TFHE 가속화를 위한 CUDA 최적화 기법 비교 연구

하승진[1], 남기빈[1], 백윤흥[1]
[1]서울대학교 전기·정보공학부, 서울대학교 반도체 공동연구소

sjha@sor.snu.ac.kr, kvnam@sor.snu.ac.kr, ypaek@snu.ac.kr

A Comparative Study of CUDA Optimisation Techniques for Accelerating TFHE

Seungjin Ha[1], Kevin Nam[1], Yunheung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center(ISRC), Seoul National University

요 약

본 논문에서는 TFHE(Fast Homomorphic Encryption over the Torus)의 가속화를 위한 다양한 CUDA 기반 라이브러리인 cuFHE, nuFHE, (RED)cuFHE의 성능을 비교 분석하였다. 각 라이브러리의 최적화 기법과 GPU 자원 활용 방식이 동형암호 연산의 처리 속도에 어떻게 영향을 미치는지 실험을 통해 확인하였으며, 이를 통해 상황에 맞는 라이브러리 선택의 중요성을 강조하였다. 특히, 본 연구는 각 라이브러리의 특징과 차이점을 상세히 분석함으로써, 효율적이고 실용적인 TFHE 가속화 방법에 대한 가이드라인을 제공한다. 이러한 연구 결과는 동형암호 기술의 실용성을 높이고, 고속화된 암호 연산이 필요한 다양한 분야에서 활용될 수 있는 기반을 마련할 것으로 기대된다.

1. 서론

동형암호(Homomorphic Encryption, HE)는 암호화된 데이터를 복호화하지 않고도 연산할 수 있는 기술로, 개인정보 보호와 데이터 보안이 중요한 상황에서 강력한 도구로 활용된다. 특히 클라우드 컴퓨팅, 금융, 의료 데이터 처리와 같은 외부 위탁 연산 환경에서 유용하며, 암호화된 상태로도 연산 결과의 정확성을 보장할 수 있다.

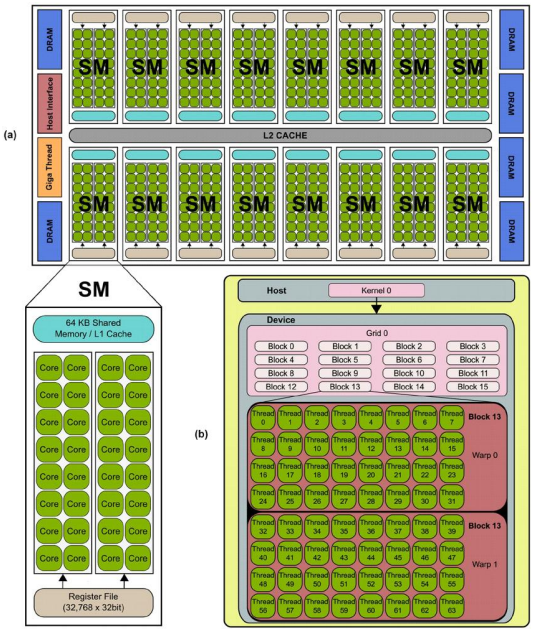
동형암호 스킴에는 CKKS[1], BFV[2], BGV[3] 등이 있으며, CKKS는 실수 연산에서 packing을 지원해 성능이 뛰어나지만, 조건부 연산 같은 비산술 연산은 근사 처리가 필요해 정확도가 떨어진다[4]. 반면, TFHE[5]는 부울 연산을 기반으로 다양한 논리 연산을 지원하며, 비트 단위의 조건부 연산 등 복잡한 연산을 빠르게 처리할 수 있어 유연한 응용이 가능하다. 이러한 특성 덕분에 TFHE는 클라우드 데이터 처리나 인공지능 연산에 적합한 기술로 평가받고 있다.

그러나 TFHE는 각 게이트 연산 후마다 부트스트래핑을 거쳐야 하기 때문에 시간이 오래 걸린다. 이를 개선하기 위해 packing이나 circuit bootstrapping 같은 알고리즘적 연구가 진행 중이다[6][7][8].

GPU 가속을 통해 TFHE 성능을 향상시키는 연구도 활발히 진행되고 있으며, nuFHE[9], cuFHE[10], (RED)cuFHE[11]와 같은 CUDA 기반 라이브러리들이 등장했다. 하지만 각 라이브러리마다 GPU 자원 활용 방식이나 FFT, NTT 같은 기술 사용 방식에 차이가 있어, 상황에 맞는 최적화 기법을 선택하는 것이 중요하다. 본 연구에서는 이러한 다양한 최적화 기법들을 정리하고자 한다.

