An Efficient and Scalable Sparse Polynomial Multiplication Accelerator for LAC on FPGA

Jipeng Zhang[1], Zhe Liu[1][2](✉), Hao Yang[1], Junhao Huang[1], Weibin Wu[1]

[1]Nanjing University of Aeronautics and Astronautics, Jiangsu, China

[2]State Key Laboratory of Cryptology, Beijing, China

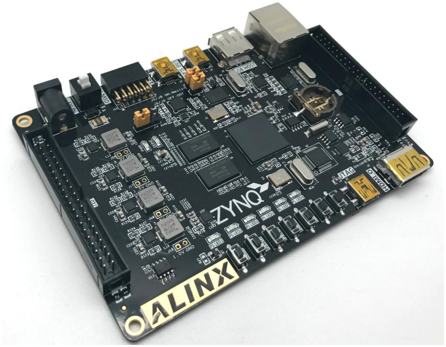
November 26, 2020

Lattice-based Cryptography

- RSA and ECC: Discrete Logarithm and Integer Factorization Problems
  - Hard problems can be solved by Shor's algorithm
- Lattice-based Cryptography: Hard for quantum computers
  - Ring-LWE Encryption schemes: proposed [EUROCRYPT'10]
  - Huge performance improvement: Newhope [USENIX'16]
  - LAC is a unique scheme (Round 2 candidates): its modulus is 251, which can be packed into a single byte.
- NIST Post-Quantum-Cryptography (PQC) Project
  - 2016, Formal call for proposals
  - 2017, Round1 algorithms announced (69 submissions)
  - 2019, Round2 algorithms announced (26 algorithms)
  - ......

Implementation Platform

- Zynq-7020 (including Artix-7 FPGA)
  - FPGA is composed of hardware resources such as logic unit, RAM, multiplier, etc.
  - FPGAs are widely used to design accelerators for cryptographic schemes.



Motivation & Contribution

- Motivation
  - The most compute-intensive routine in the LAC scheme is sparse polynomial multiplication (SPM).
  - Hardware accelerator can greatly improve the performance of SPM.
- Contributions: Efficient and Scalable SPM accelerator
  - Parallel design: Dual-For-Loop-Parallel (DFLP) technique.
  - A new modular reduction for the modulus $q = 251$.
  - Optimization of the pipeline design.
  - Scalable design can achieve various performance-area trade-offs.

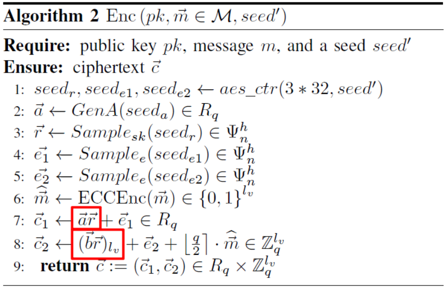
LAC scheme

- Key generation stage: KG
    - Dense polynomial: coefficients $a_i \in [0, q)$.
    - Sparse polynomial: coefficients $r_i \in \{-1, 0, 1\}$, and more than half of the coefficients are 0.
    - $\vec{a}$ is a dense polynomial and $\vec{s}$ is a sparse polynomial.
    - SPM is invoked when computing $\vec{a}\vec{s}$.
    - Public key $\left(seed_a, \vec{b}\right)$ and secret key $\vec{s}$ are obtained.

```
Algorithm 1 KG
Ensure: a pair of public key and secret key $(pk, sk)$
1: $seed \xleftarrow{\$} \{0, 1, \ldots, 255\}^{32}$
2: $seed_a, seed_{sk}, seed_e \leftarrow aes\_ctr(3 * 32, seed)$
3: $\vec{a} \leftarrow GenA(seed_a) \in R_q$
4: $\vec{s} \leftarrow Sample_{sk}(seed_{sk}) \in \Psi_n^h$
5: $\vec{e} \leftarrow Sample_e(seed_e) \in \Psi_n^h$
6: $\vec{b} \leftarrow \boxed{\vec{a}\vec{s}} + \vec{e} \in R_q$
7: return $\left(pk := \left(seed_a, \vec{b}\right), sk := \vec{s}\right)$
```

