F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption

Axel Feldmann*, Nikola Samardzic*, Aleksandar Krastev,
Srini Devadas, Ron Dreslinski, Chris Peikert, Daniel Sanchez

9/21/2021

Massachusetts Institute of Technology

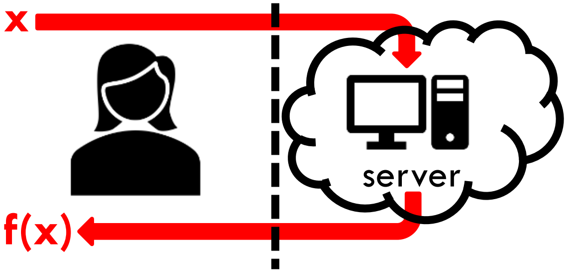
CSAIL

The University of Michigan

* Authors contributed equally

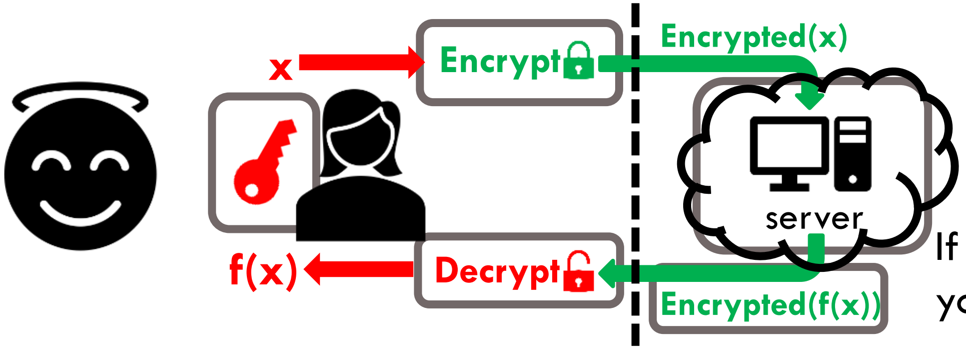
Overview

- A lot of modern software runs in the cloud



Problem: the cloud's vulnerabilities become **your** vulnerabilities

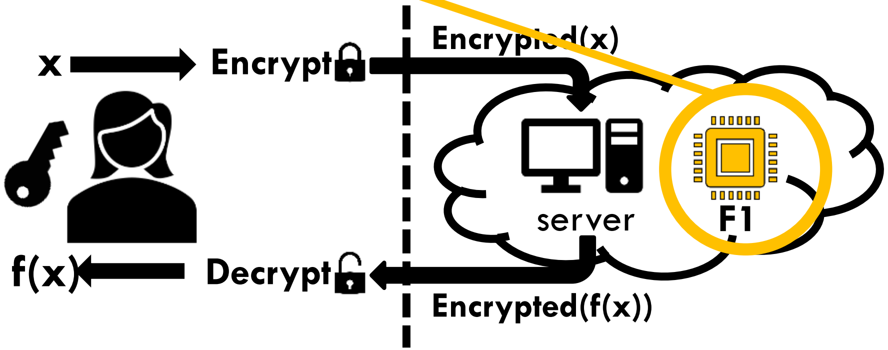
- FHE enables computation on encrypted data



If someone hacks the cloud, your data is safe!

Fully Homomorphic Encryption (FHE)

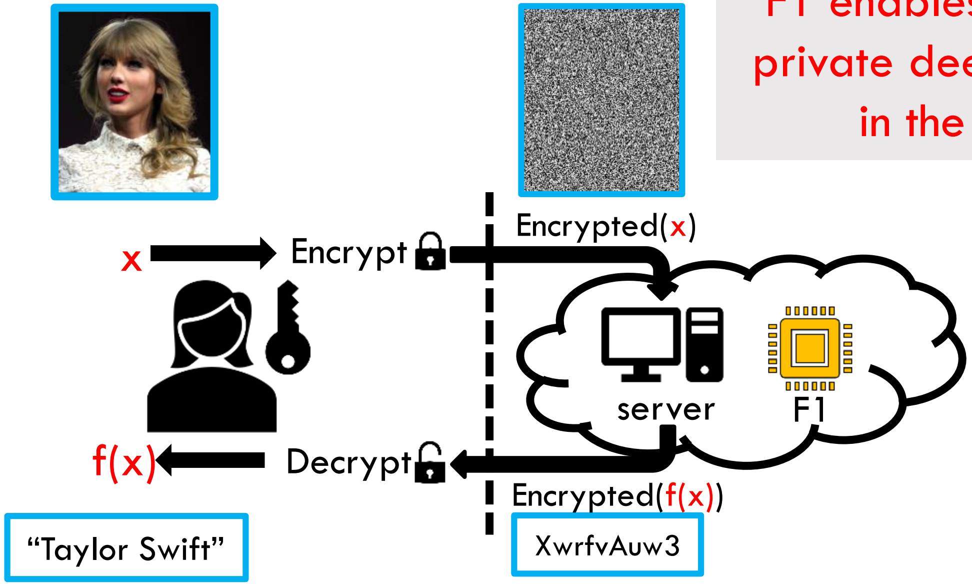
- FHE is a cryptographic system that allows us to *computation on encrypted data*
  - It allows **arithmetic operations on encrypted vectors**
  - FHE is expressive enough to implement neural network, logistic regression, etc.
- FHE computation is **10,000x slower than unencrypted computation**
- Let's accelerate it with **F1**