2. GPU 아키텍처와 CUDA를 활용한 병렬 연산

<그림 1>은 NVIDIA GPU 아키텍처를 나타낸 것이다. 이 구조에서 가장 중요한 요소는 다수의 Streaming Multiprocessors(SM)로, 각 SM은 수십 개의 CUDA 코어로 구성되어 병렬 연산을 수행한다. SM 내부에는 레지스터 파일과 공유 메모리, L1 캐시가 존재하여 데이터를 빠르게 처리할 수 있도록 돕는다. 모든 SM은 L2 캐시를 공유하며, 이는 전역 메모리(Global Memory)와의 데이터 교환 속도를 높인다.



<그림 1> NVIDIA GPU 아키텍처 [12]

GPU의 이러한 구조는 병렬 연산에 최적화되어 있으며, 각 SM은 독립적으로 다양한 스레드 블록을 처리할 수 있다. 특히, 워프(warp) 단위로 스레드가 처리되며, 각 워프는 32개의 스레드로 구성되어 병렬적으로 연산을 수행한다. 이를 통해 GPU는 대규모 병렬 처리를 효율적으로 수행할 수 있으며, 이후 CUDA 프로그래밍 모델을 통해 이러한 병렬 구조를 최적화하여 활용하게 된다.

CUDA 프로그래밍 모델은 GPU의 병렬 처리 능력을 쉽게 활용할 수 있게 해주며, 기본적으로 스레드, 블록, 그리드의 계층 구조로 구성된다. 각 스레드 블록은 하나의 SM에 할당되어 연산을 수행하며, 이를 통해 복잡한 병렬 연산이 가능해진다. GPU 자원을 어떻게 활용할지는 개발자의 선택에 따라 구현 방법이 달라질 수 있다. 예를 들어, SM마다 서로 다른 게이트 연산을 배분할 수도 있고, 하나의 게이트 연산에 여러 SM을 할당해 병렬 처리를 극대화할 수도 있다. 따라서 CUDA를 사용한 최적화는 문제의 특성과 개발자의 설계에 따라 큰 차이를 보일 수 있으며, 이를 효율적으로 설계하는 것이 중요하다.

3. cuFHE, nuFHE, (RED)cuFHE 라이브러리

TFHE를 CUDA로 가속화한 라이브러리에는 cuFHE, nuFHE, 그리고 (RED)cuFHE가 있다. 이들 라이브러리는 각각 다른 방식으로 GPU 자원을 활용하며, 특정 상황에 따라 적합한 시나리오가 다르다. 각 라이브러리가 연산 속도, 효율성, 최적화 방식에서 차이가 있기 때문에, 각기 다른 환경에서 최적화된 선택을 해야 한다.

cuFHE는 게이트 연산을 특정 SM(Streaming Multiprocessor)에 할당하여 연산할 수 있다. 이 방식은 개발자가 SM을 직접 할당함으로써, 특정 게이트 연산을 실행할 때 필요한 자원을 정확히 지정할 수 있어 성능을 최대한 끌어올릴 수 있다. 예를 들어, 중요한 연산을 여러 SM에 나누지 않고 한 SM에서 집중적으로 처리하게 하여 여러 SM 간의 데이터 교환이나 동기화 비용이 줄어들어 더 효율적인 연산이 가능하다. 또한, cuFHE는 NTT(Number Theoretic Transform) 기반으로 연산을 수행하는데, 이는 모듈로(modulo) 연산을 수반하기 때문에 FFT(Fast Fourier Transform)보다 속도가 느린 편이다. cuFHE는 각 게이트를 병렬적으로 처리할 수 있는 상황에서 유리하다. 또한, SM을 직접 지정함으로써 개별 게이트 연산의 성능이 중요한 경우에 적합하다. 하지만 연산할 게이트 수가 GPU의 SM 수보다 적을 경우 idle 상태인 SM이 발생할 수 있다.

