A FHE COMPILER PIONEERING CIRCUIT BOOTSTRAPPING

Parasol

SUNSCREEN

COED SUMMIT

01 Motivation
02 The promise of circuit bootstrapping
03 Parasol's design
04 Numbers

What started us down this rabbit hole...

Make auctions fast

CGGI/TFHE based approach

- Comparisons over encrypted data
  - BFV, CKKS not appropriate
- For our motivating use case, we were OK with...
  - Large-ish machine
  - Custom program
  - Not performing further computation after auction is complete
- Can we get better performance than what's out there in libraries today?

CGGI (aka TFHE) based approach

- Comparisons over encrypted data
  - BFV, CKKS not appropriate
- For our motivating use case, we were OK with...
  - Large-ish machine
  - Custom program
  - Not performing further computation after auction done
- Can we get better performance than what's out there in libraries today?

What approaches for computation and noise reduction exist in TFHE?

How to reduce noise (aka bootstrap)

01

Gate

- Just reduce noise
- Binary messages
- LWE —> LWE

02

Programmable (PBS)

- Reduce noise + apply function
- Messages >1 bit
- LWE —> LWE

03

Circuit (CBS)

- Just reduce noise
- Binary messages*
- Generally used for leveled computation
- LWE —> GGSW

How to perform computation

(depends on how you bootstrap!)

01

Gate

- Generate boolean circuit using standard circuit design techniques
- Bootstrap per gate

02

Programmable (PBS)

- Break up input (e.g. 4 bits per ciphertext)
- Use a series of PBS ops with different lookup tables

03

Circuit (CBS)

- Use CMUXs to realize computation
- Use CBS to reduce noise and obtain GGSW input

How to perform computation

(depends on how you bootstrap!)

01

Gate

- Generate boolean circuit using standard circuit design techniques
- Bootstrap per gate

02

Programmable (PBS)

- Break up input (e.g. 4 bits per ciphertext)
- Use a series of PBS ops with different lookup tables

03

Circuit (CBS)

- Use CMUXs to realize computation
- Use CBS to reduce noise and obtain GGSW input

How to perform computation

(depends on how you bootstrap!)

01

Gate

- Generate boolean circuit using standard circuit design techniques
- Bootstrap per gate

02

Programmable (PBS)

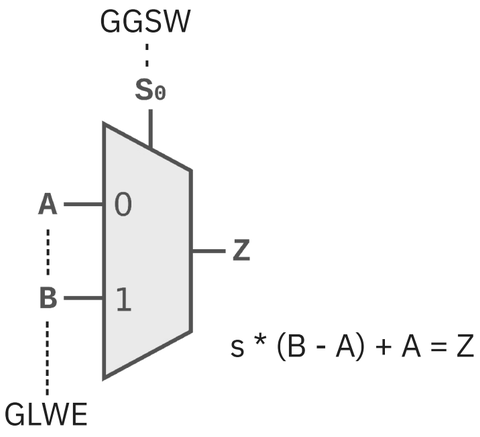
- Break up input (e.g. 4 bits per ciphertext)
- Use a series of PBS ops with different lookup tables

03

Circuit (CBS)

- Use CMUXs to realize computation
- Use CBS to reduce noise and obtain GGSW input

Multiplexers are a LOT cheaper than bootstraps!



How to perform computation

(depends on how you bootstrap!)

01

Gate

- Generate boolean circuit using standard circuit design techniques
- Bootstrap per gate

02

Programmable (PBS)

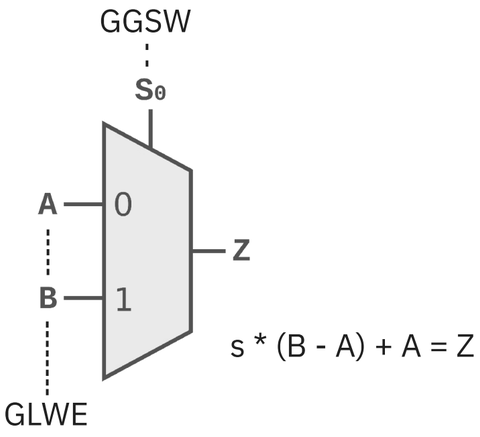
- Break up input (e.g. 4 bits per ciphertext)
- Use a series of PBS ops with different lookup tables

