정 보 보 호 학 회 지
제 33 권 제5호, 2023. 10

동형암호 가속 하드웨어 개발 기술 동향 분석

남 기 빈*, 정 헌 희*, 백 윤 흥*

요 약

동형암호는 암호화 상태에서도 연산이 가능하다는 특징을 지니고 있어 매우 유망한 프라이버시 보호 기술로 알려져 있다. 하지만 이런 장점에도 불구하고 아직 활발히 활용되지 못하고 있는데, 이는 매우 느린 연산 성능이라는 단점 때문이다. 이런 단점을 극복하기 위해서 동형암호 가속을 목표로 하는 많은 연구들이 수행되었으나, 아직 보편화 수준에 도달하지 못했으며, 특히 각 연구들이 독립적으로 산개되어 시너지효과를 누리지 못하고 있다는 특징을 보여주고 있다. 본 논문은 산업/학계의 다양한 동형암호 가속 연구들을 분류, 소개하며, 이들간의 관계를 분석하여 시너지 효과를 누릴 수 있는 가이드라인을 제시한다.

Ⅰ. 서 론

현재 클라우드 컴퓨팅이 대용량 데이터 처리의 효율성을 추구하며 널리 활용되고 있는 가운데, 개인 정보 보호에 대한 관심이 점점 커지고 있다. 일반적으로, 데이터는 암호화를 통해 보호될 수 있지만, AI 모델을 활용하는 등의 연산을 수행하기 위해선 해당 데이터의 암호를 해독해야 하고, 이러한 과정은 내부자 위협 등 다양한 위험에 데이터를 노출시킬 수 있다.

동형암호는 암호화 상태에서의 연산과 암호화 이전의 연산이 복호화될 때 동일한 결과를 나타낸다는 고유의 특성을 지닌 암호체계이다[1]. 사용자는 자신의 데이터를 암호화한 상태로 서버에 전송, 서버가 연산한 결과를 받아 복호화하여 결과값을 확인할 수 있으며, 연산 과정 중 데이터의 값은 노출되지 않기에 개인정보를 완벽하게 보호할 수 있다.

이런 장점을 지닌 동형암호지만 많은 제한점들을 지니고 있다. 특히, 동형암호는 데이터를 거대 다항식 형태로 암호화하는데, 이로 인한 데이터 용량과 연산 부하가 크게 발생하여 동형암호의 보편적 활용에 장애를 일으키고 있다. 이런 단점을 보완하기 위하여 다양한 분야의 산업/학계 많은 이들이 성능을 개선시키기 위해 노력하고 있으나, 아직 갈길이 먼 실정이다.

본 논문은 동형암호 가속 연구들의 동향을 분석, 분류 하는 구성으로 이루어져 있다. 특히, 본 논문은 수학적 접근과 HW의 특성을 활용한 접근 등 독립적으로 산개되어있는 연구결과들 중 시너지 효과를 일으켜 더욱 가속을 이룰 수 있는 방향으로 진행할 수 있도록 가이드라인을 제시하는 것을 목표로 한다.

Ⅱ. 배경 이론

본 장은 동형암호 가속 기술 연구들을 이해하는데 필요한 배경 이론을 소개한다. 세부적인 이론 내용은 주석의 참고논문들을 참고하기를 바란다.

2.1. 완전동형암호 체계

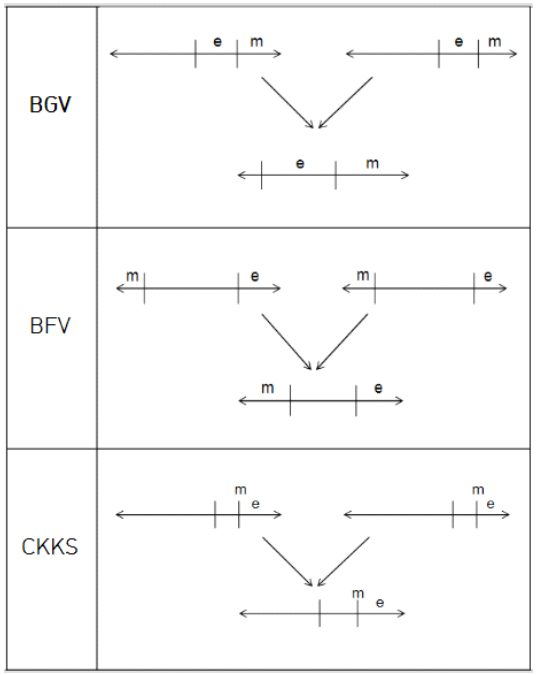
동형암호라는 개념은 1978년, RSA 암호가 개발된 지 1년 후에 처음 등장했다[2]. 암호화 상태에서 특정 연산을 수행할 수 있는 암호체계들에 붙는 개념이었으며, modular 곱셈을 지원하는 RSA[3]와 ElGamal[4], 그리고 modular 덧셈을 지원하는 Pailier[5] 등 많은 암호 체계들을 두고 동형암호라고

이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(RS-2023-00277326), 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No.RS-2023-00277060, 개방형 엣지 AI 반도체 설계 및 SW 플랫폼 기술개발), 2023년도 BK21 FOUR 정보기술 미래인재 교육연구단에 의하여 지원되었으며, IDEC에서 EDA Tool을 지원받아 수행한 결과입니다. 또한, 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.