LAC scheme

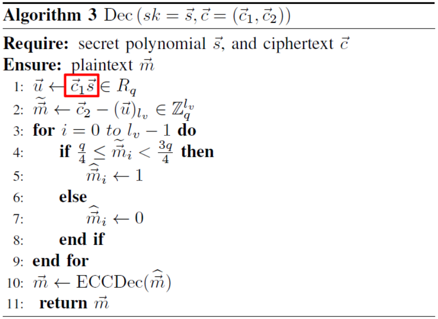
- Encryption stage: $Enc(pk, \vec{m}, seed')$
  - $\vec{a}$ and $\vec{b}$ are dense polynomial. $\vec{r}$ is a sparse polynomial.
  - SPM is invoked when computing $\vec{a}\vec{r}$ and $\vec{b}\vec{r}$.
  - Ciphertext $\vec{c}$ is obtained.



```
**Algorithm 2** Enc $(pk, \vec{m} \in \mathcal{M}, seed')$
**Require:** public key $pk$, message $m$, and a seed $seed'$
**Ensure:** ciphertext $\vec{c}$
1: $seed_r, seed_{e1}, seed_{e2} \leftarrow aes\_ctr(3 * 32, seed')$
2: $\vec{a} \leftarrow GenA(seed_a) \in R_q$
3: $\vec{r} \leftarrow Sample_{sk}(seed_r) \in \Psi_n^h$
4: $\vec{e_1} \leftarrow Sample_e(seed_{e1}) \in \Psi_n^h$
5: $\vec{e_2} \leftarrow Sample_e(seed_{e2}) \in \Psi_n^h$
6: $\widehat{\vec{m}} \leftarrow \mathrm{ECCEnc}(\vec{m}) \in \{0,1\}^{l_v}$
7: $\vec{c}_1 \leftarrow \boxed{\vec{a}\vec{r}} + \vec{e}_1 \in R_q$
8: $\vec{c}_2 \leftarrow \boxed{(\vec{b}\vec{r})_{l_v}} + \vec{e}_2 + \lfloor \frac{q}{2} \rceil \cdot \widehat{\vec{m}} \in \mathbb{Z}_q^{l_v}$
9:   **return** $\vec{c} := (\vec{c}_1, \vec{c}_2) \in R_q \times \mathbb{Z}_q^{l_v}$
```

LAC scheme

- Decryption stage: $Dec(sk, \vec{c})$
  - $\vec{c_1}$ is dense polynomial and $\vec{s}$ is sparse polynomial.
  - SPM is invoked when computing $\vec{c_1}\vec{s}$.
  - Message $\vec{m}$ is obtained.



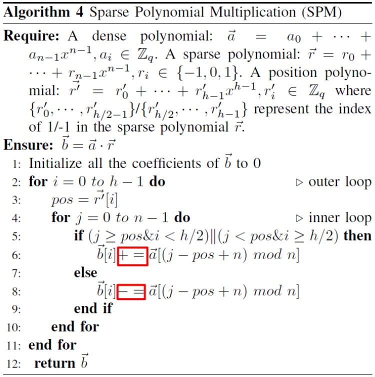
Algorithm 3 Dec $(sk = \vec{s}, \vec{c} = (\vec{c}_1, \vec{c}_2))$

Require: secret polynomial $\vec{s}$, and ciphertext $\vec{c}$
Ensure: plaintext $\vec{m}$

```
1: $\vec{u} \leftarrow \boxed{\vec{c}_1\vec{s}} \in R_q$
2: $\widetilde{\vec{m}} \leftarrow \vec{c}_2 - (\vec{u})_{l_v} \in \mathbb{Z}_q^{l_v}$
3: for $i = 0$ to $l_v - 1$ do
4:     if $\frac{q}{4} \leq \widetilde{\vec{m}}_i < \frac{3q}{4}$ then
5:         $\widehat{\vec{m}}_i \leftarrow 1$
6:     else
7:         $\widehat{\vec{m}}_i \leftarrow 0$
8:     end if
9: end for
10: $\vec{m} \leftarrow \mathrm{ECCDec}(\widehat{\vec{m}})$
11: return $\vec{m}$
```

LAC scheme

- For a sparse polynomial $\vec{r}$, its coefficients are chosen from the set $\{-1, 0, 1\}$, so the multiplication operation in the algorithm can be eliminated.
- For LAC-Light, LAC128, LAC192 and LAC256, SPM takes up 48%, 54%, 66%, and 69% of the total time respectively.



