Medha: **M**icrocoded **H**ardware **A**ccelerator for computing on **E**ncrypted **D**ata

Cryptographic Hardware and Embedded Systems (CHES), 2023

Ahmet Can Mert[1], **Aikata**[1], **Sunmin Kwon**[2], **Youngsam Shin**[2], **Donghoon Yoo**[2], **Yongwoo Lee**, **Sujoy Sinha Roy**[1]

[1] IAIK, Graz University of Technology, Graz, Austria
[2] Samsung Advanced Institute of Technology, Suwon, Republic of Korea

Outline

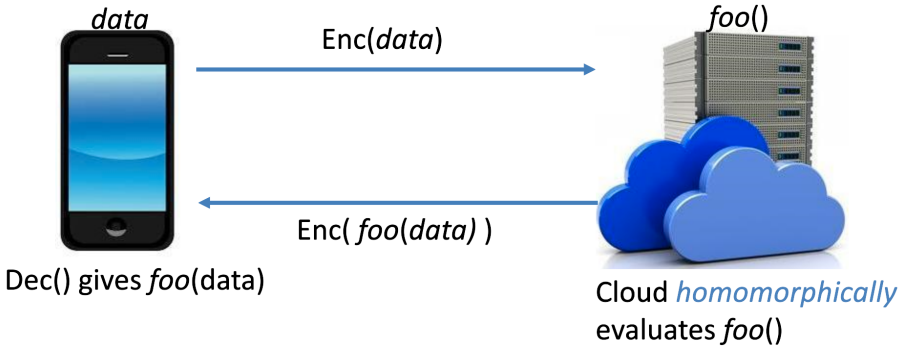
1. Motivation and Background
2. Microcoded Hardware Accelerator
   - Architecture of homomorphic processor
   - FPGA implementation with placement-friendly layout
3. Evaluation Results

Outline

1. **Motivation and Background**
2. Microcoded Hardware Accelerator
   - Architecture of homomorphic processor
   - FPGA implementation with placement-friendly layout
3. Evaluation Results

Homomorphic Encryption

Homomorphic Encryption (HE) allows computation on the encrypted data.



Challenges in accelerating Homomorphic Encryption

1. Computationally intensive:

10^4 to $10^5\times$ plain computation

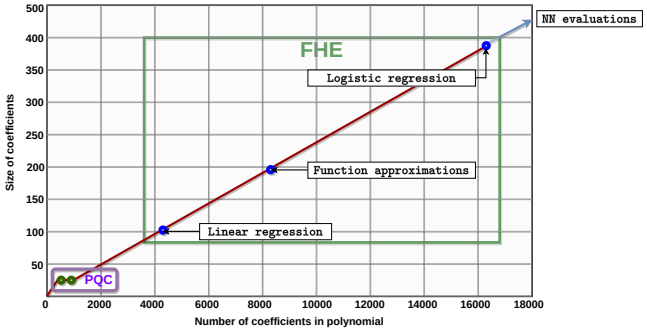
- Large polynomial arithmetic
- Long integer arithmetic

Challenges in accelerating Homomorphic Encryption

1. Computationally intensive:
10^4 to $10^5\times$ plain computation
   - Large polynomial arithmetic
   - Long integer arithmetic
2. Ciphertexts are several MBs

Challenges in accelerating Homomorphic Encryption

1. Computationally intensive:
   10^4 to $10^5\times$ plain computation
   - Large polynomial arithmetic
   - Long integer arithmetic
2. Ciphertexts are several MBs



Challenges in accelerating Homomorphic Encryption

1. Computationally intensive:
10^5 to $10^4\times$ plain computation
   - Large polynomial arithmetic $\leftarrow$ NTT
   - Long integer arithmetic
2. Ciphertexts are several MBs

$$\mathbb{O}(N^2) \rightarrow \mathbb{O}(N \log_2 N)$$

Challenges in accelerating Homomorphic Encryption

1. Computationally intensive:
10^5 to $10^4\times$ plain computation
   - Large polynomial arithmetic
   - Long integer arithmetic $\leftarrow$ RNS
2. Ciphertexts are several MBs

$$\boxed{Q \rightarrow \prod_{i=0}^{L-1} q_i}$$

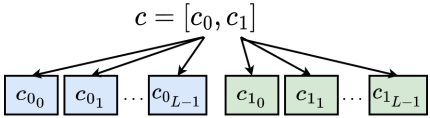
RNS-CKKS Scheme

RNS-CKKS Scheme

- Enables homomorphic computations for real numbers.

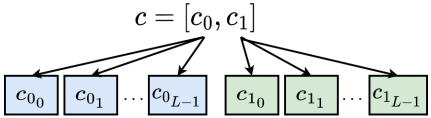
RNS-CKKS Scheme

- Enables homomorphic computations for real numbers.



RNS-CKKS Scheme

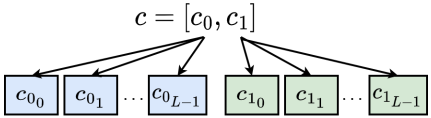
- Enables homomorphic computations for real numbers.



- Homomorphic operations:
  - Addition: $2L$ additions

RNS-CKKS Scheme

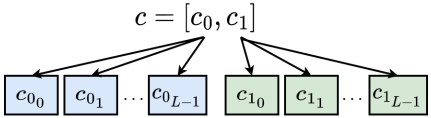
- Enables homomorphic computations for real numbers.



