CMDA 3634 Spring 2018 Homework 05

N. Chalmers

April 29, 2018

You must complete the following task by 5pm on Tuesday 04/24/18.

Your write up for this homework should be presented in a LaTeX formatted PDF document. You may copy the LaTeX used to prepare this report as follows

1. Click on this link
2. Click on Menu/Copy Project.
3. Modify the HW05.tex document to respond to the following questions.
4. Remember: click the Recompile button to rebuild the document when you have made edits.
5. Remember: Change the author.

Each student must individually upload the following files to the CMDA 3634 Canvas page at https://canvas.vt.edu

1. `firstnameLastnameHW05.tex` LaTeX file.
2. Any figure files to be included by `firstnameLastnameHW05.tex` file.
3. `firstnameLastnameHW05.pdf` PDF file.

In addition, all source code must be submitted to an online git repository as follows:

1. While on the webpage for your git repository, go to Settings → Collaborators.
2. Add nchalmer@vt.edu and zjiaqi@vt.edu as collaborators.
3. Make a folder named HW05 in you repository and store all relevant source files to this assignment in this folder. Ensure your assignment files compile with `make`.

You must complete this assignment on your own.

100 points will be awarded for a successful completion.
Extra credit will be awarded as appropriate.

ElGamal Public-key Cyptography

Through the previous assignments, we've manged to develop a functional program which performs encryption and decryption of string messages using an n bit ElGamal cryptographic system. We can set up the cryptographic system by finding the values of the public key (p, g, h). We can cast characters in strings into elements of $\mathbb{Z}_p$ and encrypt them. Finally, we can decrypt elements of $\mathbb{Z}_p$ and cast them back to characters if we know the secret key x.

In the case where we imagined that we do not have access to the secret key x, we have used MPI and OpenMP to crack our ElGamal encryption in parallel using many CPU cores. Even with that level of parallelism, it is easy to see that our program can use n bit encryption where n is large enough that we still cannot find the secret key in a practical amount of time.

In this final assignment, let's turn our previous monolithic programs into something a bit more useful by separating the basic operations into their own programs which run and save their results in files. Once we've accomplished that, let's tackle our final hurdle and use GPU acceleration to enable us to break even the strongest encryption our program can produce, namely $n = 31$ bit cryptographic systems.

Part 1: Setup, Encrypt, and Decrypt

In the HW05 folder of the instructor's github the `makefile` has been altered so that four separate programs will be compiled when you type `make` or `make all`. The names of the programs are `setup`, `encrypt`, `decrypt`, and `cudaDecrypt`. They can be compiled separately by typing `make setup`, `make encrypt`, etc. Note that the function `cudaDecrypt` requires CUDA's `nvcc` compiler, and therefore the New River cluster, to compile.

Q1(10 points) The program in `setup.c` is meant to perform the ElGamal cryptosystem setup routines and write the public key information to a file named `public_key.txt`. Complete the program so that, after querying the user for a bit length n, it write the bit length n, prime p, generator g, and number h, to a file named `public_key.txt`. For example, after running `setup`, the file `public_key.txt` could look something like,

```
25
30080879
28192225
28973487
```

Which means $n = 25$, $p = 30080879$, $g = 28192225$, and $h = 28973487$.

Q2(20 points) The program in `encrypt.c` is meant to input a message string from the user and use the public key information in the file `public_key.txt` to encrypt the message. Edit the program so that it after reading the message as a string from the user the program reads the public key information from `public_key.txt` and encrypts the message. Afterwards, write the the number of cyphertext pairs, and each cyphertext pair $(\hat{m}, a)$, into a file called `message.txt`. The contents of `message.txt` should look something like,

```
14
28395666 9350445
3562739 5794092
10156865 16818778
2858293 6363325
29107964 21484289
3274137 24856781
19487449 15592980
18341691 30012651
15930624 18293246
11025926 2833780
3862999 23130298
16935347 23161724
```

```
21230787 9009798
25866655 2102881
```

which indicates that there are 14 cyphertext entries, $(\hat{m}, a)_1 = (28395666, 9350445)$, $(\hat{m}, a)_2 = (3562739, 5794092)$, etc.