Algorithm 4 Sparse Polynomial Multiplication (SPM)

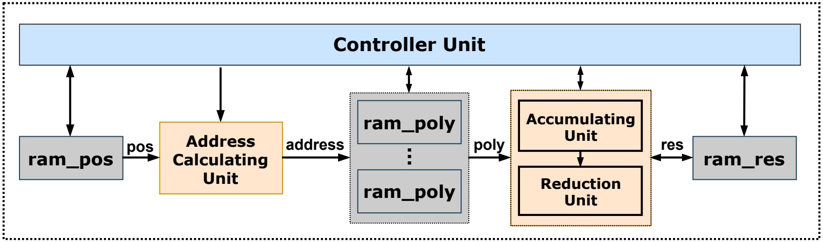
Require: A dense polynomial: $\vec{a} = a_0 + \cdots + a_{n-1}x^{n-1}, a_i \in \mathbb{Z}_q$. A sparse polynomial: $\vec{r} = r_0 + \cdots + r_{n-1}x^{n-1}, r_i \in \{-1, 0, 1\}$. A position polynomial: $\vec{r'} = r'_0 + \cdots + r'_{h-1}x^{h-1}, r'_i \in \mathbb{Z}_q$ where $\{r'_0, \cdots, r'_{h/2-1}\}/\{r'_{h/2}, \cdots, r'_{h-1}\}$ represent the index of 1/-1 in the sparse polynomial $\vec{r}$.

Ensure: $\vec{b} = \vec{a} \cdot \vec{r}$

```
1:  Initialize all the coefficients of $\vec{b}$ to 0
2:  for $i = 0$ to $h-1$ do                                   ▷ outer loop
3:      $pos = \vec{r'}[i]$
4:      for $j = 0$ to $n-1$ do                               ▷ inner loop
5:          if $(j \geq pos \& i < h/2) \| (j < pos \& i \geq h/2)$ then
6:              $\vec{b}[i] \boxed{+=} \vec{a}[(j - pos + n) \bmod n]$
7:          else
8:              $\vec{b}[i] \boxed{-=} \vec{a}[(j - pos + n) \bmod n]$
9:          end if
10:     end for
11: end for
12: return $\vec{b}$
```

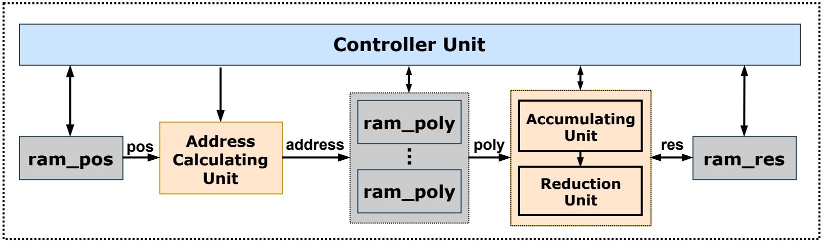
Overall Architecture

The proposed architecture consists of three memory blocks, including a position memory ***ram_pos***, p dense polynomial coefficient memories ***ram_poly***, an intermediate value and final result memory ***ram_res***, three data processing units, and a controller unit.



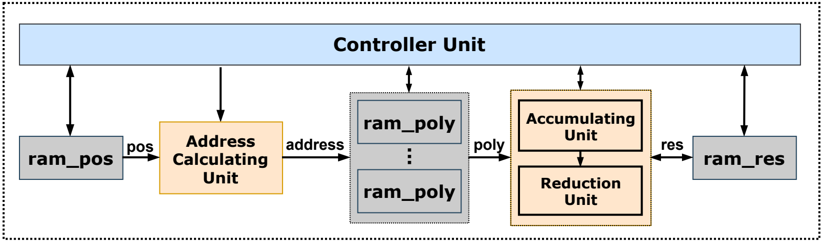
Execution Flow

Step1: Once the accelerator is started, ***controller unit*** will prepare the read address of ***ram_pos***, and then issue a read request to ***ram_pos***.
Step2: The outputs of ***ram_pos***, including the position information of sparse polynomial coefficients, are sent to ***Address Calculating Unit***.



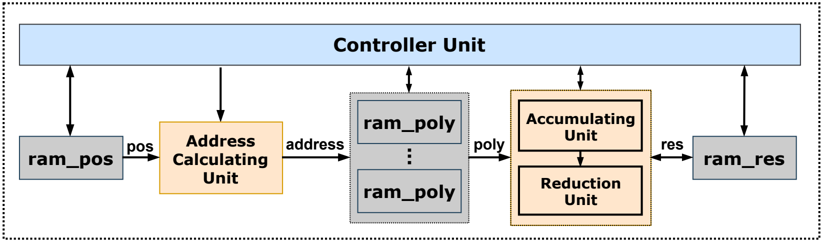
Execution Flow

Step3: The outputs of ***Address Calculating Unit*** contain p read addresses of ***ram_poly***, then p read requests to ***ram_poly*** are issued in parallel.
Step4: The ***Accumulating and Reduction Unit*** accumulates p outputs from ***ram_poly*** and the intermediate value from ***ram_res***. Then our new modular reduction algorithm is used to correct each coefficients of the result to $[0, q)$, and the corrected result is sent to ***ram_res***.