03

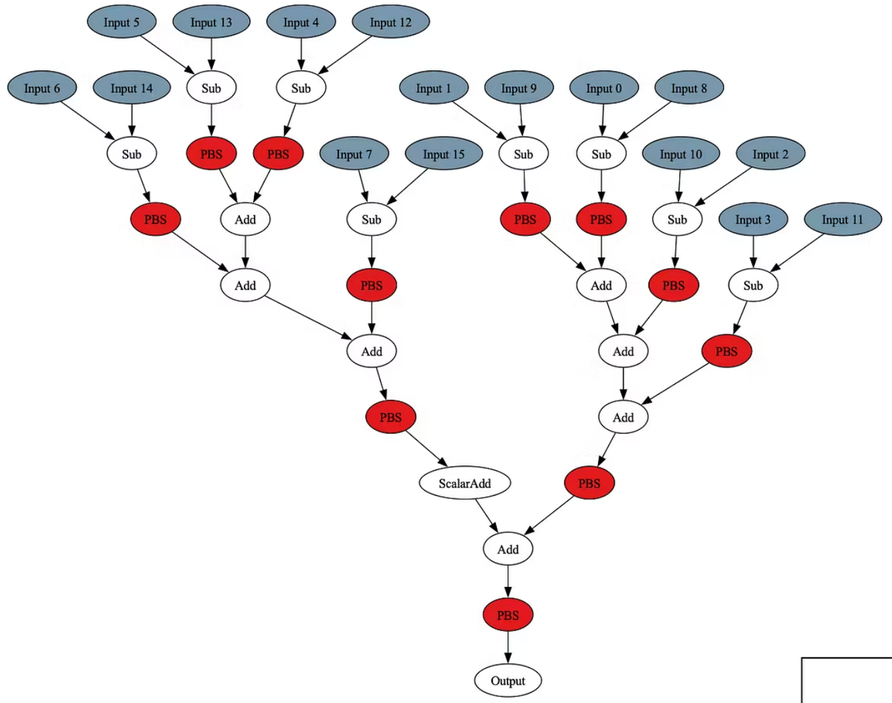
Circuit (CBS)

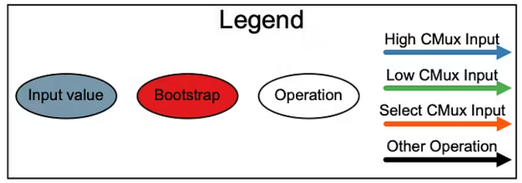
- Use CMUXs to realize computation
- Use CBS to reduce noise and obtain GGSW input

Multiplexers are a LOT cheaper than bootstraps!