Ex: Private Deep Learning In the Cloud

- Use case: inference too expensive to do on the client; data must remain private; model is too large
- State of the art: **20 minutes** per encrypted DNN inference
- **F1** reduces this to ***250 milliseconds***

F1 enables real-time private deep learning in the cloud

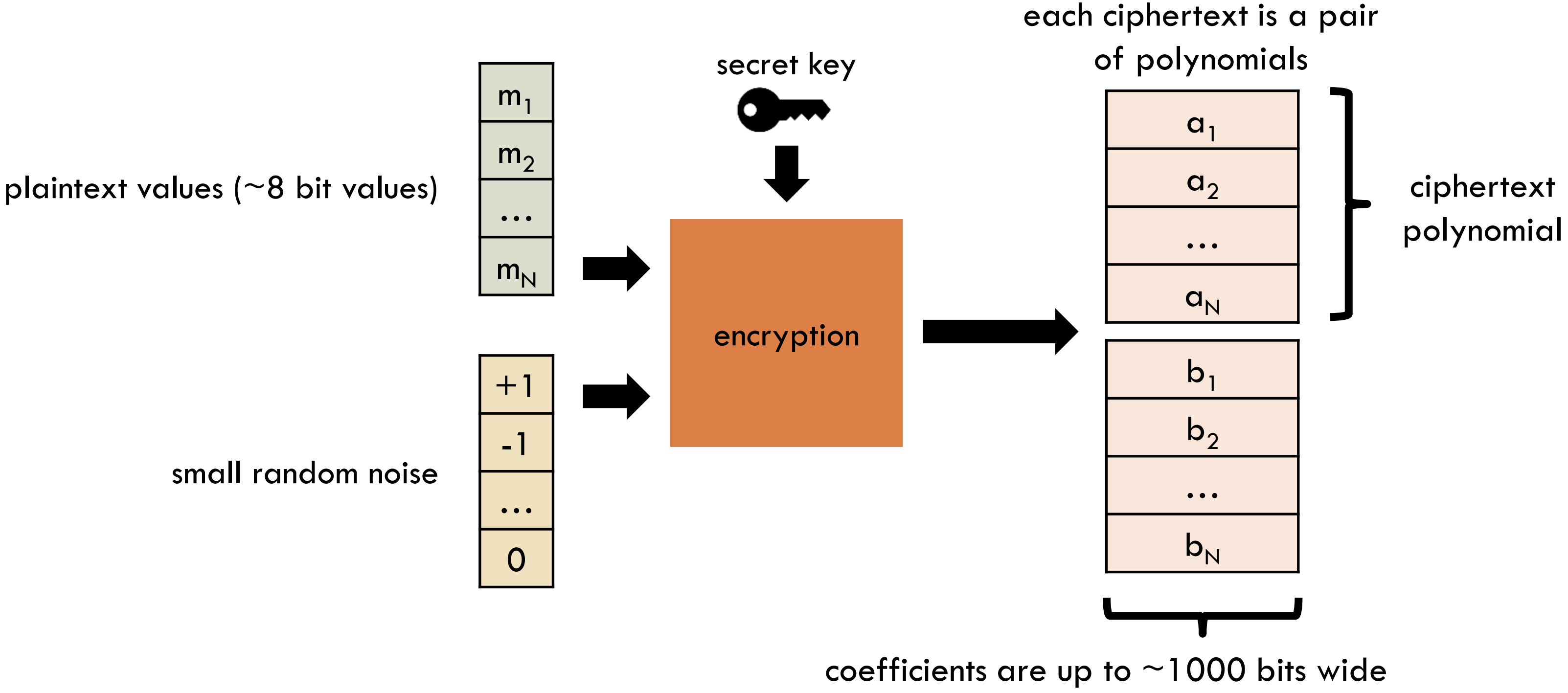


Agenda

- **Overview of FHE computations**
- Architectural characterization of FHE
- F1 design
- Evaluation and results

