The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.7, pp.291~298

eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.7.291

Annotation-guided Code Partitioning Compiler for Homomorphic Encryption Program

Dongkwan Kim† · Yongwoo Lee† · Seonyoung Cheon† · Heelim Choi† · Jaeho Lee† · Hoyun Youm†† · Hanjun Kim†††

ABSTRACT

Despite its wide application, cloud computing raises privacy leakage concerns because users should send their private data to the cloud. Homomorphic encryption (HE) can resolve the concerns by allowing cloud servers to compute on encrypted data without decryption. However, due to the huge computation overhead of HE, simply executing an entire cloud program with HE causes significant computation. Manually partitioning the program and applying HE only to the partitioned program for the cloud can reduce the computation overhead. However, the manual code partitioning and HE-transformation are time-consuming and error-prone. This work proposes a new homomorphic encryption enabled annotation-guided code partitioning compiler, called Heapa, for privacy preserving cloud computing. Heapa allows programmers to annotate a program about the code region for cloud computing. Then, Heapa analyzes the annotated program, makes a partition plan with a variable list that requires communication and encryption, and generates a homomorphic encryption-enabled partitioned programs. Moreover, Heapa provides not only two region-level partitioning annotations, but also two instruction-level annotations, thus enabling a fine-grained partitioning and achieving better performance. For six machine learning and deep learning applications, Heapa achieves a 3.61 times geomean performance speedup compared to the non-partitioned cloud computing scheme.

Keywords : Cloud Computing, Privacy Preserving, Homomorphic Encryption, Compiler

지시문을 활용한 동형암호 프로그램 코드 분할 컴파일러

김 동 관† · 이 용 우† · 천 선 영† · 최 희 림† · 이 재 호† · 염 호 윤†† · 김 한 준†††

요 약

클라우드 컴퓨팅이 널리 사용되면서, 데이터 유출에 대한 관심도 같이 증가하고 있다. 동형암호는 데이터를 암호화된 채로 클라우드 서버에서 연산을 수행함으로써 해당 문제를 해결할 수 있다. 그러나, 프로그램 전체를 동형암호로 연산하는 것은 큰 오버헤드를 가지고 있다. 프로그램의 일부분만 동형암호를 사용하는 것은 오버헤드를 줄일 수 있지만, 사용자가 직접 프로그램의 코드를 분할하는 것은 시간이 오래 걸리는 작업이고 또한 에러를 발생시킬 수 있다. 이 연구는 지시문을 활용하여 동형암호 프로그램의 코드를 분할하는 컴파일러인 Heapa를 제시하였다. 사용자가 프로그램에 클라우드 컴퓨팅 영역에 대한 코드를 지시문으로 삽입하면 Heapa는 클라우드 서버와 호스트사이의 통신 및 암호화를 적용시킨 계획을 세우고, 분할된 프로그램을 생성한다. Heapa는 영역 단위의 지시문뿐만 아니라 연산 단위의 지시문도 사용가능하여 프로그램을 더 세밀한 단계로 분할 가능하다. 이 연구에선 6개의 머신러닝 및 딥러닝 어플리케이션을 통해 컴파일러의 성능을 측정했으며, Heapa는 기존 동형암호를 활용한 클라우드 컴퓨팅보다 3.61배 개선된 성능을 보여주었다.

키워드 : 클라우드 컴퓨팅, 프라이버시 보존, 동형암호, 컴파일러

1. 서 론

클라우드 컴퓨팅이 널리 사용되고 있지만 데이터 유출의 위험은 새롭게 등장한 문제이다. 사용자들은 프라이버시 데이터를 클라우드 서버로 보내 활용하고 있지만, 악의적인 공격자들은 클라우드 서버에 저장된 암호화되지 않은 데이터에 접근할 수 있다.

동형암호[1-6]는 클라우드 컴퓨팅에서 프라이버시를 보존하기 위한 방법이다. 동형암호를 활용하면 암호화된 상태로

※ 이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No. RS-2020-II201361, 인공지능대학원지원(연세대학교); No. RS-2022-II220050, 데이터 플로우 구조 기반 PIM의 실행 및 프로그래밍 모델 개발; No. RS-2023-00277060, 개방형 엣지 AI 반도체 설계 및 SW 플랫폼 기술개발; No. RS-2024-00395134, 차세대 AI 반도체를 위한 DPU 중심의 데이터센터 아키텍처; No. RS-2024-00358765, 다양한 가속기에 이식 가능한 동형암호 컴파일러 시스템 개발). 또한 이 논문은 삼성전자의 지원을 받아 수행된 연구임.
† 비 회 원 : 연세대학교 전기전자공학과 석·박사통합과정
†† 준 회 원 : 연세대학교 전기전자공학과 석·박사통합과정
††† 종신회원 : 연세대학교 전기전자공학과 부교수
Manuscript Received : May 13, 2024
Accepted : June 2, 2024
* Corresponding Author : Hanjun Kim(hanjun@yonsei.ac.kr)

데이터 유출없이 보안적인 연산이 가능하다. 이 방법을 통해 개인의 프라이버시 데이터나 기업의 기밀 데이터를 다루는 헬스케어산업이나 금융산업에서 보안 클라우드 컴퓨팅을 적용시킬 수 있다. 그러나 동형암호를 활용한 클라우드 컴퓨팅은 하나의 연산마다 수백 밀리초의 큰 수행 시간이란 단점을 가지고 있다. 그러므로 프로그램 전체를 동형암호 프로그램으로 하는 것이 아니라, 프로그램의 보안이 필요한 일부분만 동형암호화하면 필요없는 오버헤드를 제거할 수 있다.