Q3.1(20 points) The program in `decrypt.c` is meant to read the contents of `public_key.txt` and `message.txt` and decrypt the message. After reading the files, the program will ask the user for the secret key. Complete the program so that if the user inputs $x = 0$ or an incorrect secret key, the program will attempt to break the encryption by searching for the correct secret key x. Once the correct secret key is found, decrypt the contents of `message.txt` and print the message to the terminal.

Q3.2(5 points) Using the `decrypt` function or one of your programs from previous assignments, give a rough estimate of how long you would expect a serial code to break an $n = 31$ bit encryption. Be sure to explain how you arrived at your estimate.

Running at n = 31 with the given files in the bonus, the program was able to find the key in 2.1 seconds. This is similar to other previous programs that were running in serial

Part 2: GPU Accelerated Decryption In this part, we will use Nvidia's CUDA programming language to add GPU acceleration to the `decrypt` function. In the HW05 folder is the CUDA file `cudaDecrypt.cu`. Begin this section by copying the contents of the `main` function in your complete `decrypt.c` file to to `cudaDecrypt.cu`

Q4.1(10 points) Examine the loop where we search for the secret key. Describe in words how/why we could perform this section of the program in a GPU kernel. What would the kernel require as input? How much output will the kernel produce, and how do we access it on the host? How much memory must we allocate on the device? How many threads per block would be reasonable? Why?

This loop is able to be parallelized because each loop is independent on the others. The loop is able to run without affecting the other loops, and there will not be a race condition to affect the end result. It will require the input of p, g, h, and a pointer to hold the result. This pointer will be mem copied to the device, will be given the value, and then be mem copied back to the host.

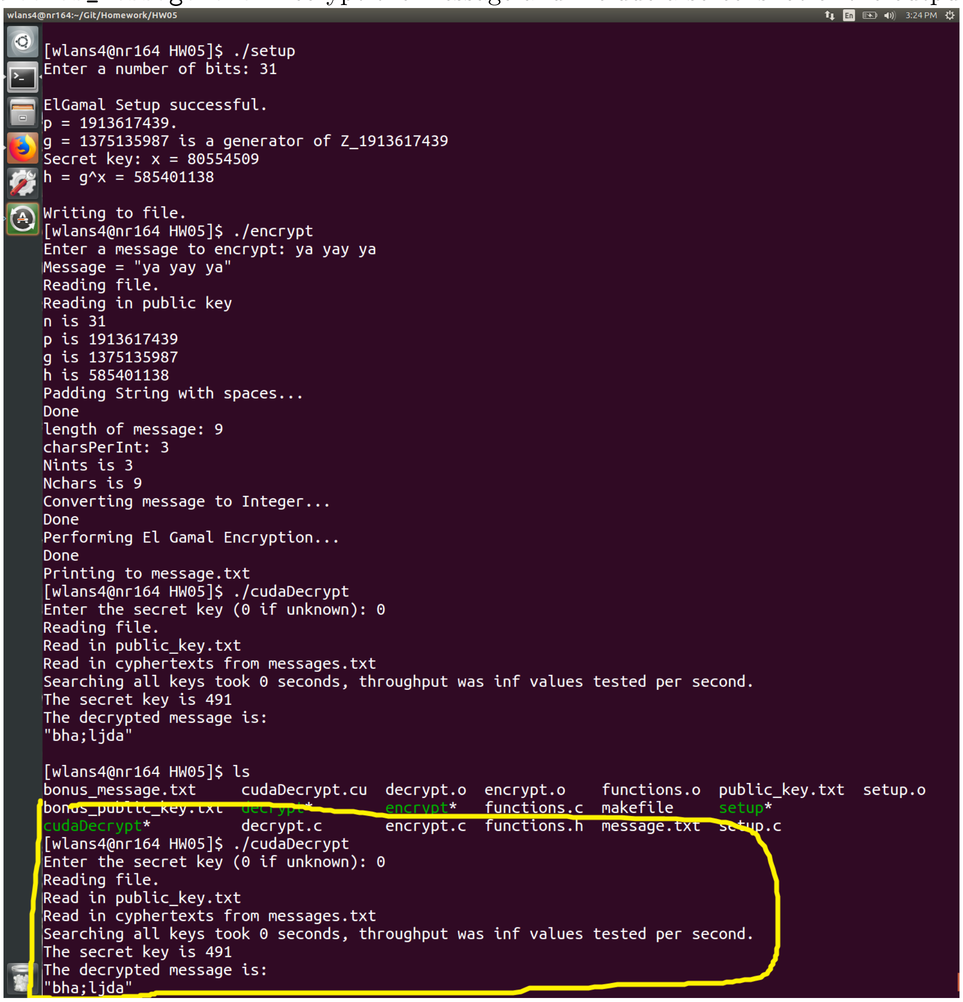
Q4.2(35 points) Edit the `cudaDecrypt.cu` file to implement the kernel you described in Q4.1, i.e. add a GPU kernel function which can find the secret key x. If any `__device__` functions are required add them to `cudaDecrypt.cu` as well (you may need to rename them to not conflict with functions in `functions.c`). Be sure to `cudaMalloc` any arrays the kernel will need, and use `cudaMemcpy` to transfer memory between host and device.