Execution Flow

Step5: The read addresses of ***ram_poly*** are updated by ***controller unit*** and then return to Step 3. Steps 3 to 5 are equivalent to the inner loop of Algorithm SPM and the loop repeats until all the memory contents of ***ram_res*** are updated.
Step6: After the inner loop, the ***controller unit*** updates the read addresses of ***ram_pos*** and then return to step 1. Steps 1 and 6 are equal to the *for* statement of the outer loop of Algorithm SPM. The counter of the outer loop is $\lceil \frac{h}{p} \rceil$.



The choice of Block RAM and Distributed RAM

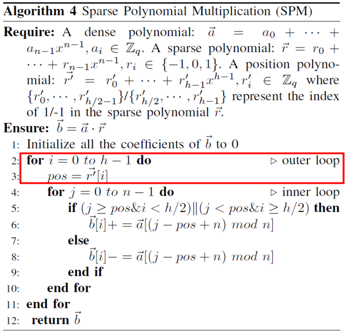
- Block RAM is dedicated but width and depth is limited.
    - The total capacity of BRAM36/BRAM18 is 36/18 Kbit, and the maximum width of them is 36/18 bits in dual-port mode.
- Distributed RAM consists of Look Up Tables (LUTs), the most basic logic element in FPGA.

RAM	Type	Width (Bit)	Depth	Mode
ram_pos	DRAM	$p \cdot \log_2(n)$	h/p	Single-Port
ram_poly	BRAM16	16	n	Dual-Port
ram_res	BRAM16	16	$n/2$	Dual-Port

- ram_pos: get p positions per cycle (use parameter p to achieve various performance-area trade-offs).
- ram_poly and ram_res: take full advantage of BRAM's bandwidth.

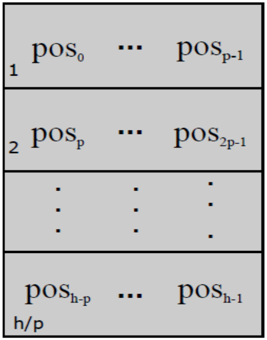
Parallel Design-Outer Loop Parallel

- The 2th and 3th lines determine the read request of ***ram_pos***.
- Only one position can be read at a time without parallel design.



```
Algorithm 4 Sparse Polynomial Multiplication (SPM)
Require: A dense polynomial: $\vec{a} = a_0 + \cdots + a_{n-1}x^{n-1}, a_i \in \mathbb{Z}_q$. A sparse polynomial: $\vec{r} = r_0 + \cdots + r_{n-1}x^{n-1}, r_i \in \{-1, 0, 1\}$. A position polynomial: $\vec{r'} = r'_0 + \cdots + r'_{h-1}x^{h-1}, r'_i \in \mathbb{Z}_q$ where $\{r'_0, \cdots, r'_{h/2-1}\}/\{r'_{h/2}, \cdots, r'_{h-1}\}$ represent the index of 1/-1 in the sparse polynomial $\vec{r}$.
Ensure: $\vec{b} = \vec{a} \cdot \vec{r}$
 1: Initialize all the coefficients of $\vec{b}$ to 0
 2: for $i = 0$ to $h - 1$ do                          ▷ outer loop
 3:     $pos = \vec{r'}[i]$
 4:     for $j = 0$ to $n - 1$ do                      ▷ inner loop
 5:         if $(j \geq pos \& i < h/2) \| (j < pos \& i \geq h/2)$ then
 6:             $\vec{b}[i] += \vec{a}[(j - pos + n)\ mod\ n]$
 7:         else
 8:             $\vec{b}[i] -= \vec{a}[(j - pos + n)\ mod\ n]$
 9:         end if
10:     end for
11: end for
12: return $\vec{b}$
```

- We can get *p* positions once at a time in our parallel design.
- Accordingly, the outer loop is accelerated by *p* times.



Parallel Design-Outer Loop Parallel

- There are $p/2$ identical copies.
- For each ***ram_poly***, each coefficient appears twice.
- The read address is random and this structure can handle both of even index and odd index.

