Final Project Phase 3

Members: Jonathan Jackson, Sriram Arjula, Ryan Luna

Our project direction is in the field of Image Processing and parallel execution of cryptography algorithms (e.g. AES). We have made significant progress towards a working implementation and have developed a larger understanding of the technical requirements to achieve our end goal. This final report will summarize and conclude our work on this project for the semester.   
 The paper we have chosen is “Parallel Implementation of Cryptographic Algorithm: AES Using OpenCL on GPUs”. The goal of the work is to accelerate the implementation of the AES algorithm using parallel computing on graphics processors. The AES algorithm was implemented using Open Computing Language (OpenCL) framework and tested on GPU devices (Nvida 8550M & 8570G). Their results showed that the parallel implementation provided up to a 98.8% speedup in comparison to that of a sequential implementation.

Table

Description automatically generated

Seeing that the work presented in the paper has a wide variety of potential applications, our objective for this project was to build a working example of one. We wished to expand on the work in this paper by integrating image processing and cryptography. Our main goal being a working implementation of a parallelized image encryption/decryption program. Fast and secure data transfer is a huge topic in the IoT and Cloud Computing era. That is our motivating factor, and we believe that parallel computing is key in achieving this.

|  |  |
| --- | --- |
| **Successes** | **Setbacks/Failures** |
| * Sequential Image Encryption/Decryption * “Partial” Parallel Image Encryption/Decryption * Runtime results gathered * Deriving technical limitations of the tools we chose to use (language, libraries, etc.) * Designing possible workarounds to the limitations * Gathering technical knowledge on a high-level implementation of parallel computing | * Working around Python’s Global Interpreter Lock (GIL) * Implementation of OpenCL in the main program * Realizing a true acceleration of the parallelized cryptography algorithm * Developing a truly parallel implementation |

The following table will summarize our successes/setbacks for the project.

We decided on Python as our programming language of choice. Given the timeframe for the project, we felt that using a high-level language might prove to accelerate development given the variety of its useful variety of packages/libraries. Seeing the true limits of parallel computing with Python, a lower-level language such as C/C++ might have been the more optimal choice. The following is our list of tools (packages/libraries) used to develop our program.

|  |  |
| --- | --- |
| **Tools** | |
| * Fernet (cryptography.fernet)   + AES algorithm * Pillow   + Image processing * Numpy   + Array management * IO   + Loading/Saving files * Glob   + File/Path management * Math   + Math * Time   + Gather execution time * Progressbar   + Aesthetically pleasing command line progress indicator | * Random   + Generate random numbers * Image Slicer   + Image processing * Pathlib   + File/Path management * Shutil   + File/Path management * Multiprocessing   + Parallelization via processes * Threading   + Parallelization via threads * Concurrent.futures   + Parallelization via processes OR threads |

The program was executed and tested on the following bit of hardware. AMD Ryzen 5 3600 6-Core Processor (12 CPUs), 16384MB RAM, AMD Radeon RX 580 Series Graphics Processor with 8171 MB dedicated memory. The results are as follows.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Program** | **Overall time (s)** | **Number of images** | **Average Encryption Time (s)** | **Average Decryption Time (s)** | **Average Time Per Item (s)** |
| Sequential | 36.4101 | 27 | 0.3687 | 0.3272 | 0.6959 |
| Parallel v.0 (Multi-Threading, w/ tiles) | 88.8663 | 27 | 0.8592 | 1.2372 | 2.0964 |
| Parallel v.1 (Multi-Threading, No tiles) | 35.9719 | 27 | 0.4494 | 0.4131 | 0.8625 |
| Parallel v.2 (Multi-Processing, No tiles) | 29.931 | 27 | 0.4066 | 0.2665 | 0.6731 |
| Parallel v.3 (Multi-Processing, w/ tiles) | 44.5434 | 27 | 0.2882 | 0.363 | 0.6512 |

Each implementation version can be seen compared against each other implementation and the sequential version.

The GIL is what hindered our progress the most as we were unable to realize a truly data parallel implementation because of it. Our research done on python’s GIL feature shows that the GIL does not have much impact on the performance of I/O-bound multi-threaded programs as opposed to CPU-bound threads. The reason being that the lock is shared between threads while they are waiting for I/O. However, a program whose threads are entirely CPU-bound, e.g., a program that processes an image in parts using threads, would not only become single-threaded due to the lock, but will also see an increase in execution time due to the acquire and release overheads added by the lock. Our second hindrance would be the missing implementation of OpenCL, or PyOpenCL which is an OpenCL wrapper for Python. OpenCL would have been used to extend our implementation to be executed directly on a GPU as the paper did. This wasn’t our main objective however because the core of work was centered around a speedup due to parallelization regardless of the specific computing unit used.

Ultimately, the timeframe was our true enemy. We have a significant amount of coding work to show for our attempt at a successful implementation. We hope that any future iteration of the project will finally yield full parallelism. **We will expand on all the topics discussed in this report in further detail during our presentation.** This concludes our final report.