- Homomorphic operations:
  - Addition: $2L$ additions
  - Multiplication: $4L$ multiplications, L additions

RNS-CKKS Scheme

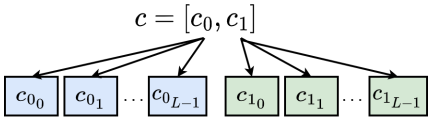
- Enables homomorphic computations for real numbers.



- Homomorphic operations:
  - Addition: $2L$ additions
  - Multiplication: $4L$ multiplications, L additions
  - Relinearization: $L(L+1)$ base conversions, $2L(L+1)$ multiplications

RNS-CKKS Scheme

- Enables homomorphic computations for real numbers.



- Homomorphic operations:
  - Addition: $2L$ additions
  - Multiplication: $4L$ multiplications, L additions
  - Relinearization: $L(L+1)$ base conversions, $2L(L+1)$ multiplications
    (Base conversion: NTT $\rightarrow q_i$ to $q_j \rightarrow$ INTT)

Challenges accelerating Homomorphic Encryption with HW

(Xilinx Alveo U250 FPGA)

Challenges accelerating Homomorphic Encryption with HW

(Xilinx Alveo U250 FPGA)

1. Computationally intensive
   - Scheduling operations
   - Target platform constraints

Challenges accelerating Homomorphic Encryption with HW

(Xilinx Alveo U250 FPGA)

1. Computationally intensive
   - Scheduling operations
   - Target platform constraints
2. Ciphertexts are several MBs
   - On-chip memory is limited
   - Off-chip transfer is very slow

Challenges accelerating Homomorphic Encryption with HW

(Xilinx Alveo U250 FPGA)

1. Computationally intensive
   - Scheduling operations
   - Target platform constraints
2. Ciphertexts are several MBs
   - On-chip memory is limited
   - Off-chip transfer is very slow

Our Solution: **Medha**

Outline

1. Motivation and Background
2. **Microcoded Hardware Accelerator**
   - Architecture of homomorphic processor
   - FPGA implementation with placement-friendly layout
3. Evaluation Results

An Overview of HW-based HE accelerators in the literature

Two main tracks:

1. Accelerator prototypes in FPGA/ASIC
2. Simulation model of accelerator

Accelerator prototypes in FPGA/ASIC
HEAWS[TRV20], HEAX[RLPD20], CoFHEE[NSA+23]

Simulation model of accelerator
F1[FSK+21], BTS[KKK+22], CraterLake[SFK+22]

[TRV20] Furkan Turan et al. HEAWS: an accelerator for homomorphic encryption on the amazon AWS FPGA. IEEE ToC, 2020.
[RLPD20] M. Sadegh Riazi et al. HEAX: an architecture for computing on encrypted data. ASPLOS 2020.
[NSA+23] Mohammed Nabeel et al. CoFHEE: A Co-processor for Fully Homomorphic Encryption Execution. DATE 2023.
[FSK+21] Axel Feldmann et al. F1: A fast and programmable accelerator for fully homomorphic encryption. MICRO 2021.
[KKK+22] Sangpyo Kim et al. BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption. ISCA 2022.
[SFK+22] Samardzic et al. CraterLake: A Hardware Accelerator for Efficient Unbounded Computation on Encrypted Data. ISCA 2022.

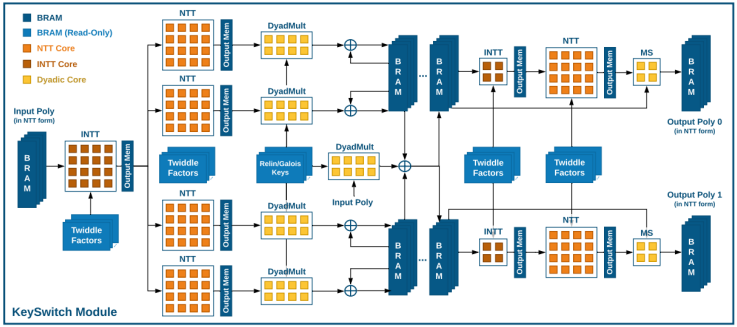
An Overview of HW-based HE accelerators in the literature

- HEAX[RLPD20] follows a block-pipelined architecture.

[RLPD20] M. Sadegh Riazi et al. HEAX: an architecture for computing on encrypted data. ASPLOS 2020.

An Overview of HW-based HE accelerators in the literature

- HEAX[RLPD20] follows a block-pipelined architecture.
  - Sub-routine specific for HE



[RLPD20] M. Sadegh Riazi et al. HEAX: an architecture for computing on encrypted data. ASPLOS 2020.