사용자가 직접 프로그램을 동형암호화 하는 것은 시간이 오래 걸리는 작업이고 에러를 발생할 가능성도 높다. 사용자는 동형암호에 대한 높은 지식을 가지고 있어야하며, 호스트와 클라우드 서버간의 통신도 고려해야하고, 이 과정에서 데이터의 보안 수준을 위반해서도 안된다. 컴파일러를 활용하여 자동으로 프로그램을 분할한다면 해당 문제점들을 해결할 수 있어 지시문을 활용하여 자동으로 프로그램을 분할하는 기법[7-11] 및 자동 동형암호화 컴파일러[12-15]가 존재하였으나 오리지널 프로그램에서 동형암호가 적용된 분할 프로그램을 자동으로 생성해주는 연구는 없다.

이 연구는 새롭게 지시문을 활용한 동형암호 프로그램 코드 분할 컴파일러를 제시한다. Heapa는 2개의 영역 단위의 지시문과 2개의 연산 단위 지시문을 도입하여 동형암호 프로그램을 분할할 수 있다. 이 지시문들을 사용하여 사용자는 쉽게 클라우드 서비스내에서 실행될 코드를 세밀하게 지정할 수 있다. Heapa 컴파일러는 지시문이 적용된 프로그램의 PDG (Program Dependence Graph)를 분석하여 호스트와 클라우드를 위한 통신 및 동형암호 연산을 추가하고, 서브 프로그램들을 생성한다.

이 연구에선 Heapa 컴파일러의 효용성을 평가하기 위해 6개의 머신러닝 및 딥러닝 어플리케이션을 사용한다. 코드 분할 없이 프로그램 전체를 동형암호화한 방법과 비교하여 Heapa 컴파일러는 평균적으로 3.61배의 속도 향상을 보인다. 또한, 지시문없이 수동적으로 분할한 프로그램과 비교하였을 때, Heapa 컴파일러를 세밀하게 분할하였을 때 평균적으로 1.09배의 속도 향상을 보였다.으로 1.09배의 속도 향상을 보였다.

2. 배경 지식

2.1 프라이버시 보존 클라우드 컴퓨팅

클라우드 컴퓨팅에서 데이터 프라이버시를 보호하기 위해 기존 연구들은 차등 프라이버시(Differential Privacy), 보안 다자간 컴퓨팅(Secure Multiparty Computation, MPC), 동형암호(Homomorphic Encryption)의 방법들을 사용하였다.

차등 프라이버시[16-19]는 연합 학습(Federated Learning)에서 자주 사용되는 기법이다. 데이터 프라이버시를 보호하기 위해, 연합 학습은 보안 데이터는 공유하지 않고, 연산 결과만을 공유하지만, 보안 데이터가 공유되지 않더라도 악의적인 그룹은 연산 결과로부터 보안 데이터를 얻어낼 수 있다. 차등 프라이버시 기법은 이를 방지하여 연산 결과로부터 보안 데이터를 얻을 수 없게 해주지만 연산 자체를 클라우드에서 실행하도록 하는 오프로딩이 불가능하다는 단점이 있다.

MPC[20-22]는 보안 프로토콜을 바탕으로 암호화된 데이터를 처리할 수 있게 한다. 그러나 특정 프로토콜을 사용하여 연산을 구성함으로써[20, 23] MPC는 연산 과정중 통신 오버헤드가 굉장히 크다.

동형암호[1-6]는 대수학적인 특성을 바탕으로 암호화된 데이터를 연산할 수 있다. 동형암호가 MPC와는 다르게 다른 길이의 연산을 계산할 수 있는데, 이로 인해 MPC만큼의 큰 통신 오버헤드를 가지고 있진 않지만 연산 오버헤드는 MPC보다 크다. 이 연구는 클라우드 컴퓨팅의 높은 연산력을 사용하며 동형암호를 사용해 데이터 프라이버시를 보호하는 것을 목표로 한다.

2.2 동형암호

Fig. 1은 동형암호[1-6]를 사용해 데이터를 암호화된 채로 연산을 수행하는 것을 나타낸다. 데이터는 Ring-lattice with error (R-LWE)를 바탕으로하는 다항식 링에 평문 $m \in Z_Q[X]/(X^N+1)$으로 표현된다. 동형암호 기법은 평문을 무작위 노이즈 e와 a, 비밀키 s로 이루어진 암호문 $(c_0,c_1)=(-as+m+e,a)$로 암호화 한다. Q는 coefficient modulus로, 최근 연구들[2, 30]은 작은 moduli를 곱하여 coefficient modulus를 만드는 방법들을 제시하고, moduli의 개수를 레벨이라 한다($Q=\Pi q_i$).

동형암호의 암호문은 데이터와 비밀키를 지키기 위해 약간의 에러를 추가하기 때문에 동형암호 연산을 하는 과정에서 노이즈가 생긴다. 노이즈에 의한 데이터 손상을 방지하기 위해 동형암호 기법들은 노이즈와 평문의 coefficient의 비율을 조절한다. BFV [5]와 BGV [1] 기법에서 평문 modulus나 CKKS [3] 기법의 scale이라 불리는 파라미터 t를 사용하여, 평문을 BFV 기법에서 $m \cdot \frac{q}{t}+e$, BGV 기법에서 $et+m$, CKKS 기법에서 $mt+e$로 표현한다. CKKS 기법에선 scaling에 의해 곱셈 연산을 할수록 암호문의 다항식의 coefficient는 증가한다. 현재 CKKS 기법은 modulus-switching 기법을 사용하여 coefficient와 modulus를 줄이고, 여러 연구들[12-14]은 modulus-switching과 관련한 파라미터를 관리하는 방법들을 제시한다.