* 서울대학교 전기정보공학부, 반도체공동연구소 (대학원생, kvnam@sor.snu.ac.kr. 대학원생, hhjung@sor.snu.ac.kr, 정교수, ypaek@snu.ac.kr)

부르곤 했다. 하지만 지원하는 연산의 종류, 연산의 정확도 등 그 개념이 정립되지 않은 채 30년이 흘렀으며, 2009년 처음으로 Craig Gentry에 의해 그 개념이 확립되었다.

Gentry의 정의에 따르면, 완전동형암호는 무한한 수의 덧셈과 곱셈을 수행하여도 그 연산 결과가 암호화 하지 않은 상태의 연산 결과와 동일한 것을 의미한다. 현재 널리 알려진 완전동형암호로는 BGV[6], BFV[7], CKKS[8], 그리고 TFHE[9]가 있다. 이들은 공통적으로 (Ring) Learning with Error Problem (LWE/RLWE)을 기반으로 하고 있으며, 다항식/벡터를 암호문 형태로 사용한다. 평문 메시지 m을 암호화하기 위해서, 다항식/벡터 형태의 암호키 s와 임의의 표본 a를 $ct = Enc_s(m) = (a, as + m + e)$과 같은 과정을 통해 암호화한다. 이 때 e는 랜덤 노이즈(noise)인데, 이로 인해 연립방정식을 통한 암호키의 추출이 어려워지지만, 복호화하여 원래의 값을 확인하는데는 문제가 발생하지 않는다. 노이즈의 주입 방법과 이에 따라 암호체계별로 세부 특징들이 달라지는데, 암호문 형태틀은 [그림 1]과 같으며, 각 체계의 원논문에 소개되어있으니 참고하기 바라며, 이후 '동형암호'라는 단어는 '완전동형암호'를 의미함을 명시하고자 한다.



〔그림 1〕 BGV, BFV, CKKS 체계의 암호문 형태〔10〕

〔표 1〕 동형암호 보안수준(λ)과 다항식/벡터의 차수(N)에 따른 권장된 항의 계수의 길이(q)

	다항식/벡터의 차수(N)			
λ	1024	2048	4096	8192
128	29	56	111	220
192	21	39	77	154
256	16	31	60	120

2.2. 완전동형암호의 파라미터 (Parameter)

앞서 언급했듯 동형암호는 거대 다항식/벡터를 암호문 형태로 사용한다. 이는 암호의 보안성과 관련이 있는데, 일반적으로 LWE/RLWE 문제는 다항식/벡터의 차수(N)가 높을수록, 각 항의 계수의 길이(q)가 작을수록 안전하다. 한편, q가 작은 경우, 암호화할 수 있는 데이터의 정밀도 (precision)이 낮아지기 때문에, 적정 수준의 q를 사용하면서 보안성을 유지하기 위해서는 높은 N을 사용할 수 밖에 없어, 결과적으로 암호문의 크기가 커지게 된다.

현재 완전동형암호의 보안성에 대해서 서술한 Homomorphic Encryption Standard[11]에 따르면, 현재 표준과 같이 활용하고 있는 $\lambda = 128$-bit 보안 수준을 사용하기 위한 (N, q) 조합으로 $(2^{10}, 64)$, $(2^{15}, 330)$과 같은 수치들을 사용할 수 있으며, 이들은 각각 8KB, 1.3MB의 용량을 사용한다. 특히, 8KB 암호문은 오로지 1-bit 데이터만을 암호화할 수 있으며, 일반 컴퓨터가 사용하는 데이터타입인 32-bit를 온전히 암호화하기 위해서는 1.3MB보다 더 큰 파라미터를 사용해야 한다. [표 1]은 동형암호 보안수준과 다항식/벡터의 차수에 따른 권장된 항의 계수의 길이를 나타낸 자료다.

2.3. 완전동형암호의 하위함수 (sub-functions)

완전동형암호의 암호문 연산은 다항식간 연산으로 이루어져 있다. 덧셈, 곱셈을 수행할 수 있는데, 이 때 noise가 누적되기 때문에, 일정 수준 연산을 진행하면 원래의 데이터 값을 온전히 복호화하지 못할 수 있다. 이런 문제를 방지하기 위해서, keyswitching,

relinearization, bootstrapping 등 다양한 하위함수(subfunction)들이 활용되며, 이들은 암호문 덧셈, 곱셈 연산 사이에 필수적으로 수행되어야 한다. 하위 함수들의 세부 정보는 각 체계의 원 논문들을 통해 확인할 수 있다. 이 하위함수들에 대해 중요한 점은 크게 두가지다. 첫째, 각 하위함수들은 매우 많은 수의 다항식 연산들로 이루어져있다. 예를들어, 표준 파라미터를 활용한 TFHE의 bootstrapping은 blind rotation이라는 다른 하위함수를 630번 반복 수행해야 하며, 각 blind rotation은 최소 12번의 암호문 곱셈을 필요로 한다.