Design Goals

Design Goals

1. A programmable hardware accelerator architecture for RNS-CKKS

Design Goals

1. A programmable hardware accelerator architecture for RNS-CKKS
    - Low-latency oriented design with flexibility

Design Goals

1. A programmable hardware accelerator architecture for RNS-CKKS
   - Low-latency oriented design with flexibility
   - Supporting $N = 2^{14}$ and $N = 2^{15}$ with RNS moduli sizes 54/60-bit

Design Goals

1. A programmable hardware accelerator architecture for RNS-CKKS
   - Low-latency oriented design with flexibility
   - Supporting $N = 2^{14}$ and $N = 2^{15}$ with RNS moduli sizes 54/60-bit
2. Verification and benchmarking on real FPGA (Xilinx Alveo U250 card)

Design Goals

1. A programmable hardware accelerator architecture for RNS-CKKS
   - Low-latency oriented design with flexibility
   - Supporting $N = 2^{14}$ and $N = 2^{15}$ with RNS moduli sizes 54/60-bit
2. Verification and benchmarking on real FPGA (Xilinx Alveo U250 card)

Medha: Microcoded Hardware Accelerator for computing on Encrypted Data

Outline

1. Motivation and Background
2. Microcoded Hardware Accelerator
   - **Architecture of homomorphic processor**
   - FPGA implementation with placement-friendly layout
3. Evaluation Results

Architecture of the Homomorphic Processor

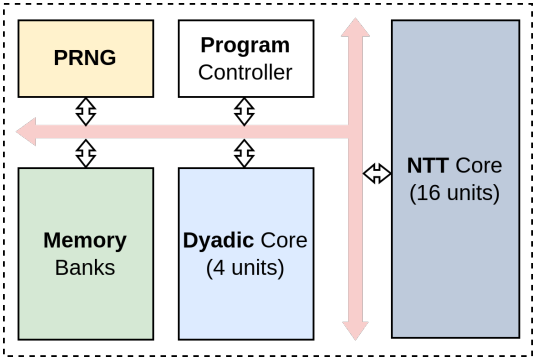
We design a *flexible instruction-set architecture* for each RNS base:

- Each RNS base $\rightarrow$ One processing element (RPAU)

Architecture of the Homomorphic Processor

We design a *flexible instruction-set architecture* for each RNS base:

- Each RNS base $\rightarrow$ One processing element (RPAU)



Architecture of the Homomorphic Processor

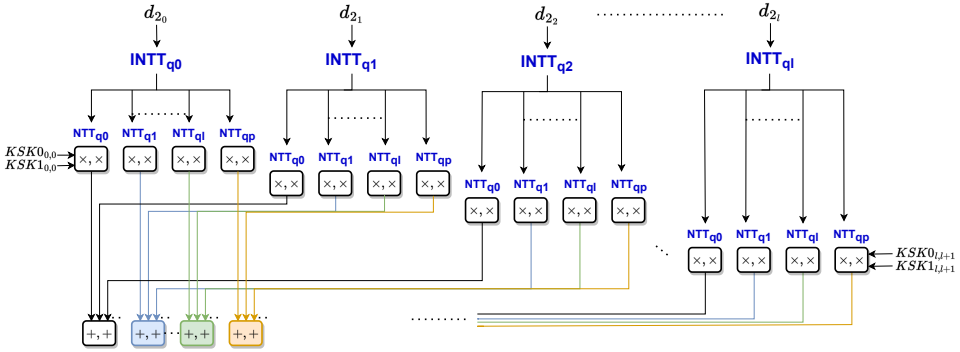
We use two cores, NTT and Dyadic, to speedup relineatization operation.

- NTT core *(for base conversion)*
- Dyadic core *(for multiplying with keys)*

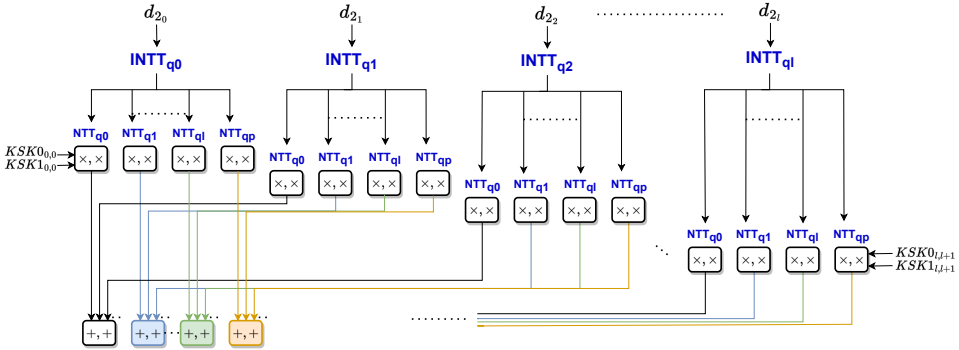
Architecture of the Homomorphic Processor

We use two cores, NTT and Dyadic, to speedup relineatization operation.

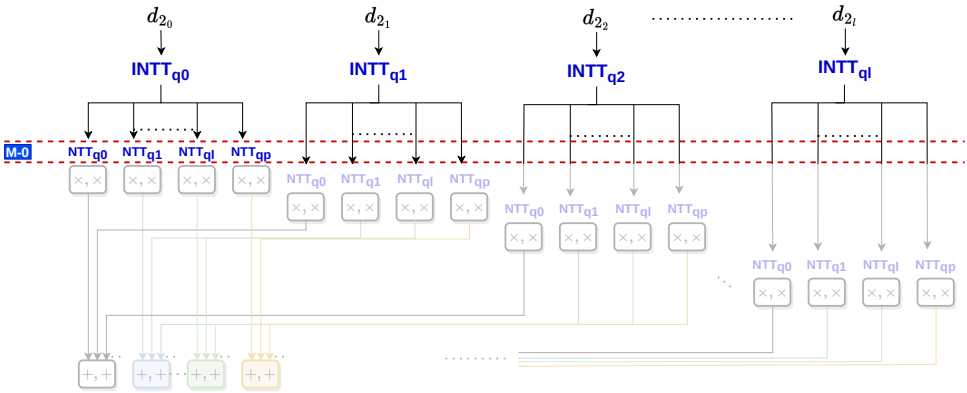
- NTT core *(for base conversion)*
- Dyadic core *(for multiplying with keys)*