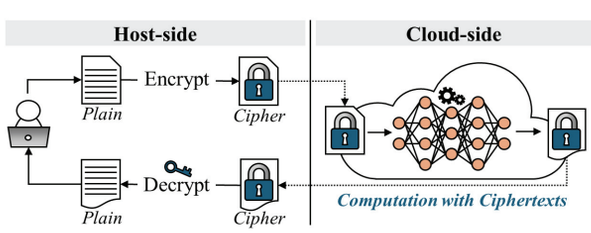


Fig. 1. Execution model of Homomorphic Encryption. HE Enables Computation with Ciphertexts without Decrypting.

2.3 동형암호 컴파일러

기존 동형암호 컴파일러들[12-14]은 프로그래밍을 용이하게 하기 위해 동형암호 프로그램의 파라미터를 자동으로 최적화해준다. EVA [12]는 프로그램의 처음부터 끝까지 사용되는 scale t을 분석하여 scale의 값이 데이터의 손상 방지를 보장하는 최소한의 값을 유지할 수 있도록 rescale이라 불리는 modulus-switching 연산을 넣어 최적화한다. EVA의 scale 관리 방법은 프로그램 전체의 scale을 낮추는 것이지만, 암호문이 가진 레벨의 관점에서 이 방법은 최적의 방법은 아니다. Hecate [14]는 다양한 scale 관리 계획을 탐색하는 방법을 제시한다. rescale 연산의 위치를 변경하며 전체 프로그램의 수행 시간을 예상하고 가장 좋은 scale 관리 계획을 선택한다. ELASM [13]은 연산의 에러와 노이즈를 분석하고 주어진 에러 한계내에서 가장 최적의 성능을 보이는 파라미터 최적화를 수행한다.

3. 연구 동기

3.1 동형암호를 위한 자동 분할 프레임워크

클라우드 컴퓨팅에서 동형암호를 사용하면 프라이버시를 보호할 수 있지만, 한 연산에 수백 밀리초나 걸리는 큰 컴퓨팅 오버헤드로 인해 사용하기 어렵다. 또한 동형암호는 연산의 종류와 암호문의 깊이(Multiplication Depth)에 따라 수행 시간이 달라지는데 Fig. 2는 깊이가 증가함에 따라 수행 시간이 증가하는 것을 나타낸다. 여기서 깊이는 해당 암호문까지 수행된 곱셈 연산의 수를 의미하는데, 동형암호 연산을 수행함으로써 깊이를 줄여 프로그램 전체의 수행 시간을 줄일 수 있다.

그러나 사용자가 직접 프로그램에서 필요한 부분만 동형암호화 하는 것은 시간도 오래걸리고 에러도 발생시킬 수 있다. 보안 데이터를 다루는 연산을 분석해야하고, 호스트 서브 프로그램과 클라우드 서브 프로그램으로 전체 프로그램을 분할해야하며, 또한 그 사이에 통신 연산도 넣어주어야한다. 게다가 클라우드 서브 프로그램에선 동형암호 지식을 가지고 기존에 존재하는 동형암호 컴파일러를 사용해야한다. 프로그램 내에 다양한 위치에서 분할할 수 있기 때문에 효율적인 성능을

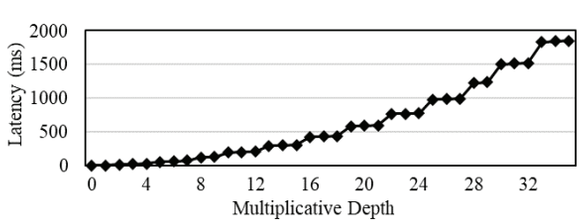


Fig. 2. Total Latency of Ciphertext Multiplications for a Given Multiplicative Depth (Same with the Number of Ciphertext Multiplication). The Latency Cubically Increases as Multiplicative Depth Grows.

가진 프로그램을 위해선 반복적으로 나눠봐야하고, 이는 사용자에게 큰 부담을 지게한다.

동형암호를 위한 자동 분할 프레임워크가 존재한다면 프로그램을 자동으로 호스트 서브 프로그램과 클라우드 서브 프로그램으로 분할할 수 있고, 또한 필요한 부분에 자동적으로 동형암호를 적용시킬 수 있다. 그러나, 기존에 존재하는 동형암호 컴파일러들[12-15]은 코드 분할을 지원하지 않으며, 자동 코드 분할 기법들[7-11]은 동형암호를 지원하지 않는다. 이 연구는 지시문을 사용한 자동 분할 컴파일러 프레임워크인 Heapa를 제시하여 호스트 서브 프로그램과 동형암호를 적용할 수 있는 클라우드 서브 프로그램을 생성할 수 있다.