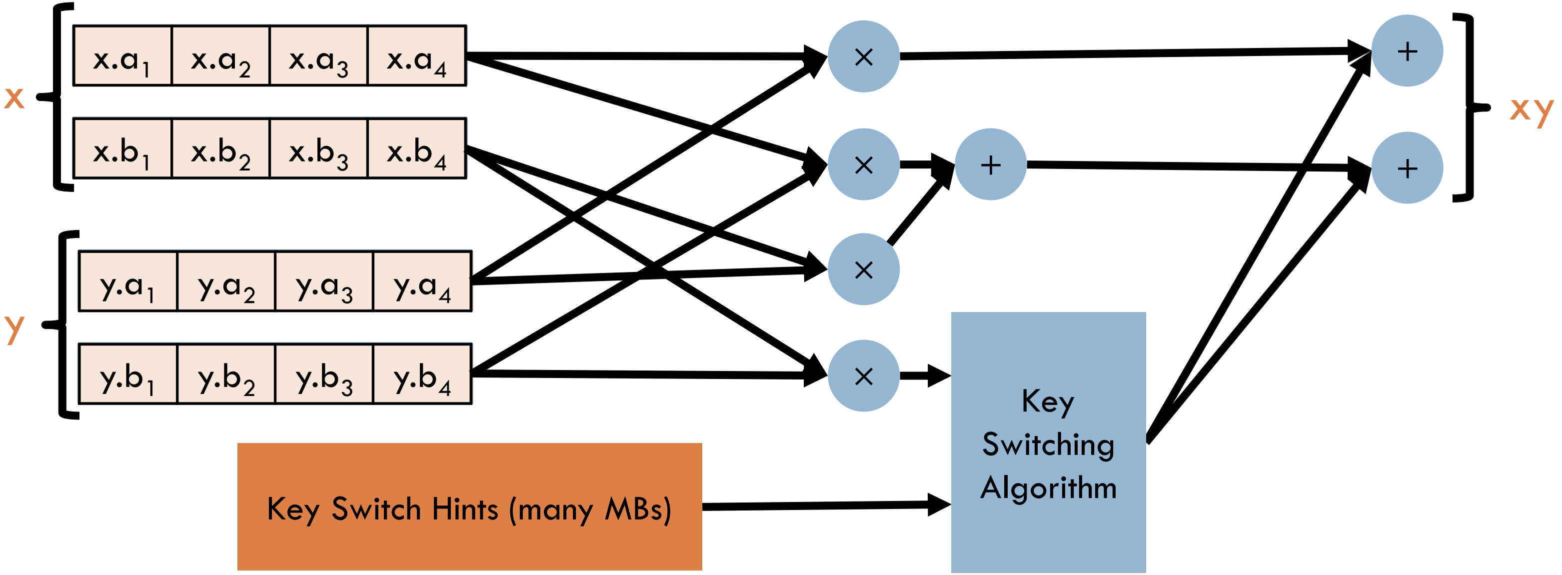
Encryption – Data Types

- Plaintext vectors are encrypted into pairs of **polynomials**
  - Polynomials are represented as vectors of coefficients



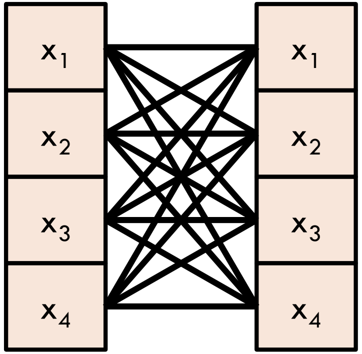
FHE Operations

- By computing on the ciphertext polynomials, FHE allows us to **add**, **multiply**, and **rotate** the underlying values
  - Operations on ciphertexts are often quite complex
  - Example: to multiply two ciphertexts x and y:



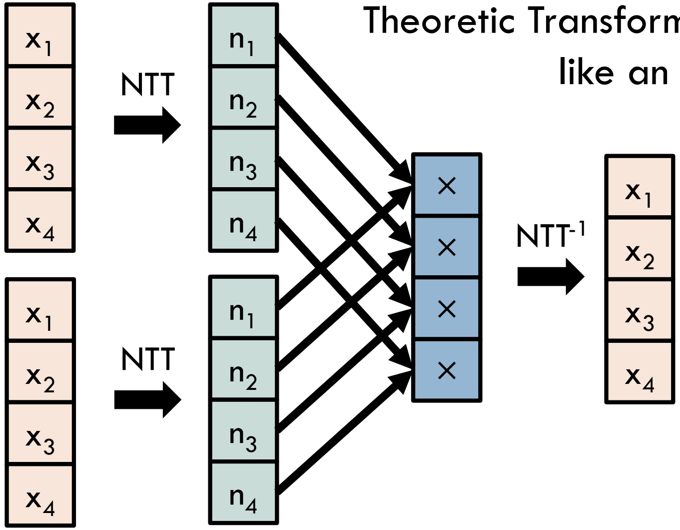
Multiplying Polynomials

- We often need to multiply polynomials



Naively, this takes $O(n^2)$ multiplications

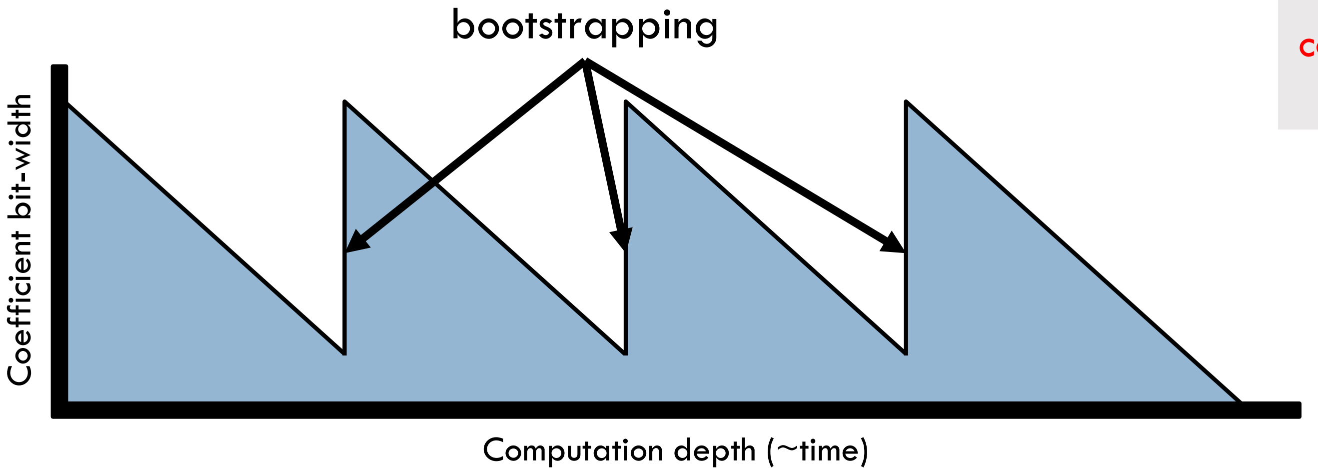
Instead, we can use the Number Theoretic Transform (NTT) which is like an FFT