16-bit less than using PBS vs CBS-CMUX

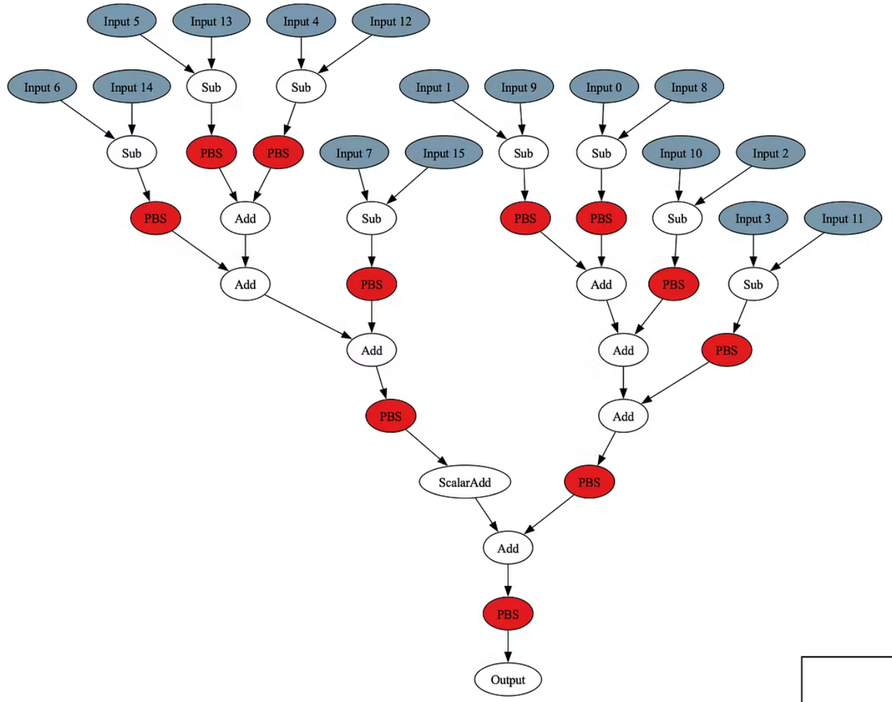


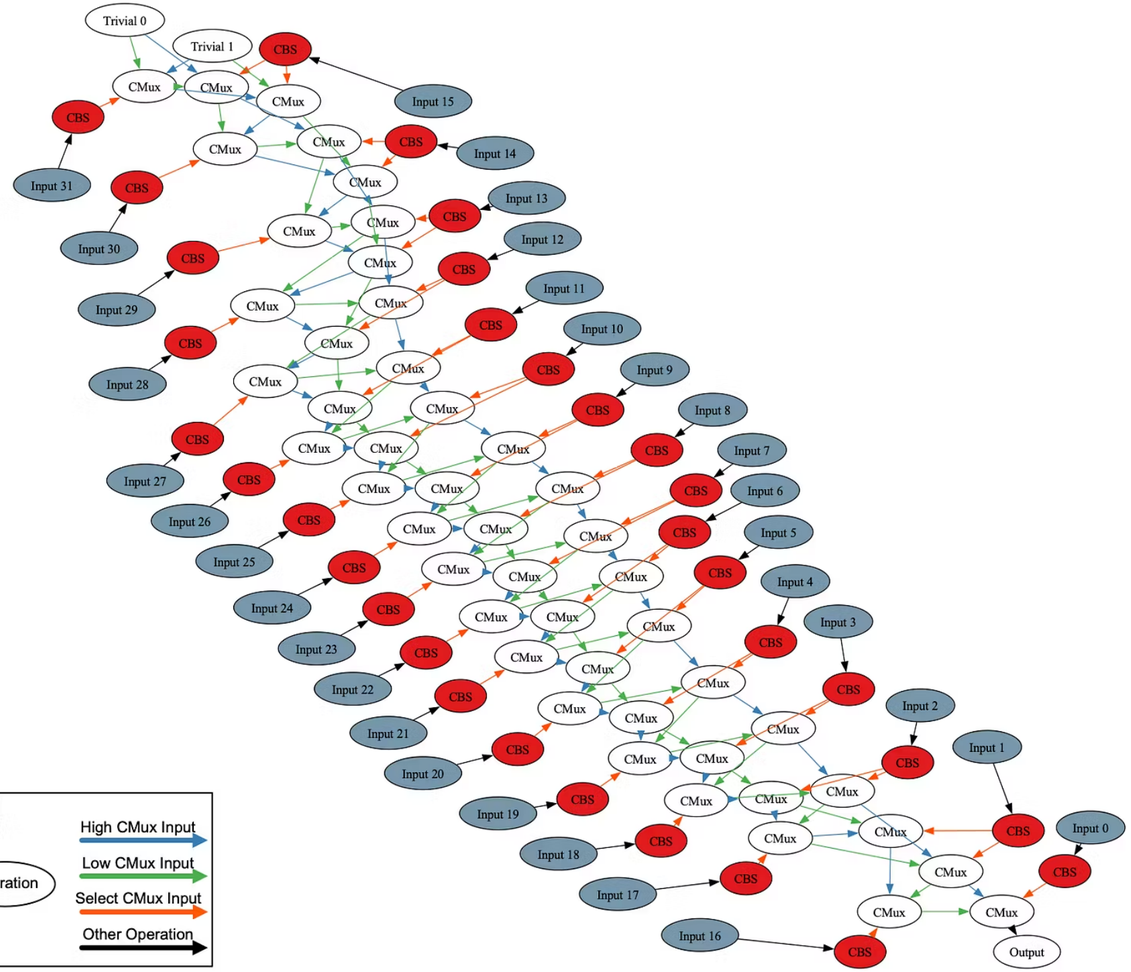


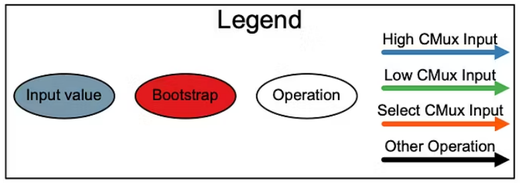
- 3 PBS ops in critical path

- 1 CBS op in critical path
- More parallelism

16-bit less than using PBS vs CBS-CMUX







- 3 PBS ops in critical path

- 1 CBS op in critical path
- More parallelism

A CBS op is slower than a PBS. Can we realize any perf improvements in practice?

Yes we can...