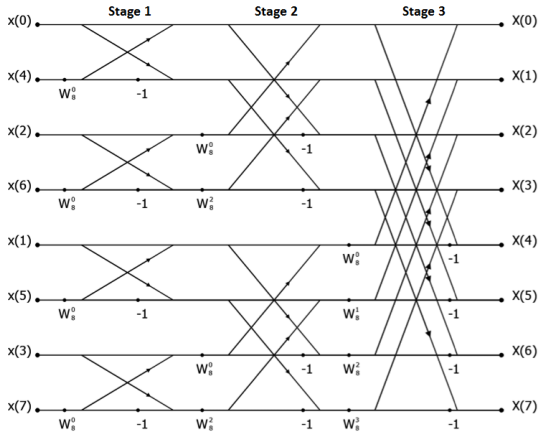
두 번째 이슈는 같은 연산을 수행하는데 있어 하위함수들을 나열하는 방법에 따라 연산의 효율이 크게 바뀐다는 점이다. 동형암호문에 담긴 메시지 m, 그리고 주입된 노이즈 e는 곱셈을 통해 그 크기가 배가 되는데, 두 번의 곱셈이 진행되면 그 bit-width가 4배 커지는 등, 기하급수적으로 증가하여, 금방 복호화 불가능 수준에 도달하게 된다. 이를 방지 하기 위하여, relinearization과 rescale을 통해 그 길이를 일정 수준으로 유지할 수 있는데, 추가 하위함수 연산 부하가 발생하지만, 복호화불가 수준에 도달할때까지 수행할 수 있는 연산의 수를 증가시켜주는 효과를 지니고 있다. 이런 교환(trade-off)가 중요한 이유는, 복호화 불가능 수준 이전에 필히 재부팅(bootstrapping)이라는 하위함수를 수행해야 하는데, 이 재부팅이 다른 하위함수들에 비해 비교 불가능한 수준으로 많은 연산 부하를 일으키기 때문이다. 즉, 동형암호 연산에 있어 복호화 불가능 수준에 도달하기 전에 최대한 많은 수의 연산을 수행하는 것이 전체적인 성능에 있어 유리하기 때문에, 다른 하위함수들을 적재적소에 수행하는 것이 중요해지는 것이다.

Ⅲ. 동형암호 가속 기술 동향

본 장은 기존 동형암호 가속 연구들의 접근법과 결과를 수학/알고리즘 중심 접근법, 컴파일러 중심 접근법, 그리고 HW 가속기 구현 연구로 분류하여 소개한다.

3.1. 수학/알고리즘 중심 접근법

앞서 서술했듯, 동형암호는 암호문 간 덧셈, 곱셈 뿐 아니라 필수적으로 여러 종류의 하위함수들을 수행해야 하며, 이들은 모두 다수의 다항식 연산들로 이루어진 일종의 알고리즘이라 할 수 있다. 이들을 최적화하기 위하여 다수의 연구들은 알고리즘의 time complexity를 낮추기 위한 연구들이 수행되어 왔다.



〔그림 2〕 Radix-2 Cooley Tukey 알고리즘을 이용한 FFT 연산 과정[12]

가장 대표적이며 고전적인 최적화의 예시로는, 다항식 곱셈을 최적화하기 위하여 Fast Fourier Transform(FFT)를 활용하는 것이다[12]. FFT는 time domain의 데이터를 frequency domain으로 변환시키는 (또는 그 반대) 연산이다. 다항식 곱셈은 한 다항식의 모든 항들이 다른 다항식의 모든 항들과 연산해야 하는 $O(n^2)$을 갖는 한편, FFT를 활용한 후, coefficient-wise 연산을 수행하는 방법으로 연산한다면 $O(n\log(n))$으로 낮출 수 있다. 이 방법은 대부분의 동형암호 가속 연구들이 기본적으로 차용하는 기본 최적화 기법으로, 다항식을 다루는 완전동형암호의 전반적인 연산 복잡도를 낮추어주는 역할을 한다고 할 수 있다.

한편, 동형암호 암호문을 구성하는 데이터들은 대부분 modular 연산을 수행하는 정수들로 구성되어 있다. 따라서, double floating point 데이터로 연산되는 FFT를 수행하기 위해서는, 데이터타입의 변환(casting)이 필요한데, 이는 이후 소개할 HW 가속기들 구현에 있어 다른 종류의 연산기를 필요로 한다는 점에서 비효율적이다. 이에 동일한 modular 연산을 수행하는 형태로 FFT를 수행할 수 있는 기법으로 Number Theoretic Transform(NTT)를 활용하여 동형암호 다항식 곱셈을 수행할 수도 있다.

이런 FFT와 NTT는 주로 Cooley-Tukey 알고리즘

을 통한 divide and conquer 방식으로 수행되곤 한다. 이 방식은 [그림 2]와 같이 FFT를 여러 단계(stage)로 나누어 수행하는데, 이 때 두 항을 더하고 빼는 연산하는데 사용되는 단위 연산을 butterfly operation 이라고 부른다.