3.2 위험 모델

Heapa 프레임워크에서 수행하는 클라우드 컴퓨팅은 호스트와 클라우드의 두 영역으로 구분한다. 데이터 유출이 걱정없는 부분의 코드는 호스트와 클라우드가 서로 공유할 수 있고, 사용자의 입력값이나 모델 가중치와 같은 데이터는 호스트와 클라우드에서 각각 보안 데이터로 가정한다. 호스트와 클라우드는 각각 주어진 프로그램을 수행하고, 데이터의 암호화는 호스트에서 수행된다. 호스트는 클라우드로 데이터를 보내기 전에 보안 데이터를 암호화하고, 결과를 받은 후엔 복호화한다. 이 연구에서 서비스 제공자는 클라우드 서버를 믿는다고 가정하기 때문에 서비스 제공자를 클라우드 서버로써 역할을 한다고 본다.

클라우드 서버는 신뢰성 있는 인프라를 제공하여 보안 통신 과정 중에 중간자 공격(Man-in-the-middle Attack)은 방어할 수 있다. 그러나, 해당 인프라가 클라우드 서버 내에서의 데이터 보안을 의미하진 않는다. 이전 연구들[24-28]은 외부에서 데이터를 저장하거나 처리하는 것에 대한 문제를 지적하였고, 클라우드 제공자나 공격자에 의한 위험을 언급하였다.

기존 연구들[24, 26, 28]은 클라우드 서버를 honest-but-curious, 즉, 요청받은 수행은 실제로 실행하지만 이와 별개로 사용자 데이터를 서버 내에 남겨놓을 수 있다고 보았다. 예를 들어, 클라우드 서버는 금융적인 목적으로 사용자 데이터를 지우지 않고 가지고 있거나, 요청받은 수행과는 관련없는, 물체 인식이나 행동 예측같은 모델 트레이닝에 사용자 데이터를 사용할 수도 있다.

이 연구는 Heapa 런타임이 호스트와 클라우드 플랫폼에 설치되어있고, 서로 통합적으로 수행된다고 가정한다. 런타임은 동형암호 수행에 필요한 키나, 생성된 프로그램을 자동으로 호스트와 클라우드에 전달한다. 호스트와 클라우드 사이의 데이터 전달은 정확히 이루어지며, 데이터 변조는 일어나지 않는다. 동형암호 라이브러리로 SEAL 라이브러리[29]를 사용하여, 해당 라이브러리가 지원하지 못하는 조건문과 같은 연산 및 관련된 공격에 대해서 이 연구에서 다루지 않는다.

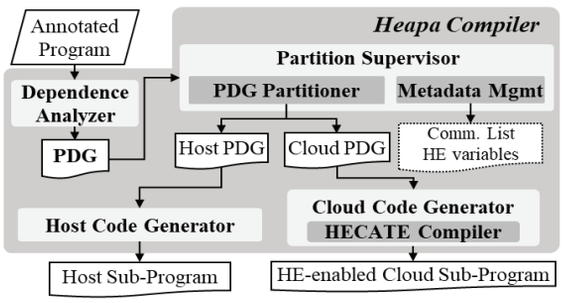


Fig. 3. Overall Compilation Procedure of the Heapa Compiler. The Dotted Box Represents Metadata, which Includes the Communication List and HE Variables.

4. Heapa 지시문 및 컴파일러

이 연구는 동형암호를 지원하는 지시문을 사용한 분할 컴파일러, Heapa를 제시하였다. Heapa는 Heapa 지시문, 컴파일러 및 런타임 시스템으로 구성되어있다. Heapa 지시문은 사용자가 클라우드 서버에서 실행될 코드 영역을 세밀하게 지정할 수 있다(Section 4.1). Fig. 3은 Heapa 컴파일러의 과정을 나타낸다. Heapa 컴파일러는 지시문이 있는 프로그램의 PDG를 분석하고, 클라우드 서버에서 실행될 코드 영역을 찾아낸다(Section 4.2). 컴파일러는 통신 및 동형암호가 적용된 분할 PDG를 생성하고(Section 4.3), 이를 바탕으로 호스트 서브 프로그램과 클라우드 서브 프로그램을 생성한다(Section 4.4).

4.1 Heapa 지시문이 포함된 프로그램 코드

클라우드 컴퓨팅을 위한 동형암호 프로그램을 효율적으로 만들 수 있도록 이 연구는 자동으로 코드를 분할하는데 사용할 지시문을 새롭게 제시하였다. Table 1은 Heapa에서 사용하는 지시문들을 표현한다. 지시문은 2개의 영역 단위 지시문과(#pragma Heapa Begin and End) 2개의 연산 단위 지시문으로(#pragma Heapa Host and Cloud) 구성되어 있다. 해당 지시문들을 사용하여 사용자는 쉽게 클라우드에서 사용할 코드를 지정할 수 있다. 먼저 사용자는 영역 단위 지시문을 사용하여 클라우드 서버에서 실행될 코드들을 지정한다. 그 후, 연산 단위의 지시문을 사용해서 영역 단위에 포함되더라도 호스트에서 실행되거나, 혹은 영역 단위에 포함되지 않더라도 클라우드에서 실행되도록 포함시킨다. 두 단계로 이루어진 지시문을 활용하여 사용자는 쉽게 코드 분할을 할 수 있다.

Listing 1은 파란색으로 표시된 Heapa 지시문을 사용한 예시 DNN 프로그램이다. conv2(B, W2)와 AvgPool(C)를 클라우드에서 실행시키도록 지시하되, rot(I[k], a)는 호스트에서 실행시키도록 한다.