NTTs and NTT^{-1} each take $O(n\log n)$ multiplies, making the whole operation $O(n\log n)$

Rough Shape of FHE Programs

- Ciphertexts start with some initial noise and coefficient width
- As we compute on them, they become noisier, and we chop off the noise, also reducing the coefficient width
- **Bootstrapping** is an expensive procedure to refresh ciphertexts



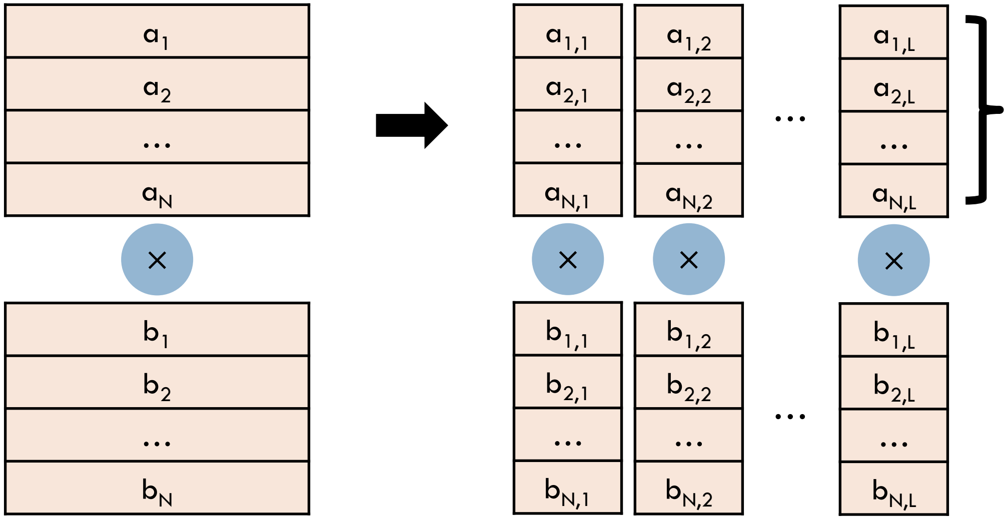
We must perform computation at multiple bit-widths!

Wide Arithmetic Using RNS

- Problem: our polynomial coefficients are extremely wide (up to ~1000 bits)
  - We also need to support computation on narrower ones

Advantage: we can perform arbitrarily wide modular arithmetic with 32-bit multipliers

Residue Number System: we can represent a single wide polynomial modulo some large Q as L many polynomials each mod a smaller q_i where $q_1 q_2 \ldots q_L = Q$



Residue polynomials are F1's primitive datatype

Agenda

- Overview of FHE computations
- **Architectural characterization of FHE**
- F1 design
- Evaluation and results

Architectual Characteristics of FHE

- FHE enables many algorithms on encrypted data, not just a single application
- Homomorphic operations all rely on big polynomial arithmetic
- Ciphertexts are large (some are many MBs), so data movement is extremely important
- Dataflow is completely static

Accelerating Whole FHE Programs

- <u>FHE enables many algorithms on encrypted data, not just a single application</u>

- FUs need to be flexible enough to support various polynomial sizes and coefficient widths

- Needs to support all possible FHE programs
  - General, without sacrificing performance

- Prior work only accelerates some FHE operations

Polynomial Arithmetic Primitives

- Homomorphic operations all rely on big polynomial arithmetic

- F1 accelerates polynomial arithmetic primitives
  - We support various FHE schemes
  - Within FHE schemes, we support multiple implementations of ciphertext operations

- Prior work builds overspecialized pipelines

Large Ciphertexts

- <u>Ciphertexts are large (some are many MBs), so data movement is extremely important</u>

- We need a large scratchpad with decoupled loads

- Operand size limits parallelism
  - Only a small number of operands fit on chip at any time
  - **FU latency is critical**

- Prior work targets FPGAs with limited compute, bypassing data movement problems

Static Dataflow

- <u>Dataflow is completely static</u>

```
if (x > 37):
    do something
```

Branching is **not possible.** We are computing on **encrypted data.**
If we can't decrypt x, we can't branch on it!