알고리즘 수준에서의 최적화를 위한 연구들도 다수 존재한다. 이들은 하위함수의 구조 자체를 바꾸어 버리는 경우들이 많다. 예를 들어, key decomposition을 위해 기존에 사용되던 기반 수치(bases)들의 구성을 바꾸어, 보다 가속하는 연구, 그리고 기존에는 매우 많은 횟수를 수행하는 loop이 필요한 재부팅 알고리즘을 loop의 각 단계들 간 데이터 의존도를 제거하여 병렬처리 가능하도록 중간 암호문 형태를 변경, 일부 정확도를 포기하는 대신 연산의 병렬처리 가능성을 열어주는 알고리즘 개선 연구가 존재한다.

이런 수학/알고리즘 중심 접근법들은 많은 동형암호 연산기들로부터 차용받고 있지만, 일부는 실제 HW에 구현되는데 있어 한계를 보이기도 한다.

3.2. 컴파일러 중심 접근법

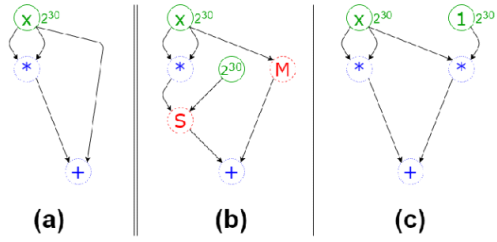
컴파일러 기반 동형암호 가속 연구들은 크게 하위함수 스캐줄링, 그리고 최적의 명령어 집합체를 목표로하는 low level language 컴파일러 연구의 두가지 방향으로 분류할 수 있다.

〔표 2〕 Microsoft SEAL을 활용한 N=8192일 때 CKKS 체계의 여러 하위함수들의 연산 시간

하위함수	연산시간(ms)
add	130
mult	4,460
relinearization	7,965
rotation	33,733

3.2.1. 하위함수 스캐줄러

동형암호를 사용한 프로그래밍의 큰 진입장벽 중 하나는 AI와 같은 주어진 응용의 연산과정을 동형암호 연산으로 효율적으로 변환하는 능력에 있다. [표 2]는 Microsoft SEAL[13] 동형암호 라이브러리를 활용하여 CKKS의 여러 하위함수들의 연산 시간을 측정한 결과를 나타낸다. 단순 프로그래밍 능력 뿐 아니라, 앞서 서술했듯, 동형암호의 하위함수들을 어떻게



〔그림 3〕 EVA 컴파일러가 x^2+x 를 연산하는 다양한 방법. (a) 기본 (b) 항상 rescale 우선 (c)scale 맞춤 우선 방법[15]

배열하여 수행하는지에 따라 같은 연산이라도 그 정확도와 성능이 달라지기에, 동형암호 표현 능력은 연산 성능에 큰 영향력을 미친다는 것을 알 수 있다. 많은 연구자들은 주어진 연산을 동형암호 하위함수들로 효율적으로 변환하여 배치하는 하위함수 스캐줄러 연구들을 수행했다.

하위함수 스캐줄링의 대표적인 예시로 데이터 패킹(packing)과 암호문 내 인덱스 회전(rotation) 간의 교환(trade-off)를 들 수 있다. BFV와 CKKS 같은 체계들은 한 암호문에 여러 데이터를 배열화하여 암호화할 수 있다. 이 때 기본적으로 데이터 간 연산은 같은 인덱스의 데이터 끼리만 수행되며, 서로 다른 인덱스의 데이터 간의 연산을 수행하고 싶은 경우, 암호문 내 데이터를 다른 인덱스로 옮기는 회전 연산을 수행해야 하는데, 이 회전 연산은 부하가 큰 편에 속하기 때문에, 이를 줄이는 방향으로 스캐줄링하는 것이 중요하다. CHET[14]은 CNN연산에 필요한 데이터 패킹과 rotation 연산 수행의 최적화를 위해 개발된 스캐줄러이다. CHET는 CNN의 입력 이미지 픽셀들을 최적의 순서로 패킹하여, rotation 수를 최소화하도록 하며, 이로 인해 세부 하위함수 수행 순서도 정해진다.

하위함수 스캐줄링의 다른 예시로, CKKS의 relinearization과 rescaling의 스캐줄링이 있다. 앞서 서술했듯, 기본 형태의 동형암호 암호문을 곱하면, 메시지 영역과 노이즈 영역이 증폭되며, 암호문의 level이 올라간다고 표현한다. 동일한 level의 암호문들 간에만 연산이 가능하기 때문에, 한번 연산이 수행된 level=2 암호문과 연산하기 위해서 level=1 암호문은 강제로 level=2로 바꾸어주어야 하며, 이는 일종의 곱셈과 동일한 수행시간을 필요로 한다. 이런 복합적인 상관관계를 Direct Acyclic Graph(DAG)를 활용하여

해석하고 분석하여 최적화된 하위함수 스캐줄링을 위해 만들어진 EVA[15]는 다양한 기본 원리들을 적용하여 하위함수 DAG를 최적화하는 시도들 끝에 가장 최적의 연산 시퀀스를 사용자에게 출력해주는 동형암호 컴파일러다. [그림 3]은 같은 수식을 연산하는 여러 방법을 소개하는데, 3가지 방법이 모두 서로 다른 수의 연산을 요구한다는 것을 확인할 수 있다.