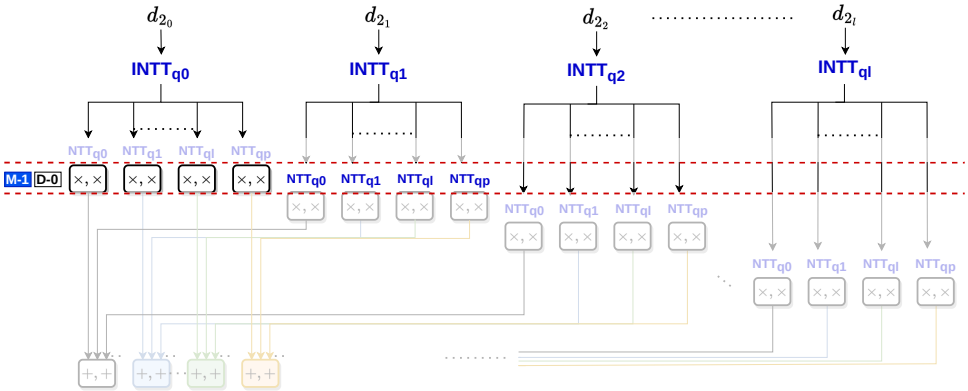
Architecture of the Homomorphic Processor



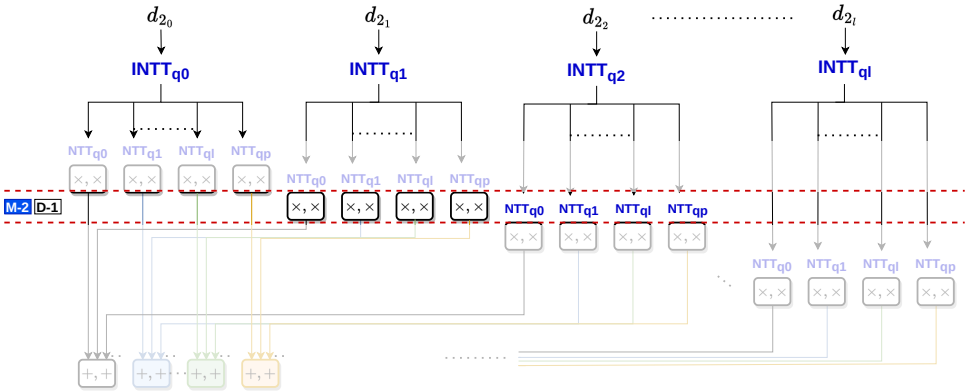
Architecture of the Homomorphic Processor



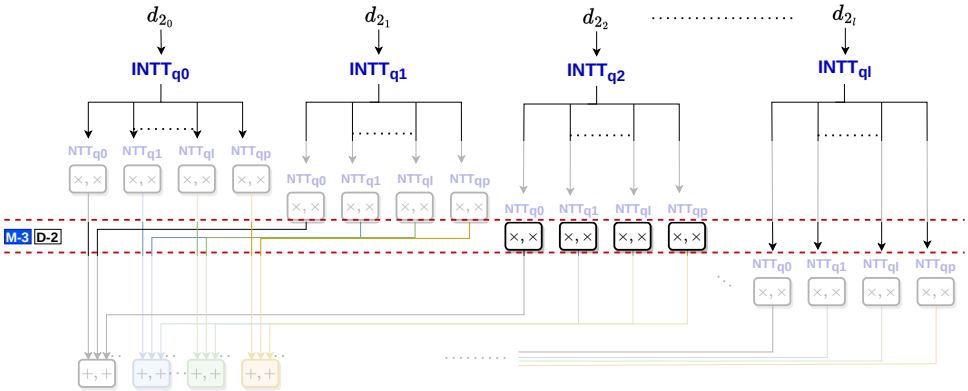
Architecture of the Homomorphic Processor



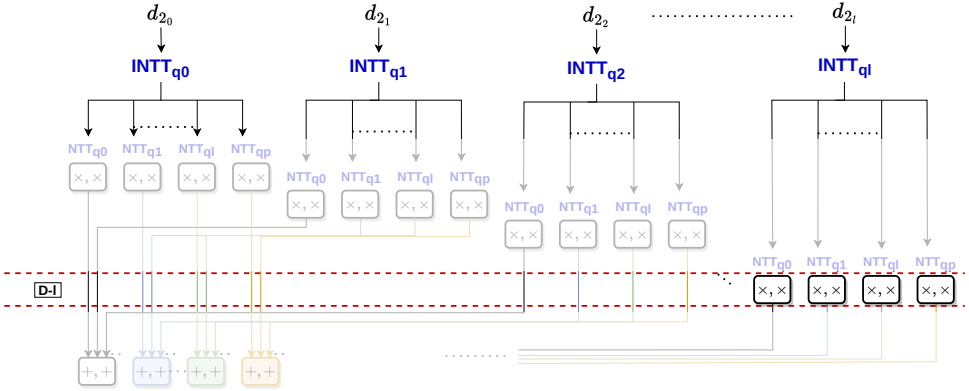
Architecture of the Homomorphic Processor



Architecture of the Homomorphic Processor



Architecture of the Homomorphic Processor



Parallel execution of NTT & dyadic cores results in significant cycle reduction!