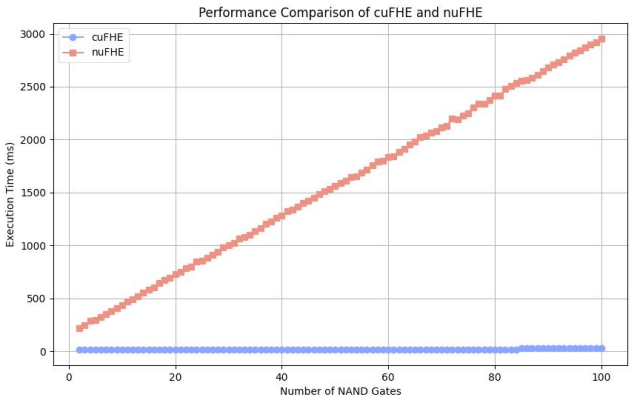
nuFHE는 FFT와 NTT를 모두 지원하며, 이를 상황에 맞게 사용할 수 있다. 그러나 pyCUDA를 사용하기 때문에 다른 C++ 라이브러리와의 링킹이 상대적으로 어렵다는 단점이 있다. 또한, nuFHE는 SM을 지정하여 게이트 연산을 할 수 없고, CUDA의 자동 스케줄링에 의존하여 스레드 블록이 SM에 할당된다. 이는 개발자가 직접 연산을 제어하지 않기 때문에 최적화가 어려울 수 있지만, 자동화된 환경에서는 개발 편의성을 제공한다. nuFHE는 연산할 데이터가 많고, 게이트 연산이 많은 상황에서 FFT와 NTT를 모두 지원하여 유연하게 선택할 수 있다는 장점이 있다. 또한, 파이썬 언어 특성상 pyCUDA를 사용한 환경에서 개발할 때 편리한 점이 많으며, 자동 스케줄링을 통해 GPU 자원을 최대한 활용할 수 있다. 하지만 SM을 지정할 수 없기 때문에 특정 게이트에 집중된 자원 활용은 어려울 수 있으며, 상대적으로 성능이 최적화되지 못하는 경우도 있을 수 있다.

(RED)cuFHE는 cuFHE를 기반으로 구현되었으며, NTT 기반의 연산을 사용하고, SM별로 게이트 연산을 지정할 수 있다. 이 라이브러리는 또한 TFHE의 연산을 이진 도메인(binary domain)과 정수 도메인(integer domain) 사이에서 상황에 맞게 전환하여 연산 효율을 극대화하는 알고리즘을 사용한다. 이진 도메인은 0과 1로 이루어진 부울 연산에 적합하고, 정수 도메인은 숫자를 더하거나 곱하는 산술 연산에

적합하다. (RED)cuFHE는 이러한 두 도메인을 상황에 맞게 전환하여, 각각의 연산에서 가장 빠른 방식을 선택한다.[13] 이는 마치 컴파일러가 연산의 특성에 따라 최적의 방법을 선택하는 것과 유사하다. 이러한 방식 덕분에 연산 성능이 크게 향상되며, cuFHE나 nuFHE보다 훨씬 빠른 속도를 자랑한다. (RED)cuFHE는 연산이 많은 대규모 작업이나, 높은 성능이 요구되는 상황에서 특히 유리하다. 특히, 이진 연산과 정수 연산을 전환하여 최적화된 연산을 수행하기 때문에, 다양한 종류의 연산이 섞여 있는 환경에서도 높은 성능을 기대할 수 있다.

4. cuFHE, nuFHE, (RED)cuFHE 라이브러리 비교

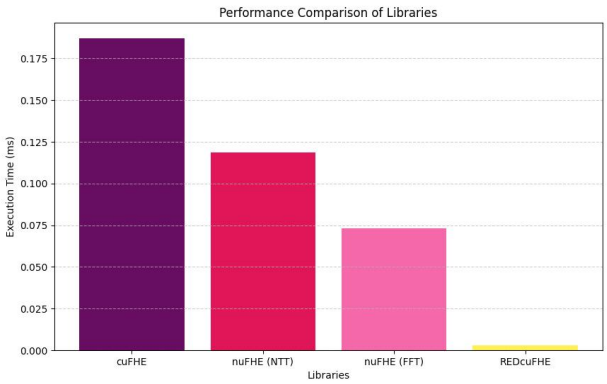
본 연구에서는 CPU로 Intel Xeon Gold 6326 2.9GHz 프로세서 2개를 사용하였으며, GPU로는 NVIDIA RTX A6000을 사용하였다. RTX A6000은 48GB의 GDDR6 메모리를 탑재하고 있으며, 84개의 Steaming Multiprocessor(SM)를 갖추고 있어 대규모 병렬 연산에 적합하다. 이러한 하드웨어 구성은 동형암호 라이브러리의 GPU 가속 성능을 측정하는 데 최적화되어 있으며, 실험을 통해 cuFHE, nuFHE, (RED)cuFHE의 연산 효율성을 비교하였다. 사용된 TFHE 파라미터는 $n = 500$, $N = 1024$이다.



<그림 2> 연산하는 게이트 개수에 따른 실행 시간

<그림 2>는 각 라이브러리별로 NAND 게이트의 개수가 증가함에 따라 실행 시간이 어떻게 변화하는지 관찰하였다. (RED)cuFHE는 cuFHE를 기반으로 구현되었기 때문에 cuFHE와 유사하게 동작한다. cuFHE의 경우, 각 SM이 하나의 게이트 연산을 담당하기 때문에, RTX A6000 GPU의 84개 SM이 모두 활용될 때까지는 실행 시간이 거의 일정하게 유지된다. 반면, nuFHE는 SM별로 연산이 할당되는 것이 아니라 스레드 단위로 자동 할당되기 때문에 게이트 수가 증가할수록 선형적으로 실행 시간이 늘어났다.