이런 기본 원리를 시작으로, 다양한 최적화 시도들 끝에 기존 연구들보다 개선된 연구들이 발표되고 있으나 판을 뒤흔들 수준의 성능 개선을 이루는 연구는 아직 등장하지 않고 있다. 특히, 대부분의 연구들이 CHET와 EVA가 문제로 지적한 스캐줄링의 필요성을 기반으로 하고 있다는 점에서, 컴파일러 최적화만의 접근법은 한계가 있음을 알 수 있다.

3.2.2. low level language 동형암호 컴파일러

새로운 명령어들이 등장함에 따라, 자연스럽게 high level language의 컴파일 결과가 최적의 명령어 집합체로 이어지면 좋으나, 기존 컴파일러들의 한계들로 인해 최신 명령어들을 적극적으로 활용하지 못하는 경우들이 많다. 동형암호와 직결되는 대표적인 예시는 FFT를 가속하기 위해 만들어진 다양한 add-on 라이브러리들이 있다. 기본적으로 코드로 구현된 FFT의 경우, 컴파일러의 능력에 따라 그 성능이 다르지만, 대체적으로 주어진 규칙에 따라 연산을 수행하기 때문에 직접 일일이 최적의 스캐줄링을 구현한 명령어 집합체들에 비해 느릴 수 밖에 없다. 이에 FFT를 빠르게 구현한 명령어 집합체를 pre-define하여 활용하도록 FFT 라이브러리들을 pre-built 상태로 활용하여 동형암호를 가속하는 연구들이 다수 존재한다.

또한, 새로운 명령어들이 등장함에 따라 이들을 최대한 활용하고자 하는 연구들도 등장했다. Intel의 HEXL은 최신 AVX-512 명령어를 활용하도록 구현된 동형암호 라이브러리이다. Microsoft SEAL과 PALISADE에 활용되고 있으며, 기본 C/C++ 컴파일러들에 비해 훨씬 빠른 성능의 동형암호 연산을 하도록 명령어 집합체를 생성하는 컴파일러로써 기능을 수행한다.

3.3. HW를 활용한 동형암호 가속기 구현

여러 응용들에 대한 가속기들이 개발되었듯 동형암호 역시 HW를 활용한 가속 연구들이 진행되었다. 본 장은 고전적인 가속방식들을 활용한 동형암호 가속의 어려움들을 보여주고, 이를 해소하기 위한 기존 가속기들의 접근법들을 소개하고자 한다.

3.3.1. 병렬처리와 데이터 재사용

가장 기본적인 HW 가속기술로 우리는 병렬처리를 떠올려볼 수 있다. CPU의 core, GPU의 vector lane 등 다양한 단위로 병렬처리를 수행할 수 있는데, 동형암호는 병렬처리 가능한 연산단위가 다수 존재한다. 가장 대표적으로 우리는 요소 단위 병렬처리(coefficient-level)을 생각해볼 수 있다. FFT를 수행하는데 있어 다항식의 모든 항들은 두 개씩 쌍을 이루어 butterfly operation을 수행하고 이를 단위로 병렬처리를 구성할 수 있다. 또한, 중간값을 포함하여 다항식 단위로 병렬처리를 수행할 수 있으며, 더 나아가 각 암호문들 단위로도 병렬처리를 할 수도 있다.

하지만 우리는 암호문이 매우 큰 단위임을 명심해야 한다. 데이터 재사용, 혹은 캐싱과 같이 주 메모리에 접근하지 않고 연산을 이어나갈 수 있는 HW의 능력은 전체 연산 속도를 매우 개선시킬 수 있는 주요 특징 중 하나이다. 하지만, 암호문이 매우 크기 때문에, 암호문 단위, 혹은 다항식 단위 병렬처리를 수행할 경우 매우 큰 사이즈의 임시 메모리 혹은 캐시가 필요하다. CKKS의 표준 암호문 크기가 12.5MB 정도임을 고려하면, 두 개의 암호문을 담기위해서는 적어도 25MB의 크기가 필요하며, 하위함수 수행하는데 필요한 상수값(twiddle factor 등)을 고려하면 훨씬 더 많은 데이터가 필요함을 알 수 있다.

요소 단위 병렬처리의 경우, 한 연산기가 들고 있는 데이터를 재사용하기 어렵다는 단점이 있다. Cooley Tukey 방식의 FFT 연산 수행을 생각해보자. 한 stage에서 하나의 butterfly operation에 사용된 데이터는 다음 stage에서 다른 연산기에 필요하기에 데이터 이동이 필요하다. 이는 모든 stage에 걸쳐 한번도 중복되지 않기에 사실상 데이터 재사용이 없다고 생각해볼 수 있다.