Customized on-chip memory design

Customized on-chip memory design

Utilizing left-over bits in BRAM/URAM

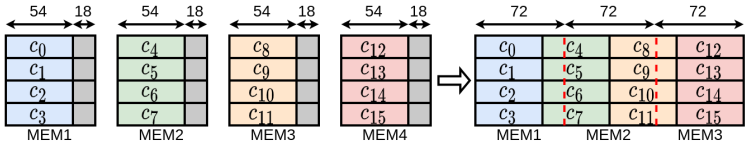
- One URAM address can store 72-bits
- One BRAM address can store 18/36/72-bits
- We use 54-bits RNS bases

Utilizing left-over bits in BRAM/URAM

- One URAM address can store 72-bits
- One BRAM address can store 18/36/72-bits
- We use 54-bits RNS bases

We created a virtual memory to utilize left-over bits in BRAM/URAM.

- Example: 54-bit coefficient storage in URAM



Outline

1. Motivation and Background
2. Microcoded Hardware Accelerator
   - Architecture of homomorphic processor
   - **FPGA implementation with placement-friendly layout**
3. Evaluation Results

Placement-friendly Layout

Placement-friendly Layout

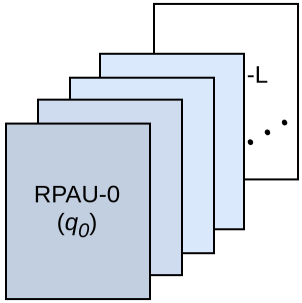
1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial

Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.

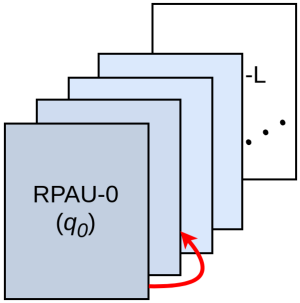
Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.



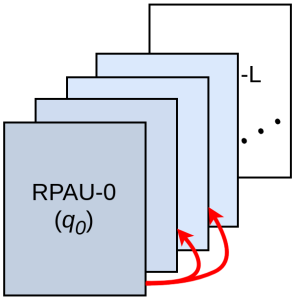
Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.



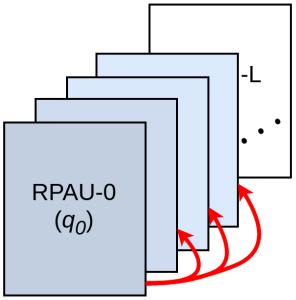
Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.



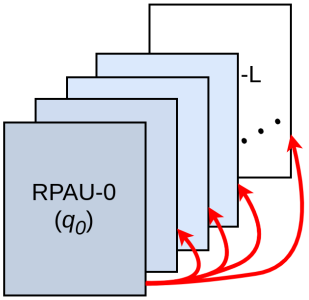
Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.



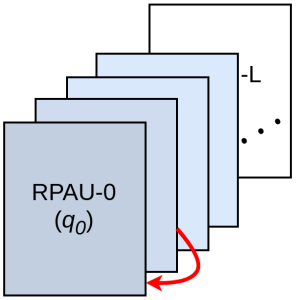
Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.



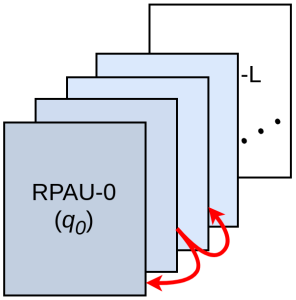
Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.



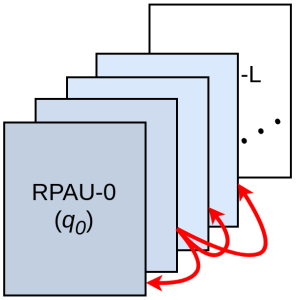
Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.



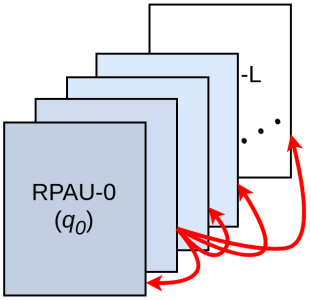
Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.



Placement-friendly Layout

1. Each RNS base arithmetic is implemented by one processing element.
   - Each processing element stores its residue polynomial
   - Relinearization operation requires polynomials to be exchanged between RPAUs.



