GPU상에서의 Pilsung 블록 암호 구현

엄시우*, 김현준*, 권혁동*, 서화정**

*한성대학교 (대학원생)
**한성대학교 (교수)

Implementation of Pilsung Block Cihper on GPU

Si-Woo Eum*, Hyun-Jun Kim*, Hyeok-Dong Kwon*, Hwa-Jeong Seo**

*Hansung University(Graduate student)
**Hansung University(Professor)

요 약

북한에서 개발한 Pilsung 블록 암호는 AES 블록 암호를 기반으로 한 SPN 구조의 블록 암호이다. 본 논문에서는 GPU 상에서의 Pilsung 블록 암호 구현을 진행한다. Host에서 메모리를 할당할 때, Pinned 메모리를 바로 할당하는 함수를 활용하여 Host에서 Device로 데이터를 복사하는 비용을 최소화하였다. 또한 Pilsung 블록 암호에서의 Sbox, Pbox 테이블 확장 연산을 Device상에서 구현함으로써 최적화하였다. 해당 기법을 적용함으로써 1.33배의 성능 향상을 확인하였으며, GPU 구현에서 Host와 Device 사이의 데이터 복사 비용이 성능에 큰 영향을 미치는 것을 확인한다.

I. 서론

Pilsung 블록 암호는 북한에서 개발한 운영체제인 붉은별 3.0 OS에서 볼 수 있는 블록 암호이다[1]. AES[2]를 기반으로 한 블록 암호이며, 암호화 과정보다 암호화에 필요한 테이블을 연산 과정이 더 오래 걸리는 암호이다. 즉, 키 변경에 따른 비용이 매우 크다. 따라서 키 변경 빈도가 많은 환경에서 사용하기에는 비효율적인 암호이다.

본 논문에서는 GPU 상에서의 Pilsung 블록 암호 구현을 진행한다. GPU는 대량 병렬 연산이 뛰어난 프로세서이다. 따라서 매우 큰 데이터의 병렬 암호화를 통해 빠른 시간안에 암호화하는 것이 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 Pilsung 블록 암호와 GPU에 대해 설명한다. 3장에서는 GPU 상에서의 최적화 구현을 위한 기법을 설명한다. 4장에서는 구현 기법을 적용한 구현의 성능 평가를 진행한다. 마지막으로 5장에서는 결론을 내린다.

II. 관련 연구

2.1 Pilsung 블록 암호

Pilsung 블록 암호는 AES 블록 암호에 기반을 둔 SPN 구조의 알고리즘이다. 블록 길이는 128-bit 이며, 키 길이는 256-bit이다. AES 블록 암호에 기반을 둔 암호이기 때문에 라운드 함수(SubByte, ShiftRows, MixColumns, Addroundkey)가 동일하다. 하지만 SubByte와 ShiftRows 함수, 그리고 키 확장 알고리즘에서 AES 블록 암호와의 차이가 있다[3].

Pilsung의 키 확장 알고리즘에서는 SHA-1 해시함수[4]가 사용된다. 입력된 256-bit 키는 SHA-1 해시함수를 통해 160-bit의 해시 값이 생성되고, 생성된 160-bit의 해시 값을 AES의 키 확장 알고리즘을 활용하여 11개의 128-bit

라운드키를 생성한다.

SubByte에서는 Sbox 테이블을 활용한 치환 연산을 한다. AES에서는 하나의 테이블을 활용하여 연산이 진행되는데, Pilsung에서는 Byte 단위로, 라운드마다 서로 다른 Sbox 테이블이 사용된다. 즉, 160개(16-byte*10라운드)의 테이블이 사용된다. 160개의 테이블은 생성된 라운드 키이 사용하여 Rao-Sandelius 셔플을 활용한 임의의 비트 순열이 기존의 Sbox 테이블에 연산되어 생성된다. 또한 이렇게 생성된 테이블을 통해 치환된 값은 마지막에 3과 XOR 연산이 진행된다.

ShiftRow에서는 각 라운드마다 서로 다른 순열 연산을 진행한다. 서로 다른 순열은 Sbox 생성과 마찬가지로 라운드키를 활용하여 임의의 바이트 순열을 생성하여 연산을 진행한다.

2.2 Graphics Processing Unit(GPU)

GPU는 대량 병렬 연산이 가능하여, 이미지 및 그래픽 부분에서 널리 사용되어 왔다. 하지만 최근에는 머신 러닝, 암호화 연산 가속화 등 여러 응용 프로그램에도 널리 사용되고 있다[5, 6].

GPU의 대표적인 제조사 NVIDIA는 2006년 CUDA를 발표했다. CUDA는 GPGPU(범용 컴퓨팅 기반 그래픽 처리 장치) 기술의 사용을 가능하게 하는 병렬 컴퓨팅 플랫폼 및 API 모델이다[7]. CUDA는 다양한 프로그래밍 언어로 사용 가능하며, 효율적인 GPU 구현을 위한 다양한 함수를 제공하고 있다.

