GPU를 활용한 동형암호 구현 동향

엄시우[1], 김현준[2], 임세진[1], 서화정[1]
[1]한성대학교 IT융합공학과
[2]한성대학교 정보컴퓨터공학과
shuraatum@gmail.com, khj930704@gmail.com,
dlatpwls834@gmail.com, hwajeong84@gmail.com

Trends in Implementation of Homomorphic Encryption using GPU

Si-Woo Eum[1], Hyun-Jun Kim[2], Se-Jin Lim[1], Hwa-Jeong Seo[1]
[1]Dept. of IT Convergence Engineering, Han-Sung University
[2]Dept. of Information Computer Engineering, Han-sung University

요 약

빅데이터, 인공지능, 클라우드 등의 기술이 발전함에 따라서 개인 정보나 중요 데이터가 많이 노출되고 있다. 동형암호는 암호화된 데이터에 대해서 직접 연산이 가능한 암호체계이다. 이러한 특성은 오늘날 클라우드 컴퓨팅 플랫폼에 매우 중요한 기술이지만, 많은 연산으로 인해 처리 시간이 오래 걸려 많이 사용되어 오고 있지 않다. GPU는 병렬 연산의 특성을 활용하여 CPU가 담당하는 작업을 훨씬 효율적으로 작업하는 것이 가능하다. 본 논문에서는 GPU를 활용하여 동형 암호의 속도 향상을 위한 기법 연구 동향에 대해 알아본다.

1. 서론

빅데이터, 인공지능, 클라우드 등의 기술이 발전함에 따라서 인터넷상에서 중요한 데이터나 개인정보가 많이 노출되고 있다. 특히 인공지능의 발전으로 인해 클라우드 상에서의 학습을 위해 민감한 데이터가 신뢰할 수 없는 원격 호스팅 서버와 같은 곳으로 전송된다. 학습을 위해서는 암호화된 데이터를 복호화를 하고 학습을 진행하게 되는데, 이때 데이터가 노출될 수 있는 문제가 있다[1].

동형암호는 암호화된 데이터에 대해 직접 임의의 연산을 할 수 있는 암호체계이다. 이러한 특성은 오늘날 클라우드 컴퓨팅 플랫폼에 매우 중요하다. 하지만 동형암호는 효율성이 매우 떨어지기 때문에 많이 사용되고 있지 않다.

본 논문에서는 동형암호의 효율성 개선을 위해 GPU를 활용 구현 동향에 대해 알아본다.

2. 관련 연구

2.1 동형 암호

일반적으로 암호화된 값의 연산을 위해서는 암호화된 값을 복호화 한 후 복호화된 메시지간의 연산을 진행하고, 다시 암호화를 하는 식으로 동작하게 된다.

동형암호는 암호화된 메시지들을 복호화 하지 않고 더하기나 곱하기와 같은 연산을 하였을 때, 그 결과 값을 복호화한 값이 복호화를 하고 연산하였을 때의 연산 값이 나오는 암호체계이다. 즉, 평문 a와 평문 b의 암호화 값이 E(a), E(b)라고 하였을 때, E(a)+E(b) = E(a+b)라고 볼 수 있으며, 이는 덧셈 동형성질이라고 한다.

동형암호는 크게 세 개의 종류가 있다. 첫 번째로 더하기와 곱하기 연산 중 하나의 연산만 가능한 동형암호 체계를 부분동형암호(Partial Homomorphic Encryption, PHE)라고 한다.

두 번째는 준동형암호(Somewhat Homomorphic Encryption, SWHE)이다. 두 연산이 모두 가능한 동형암호의 경우 여러 공격으로 해독하거나 키가 노출되는 문제가 있었다. 이 문제는 Gentry가 2009년 암호문에 에러를 주입함으로써 기존의 공격으로부터 안전한 동형암호 체계를 만들었다. 하지만 많은 연산이 반복될 경우 에러가 점점 커지는 문제로 인하여 복호화 후 기존의 메시지가 복원되지 않는 문제가 발생한다. 이러한 문제가 발생하지 않도록 연산의 횟수를 제한하였으며, 이러한 동형암호 체계를 준동형암호라고 한다[2].