Placement-friendly Layout

2. Target platform (Xilinx Alveo U250 FPGA) constraints

Image is retrieved from https://docs.xilinx.com/v/u/en-US/wp380_Stacked_Silicon_Interconnect_Technology

2. Target platform (Xilinx Alveo U250 FPGA) constraints

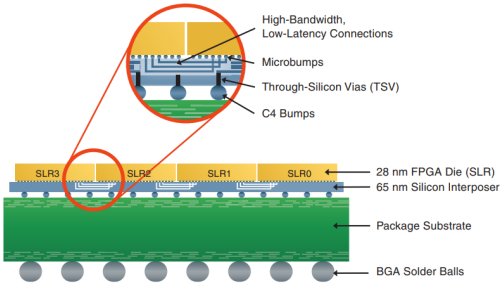
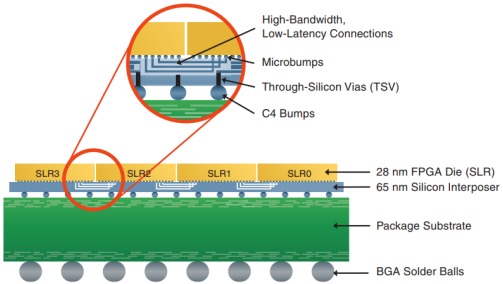


Image is retrieved from `https://docs.xilinx.com/v/u/en-US/wp380_Stacked_Silicon_Interconnect_Technology`

Placement-friendly Layout

2. Target platform (Xilinx Alveo U250 FPGA) constraints



- Two neighboring SLRs are connected using a limited number of wires.

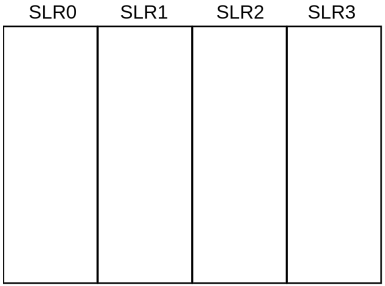
Image is retrieved from `https://docs.xilinx.com/v/u/en-US/wp380_Stacked_Silicon_Interconnect_Technology`

Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!

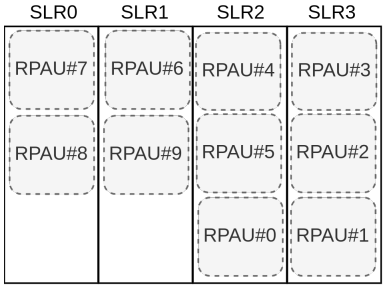
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



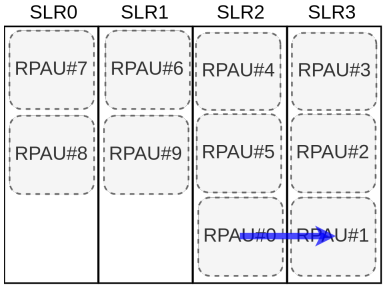
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



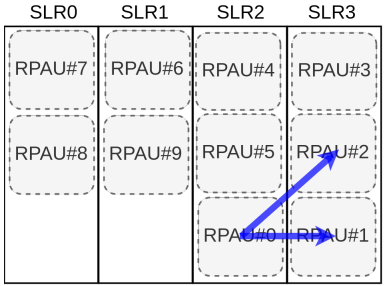
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



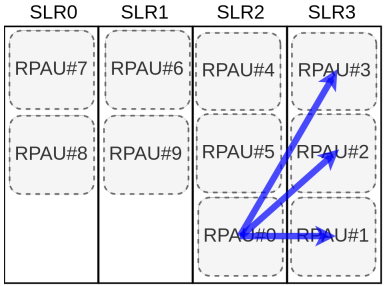
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



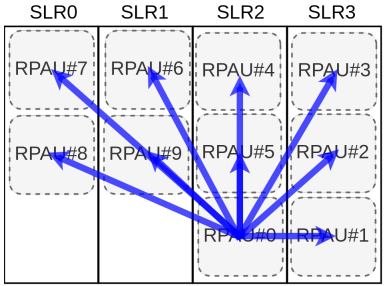
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



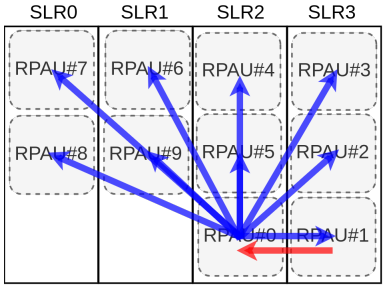
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



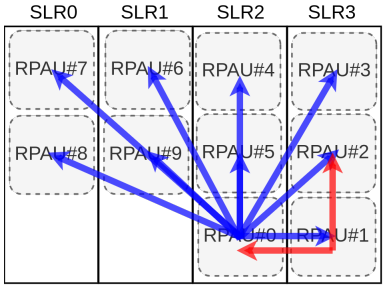
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



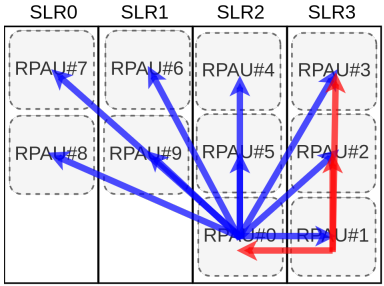
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



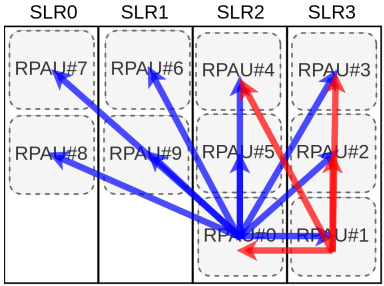
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