4.2 프로그램 분석

Heapa 컴파일러는 프로그램의 PDG를 분석하여 분할 포인트를 찾는다. Fig. 4A에서 Heapa는 먼저 주어진 프로그램의 변수로 구분하고, 지시문으로 표현된 영역에 대해 표시한다.

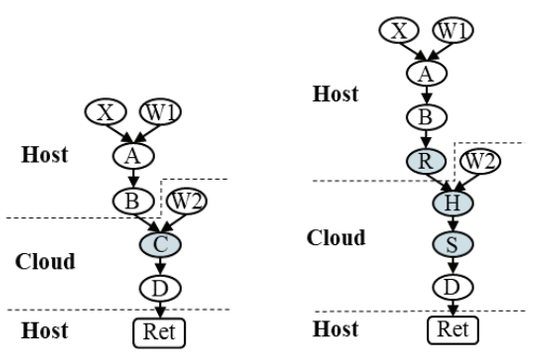
Table 1. Meaning of the Heapa Annotations.

Annotation	Meaning
#pragma Heapa Begin	Starting point of cloud section
#pragma Heapa End	End point of cloud section
#pragma Heapa Host	Instruction executed at host
#pragma Heapa Cloud	Instruction executed at cloud

```
1  conv2(I, W) {
2    res = 0
3    for k in [0, 6):
4      for a in [0, 25):
5        #pragma Heapa Host
6        R = rot(I[k], a)
7        H = R * W[k][a]
8        res = res + H
9    return res; }
```

```
10 DNN(x) {
11   A = conv1(x, W1)
12   B = AvgPool(A)
13   #pragma Heapa Begin
14   C = conv2(B, W2)
15   D = AvgPool(C)
16   #pragma Heapa End
17   return D }
18
```

Listing 1. Example DNN Pseudocode. Programmers Only Need to Add the Heapa Annotations (Blue Colored) to an Existing Program in Order to Extract Cloud Code Regions for Privacy Preserving Cloud Computing.



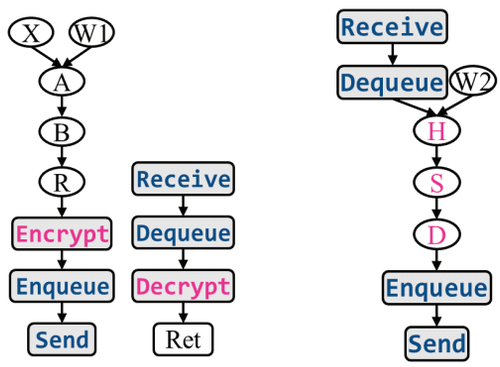
(A) Coarse-grained Partitioning (B) Fine-grained Partitioning

Fig. 4. Program Dependence Graphs (PDGs) of the example program with the coarse- and fine-grained partitioning plans.

그 후, Fig. 4B에서 세밀한 연산 단위에 대해 지시문에 따라 클라우드 프로그램에 포함시키거나 호스트 프로그램에서 제외한다. Heapa 컴파일러는 PDG 분석 결과를 바탕으로 분할된 프로그램의 PDG를 각각 생성한다.

4.3 통신 및 동형암호 적용

분할 PDG를 바탕으로 Heapa 컴파일러는 호스트와 클라우드 사이의 통신 및 동형암호가 적용된 서브 프로그램을 생성한다. Fig. 5는 각 서브 프로그램을 의미한다. 분할 PDG 간의 경계에서 컴파일러는 Enqueue, Dequeue 및 Send, Receive 연산을 추가하여 호스트와 클라우드가 서로 데이터를 주고 받을 수 있도록한다. 호스트 서브 프로그램의 코드인 Listing 2와 클라우드 서브 프로그램의 Listing 3에서 파란색으로 표시된 코드는 해당 통신 연산들을 의미한다.



(A) Host PDG (B) Cloud PDG

Fig. 5. Partitioned PDGs. Instructions and Variables Related to HE are Marked in Red, and Instructions for Communication are Indicated in Blue.

```
1  conv2_Host(Q, I) {
2    for k in [0, 6):
3      for a in [0, 25):
4        R[k][a] = rot(I[k], a)
5        Encrypt(R[k][a])
6        Enqueue(Q, R[k][a]) }
7
8
9
10
11
12
```

```
13 DNN_Host(x) {
14   Q_1 = InitQueue()
15   A = conv1(x, W1)
16   B = AvgPool(A)
17   conv2_Host(Q_1, B)
18   Send(Q_1)
19   // Offloaded code
20   // executed at cloud
21   Q_2 = Receive()
22   D = Dequeue(Q_2)
23   ret = Decrypt(D)
24   return ret  }
```

Listing 2. Partitioned Code for the Host The Heapa Compiler Inserts Communication Codes And Encryption/Decryption Codes.

```
1  conv2_cloud(Q, W) {
2    S=0
3    for k in [0, 6):
4      for a in [0, 25):
5        R=Dequeue(Q)
6        H=HE_mul(R,W[k][a])
7        S=HE_add(S,H)
8    return S  }
```

```
9  DNN_cloud() {
10   Q_2 = InitQueue()
11   Q_1 = Receive()
12   C = conv2_cloud(Q_1, W2)
13   D = HE_AvgPool(C)
14   Enqueue(Q_2, D)
15   Send(Q_2)  }
16
```

Listing 3. Partitioned Code for the Cloud. The Heapa Compiler Inserts Communication Codes and Transforms the Plain Operations into Homomorphic Operations.