- We can avoid expensive scheduling hardware

- Decoupling loads is easier

F1 vs. Prior Work

Prior Work	F1
Built for FPGAs → Limited compute, ignore data movement bottlenecks	Targets ASICs → Designed to minimize off-chip data movement
Accelerate only some FHE operations, defer others to a host processor	Accelerates all FHE operations
Build overspecialized pipelines with simple FUs → Hinder algorithmic diversity	Accelerates *primitive operations* with high throughput FUs

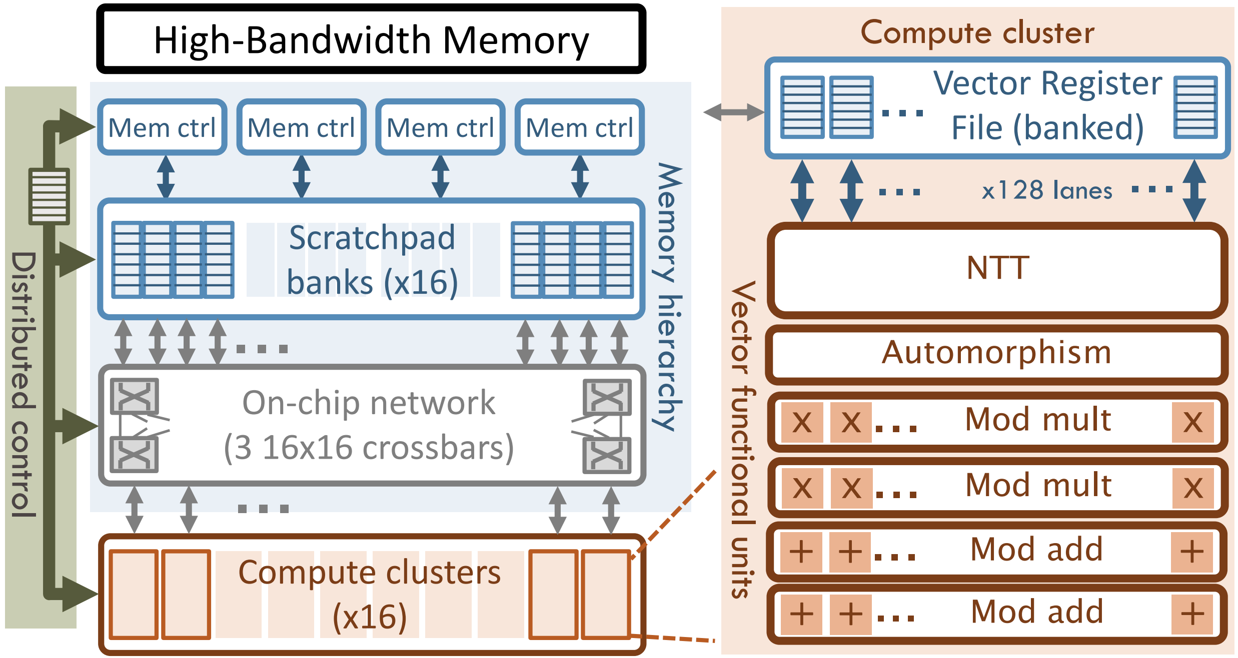
Prior work:

- M Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. HEAX: An architecture for computing on encrypted data. In *Proceedings of the 25th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XXV).*
- Sujoy Sinha Roy, Furkan Turan, Kimmo Järvinen, Frederik Vercauteren, and Ingrid Verbauwhede. 2019. FPGA-Based High-Performance Parallel Architecture for Homomorphic Computing on Encrypted Data. In *Proceedings of the 25th IEEE international symposium on High Performance Computer Architecture (HPCA-25).*
- Vincent Migliore, Cédric Seguin, Maria Mendez Real, Vianney Lapotre, Arnaud Tisserand, Caroline Fontaine, Guy Gogniat, and Russell Tessier. 2017. A High-Speed Accelerator for Homomorphic Encryption using the Karatsuba Algorithm. *ACM Trans. Embedded Comput. Syst.* 16, 5s (2017), 138:1ś138:17.

Agenda

- Overview of FHE computations
- Architectural characterization of FHE
- **F1 design**
- Evaluation and results