데이터 재사용에 대해서는 다음과 같이도 분석해

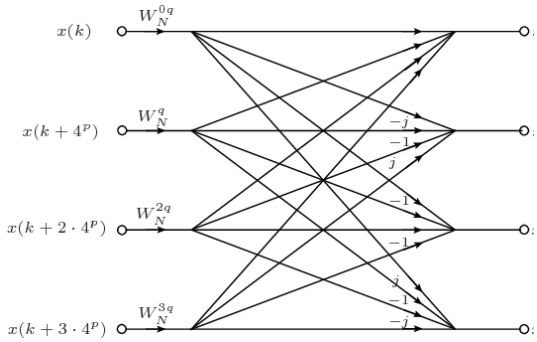
볼 수 있다. 하나의 특정 다항식의 특정 항이 다시 연산에 활용되기 위해서는 우선 한 다항식의 모든 항에 대한 연산 (예를들어 FFT의 하나의 stage)를 모두 수행한 후에야 가능하다. 이는 상당히 긴 시간을 필요로 하며, 그동안의 모든 데이터를 담을 수 있을 만큼 캐시가 커야만 해당 데이터가 캐시에서 지워지지 않고 남아있을 수 있다. 따라서, 동형암호 연산에 있어 충분한 데이터 재사용을 누리기 위해서는 매우 큰 임시 메모리, 레지스터, 혹은 캐시가 필요하다.

3.3.2. HW구현과 수학적 접근법의 한계

앞서 우리는 알고리즘 수준의 최적화를 통한 병렬처리 기회를 증가시켜준 연구와 같이, 수학적으로 의미를 갖는 기술들을 살펴보았는데, 이들은 HW에 매핑하는데 있어 다양한 한계를 보여준다. 병렬처리 단위 하나에 필요한 데이터를 저장하기 위해 필요한 임시 메모리의 크기는 상당히 크게 나타난다. 따라서, 이상적인 상황을 고려한 수학적 해석과 다르게, HW의 현실적인 구조적 특징으로 인해 누리기 어려운 병렬처리들이 존재하고, 이런 한계로 인해 수학적 개선 연구들은 빛을 발하지 못하곤 한다.

다른 예시로는 Cooley Tukey 접근을 활용한 FFT 연산의 다양한 내면을 예시로 들 수 있다. 첫째로, 기존 modular 연산을 수행하는 정수 연산기들은 FFT에서 활용하지 못하기 때문에, 별도의 연산기를 두어야 하여 resource 측면에서 낭비로 바라볼 수 있다. 둘째로, NTT를 사용할 경우, modular reduction이 필요한데, 이는 CPU나 GPU의 경우 매우 오래 걸리는 연산이며, FPGA나 ASIC으로 별도의 커스텀 연산기를 사용한다 하더라도 해당 연산기가 매우 많은 resource를 필요로 한다. 특히, 주어진 상수에 대한 reduction이 아닌 다수의 수들에 대한 reduction, 즉 나눗셈이 필요한 경우, CPU 혹은 GPU와 동일한 수준의 연산 부하가 발생한다. 때문에 동형암호 가속기 연구들은 대게 특정 파라미터들을 미리 선언하고, 이들에 대해서만 연산을 수행하도록 구현되곤 한다. 따라서 일반적으로 모든 파라미터를 수행하지 못한다는 단점을 지니고 있다고 할 수 있다.

다음으로, FFT를 구성하는 butterfly operation의 radix 단위에 따른 trade-off를 생각해 볼 수 있다. [그림 4]는 radix-4 FFT 단위 연산을 나타낸 것이다. 앞



〔그림 4〕 Radix-4 FFT 단위 연산

서 [그림 2]에서 소개한 radix-2와 다르게, 4개의 입력값을 한번에 처리하는 방식이다. 각 stage를 하나의 행렬 연산으로 여긴다면, radix 단위가 높아질수록 연산 수가 줄어든다고 표현할 수 있으며, 실제로 FFT 알고리즘의 개선을 언급하는 수학 분야 연구들 중 radix를 높이는 방법을 기본적으로 추천하는 경우가 존재한다[16]. 하지만, 실제 HW 구현을 고려해보면, 한번에 처리해야하는 연산 datapath는 깊어진다. 데이터 이동이 상대적으로 줄어든다는 장점을 지니고 있지만, 단위 모듈에 대해 더 많은 resource를 사용해야 하기에, 대다수의 동형암호 가속 연구들은 radix-2를 기본적으로 채택하고 있는 실정이다.

3.3.3. 주 메모리 접근 최적화

이상적인 메모리는 연산기에게 매 단위 시간(cycle)마다 데이터를 입력시켜줄 수 있겠으나, 현실 속 메모리인 DRAM과 같은 기기들은 데이터를 읽고 쓰는데 일정 수의 cycle을 소모하고 대기 시간등이 존재한다. 또한, 데이터 접근 패턴에 따라 이런 시간들에 차이가 발생하는데, 이로 인해 실제 원할때마다 데이터를 바로 입력받아 사용할 수 없기에 지연 시간이 발생한다. 따라서, 연산을 최적화하는 접근법을 활용한 연구만으로는 이런 메모리 지연시간으로 인한 부하를 해소하지 못하며, 특히 동형암호는 복잡한 데이터 접근 패턴을 갖고 있기에 더욱이 이런 지연시간은 눈에 띄게 나타날 것이다. 이런 한계를 극복하고자 동형암호의 메모리 패턴을 분석, 데이터를 중복하여 저장하더라도 최대한 연속적으로 데이터를 읽어내어 지연시간을 최소화하기 위한 연구가 존재한다[17]. 그 결과 다른 연구들에 비해 상대적으로 적은 (비슷한)

수의 연산기를 활용하지만, 훨씬 빠른 성능을 이룰 수 있었다.

