Project Objective

This project's goal is to create an anti-virus program that uses GPU to parallelize the virus pattern-matching procedure for incoming files. Using CUDA, this project parallelizes the scanning and maintains the effectiveness of finding the matching patterns. ClamAV, an open source anti-virus program [1], provides the virus database for this project. The pattern-matching algorithm chosen in this case is Boyer-Moore algorithm. Note that the program does not attempt to parallelize the algorithm itself; instead, the program launches many cores to execute many copies of the algorithm in parallel.

Compiling and Running the Program

The submission folder contains all the results and compiled binary file. Please find details in table 1.

Folders/Files	Comments
files/	The folder contains all the files that can be used to test the parallel virus scan program
mainPack/	The folder contains one of the virus database files for virus scan
dailyPack/	The foulder contains another virus database files for virus scan
dailyGPUvirus.ndb	Virus dictionary for interpret the output of the program
mainGPUvirus.ndb	Another virus dictionary for interpret the output of the program
convert.py	The python script that converts default database to GPU compatible database
compile.sh	The script to load the GPU module and compile the cuda program
parallel.cu	The CUDA program that does parallel virus scan for each input file
Makefile	Makefile for compiling the program
***.o**	object files from "make"
parallel	precompiled executable for the program

Table 1, Top-tier directory

To **compile** the program, one needs to make sure CUDA is properly installed on the machine. For compilation, run:

```
Bash$ ./compile.sh
```

or

```
Bash$ make
```

Once the compilation is done, to **run** the program for scanning *yourfile.bin*, one can do:

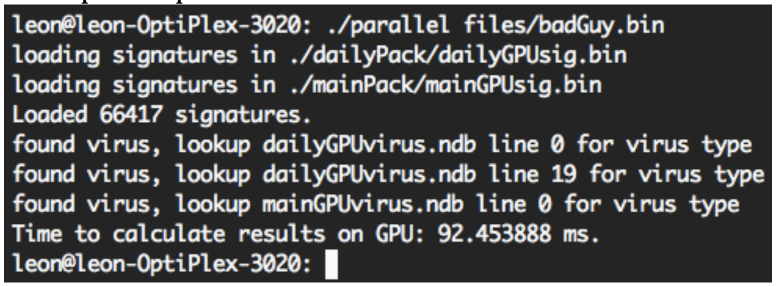
```
Bash$ ./parallel yourfile.bin
```

Note that the usage of the *parallel* program. It is required to give a file as its parameter. I have provided infected binary files in *files* folder. One needs not to worry about the infected files exploiting the host. The files are simply written with some of the known virus signatures from the database and compiled to binary. The files are not runnable and are treated simply as binary dump.

To scan the sample-infected files, one can do:

Bash$ ./parallel files/badGuy.bin

A simple output of this command would be as shown in figure 1.



```
leon@leon-OptiPlex-3020: ./parallel files/badGuy.bin
loading signatures in ./dailyPack/dailyGPUsig.bin
loading signatures in ./mainPack/mainGPUsig.bin
Loaded 66417 signatures.
found virus, lookup dailyGPUvirus.ndb line 0 for virus type
found virus, lookup dailyGPUvirus.ndb line 19 for virus type
found virus, lookup mainGPUvirus.ndb line 0 for virus type
Time to calculate results on GPU: 92.453888 ms.
leon@leon-OptiPlex-3020: 
```

Figure 1, program output

The program would output total number of signatures loaded, number of virus found, and total execution time for the scanning. More information on how to interpret the virus types would be discussed in one of the following sections. The following sections discuss the background information, implementation approach, and results of the project.

Background

As I mentioned in my project proposal, most of the existing anti-virus software not only scans host data in a very time consuming fashion but also takes large amount of CPU resources. The scanning scheme is usually a single-thread program that runs a form of pattern matching algorithm to pair the virus signatures with the bit-stream coming from the files. That being said, there are two approaches to the parallelization: 1. Scanning the incoming file data stream in parallel against different signatures in the virus database. 2. Reading multiple incoming files in parallel and processing them against one signature at a time. In this project, I choose to explore the first approach.

GPU , a SIMD processor, can process regular fixed-size data, meaning that if one of the CUDA cores is working on one element of the array, the other parallel CUDA cores have to work on the elements of the same byte length in the array to achieve parallelization. An array containing elements with different byte length will not work on GPU. Given that pre-requisite, we look into the virus signature pattern and see if it is a good application for GPU.

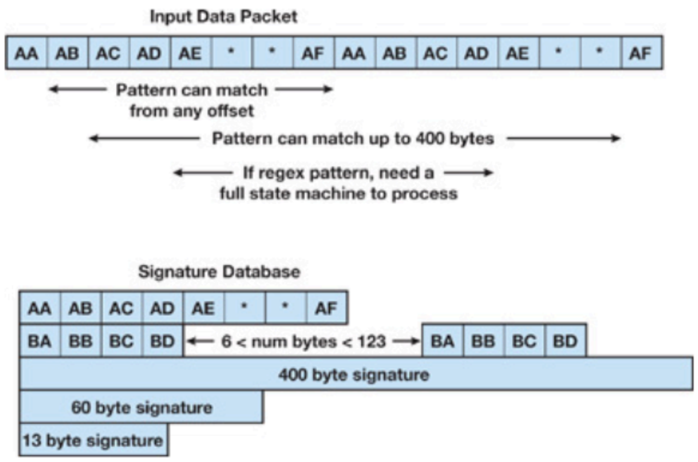

Input Data Packet
AA AB AC AD AE * * AF AA AB AC AD AE * * AF
Pattern can match from any offset
Pattern can match up to 400 bytes
If regex pattern, need a full state machine to process
Signature Database
AA AB AC AD AE * * AF
BA BB BC BD 6 < num bytes < 123 BA BB BC BD
400 byte signature
60 byte signature
13 byte signature