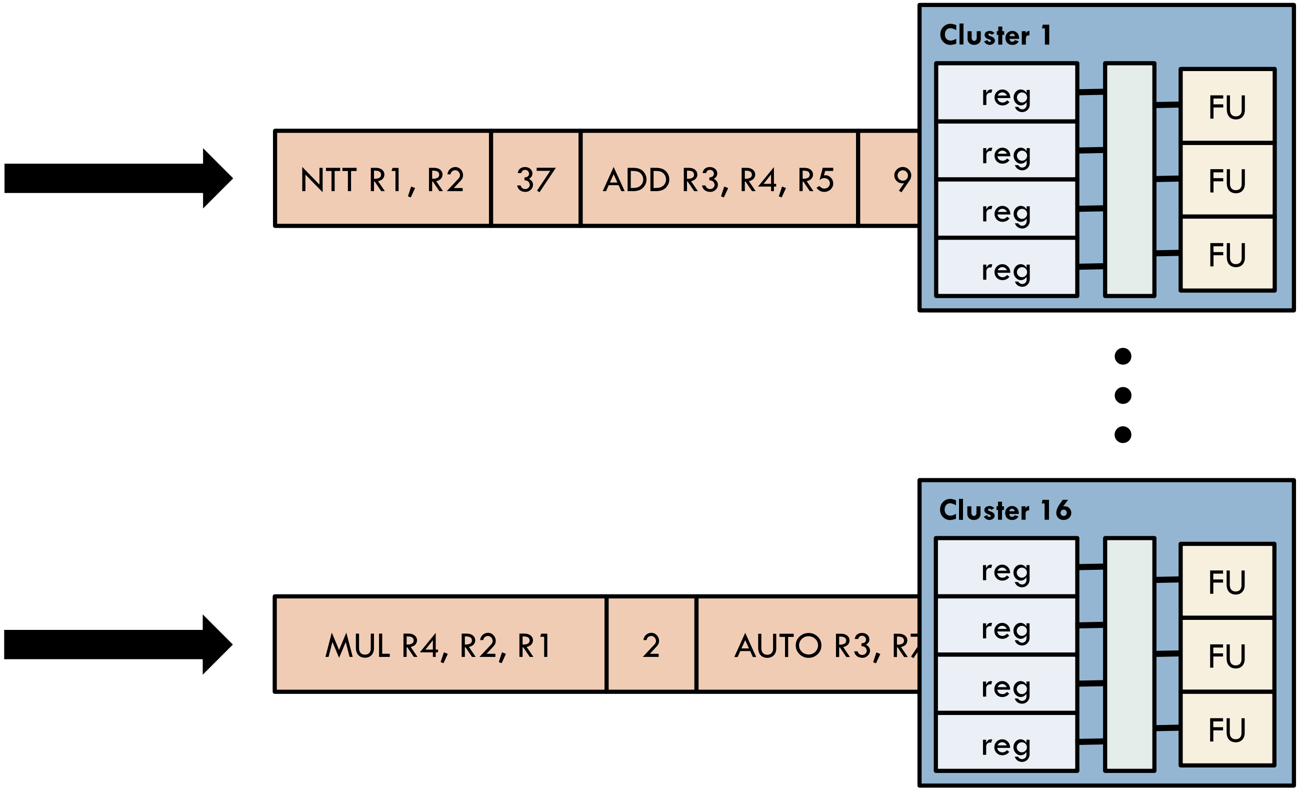
F1 Architecture Overview



- 64 MB scratchpad
- 16 clusters
- 1 TB/s HBM2

Decentralized Control

- Each compute cluster has its own independent instruction stream

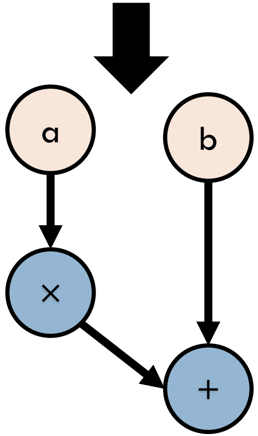


Static Scheduling

- We design and implement a complete software stack that compiles a simple DSL to F1 instructions

```
a = Ciphertext()
b = Ciphertext()

c = a * a + b
```



- FHE programs are static dataflow graphs
  - All dependences are precisely known at compile-time

- We use an explicitly managed memory hierarchy
  - Data is fetched ahead-of-time and replaced using an approximation of Bélády's Min

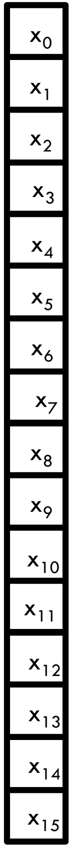
Scheduling Algorithms

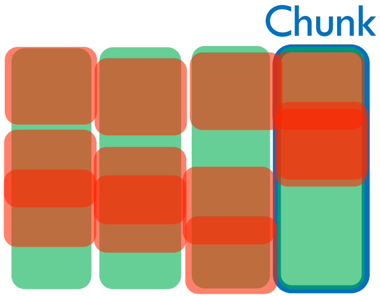
- Traditional VLIW scheduling algorithms don’t scale to our problem size
- We schedule primarily to minimize off-chip data movement
  - Data movement is largely dominated by Key Switch Hints, which are required for most ciphertext operations
  - We schedule to maximize KSH reuse

Specialized Functional Units

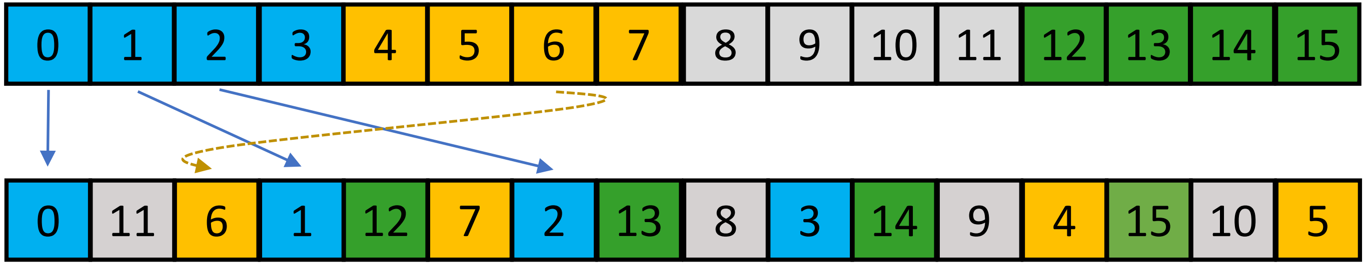
- Vector additions
- Vector multiplications
- NTTs
- Automorphisms
  - Primitive ciphertext polynomial operation that enables rotations of encrypted slots