3.3.4. 파이프라이닝

한편, 파이프라이닝을 활용한 동형암호 가속기 구현 연구들도 존재한다. 이들은 병렬처리 뿐 아니라, 매우 긴 datapath를 구현하여, 단일 동형암호 연산은 느리지만, 전체 throughput이 높고 추가적인 데이터 이동으로 인한 지연시간 증가를 방지한다는 장점을 지니고 있다. 하지만 이런 접근법들은 매우 큰 HW를 필요로 한다는 점, 그리고 주어진 datapath 외의 다른 최적화 기법을 적용할 수 없다는 단점을 지니고 있다.

MemFHE[18], F1[19], CraterLake[20] 모두 이런 특징을 지닌 가속기들이다. 그중 MemFHE는 비현실적인 (현실에 존재하기 어려운) 가상의 HW를 전제로 하여 동형암호 연산 한 주기 전체를 파이프라인 datapath를 활용하여 연산한다. F1과 CraterLake는 매우 큰 크기의 asic을 활용하여, 가장 중요한 datapath에 대한 파이프라인을 형성하여 활용한다. 이런 연구의 또다른 단점은, 미리 설계된 datapath와 다른 연산을 수행해야 하는 경우 (알고리즘 개선 등) 쉽게 적용하기 어렵다는 점이다. 특히, 나날이 동형암호의 수학적 알고리즘 자체가 발전하고 있는 가운데, 정해진 datapath만을 연산하는 설계 방식은 최신 알고리즘들에 대한 적용력을 떨어뜨린다는 단점을 지니고 있다.

Ⅳ. 다양한 가속 연구 결과의 시너지 효과를 누릴 수 있는 동형암호 가속 HW 설계

앞에서 소개한 다양한 가속기술들은 3장에서 소개한 것과 같이 양립하기 어려운 경우들이 다반사 존재하며, 특히 이는 HW의 현실적인 특징을 고려하지 않아 발생하는 경우가 많다. 우리는 이런 최근 연구들의 동향 분석을 바탕으로 다양한 연구들이 시너지효과를 내기 위해서는 다음과 같은 조건이 고려될 수 있음을 파악할 수 있다.

첫째, 수학적 알고리즘 개선에 있어 메모리 사용량이 고려되어야 한다. 이는 단순히 병렬처리 가능성만을 언급하는 것이 아닌, 각 병렬처리 thread가 필요로 하는 memory footprint와 메모리 재사용 주기를 명시하여, HW가 필요로 하는 최소 캐시 및 임시 데이터 크기를 예측하여 병렬처리 가능성을 고려하는 것이 필요하다는 것이다.

둘째, 총 연산수가 줄어드는 것 만큼, 단위 연산의 datapath 크기도 중요함을 항상 명심해야 한다. FFT의 radix 크기에 따른 trade-off를 통해 알 수 있듯이, 총 연산수는 줄어들지만, 더 큰 datapath를 독립적으로 구현해야 하기에 HW의 크기는 커질 것이고, 특히 resource의 재사용이 줄어들이 비효율적인 HW가 구현될 수 있다. 단 한번만 사용할 특수한 연산기들을 나열한 가속기보다는, 조금 성능이 느리더라도 재사용률이 높은 연산기들로 구성된 HW가 효율적인 설계를 이루었다고 하는 것은 통용된 사실이다. 동형암호 가속기도 이런 전통 HW 설계 방향을 따라야할 것이다.

마지막으로 본 논문이 제시하고 싶은 방안은, 특수한 파라미터와 체계, 그리고 알고리즘만을 위한 가속기보다는 보편적인, 공통적인 단위 연산기를 병렬적으로 배치하여 다양한 파라미터와 체계들을 지원하는 방안이다. 이런 방식은 특수한 가속기에 비해서 성능이 떨어질 수 있으나, 현재 아직 동형암호는 표준이 정해지지도 않았으며, 하루가 다르게 알고리즘에 변화가 발생하고 있다. 예를들어, 2021년도에 발표된 TFHE의 가속기[21]는 다양한 파라미터를 지원하지 못하며, 현재 사용하지 않는 알고리즘만을 가속하는 datapath로 이루어져 있는 한편, 바로 다음해 2022년도에 발표된 가속기[17]는 개선된 알고리즘과 다양한 파라미터들을 모두 지원할 수 있는 구조를 띄고 있어 새로운 알고리즘들에 대한 적용력이 높다.

이는 컴파일러들과의 호환 능력과도 연관이 있다. 기본적으로, GPU나 CPU를 활용하는 것에 비해서 ASIC을 활용한 특수 datapath를 구현한 F1과 CraterLake와 같은 가속기들은 해당 가속기가 제안한 스캐줄러 외 다양한 컴파일러 및 스캐줄러를 활용할 수 없다. 따라서 EVA와 같은 연구들이 등장하더라도, 적용할 수 없기에, 컴파일러의 등장과 함께 또다른 맞춤형 가속기가 등장하여야 하여 독립적인 연구들이 이어 등장할 수 밖에 없는 것이 현실이다. 이에 최근 XHEC[17]과 BASILISC[22]와 같이 체계들의 기본 연산들은 지원하되, 특수 datapath를 지원하기보다는 보편적인 가속기로 설계된 HW들이 등장하였는데, 이들은 각기 지원하는 체계들의 개선된 알고리즘들도 지원할 수 있다는 점에서, 큰 장점을 지닌다.