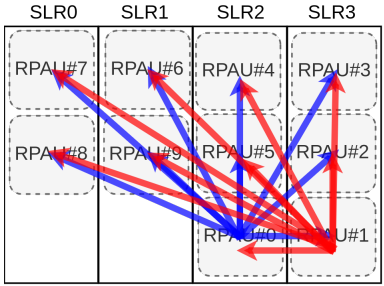
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



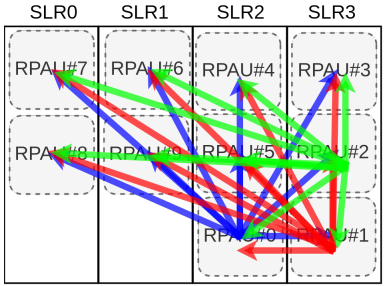
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



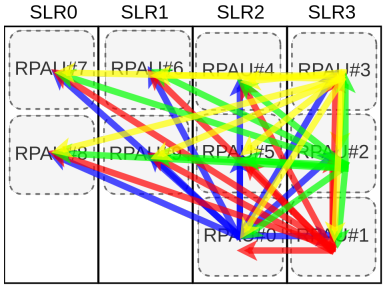
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



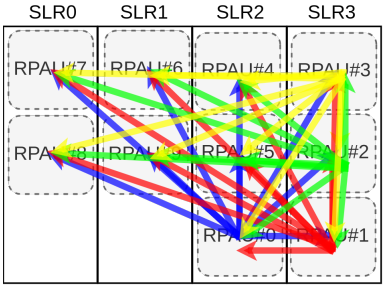
Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



Placement-friendly Layout

Placing/Interconnecting RPAUs is a huge engineering challenge!



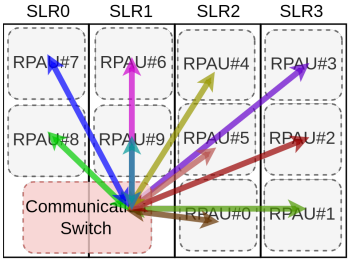
We need an efficient method to interconnect and map RPAUs!

Placement-friendly Layout

We must consider SLR-to-SLR connection constraints.

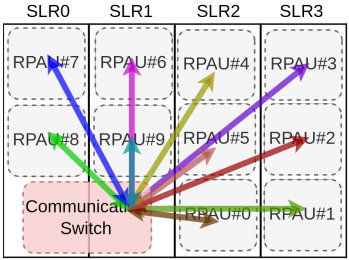
Placement-friendly Layout

We must consider SLR-to-SLR connection constraints.



Placement-friendly Layout

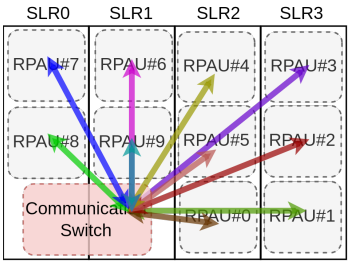
We must consider SLR-to-SLR connection constraints.

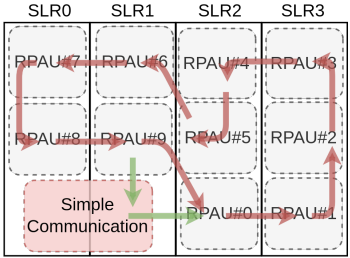


- Wiring complexity: $\mathbb{O}(L^2) \rightarrow \mathbb{O}(L)$
- Still many SLR-crossing nets

Placement-friendly Layout

We must consider SLR-to-SLR connection constraints.

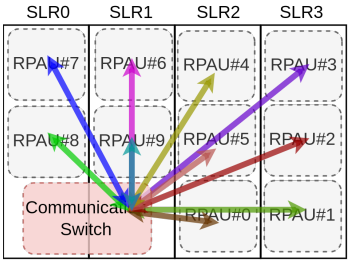




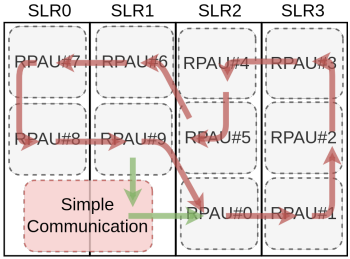
- Wiring complexity: $\mathbb{O}(L^2) \rightarrow \mathbb{O}(L)$
- Still many SLR-crossing nets

Placement-friendly Layout

We must consider SLR-to-SLR connection constraints.



- Wiring complexity: $\mathbb{O}(L^2) \rightarrow \mathbb{O}(L)$
- Still many SLR-crossing nets



- Only neighbouring RPAUs connected
- Few SLR-crossing nets

Outline

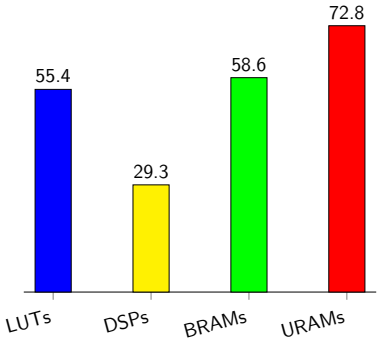
1. Motivation and Background
2. Microcoded Hardware Accelerator
   - Architecture of homomorphic processor
   - FPGA implementation with placement-friendly layout
3. **Evaluation Results**

Evaluation Results

As a proof of concept, our implementation employs 10 PEs and two parameter sets.