Heapa 컴파일러는 서브 프로그램에 동형암호 연산을 추가적으로 적용한다. 호스트 서브 프로그램의 경우, 클라우드로 보내기 전에 데이터를 암호화하거나 결과를 받아 복호화한다. 클라우드 서브 프로그램의 경우, 프로그램내에서 실행되는 연산들을 동형암호 라이브러리를 사용하도록 변환한다. 동형암호 연산의 경우 Listing 2와 Listing 3에서 빨간색으로 표시하였다.

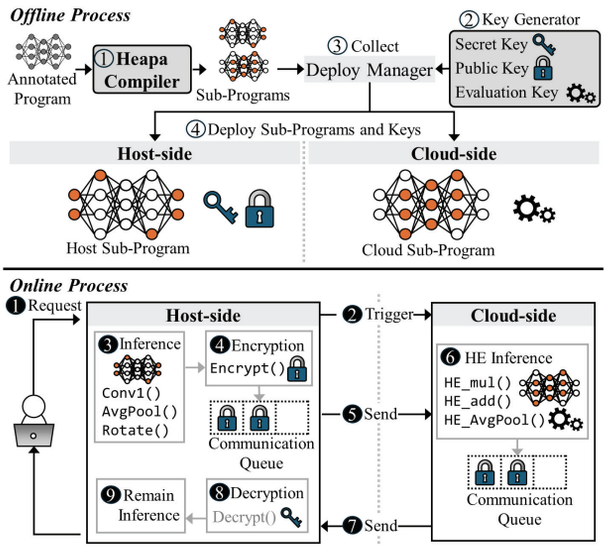


Fig. 6. Key Management and Execution Process of the Heapa Runtime. The Key Generation and Distribution Take Place in an Offline Process. The Online Process Shows How Computation is Performed on Both the Host Side and the Cloud Side.

4.4 Heapa 런타임

Heapa 런타임은 오프라인과 온라인 파트로 구분하며, Fig. 6은 런타임의 전반적인 구조를 표현한다. 오프라인 파트는 4단계로 이루어져있다. 먼저, Heapa 컴파일러가 프로그램을 분할하여 서브 프로그램을 생성한다. 그 이후, 동형암호 연산에 필요한 비밀키, 공개키, 평가키를 생성한다. Heapa 런타임은 생성된 프로그램과 키들을 호스트와 클라우드로 전달한다. 온라인 파트에서 호스트와 클라우드는 서브 프로그램을 실행한다. 사용자가 수행을 요청하였을 때, 호스트와 클라우드는 통신을 준비하게되고, 먼저 호스트 서브 프로그램의 연산을 수행한다. 호스트에서의 결과는 암호화되어 클라우드 서버로 보내지고, 클라우드 서브 프로그램은 데이터를 암호화된 채로 동형암호를 사용하여 연산한다. 연산이 모두 끝나면 결과는 다시 호스트 서브 프로그램으로 보내지고, 호스트는 결과를 복호화하여 평문으로 나머지 연산을 수행하여 최종 결과를 사용자에게 전달한다.

5. 실험 결과

이 연구는 호스트-클라우드 컴퓨팅 환경을 가정하여 실험을 수행하였다. 호스트는 ARM big-LITTLE 아키텍쳐인 Odroid N2를 사용하였고, 클라우드 서버로는 Intel Core i7-8700의 CPU를 가진 데스크탑 서버를 사용하였다. 동형암호 라이브러리는 마이크로소프트 SEAL Library (Release 3.5.9) [29]를 사용하였다.

이 연구는 전체를 동형암호를 사용하였던 Hecate [14]와 영역 단위 분할(Coarse-grained), 영역과 연산 단위 분할

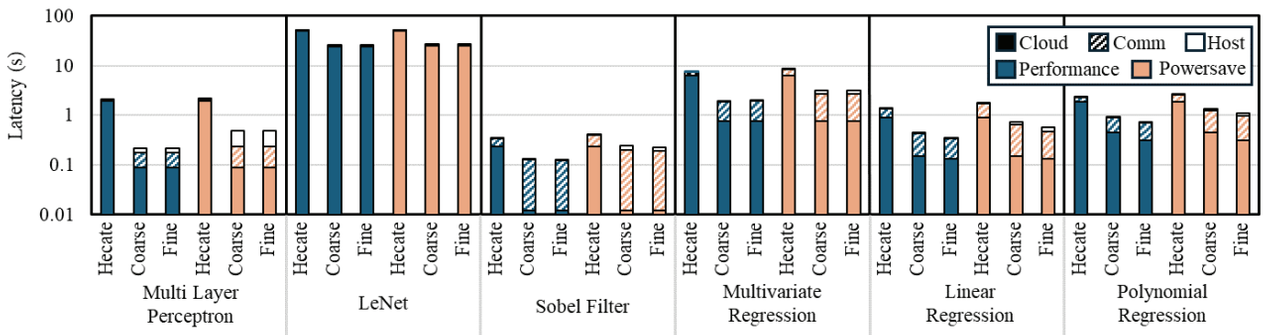


Fig. 7. Latency of Benchmarks Under the Performance Governor and Powersave Governor.

(Fine-grained)의 3가지 방법을 비교하였다. 분할된 프로그램들은 Hecate로 컴파일된 프로그램과 동작상 같은 결과를 내도록 생성된다. 또한, 호스트의 컴퓨팅 성능에 따른 결과를 확인하기 위해 CPU의 주파수(Frequency)를 변경하여 최대 주파수(Performance) 및 최소 주파수(Powersave)의 성능을 비교하였다.