F1 Vector Datapath



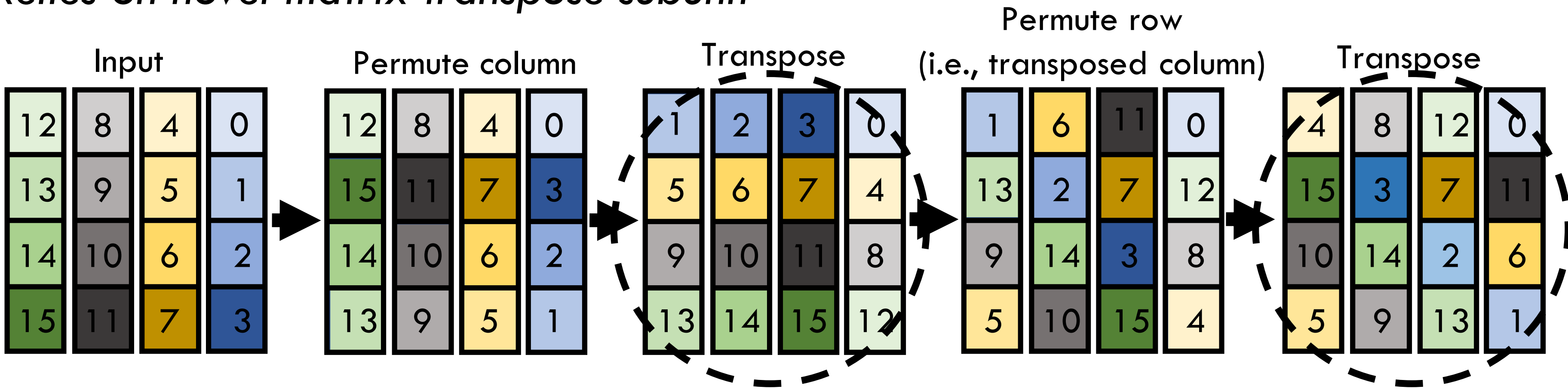


- Polynomials divided into 128-coefficient **chunks.**
- Datapath is 128 lanes wide.
- Vector adds and multiplies act coefficient-wise. Easy to pipeline.
- NTTs and automorphisms have dependencies *across chunks* making them hard to pipeline.



Vectorized Automorphism Unit

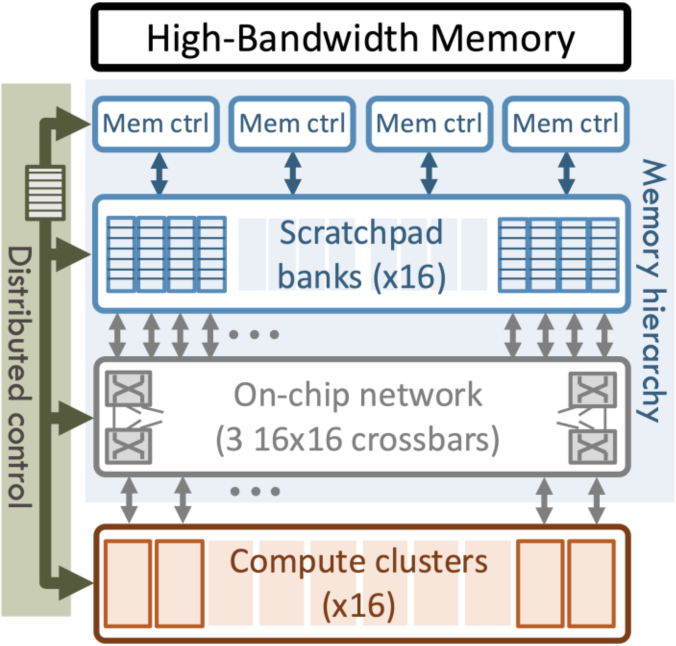
- Decomposes automorphisms into a pipeline of fixed permutations
- Each permutation only applied to *one chunk at a time*
- *Relies on novel matrix transpose subunit*



Agenda

- Overview of FHE computations
- Architectural characterization of FHE
- F1 design
- **Evaluation and results**

Area and Power Breakdown



Component	Area %	TDP %
16× compute clusters	42%	78%
Scratchpad (16×4MB banks)	32%	11%
3×NoC (16×16 512B bit-sliced)	6%	11%
Memory interface (2×HBM2 PHYs)	20%	0%
Total F1	**151.43mm^2**	**180.45W**

- Uses commercial 14/12nm process

Benchmarks

- Low Latency CryptoNets (LoLa)
  - LoLa-MNIST
    - Simple LeNet-style network
    - Used on the MNIST dataset
    - Available with both encrypted and unencrypted weights
  - LoLa-CIFAR
    - Large 6-layer network similar to MobileNet v3
    - Used on the CIFAR-10 dataset
- Logistic regression
  - HELR algorithm for logistic regression in FHE
  - Implements logistic regression training with up to 256 features and 256 samples per batch
- Database lookup
  - HELib’s database lookup example
- BGV/CKKS Bootstrapping