<그림 3>은 각 라이브러리별로 NAND 게이트 한 개를 연산하는 데 걸리는 시간을 비교하였다. nuFHE의 경우, NTT보다 FFT가 정수 modulus 연산을 피할 수 있어 더 빠르게 동작함을 확인할 수 있었다. 또한, REDcuFHE는 cuFHE에 추가적인 최적화가 적용되어 가장 빠른 속도를 보여주었다. nuFHE는 cuFHE보다 빠른 성능을 보이지만, pyCUDA로 구현되어 있어 다른 C 기반 라이브러리와의 연동에 한계가 있다. 반면, cuFHE는 C++로 구현되어 nuFHE와 비교했을 때 언어적으로 호환성 측면에서 차이가 있다. 이러한 결과는 두 라이브러리 간의 속도 차이가 존재하더라도 구현 언어와 사용 환경에 따라 선택이 달라질 수 있다는 점을 보여준다.



<그림 3> 게이트 한 개당 연산 시간

4. 결론

본 연구에서는 TFHE의 가속화를 위한 CUDA 기반 라이브러리인 cuFHE, nuFHE, (RED)cuFHE의 성능을 비교하고 분석하였다. 각 라이브러리는 고유한 최적화 기법과 구현 방식에 따라 성능이 달라지며, 이로 인해 동형암호 연산을 가속화하는 상황에서는 적합한 라이브러리를 선택하는 것이 중요하다. 예를 들어, cuFHE는 SM 할당 방식을 직접 관리할 수 있어 자원 활용의 효율성을 높일 수 있고, nuFHE는 FFT와 NTT를 모두 지원하지만 pyCUDA로 구현되어 C 라이브러리의 통합에 제한이 있다. 따라서 TFHE를 활용한 고속화 환경에서는 각 라이브러리의 특성과 적용 환경을 고려한 선택이 필요하다.

향후 연구에서는 packing이나 circuit bootstrapping 등의 TFHE 최적화 기법을 CUDA로 추가로 구현하거나, TFHE 알고리즘 자체를 GPU 아키텍처에 더욱 최적화하는 방안을 모색할 계획이

다. 이를 통해 GPU 기반 동형암호 연산의 효율성과 성능을 향상시킴으로써 대규모 데이터 처리에서의 동형암호 기술의 실용성을 극대화할 수 있을 것으로 기대한다.

5. ACKNOWLEDGEMENT

이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (RS-2023-00277326). 이 논문은 2024년도 BK21 FOUR 정보기술 미래인재 교육연구단에 의하여 지원되었음. 이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임. (IITP-2023-RS-2023-00256081) 본 연구는 반도체 공동연구소 지원의 결과물임을 밝힙니다. 이 논문은 2024년도 정부(산업통상자원부)의 재원으로 한국산업기술기획평가원의 지원을 받아 수행된 연구임.(No. RS-2024-00406121, 자동차보안취약점기반위협분석시스템개발(R&D)).

참고문헌

[1] Cheon, Jung Hee, et al. "Homomorphic encryption for arithmetic of approximate numbers." Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23. Springer International Publishing, 2017.

[2] Fan, Junfeng, and Frederik Vercauteren. "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive (2012).

[3] Brakerski, Zvika, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping." ACM Transactions on Computation Theory (TOCT) 6.3 (2014): 1-36.

[4] Gilad-Bachrach, Ran, et al. "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy." International conference on machine learning. PMLR, 2016.

[5] Chillotti, Ilaria, et al. "TFHE: fast fully homomorphic encryption over the torus." Journal of Cryptology 33.1 (2020): 34-91.

[6] Chillotti, Ilaria, et al. "Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE." International Conference on the Theory and Application of Cryptology and Information Security. Cham: Springer International Publishing, 2017.

[7] Chillotti, Ilaria, et al. "Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE." Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27. Springer International Publishing, 2021.

[8] Wang, Ruida, et al. "Circuit bootstrapping: faster and smaller." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Cham: Springer Nature Switzerland, 2024.

[9] https://github.com/nucypher/nufhe.git

[10] https://github.com/vernamlab/cuFHE.git

[11] https://github.com/TrustworthyComputing/REDcuFHE.git

[12] Hernández, Moisés, et al. "Accelerating fibre orientation estimation from diffusion weighted ma gnetic resonance imaging using GPUs." PloS one 8.4 (2013): e61892.

[13] Folkerts, Lars Wolfgang, Charles Gouert, and Nektarios Georgios Tsoutsos. "REDsec: Running E ncrypted Discretized Neural Networks in Seconds. “