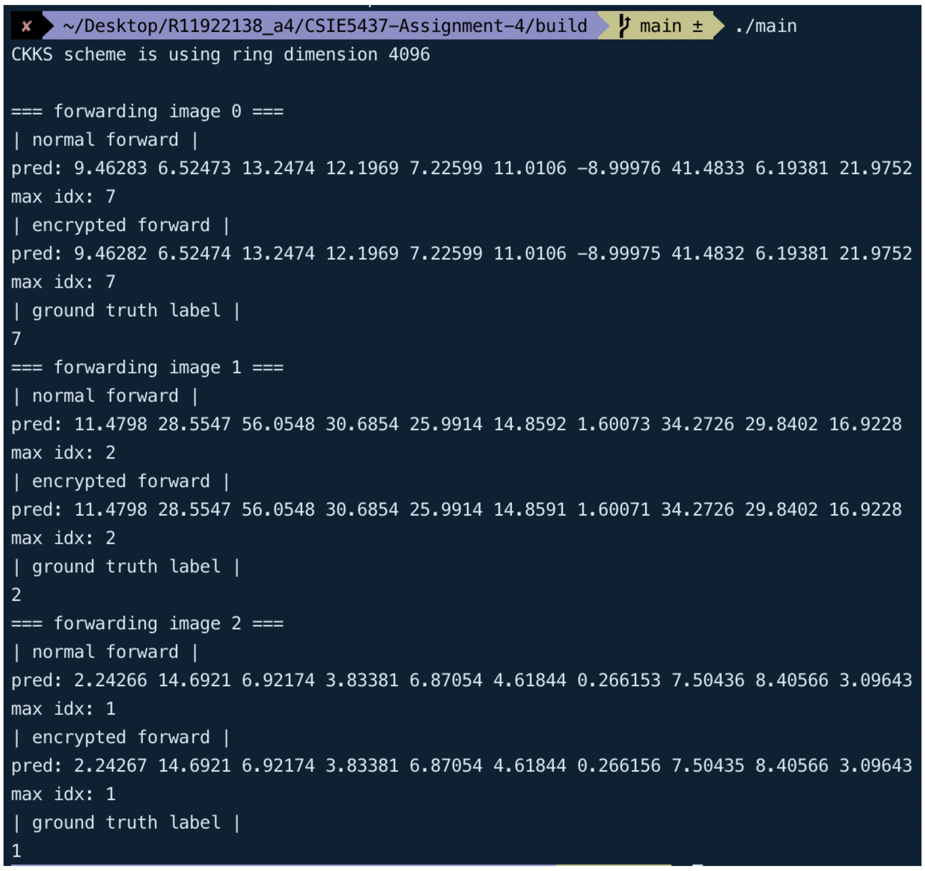
With break statement: 10:36min(636sec)

Speedup: 3.2284



```
for (int i = 0; i < n; i++) {
  if (i >= (int)M[0].size())
    break;
  // TODO: element-wise multiplication (cur = v * diagonal[i])
  ptxt = context->MakeCKKSPackedPlaintext(diagonal[i]);
  cur = context->EvalMult(v, ptxt);
  // TODO: element-wise addition (sum = sum + cur)
  sum = context->EvalAdd(sum, cur);
  // TODO: rotate ciphertext (left_rotate(v, 1))
  v = context->EvalRotate(v, 1);
}
```

Result:



```
✘ ~/Desktop/R11922138_a4/CSIE5437-Assignment-4/build  main ±  ./main
CKKS scheme is using ring dimension 4096

=== forwarding image 0 ===
| normal forward |
pred: 9.46283 6.52473 13.2474 12.1969 7.22599 11.0106 -8.99976 41.4833 6.19381 21.9752
max idx: 7
| encrypted forward |
pred: 9.46282 6.52474 13.2474 12.1969 7.22599 11.0106 -8.99975 41.4832 6.19381 21.9752
max idx: 7
| ground truth label |
7
=== forwarding image 1 ===
| normal forward |
pred: 11.4798 28.5547 56.0548 30.6854 25.9914 14.8592 1.60073 34.2726 29.8402 16.9228
max idx: 2
| encrypted forward |
pred: 11.4798 28.5547 56.0548 30.6854 25.9914 14.8591 1.60071 34.2726 29.8402 16.9228
max idx: 2
| ground truth label |
2
=== forwarding image 2 ===
| normal forward |
pred: 2.24266 14.6921 6.92174 3.83381 6.87054 4.61844 0.266153 7.50436 8.40566 3.09643
max idx: 1
| encrypted forward |
pred: 2.24267 14.6921 6.92174 3.83381 6.87054 4.61844 0.266156 7.50435 8.40566 3.09643
max idx: 1
| ground truth label |
1
```