III. 제안 기법

본 논문에서는 Pinned Memory 사용을 통한 최적화와 Sbox, Pbox(ShiftRows에서의 순열 테이블) 확장 연산을 GPU에서 병렬로 구현한 최적화 구현을 진행한다.

3.1 Pinned Memory 사용

GPU 구현에서 성능에 영향을 주는 부분 중 하나로 Host(CPU)와 Device(GPU)간의 데이터 복사가 있다. GPU 구현의 경우 GPU 커널 함수를 동작하기 전 Host에서 Device로 데이터를 복사하고 해당 데이터를 활용한 연산 결과값을 다시 Device에서 Host로 복사하는 과정으로 동작한다. 단순한 복사 과정이지만 성능에 작지 않은 영향을 미친다.

기본적으로 Malloc() 함수를 통해 Host에서 메모리를 할당해주면 Pageable 메모리이다. Pageable 메모리는 GPU 메모리로 바로 복사가 불가능하다. Pageable 메모리를 GPU 메모리로 복사하기 위해서는 GPU driver가 Pinned 메모리를 할당해야 한다. 이후에 Pageable 메모리에서 Pinned 메모리로 데이터가 먼저 복사된 후에 Pinned 메모리에서 GPU 메모리로 복사된다.

이때 Pageable 메모리에서 Pinned 메모리로 데이터를 복사하는 과정을 생략하기 위해 Host에서 메모리를 할당할 때, Pinned 메모리로 바로 할당하는 방법을 사용한다. 이는 CUDA에서 지원하는 cudaMallocHost()나 cudaHostAlloc() 함수를 통해 Pinned 메모리 할당이 가능하다[8]. 해당 함수를 사용함으로써 Host에서는 바로 Pinned 메모리 할당이 가능하며, Pageable 메모리에서 Pinned 메모리로 복사하는 과정을 생략할 수 있다. 해당 과정은 [그림 1]과 같다.

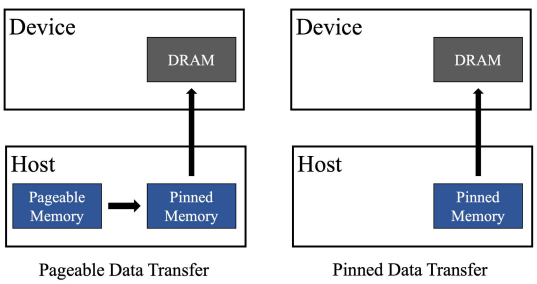

Fig. 1. Processes of Pageable and Pinned Data Transfer

3.2 테이블 확장 최적화

Pilsung에서의 SubByte 함수는 각 라운드, 각 바이트 마다 서로 다른 Sbox를 사용하기 때문에 라운드 함수가 연산되기 전에 테이블을 확장시켜주는 연산이 진행된다. 또한 마찬가지로

ShiftRows 함수에서도 고정된 순열이 아닌 라운드마다 서로 다른 임의의 순열(Pbox를 활용한) 연산이 진행된다.

이러한 Sbox, Pbox 테이블 확장 연산은 키 확장을 통해 생성된 라운드 키를 기반으로 테이블이 확장되고 확장된 테이블을 활용하여 암호화가 진행된다. 이때 Sbox의 경우 각 라운드, 각 Byte마다 다른 테이블이 사용되기 때문에 160번의 반복을 통한 확장이 이루어지며 이로 인한 연산 비용이 매우 크다.

본 논문에서는 최적화를 위해 키 확장은 Host에서 진행되고, 테이블 확장부터 Device(GPU)에서 진행한다. 테이블 확장의 경우 라운드에 따라서 병렬 연산이 가능하기 때문에 GPU의 Thread를 나누어 병렬 연산으로 Host에서 구현 시 160번 반복을 통한 확장이 이루어지는 것 대신, 10개의 Thread를 사용하여 병렬 연산을 통해 16번 반복만으로 확장이 가능하다.

병렬 연산으로 인한 성능 향상뿐만 아니라, Host에서 Device로 복사해야 하는 데이터의 크기가 줄어들기 때문에 추가적인 성능 향상을 얻을 수 있다. 테이블 확장을 Host에서 진행하게 되면 테이블이 확장된 만큼 복사 비용이 증가하게 되지만, Device에서 확장을 진행할 경우 테이블이 Device 내부에서 정의되고 확장되기 때문에 복사 과정의 생략이 가능하다. 전체적인 동작 과정은 [그림 2]와 같다.