마지막 세 번째는 완전동형암호(Fully Homomorphic Encryption, FHE)이다. 기존의 연산이 반복될 수 록 에러가 커지는 문제로 인해 연산 횟수가 제한되었는데, 커진 에러를 줄여줄 수 있는 기법인 부트스트래핑이 연구되었다. 부트스트래핑은 반복된 연산으로 인해 커진 에러를 갱신하여 연산을 제한 없이 할 수 있는 알고리즘이다. 이 알고리즘을 적용하여 두 연산을 제한 없이 사용 가능한 동형암호 체계를 완전동형암호라고 한다.

PHE는 하나의 연산만 가능한 단점이 있지만, 연산 횟수의 제한이나 에러를 갱신하는 부트스트래핑과 같은 과정이 없기 때문에 연산량이 적은 장점이 있다. SWHE와 FHW는 두 연산이 모두 가능한 장점이 있지만, 연산 횟수의 제한이 있거나, 추가적인 연산 과정이 필요한 단점이 있다[3].

2.2 Graphic Processing Unit(GPU)

GPU는 그래픽 처리 장치의 약어로 그래픽 처리, 특히 3D 모델링을 위한 프로세서로 개발되었다. GPU의 주된 역할은 2D 및 3D 그래픽의 연산 및 생성이지만 CPU가 담당하던 작업을 GPU를 활용하면서 현대의 GPU는 그래픽 연산외의 다양하게 활용되고 있다. GPU의 자원을 이용해 그래픽 작업 이외의 범용 작업을 하는 것을 GPGPU(General Purpose Computing on Graphics Processing Units)라고 하며 암호 해독, 기상 변화 예측, 머신 러닝 등의 분야에서 사용되고 있다[4, 5].

GPGPU는 GPU의 대표적인 제조사 Nvidia에서 2006년 CUDA의 발표로 더욱 활발해졌다. CUDA는 GPGPU기술의 사용을 가능하게 하는 병렬 컴퓨팅 플랫폼 및 API 모델이다. CUDA는 C, C++등 여러 프로그래밍 언어에서 사용할 수 있으며, 효율적인 GPU 자원을 사용하기 위한 다양한 기능을 제공한다[6].

3. GPU를 활용한 동형암호 연구

3.1 Accelerating FHE Using GPU[7]

2012년 Wang et al.은 Gentry-Halevi FHE(Fully Homomorphic Encryption)을 처음으로 GPU를 활용하여 구현하였다. FHE는 크게 4개의 KeyGen, Encrypt, Decrypt, Recrypt 함수로 구성되는데, 이 중에서 KeyGen을 제외한 나머지 함수에 대해서 최적화 구현을 진행하였다.

재귀 호출을 지원하지 않는 GPU에서 다항식 계산을 위해 직접적인 접근 방식을 사용하고, 슬라이딩 윈도우 기법을 적용하여 다항식 계산을 하였다. 또한 사전 연산을 통해 더 빠른 연산을 가능하게 하였다. 하지만 윈도우 크기를 크게 설정할 경우 연산의 수가 줄어들어 빠른 연산이 가능하지만 사전 연산된 값을 저장하는 공간이 많이 필요하다. 따라서 적절한 윈도우 크기로 설정하여야 한다.

또한 Strassen의 FFT 기반 정수 곱셈 알고리즘과 Barrett의 Modular reduction 알고리즘을 결합하여 매우 큰 비트 크기의 연산을 지원하는 효율적인 Modular 곱셈기를 구현하여 최적화 구현을 하였다.

결과적으로 CPU 구현과 비교할 때, Encrypt의 경우 8배, Recrypt의 경우 7.6배의 속도 향상을 하였다.

3.2 Accelerating NTT for Bootsrappable HE on GPU[8]

2020년 Kim et al.은 NTT와 DFT의 알고리즘적 특성을 분석하고 최신 GPU에서 DFT와 NTT 모두에 일반적으로 적용할 수 있는 최적화 기법을 NTT 구현에 적용하였을 때의 성능을 평가하였다. FFT 기반의 최적화 및 알고리즘을 NTT 구현에 적용하였을 때, Trade-off 및 성능 문제에 대해 연구를 진행하였다. 특히, GPU에서 지원하는 공유메모리와 같은 GPU의 하드웨어 기능과 다양한 Radix 값을 사용한 분석을 통해 DFT와 NTT 구현에서 최적의 Radix 값에 대한 연구를 진행하였다.