사용한 벤치마크는 기존 연구인 EVA [12]와 Hecate [14]에서 사용한 다층 퍼셉트론(MLP), LeNet, 소벨 필터(Sobel Filter), 다변량 회귀(Multivariate Regression), 선형 회귀(Linear Regression), 다항 회귀(Polynomial Regression)을 사용하였다.

5.1 수행 시간

Fig. 7은 분할 방법 및 호스트 성능에 따른 수행 시간 결과를 나타낸다. 프로그램 수행 시간은 호스트와 클라우드에서의 연산 수행 시간 및 통신 시간을 측정하였다. 실험 결과는 모든 벤치마크에서 전체를 암호화한 프로그램보다 영역 단위로만 분할 하였을 때 3.32배, 연산 단위까지 세밀하게 분할한 프로그램은 3.61배 빠른 수행시간을 보여주었다. 클라우드보다 낮은 컴퓨팅 파워를 가지고 있는 호스트가 프로그램의 일부 연산을 계산하더라도 동형암호의 오버헤드를 제거할 수 있어 프로그램의 전체적인 성능 향상을 확인할 수 있다.

실험 결과에서 연산 단위까지의 세밀한 분할이 영역 단위보다 1.09배 빠른 성능을 가진다는 점도 확인할 수 있다. Heapa 지시문을 사용하여 사용자는 연산 영역에 대해 쉽게 세밀한 조절이 가능하면서도 좋은 성능을 가질 수 있다.

호스트 성능과 관련해서 최소 주파수를 가졌을 때, 연산 영역 단위의 세밀한 분할은 Hecate의 결과보다 2.55배, 영역 단위 분할보다 1.09배 빠른 결과를 보여주었다. 통신을 제외하고 연산 오버헤드에서 호스트 성능은 결과에 크게 영향을 보이지 못했는데, 이는 프로그램 전체 오버헤드에 비해 동형암호를 사용하지 않는 호스트의 오버헤드가 상대적으로 작은 부분을 차지하기 때문이다.

5.2 사용성 및 확장성

Table 2에서 원본 코드, 사용자가 직접 분할 및 동형암호를 적용한 코드, Hecate [14]를 사용하지만 분할은 직접한 코드, Heapa 컴파일러 코드의 Line of Code (LoC)를 비교하였다. 사용자가 직접 코드를 분할하고, 통신 및 동형암호를 적용하는 경우 원본 코드보다 1.98배 많아졌으며, 단순히 코드만 늘어나는게 아니라 사용자는 동형암호 지식을 필요로 하며 데이터가 유출되지 않도록 정확한 분석이 필요하다.

Table 2. Line of Code (LoC). Geomean Means the Geomean Value of Increased LoC Ratio Over the Original Code.

Benchmark	Original	Manual	Manual +Hecate[14]	Heapa
MLP	64	90	81	66
LeNet	266	387	280	268
SobelFilter	25	54	39	28
MR	53	123	88	58
LR	48	102	76	52
PR	44	122	75	49
Geomean	1	1.98	1.45	1.07

동형암호 컴파일러를 사용하는 경우 원본 코드보다 1.45배의 코드로 작성할 수 있지만, 여전히 데이터 통신 및 암호화를 고려하여 분할하여야한다. 그러나, 코드에 지시문을 추가하는 방식으로 Heapa 컴파일러를 사용하면 원본 코드보다 단 1.07배의 코드로 데이터 통신 및 프라이버시 보존이 되는 분할 프로그램을 작성할 수 있다.

6. 결 론

동형암호를 사용한 효율적인 프라이버시 보존 클라우드 컴퓨팅을 위해 이 연구는 Heapa 지시문와 컴파일러를 제시하였다. Heapa 지시문은 사용자가 통신 및 동형암호에 대한 고려없이 쉽게 호스트와 클라우드의 프로그램을 만들 수 있게 해준다. Heapa 컴파일러는 지시문을 바탕으로 프로그램 코드를 분할하며, 통신을 위한 코드를 삽입하고 동형암호로 연산할 수 있게 프로그램을 변환한다. 이 연구는 6개의 딥러닝 및 머신러닝 벤치마크를 사용하여 기존의 분할하지않은 동형암호 프로그램보다 Heapa가 3.61배 성능 향상을 갖는 것을 보여주었다. 또한, 연산 단위의 세밀한 지시문을 활용하여 영역 단위보다 1.09배의 개선된 성능을 가짐을 보였다.

References

[1] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption without Bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference,* pp.309-325, 2012.

[2] J. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Selected Areas in Cryptography*, 2019.

[3] J. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology-Asiacrypt*, pp.409-437, 2017.

[4] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *in Journal of Cryptology*, pp.34-91, 2020.

[5] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," in *Cryptology ePrint*, 2012.

[6] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, pp.169-178, 2009.

[7] N. Giang, M. Blackstock, R. Lea, and V. Leung, "Developing IoT applications in the fog: A distributed dataflow approach," in *Proceedings of 2015 International Conference on the Internet of Things*, 2015.

[8] N. Giang, M. Blackstock, R. Lea, and V. Leung, "Distributed data flow: A programming model for the crowdsourced internet of things," in *Proceedings of the Doctoral Symposium of the 16th International Middleware Conference*, 2015.