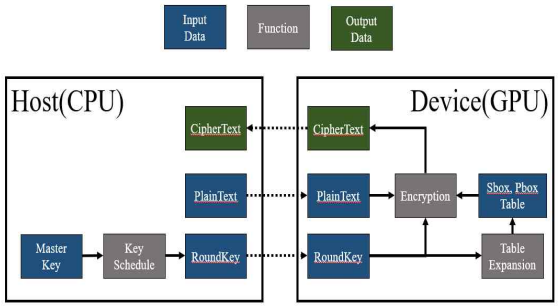

Fig. 2. Entire encryption process including data flow

IV. 성능 평가

성능 평가를 위해 Nivida GTX 3060 Laptop 상에서 구현 및 성능 측정을 진행하였다. Visual Studio상에서 구현을 진행하였으며, Cuda 11.7 runtime 버전을 사용하였다. 프로젝트 빌드 시 Release 방식으로 빌드하여 동작한 성능을 측정하였다. Gird 수는 32768(1024*32)로 고정하고, Thread 수를 늘려가며 측정하였다. 성능 측정 범위는 메모리 복사(암호화 값의 복사는 미포함)를 포함한 키 확장 함수부터 암호화 함수까지의 동작 시간(ms)을 측정하여 비교를 진행한다. 평문 데이터의 크기는 Grid수*Thread수*블록길이(16byte)의 크기가 사용된다. 즉 Thread 수가 32 일 때의 평문 데이터의 크기는 16MB(1024*32*32*16)이다.

구현은 Pinned 메모리 사용과 테이블 확장 최적화 구현의 적용 여부에 따라서 4개의 구현으로 나누어 측정을 진행한다. 측정결과는 다음 <표 1>과 같다.

Impl.	Thread					
	32	64	128	256	512	1024
None	10	14.9	24.1	40.7	75.3	145.2
O	9.6	12.7	20.3	35.9	69.3	142.1
P	8.9	12.7	19.5	32.1	58.4	112.7
PO	8.4	10.7	15.6	27.3	51.8	108.5

Table 1. Performance result(GridSize: 1024*32, P: Using Pinned mermory, O: Optimized Table Expansion, unit: ms)

Thread 수가 1024일 때, 측정 결과 기법을 적용하기 전과 두 기법을 모두 적용하였을 때의 성능 차이는 약 1.33배의 성능 차이를 보여준다.

마찬가지로 Thread 수가 1024일 때, 테이블 확장 최적화 구현의 성능 향상은 약 1.02배로 다소 낮은 성능 향상 폭을 보여주지만, Pinned 메모리를 사용한 구현의 성능 향상은 약 1.29배로 매우 큰 성능 향상을 보여준다. 이를 통해 GPU 구현에서 데이터의 복사가 성능에 미치는 영향이 큰 것을 알 수 있다.

V. 결론

본 논문에서는 Pinned 메모리의 사용과 테이블 확장을 Device(GPU)에서 구현함으로써 GPU상에서의 Pilsung 블록 암호 구현을 진행하였다. Pinned 메모리의 사용만으로도 큰 성능 향상을 확인할 수 있었으며, 이를 통해 GPU 구현에서 데이터의 복사가 성능에 미치는 영향을 알 수 있다. 해당 기법은 해당 블록 암호에만 국한된 기법이 아니기 때문에 다양한 블록 암호 GPU 구현에 적용 가능하다. 추후 연구로는 다양한 블록 암호 운영 모드의 GPU 최적화 구현을 제안한다.

VI. Acknowledgment

이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2021-0-00540, GPU/ASIC 기반 암호알고리즘 고속화 설계 및 구현 기술개발, 100%).

[참고문헌]

[1] Kryptos Logic. A Brief Look At North Korean Cryptography. https://www.kryptoslogic.com/blog/2018/07/a-brief-look-at-north-korean-cryptography/. July 2018.

[2] Daemen, Joan, and Vincent Rijmen. "AES proposal: Rijndael." (1999).

[3] Kryptos Logic. pilsung.c. https://blog.kryptoslogic.com/assets/pyongyang/pilsung.c. July 2018.

[4] Burrows, James H. Secure hash standard. Department of Commerce Washington DC, 1995.

[5] An, SangWoo, et al. "Parallel implementations of ARX-based block ciphers on graphic processing units." Mathematics 8.11 (2020): 1894.

[6] Owens, John D., et al. "GPU computing." Proceedings of the IEEE 96.5 (2008): 879-899.

[7] NVIDIA. CUDA Toolkit-Develop, Optimize and Deploy GPU-Accelerated Apps. Available online: https://docs.nvidia.com/cuda/ (accessed on 21 August 2020)

[8] Negrut, Dan, et al. "Unified memory in cuda 6.0. a brief overview of related data access and transfer issues." SBEL, Madison, WI, USA, Tech. Rep. TR-2014-09 (2014).