이러한 분석을 통해 NTT의 알고리즘적 특성에 의한 메인메모리 대역폭 병목 현상을 식별하였다. 메인메모리 대역폭 문제의 해결과 Modular 곱셈 비용을 줄이기 위해 On-the-fly Twiddling(OT)라고 하는 새로운 NTT 관련 On-the-fly 루트 생성 박식을 제안하였다. OT기법을 사용하여 Twiddle Factor의 일부를 즉석에서 계산하여, 메인메모리 접근을 최소화하였다.

결론적으로 기존 Radix-2 NTT 구현과 비교하여 OT 구현을 포함한 모든 최적화 구현을 적용한 경우 최신 GPU에서 4.2배의 속도 향상을 달성하였다.

3.3 Accelerating Polynomial Multiplication for HE on GPU[9]

2022년 Shivdikar et al.은 격자 기반 동형암호 시스템의 주요 계산인 다항식 곱셈의 GPU 기반 구현의 특성을 제시하고 최적화 구현을 하였다.

모듈러 축소는 격자 기반 암호화의 핵심 연산이면

서 병목 현상의 원인이다. 동형암호에 최적화된 Barrett 모듈러 축소 알고리즘[10]의 여러 변형을 분석하고 최적화된 모듈러 축소 변형을 제안하고 있다.

또한 NTT 커널이 GPU의 DRAM 대기 시간으로 인해 병목 현상이 심한 작업임을 분석하였다. 버터플라이 연산은 NTT 커널 내에서 중요한 계산 중 하나이다. 이 작업은 각 단계에 따라 보폭이 달라지는 스트라이드 접근이 특징이다. 보폭의 변경은 비순차적 메모리 접근으로 이어져 NTT 커널의 공간적 지역성을 줄인다. 메모리 병합을 효과적으로 활용하기 위해 CUDA 스레드에서 데이터를 분할 할 수 있다. 해당 논문에서는 각기 다른 입력 크기에 최적화되고, 다른 데이터 분할 기술을 사용하는 세 가지의 NTT 커널 구현을 제안하고 있다.

마지막으로 속도 향상을 위해 중복 메모리 접근을 줄이는 공유 메모리 최적화를 진행하고, karatsuba 알고리즘을 적용하여 Hadamard 곱셈과 인접한 버터플라이 연산을 융합하는 융합 다항식 곱셈 알고리즘을 제안하였다. 이를 통해 Twiddle factor의 메모리 사용량을 50% 감소시켰다.

결론적으로 NTT 최적화를 통하여 기존 최신 CPU, GPU 구현보다 각각 약 123배, 약 2.3배의 속도 향상을 보여준다.

4. 결론

클라우드의 발전은 동형암호의 필요성을 점점 증가시키고 있다. 하지만 아직 동형암호의 효율성이 떨어지는 문제로 많이 사용되고 있지 않다. 본 논문에서는 GPU를 활용한 동형 암호 최적화 구현 연구에 대해서 알아보았다. 최근 연구에서는 GPU의 공유메모리를 활용하며, 다항식 곱셈의 최적화를 위해 모듈러 축소 알고리즘에 대한 분석과 NTT의 최적화 연구가 진행되었다. 인공지능과 클라우드의 발전에 맞춰서 꾸준한 동형암호에 대한 연구가 필요하다고 생각된다.

5. Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00540, Development of Fast Design and Implementation of Cryptographic Algorithms based on GPU/ASIC).

참고문헌

[1] 조은지, 문수빈 and 이윤호. "완전 동형 암호 라이브러리의 성능 분석" 한국정보기술학회논문지 16, no.2. 2018.

[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," Proc. of the forty-first annual ACM symposium on Theory of pp. 169-178, May 2009.

[3] 이승범, 이주원 and 심형보. "사이버 물리 시스템의 보안을 위한 동형암호 소개". 제어.로봇.시스템학회 논문지 27, no.3, 177-184. 2021.

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

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

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

[7] Wang, Wei, et al. "Accelerating fully homomorphic encryption using GPU." 2012 IEEE conference on high performance extreme computing. IEEE. pp. 1-5. 2012.

[8] Kim, Sangpyo, et al. "Accelerating number theoretic transformations for bootstrappable homomorphic encryption on gpus." 2020 IEEE International Symposium on Workload Characterization (IISWC). IEEE, p. 264-275. 2020.

[9] Shivdikar, Kaustubh, et al. "Accelerating Polynomial Multiplication for Homomorphic Encryption on GPUs." arXiv preprint arXiv:2209.01290 (2022).

[10] P. Barrett, "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor," in Advances in Cryptology - CRYPTO' 86, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 311-323.