Done, on github

Q4.3(5 points) Experiment with decrypting several different n-bit encryptions. In general, how much faster (in time and throughput) is the GPU accelerated version of your decryption program? Assuming you could scale perfectly in parallel with MPI or OpenMP, how many CPU cores would you need in order to achieve the same performance as one of the P100 GPUs on the New River cluster?

In general, the cuda version was much faster. When using 31 bit encryption without cuda, sometimes the program would even hang and never ouput a result. Conversely, the cuda file was able to decrypt within a few seconds, usually under 5. It is hard to say how many cores would be needed to decrypt at the same rate. Based on examples where cuda ran 300 times faster, it would seem to need 300 times the number of cores

Bonus:(20 points) In the HW05 folder of the instructor's github there is a file named `bonus_public_key.txt` and `bonus_message.txt`. Decrypt the message and include a screenshot of the output here.



```
[wlans4@nr164 HW05]$ ./setup
Enter a number of bits: 31

ElGamal Setup successful.
p = 1913617439.
g = 1375135987 is a generator of Z_1913617439
Secret key: x = 80554509
h = g^x = 585401138

Writing to file.
[wlans4@nr164 HW05]$ ./encrypt
Enter a message to encrypt: ya yay ya
Message = "ya yay ya"
Reading file.
Reading in public key
n is 31
p is 1913617439
g is 1375135987
h is 585401138
Padding String with spaces...
Done
length of message: 9
charsPerInt: 3
Nints is 3
Nchars is 9
Converting message to Integer...
Done
Performing El Gamal Encryption...
Done
Printing to message.txt
[wlans4@nr164 HW05]$ ./cudaDecrypt
Enter the secret key (0 if unknown): 0
Reading file.
Read in public_key.txt
Read in cyphertexts from messages.txt
Searching all keys took 0 seconds, throughput was inf values tested per second.
The secret key is 491
The decrypted message is:
"bha;ljda"

[wlans4@nr164 HW05]$ ls
bonus_message.txt     cudaDecrypt.cu  decrypt.o  encrypt.o    functions.o  public_key.txt  setup.o
bonus_public_key.txt  decrypt*        encrypt*   functions.c  makefile     setup*
cudaDecrypt*          decrypt.c       encrypt.c  functions.h  message.txt  setup.c
[wlans4@nr164 HW05]$ ./cudaDecrypt
Enter the secret key (0 if unknown): 0
Reading file.
Read in public_key.txt
Read in cyphertexts from messages.txt
Searching all keys took 0 seconds, throughput was inf values tested per second.
The secret key is 491
The decrypted message is:
"bha;ljda"
```