- c7a.16xlarge machine (64 cores)
- 128 bits of security
- LHS uses tfhe-rs 1.1.3
- RHS is our own implementation
  - Additional optimization in multiply

	PBS	CBS
8-bit Greater than	34.47 ms	**28.88 ms**
16-bit Greater than	51.50 ms	**32.54 ms**
32-bit Greater than	70.07 ms	**47.09 ms**
8-bit Add	51.66 ms	**29.65 ms**
16-bit Add	51.71 ms	**33.34 ms**
32-bit Add	72.72 ms	**47.89 ms**
8-bit Multiply	88.67 ms	**34.25 ms**
16-bit Multiply	137.12 ms	**54.24 ms**
32-bit Multiply	279.35 ms	**166.08 ms**

Downsides of CBS-CMUX approach

- More memory-intensive than gate bootstrapping
- Need sufficient cores to parallelize computation
- Extremely complex to build performant programs with this approach

How can we bring the CBS-CMUX approach to non-experts?

Build an FHE compiler!

Design goals of Parasol

- Better performance (in part due to the CBS-CMUX approach)
- Write in a mainstream language (no eDSL required)
- Compact programs (not covered in this talk!)

```c
#include <stdbool.h>
#include <stdint.h>

typedef struct Winner {
    uint16_t bid;
    uint16_t idx;
} Winner;

void auction(
    uint16_t *bids,
    uint16_t len,
    Winner *winningBid
) {
    winningBid->bid = bids[0];
    winningBid->idx = 0;

    for (uint16_t i = 1; i < len; i++) {
        bool isWinner = bids[i] ⩾ winningBid->bid;

        winningBid->bid = isWinner ? bids[i] : winningBid->bid;
        winningBid->idx = isWinner ? i : winningBid->idx;
    }
}
```

```c
#include <stdbool.h>
#include <stdint.h>

typedef struct Winner {
    uint16_t bid;
    uint16_t idx;
} Winner;

[[clang::fhe_program]] void auction(
    [[clang::encrypted]] uint16_t *bids,
    uint16_t len,
    [[clang::encrypted]] Winner *winningBid
) {
    winningBid->bid = bids[0];
    winningBid->idx = 0;

    for (uint16_t i = 1; i < len; i++) {
        bool isWinner = bids[i] ⩾ winningBid->bid;

        winningBid->bid = isWinner ? bids[i] : winningBid->bid;
        winningBid->idx = isWinner ? i : winningBid->idx;
    }
}
```

Parasol

Looking under the hood...

LLVM-based compiler

- Developers write in C, tag FHE functions and inputs appropriately
- Modified version of clang that supports the Parasol processor

---

Virtual processor

- Custom ISA to execute programs over a mix of plaintext and encrypted data
- OUt-of-order processor design
- Backend circuit processor implements CBS-CMUX approach

TFHE library

- Comprehensive implementation of TFHE scheme
- Features recent optimization for CBS

Parasol

Looking under the hood...

LLVM-based compiler

- Developers write in C, tag functions and inputs appropriately
- Modified version of clang that supports the parasol processor

---

Virtual processor

- Custom ISA to execute programs over a mix of plaintext and encrypted data
- Out-of-order processor design
- Backend circuit processor implements CBS-CMUX approach

TFHE library

- Comprehensive implementation of TFHE scheme
- Features recent optimization for CBS

Parasol

Looking under the hood...

LLVM-based compiler

- Developers write in C, tag functions and inputs appropriately
- Modified version of clang that supports the parasol processor

Virtual processor

- Custom ISA to execute programs over a mix of plaintext and encrypted data
- OUt-of-order processor design
- Backend circuit processor implements CBS-CMUX approach

TFHE library

- Comprehensive implementation of TFHE scheme
- Features recent optimization for CBS (WHS+25)

So what ever happened to that auction?

- ~15-17x faster than the current SotA
- c7a.16xlarge machine
- 16-bit precision
- Parasol/Concrete provide ~128 bits of security; Google/Cingulata/Juliet are ~118 bits; E3 ~94 bits

	Runtime (secs) for number of bids				
Compiler	2	4	8	16	32
Parasol Compiler	**0.098**	**0.275**	**0.625**	**1.315**	**2.714**
Concrete[1]	24.1	86.4	264	694	1690
Google Transpiler	2.36	6.72	15.4	33.1	68.2
E3	12.4	36.6	84.8	182	379
Cingulata	1.48	4.33	10.3	22.4	47.2
Juliet	5.54	16.6	38.7	82.7	171

(1) We pass 16-bit input examples to maintain same input precision as other frameworks.

Takeaways for the community

- **CBS-CMUX approach is promising**
  - Especially in the context of hardware acceleration!
  - Potential to unlock higher throughput + lower latency
- **FHE compiler devex still has some way to go**
  - LLVM-based approach is technically ambitious

Find out more

EPRINT: 2025/1144

USE PARASOL: DOCS.SUNSCREEN.TECH