Ⅴ. 결 론

본 논문은 최근 동형암호 가속 연구들에 대한 동향을 분석, 이들을 연계하여 시너지 효과를 내며 가속기를 설계할 수 있는 방향을 제시했다. 특히 현재 다양한 분야에서 이루어지고 있는 연구들이 독립적으로 산개되어있는 상황에서, 각 연구들이 서로 시너지 효과를 내기 위해서는 HW 특성을 고려하여 메모리 크기, datapath의 사이즈 등 다양한 요소들을 고려하고, 보다 일반적인 보편화된 동형암호 가속기 설계로 이어지는 것의 필요성을 제안한다. 본 논문에서 제안한 방안을 바탕으로 다양한 연구들의 결과들이 합쳐진 가속기 연구가 이루어질 수 있었으면 하는 바램과 함께 본 논문을 마친다.

참 고 문 헌

[1] Gentry, Craig. "Fully homomorphic encryption using ideal lattices." Proceedings of the forty-first annual ACM symposium on Theory of computing. 2009.

[2] Rivest, Ronald L., Len Adleman, and Michael L. Dertouzos. "On data banks and privacy homomorphisms." Foundations of secure computation 4.11 (1978): 169-180.

[3] Rivest, Ronald L., Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems." Communications of the ACM 21.2 (1978): 120-126.

[4] ElGamal, Taher. "A public key cryptosystem and a signature scheme based on discrete logarithms." IEEE transactions on information theory 31.4 (1985): 469-472.

[5] Paillier, Pascal. "Public-key cryptosystems based on composite degree residuosity classes." International conference on the theory and applications of cryptographic techniques. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

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

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

[8] 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.

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

[10] Nam, Kevin, et al. "Partially Homomorphic Encryption HW accelerator." Proceedings of the Korea Information Processing Society Conference. Korea Information Processing Society, 2020.

[11] Albrecht, Martin, et al. "Homomorphic encryption standard." Protecting privacy through homomorphic encryption (2021): 31-62.

[12] Lopes, João & Sousa, José. (2017). Fast Fourier Transform on the Versat CGRA.

[13] Chen, Hao, Kim Laine, and Rachel Player. "Simple encrypted arithmetic library-SEAL v2. 1." Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21. Springer International Publishing, 2017.

[14] Dathathri, Roshan, et al. "CHET: an optimizing compiler for fully-homomorphic neural-network inferencing." Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation. 2019.

[15] Dathathri, Roshan, et al. "EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation." Proceedings of the 41st ACM SIGPLAN conference on pro-

gramming language design and implementation. 2020.

[16] Jones, Douglas L. "Radix-4 FFT Algorithms." Connexions module: m12027 (2006).

[17] Nam, Kevin, et al. "Accelerating n-bit operations over tfhe on commodity cpu-fpga." Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design. 2022.

[18] Gupta, Saransh, Rosario Cammarota, and Tajana Šimunić Rosing. "Memfhe: End-to-end computing with fully homomorphic encryption in memory." ACM Transactions on Embedded Computing Systems (2022).

[19] Samardzic, Nikola, et al. "F1: A fast and programmable accelerator for fully homomorphic encryption." MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. 2021.

[20] Samardzic, Nikola, et al. "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data." Proceedings of the 49th Annual International Symposium on Computer Architecture. 2022.

[21] Jiang, Lei, Qian Lou, and Nrushad Joshi. "Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus." Proceedings of the 59th ACM/IEEE Design Automation Conference. 2022.

[22] Geelen, Robin, et al. "BASALISC: Programmable Hardware Accelerator for BGV Fully Homomorphic Encryption." IACR Transactions on Cryptographic Hardware and Embedded Systems (2023): 32-57.

〈저 자 소 개〉

남 기 빈 (Kevin Nam)

2020년 2월 : 서울대학교 전기정보공학부 졸업
2020년 2월~현재 : 서울대학교 전기정보공학부 석박사통합과정
<관심분야> 프라이버시 보호 HW 보안, SW보안, AI 보안

정 헌 희 (Heonhui Jung)

2022년 2월 : 서울시립대학교 전자전기컴퓨터공학부 졸업
2022년 3월~현재 : 서울대학교 전기정보공학부 석사과정
<관심분야> 컴퓨터 아키텍처, 디지털 SoC, 보안

백 윤 흥 (Yunheung Paek)

종신회원

1988년 2월 : 서울대학교 컴퓨터공학과 졸업
1990년 2월 : 서울대학교 컴퓨터공학과 석사
1997년 4월 : 일리노이 주립대 전산학과 박사
1997년 9월~1999년 7월 : 뉴저지 주립 공대 전산과 조교수
1999년 8월~2003년 2월 : KAIST 전자전산학과 조교수/부교수
2003년 3월~현재 : 서울대학교 전기.컴퓨터공학부 부교수/정교수
<관심분야> SW 보안, HW 보안, 프라이버시 보호, AI 보안