Figure 2, Pattern Matching scheme

Figure 2 shows that each signature pattern can have different length ranging from 4 bytes to 400 bytes. This character prevents the GPU to perform a full scale scanning; however, if the program's goal is to scan the file for certain types of virus, the viruses that have at least 10-byte signatures, the GPU would rapidly increase the pattern matching for all the 10-byte signatures in the database. Since the total number of the exactly 10-byte signatures is rather minimal, I decide to crop the first 10 bytes of all signatures and execute the pattern-matching algorithm. In this way, I use the GPU as an anti-virus accelerator before completing the signature-matching work on the CPU. The GPU uses 10 bytes to determine whether there is a likely match against a signature in the database. If the GPU can find a 10-byte matching, it can narrow down the potential virus thread type. The CPU can then screen the file with the full-length signatures of the potential virus. Nvidia also uses similar approach for anti-virus applicatios [1]. That being said, this project mainly focuses on the GPU parallelizing the scanning and the effectiveness of finding the matching pattern.

Implementation

As I mentioned in the previous section, each of the signature is being cropped to 10-byte chunk (first 10-byte of the signature). I use a python script to do the signature conversion. The raw virus patterns are included in the folders, *mainPack* and *dailyPack*. The files with extension .ndb are the raw files taken from ClamAV [1]. The convert.py file then uses the .ndb as input files to do the conversion.

Once all the data (signatures) is ready for GPU, the CUDA Kernel in parallel.cu can execute the pattern-matching algorithm. Figure 3 below presents an overview of the parallel implementation on GPU of this project.

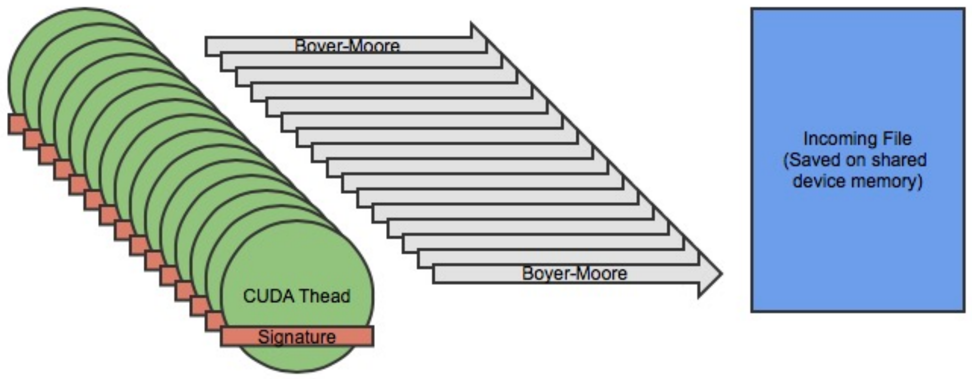


Figure 3, GPU pattern matching

From figure 3, we see that one CUDA thread is allocated for each of the unique signature (10-byte long). Since there are 66417 valid signatures in this case, the total number of threads needed is at least 66K. For convenience, I configured block dimension as 32 x 32 and grid dimension as 32 x 32. The boundary check is done in the CUDA kernel.

Given that each signature is 10-byte long, and we have 64417 signatures. The signatures are loaded into shared device memory as 64417 x 10 matrix, and each matrix element is 8-byte unsigned char. This way, each row of the matrix is a signature. Both the signature matrix and file contents are allocated on the shared device memory. Each thread would occupy one signature from the shared memory, and execute Boyer-Moore pattern-matching algorithm to find the matching where applicable. The Boyer-Moore algorithm is implemented on GPU device functions. One can inspect parallel.cu for source code and references.

Results

I have created a number of sample binary files in the *files* folder to validate the effectiveness of the scanning. Figure 1 presented above shows that the program is able to detect the virus pattern in a relatively short period of time. However, since ClamAV does not provide the source code for their pattern-matching algorithm, I cannot fairly compare the timing results between my GPU program and the CPU program from ClamAV. Also, in order to run the ClamAV program from a host system with unprivileged account, one need to go through messy installation procedure. For that reason, I decide to leave out the ClamAV performance comparison and focus on the functionality of the parallel virus scanning on GPU.

Once the program finishes running there would be a prints out as shown in figure 1. User can now use the directory (mainGPUvirus.ndb and dailyGPUvirus.ndb) to find which virus pattern found matching. The directories are simply collections of different virus names. The order of the pattern matches the order of the virus name, so using the line number provided by the program can provide user what virus is in effect. I am taking this quick approach for user feedback for 2 reasons: 1. The project mainly focuses on exploring the GPU programming. 2. In order to print virus name directly from C program, large buffer is required from device to host, which reduces the memory efficiency of the program.

In conclusion, the program is able to detect matching pattern in parallel in a relatively short time, using the virus database from ClamAV.

References

[1] available online: http://www.clamav.net/index.html
[2] available online:
http://http.developer.nvidia.com/GPUGems3/gpugems3_ch35.html