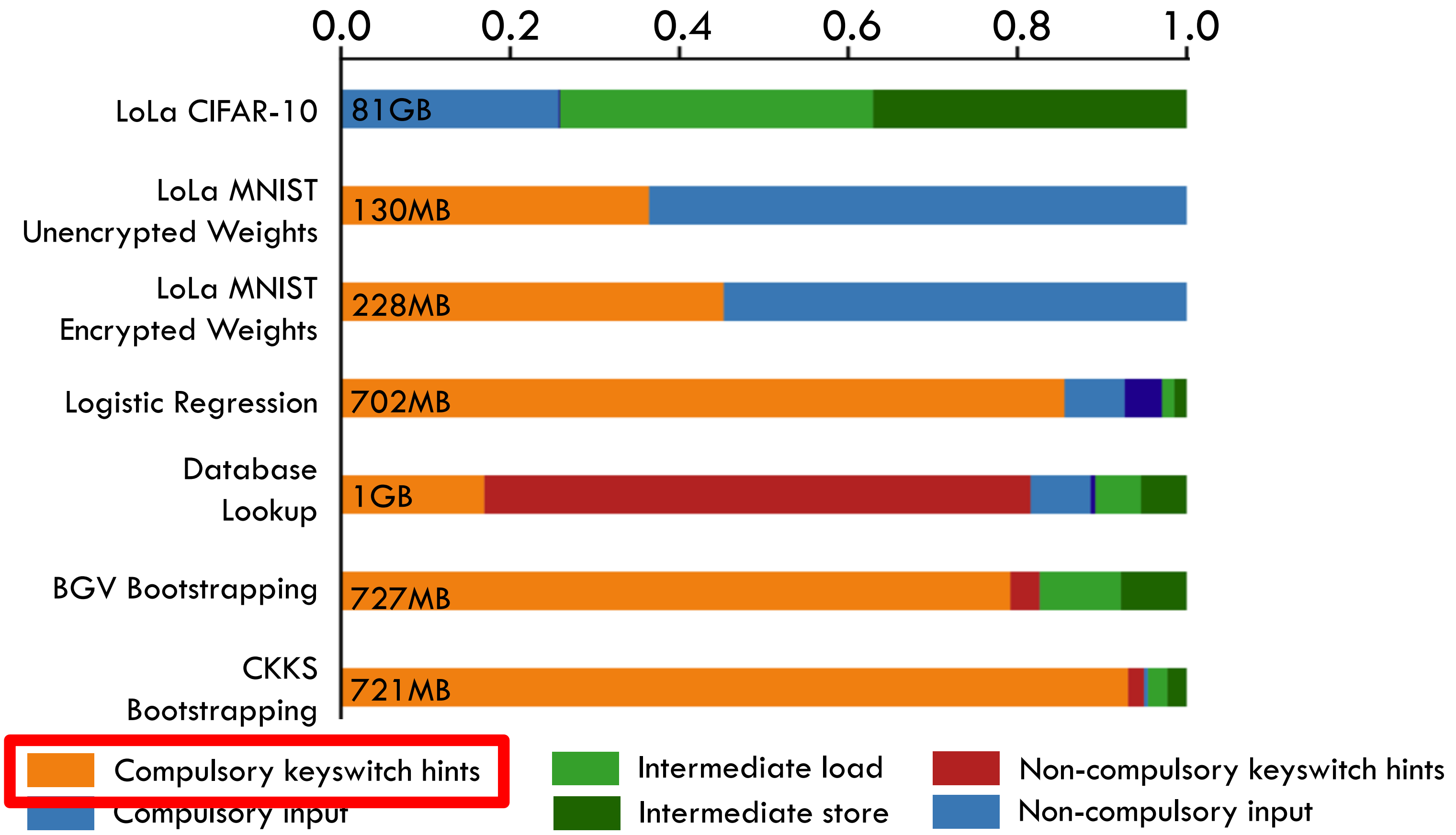
Results

Benchmark	Speedup
LoLa-CIFAR Unencrypted Weights	5,011×
LoLa-MNIST Unencrypted Weights	17,412×
LoLa-MNIST Encrypted Weights	15,086×
Logistic Regression	7,217×
Database Lookup	6,722×
BGV Bootstrapping	1,830×
CKKS Bootstrapping	1,195×
gmean speedup	**5,432×**

Speedup Breakdown

Speedup on benchmark	vs. wimpy NTT	vs. naïve automorph.	vs. VLIW scheduler
LoLa-Cifar Unencr. Wghts	3.5×	12.1×	--- *
LoLa-MNIST Unencr. Wghts	5.0×	4.2×	1.1×
LoLa-MNIST Encr. Wghts	5.1×	11.9×	7.5×
Logistic Regression	1.7×	2.3×	11.7×
Database Lookup	1.6×	1.1×	5.4×
BGV Bootstrapping	1.1×	1.2×	2.7×
CKKS Bootstrapping	2.8×	2.2×	---*
gmean speedup	**2.6×**	**3.3×**	**4.2×**

Off-chip Data Movement Breakdown



Conclusions

- FHE enables computational offloading with guaranteed security
- High computational overhead limits applicability
- F1 accelerates FHE, enabling new applications
- Demonstrates ASIC-level performance ***without sacrificing programmability***

Thanks For Your Attention!

Questions are welcome!