[9] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, 2013.

[10] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, "Wishbone: Profile-based partitioning for sensornet application," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.

[11] T. Szydlo, R. Brzoza-Woch, J. Sendorek, M. Windak, and C. Gniady, "Flow-based programming for IoT leveraging fog computing," in *IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2017.

[12] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvath, "EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020.

[13] Y. Lee, S. Cheon, D. Kim, D. Lee, and H. Kim, "ELASM: Error-latency-aware scale management for fully homomorphic encryption," in *32nd USENIX Security Symposium*, 2023.

[14] Y. Lee, S. Heo, S. Cheon, S. Jeong, C. Kim, E. Kim, D. Lee, and H. Kim, "HECATE: Performance-aware scale optimization for homomorphic encryption compiler," in *2022 IEEE/ACM International Symposium on Code Generation and Optimization*, pp.193-204, 2022.

[15] A. Viand, P. Jattke, M. Haller, and A. Hithnawi, "HECO: Fully homomorphic encryption compiler," in *32nd USENIX Security Symposium*, 2023.

[16] J. Duchi, M. Jordan, and M. Wainwright, " Local privacy and statistical minimax rates," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 2013.

[17] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Advances in Cryptology-Asiacrypt*, 2006.

[18] N. Holohan, D. Leith, and O. Mason, "Optimal differentially private mechanisms for randomised response," in *IEEE Transactions on Information Forensics and Security*, 2017.

[19] P. Kairouz, K. Bonawitz, and D. Ramage, "Discrete distribution estimatino under local privacy," in *International Conference on Machine Learning*, pp.2436-2444, 2016.

[20] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pp.486-498, 2008.

[21] A. Shamir, "How to Share a Secret," in *Commun*, 1979.

[22] A. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science*, 1986.

[23] A. Choudhury, J. Loftus, E. Orsini, A. Patra, and N. Smart, "Between a rock and a hard place: Interpolating between MPC and FHE," in *Advances in Cryptology-Asiacrypt*, 2013.

[24] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data" in *IEEE Transactions on Parallel and Distributed Systems*, pp.222-233, 2014.

[25] Z. Mo, Y. Qiao, and S. Chen, "Two-Party Fine-Grained Assured Deletion of Outsourced Data in Cloud Systems," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, 2014.

[26] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, 2013.

[27] Y. Tang, P. Lee, J. Lui, and R. Perlman, "Secure overlay cloud storage with access control and assured deletion," in *IEEE Transactions on Dependable and Secure Computing*, 2012.
[28] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," in *IEEE Transactions on Parallel and Distributed Systems*, pp.340-352, 2016.
[29] SEAL, Microsoft SEAL (release 3.5.9), 2020.
[30] J. Kim, M. Lee, A. Yun, and J. Cheon, "CRT-based fully homomorphic encryption over the integers," in *Cryptology ePrint*, 2013.

김 동 관
https://orcid.org/0000-0002-9611-0233
e-mail : dongkwan@yonsei.ac.kr
2020년 연세대학교 전기전자공학부(학사)
2010년 ~ 현 재 연세대학교 전기전자공학과 석 · 박사통합과정
관심분야 : Security & Compiler

이 용 우
https://orcid.org/0000-0002-7458-8885
e-mail : dragonrain96@yonsei.ac.kr
2019년 포항공과대학교 컴퓨터공학과(학사)
2019년 ~ 현 재 연세대학교 전기전자공학과 석 · 박사통합과정
관심분야 : Compiler & Homomorphic Encryption

천 선 영
https://orcid.org/0009-0005-3463-716X
e-mail : seonyoung@yonsei.ac.kr
2021년 연세대학교 전기전자공학부(학사)
2021년 ~ 현 재 연세대학교 전기전자공학과 석 · 박사통합과정
관심분야 : Homomorphic Encryption & Compiler & System Security

최 희 림
https://orcid.org/0000-0002-1885-0578
e-mail : heelim@yonsei.ac.kr
2020년 연세대학교 전기전자공학부(학사)
2020년 ~ 현 재 연세대학교 전기전자공학과 석 · 박사통합과정
관심분야 : Compiler & Processing-In-Memory

이 재 호
https://orcid.org/0000-0002-2735-0647
e-mail : jaeho@yonsei.ac.kr
2020년 연세대학교 전기전자공학부(학사)
2020년 ~ 현 재 연세대학교 전기전자공학과 석 · 박사통합과정
관심분야 : Memory-aware GPU Compiler

염 호 윤
https://orcid.org/0009-0005-6623-3995
e-mail : hoyun@yonsei.ac.kr
2024년 연세대학교 전기전자공학부(학사)
2024년 ~ 현 재 연세대학교 전기전자공학과 석 · 박사통합과정
관심분야 : Compiler & Homomorphic Encryption

김 한 준
https://orcid.org/0000-0002-0762-7901
e-mail : hanjun@yonsei.ac.kr
2007년 서울대학교 전자공학과(학사)
2009년 Princeton University Computer Science(석사)
2013년 Princeton University Computer Science(박사)
2013년 ~ 2018년 포항공과대학교 창의IT공학과, 컴퓨터공학과 조교수
2018년 포항공과대학교 창의IT공학과, 컴퓨터공학과 부교수
2018년 ~ 현 재 연세대학교 전기전자공학과 부교수
관심분야 : Computer Architecture & Compiler Optimization