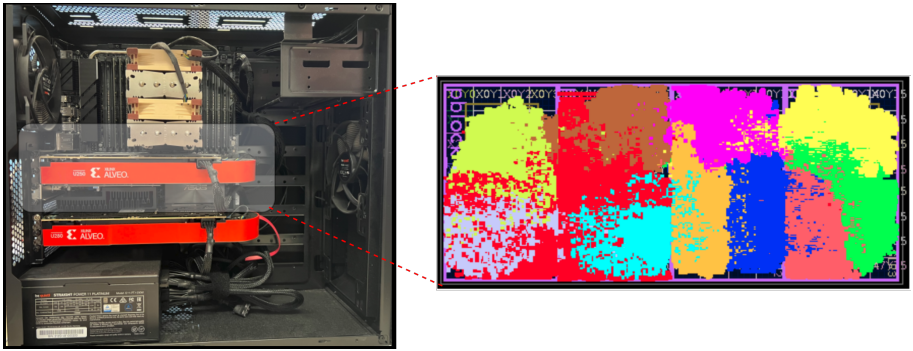
- **Set-1:** $\log_2(pQ) = 438,\ N = 2^{14}$
- **Set-2:** $\log_2(pQ) = 546,\ N = 2^{15}$

Resource utilization on Xilinx Alveo U250 FPGA (in %)



Evaluation Results

Placement of Medha on the Alveo U250 FPGA chip.



Evaluation Results

Performance on Alveo U250 board running at 200 MHz.

[SEA20] Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.
[RLPD20] M. Sadegh Riazi et al. HEAX: an architecture for computing on encrypted data. ASPLOS 2020.

Evaluation Results

Performance on Alveo U250 board running at 200 MHz.

- Homomorphic operation benchmark
  - Hom. Mult. + Relin. for Set-1: **497μ sec**
  - Hom. Mult. + Relin. for Set-2: **1,374μ sec**

[SEA20] Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.
[RLPD20] M. Sadegh Riazi et al. HEAX: an architecture for computing on encrypted data. ASPLOS 2020.

Evaluation Results

Performance on Alveo U250 board running at 200 MHz.

- Homomorphic operation benchmark
  - Hom. Mult. + Relin. for Set-1: **497**μ **sec**
  - Hom. Mult. + Relin. for Set-2: **1,374**μ **sec**
- End-to-end application benchmark (logistic regression)
  - **64**$\times$ speedup compared to the implementation in SEAL [SEA20]

[SEA20] Microsoft SEAL (release 3.6). `https://github.com/Microsoft/SEAL`, November 2020. Microsoft Research, Redmond, WA.
[RLPD20] M. Sadegh Riazi et al. HEAX: an architecture for computing on encrypted data. ASPLOS 2020.

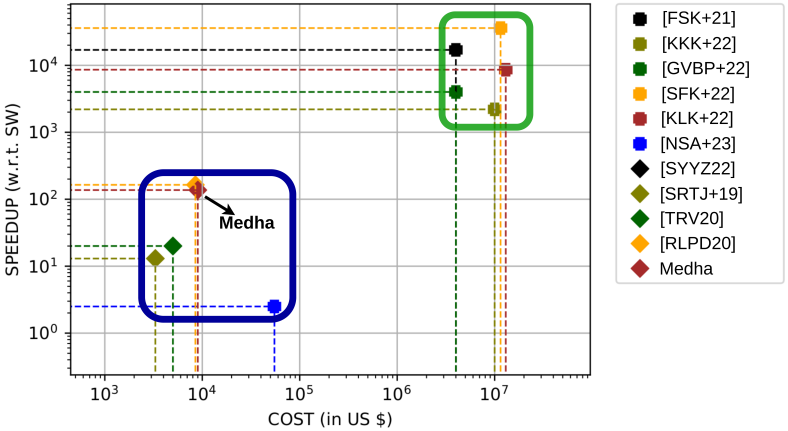
Evaluation Results

Performance on Alveo U250 board running at 200 MHz.

- Homomorphic operation benchmark
  - Hom. Mult. + Relin. for Set-1: **497**μ **sec**
  - Hom. Mult. + Relin. for Set-2: **1,374**μ **sec**
- End-to-end application benchmark (logistic regression)
  - **64**$\times$ speedup compared to the implementation in SEAL [SEA20]
- Comparison with HEAX [RLPD20]
  - **2.3**$\times$ better latency

[SEA20] Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.
[RLPD20] M. Sadegh Riazi et al. HEAX: an architecture for computing on encrypted data. ASPLOS 2020.

Evaluation Results



Medha: **M**icrocoded **H**ardware **A**ccelerator for computing on **E**ncrypted **D**ata

Cryptographic Hardware and Embedded Systems (CHES), 2023

Ahmet Can Mert[1], **Aikata**[1], **Sunmin Kwon**[2], **Youngsam Shin**[2], **Donghoon Yoo**[2], **Yongwoo Lee**, **Sujoy Sinha Roy**[1]

[1] IAIK, Graz University of Technology, Graz, Austria
[2] Samsung Advanced Institute of Technology, Suwon, Republic of Korea