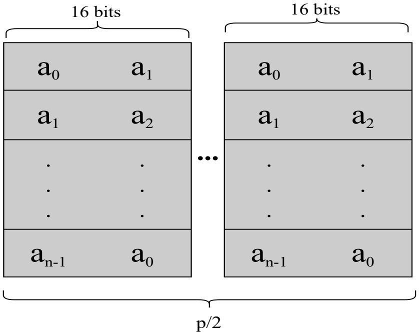
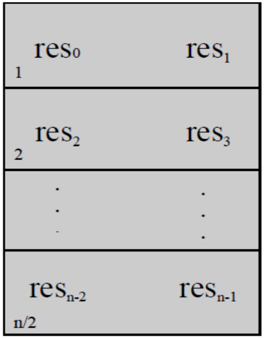


Figure: ram_poly structure

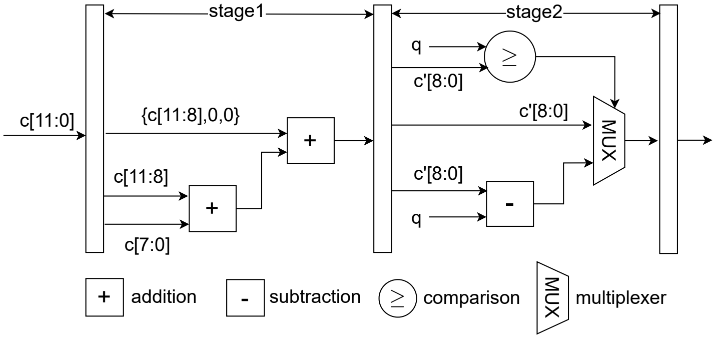
Parallel Design-Inner Loop Parallel

- The maximum width of BRAM16 is 16bits in dual-port mode.
- This means that 32 bits can be read from BRAM16 at a time, which can accommodate four polynomial coefficients.
- One port is used for reading, while another port is dedicated for writing.
- Inner loop is accelerated by 2 times.



New Modular Reduction

- Take $p = 16$ as an example, 16 8-bit coefficients are added and stored in 12 bits.
- $2^8 \equiv 2^2 + 1 (\text{mod } 251)$
- $c \equiv 2^8 c[11:8] + c[7:0] \equiv 2^2 c[11:8] + c[11:8] + c[7:0]$
- An additional comparison and subtraction are needed for correcting the result to $[0, q)$.

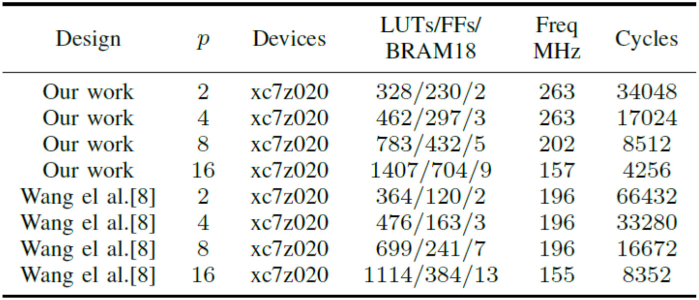


New Modular Reduction

- Advantages of the new modular reduction:
    - It is more hardware friendly, because multiplication is not used.
    - Compared with compare-and-subtraction method, this method can save hardware resources.
    - Thanks to the two-stage pipeline design, this method can achieve higher hardware frequency.

Implementation Results for LAC128

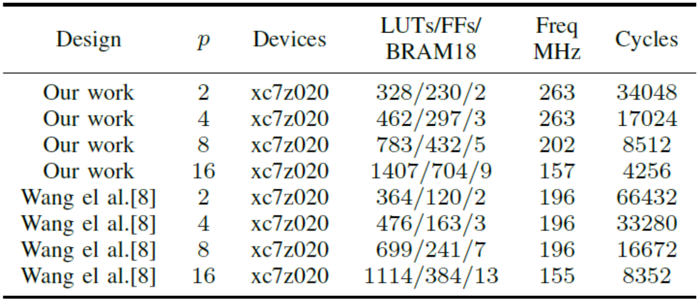
- Hardware resources and frequency
  - The number of LUTs and FFs has been slightly increased, but the frequency has been increased by up to 34%.
  - Thanks to the full use of BRAM18, our design reduced 2,4 BRAM18 for p=8,16, respectively.



Design	p	Devices	LUTs/FFs/ BRAM18	Freq MHz	Cycles
Our work	2	xc7z020	328/230/2	263	34048
Our work	4	xc7z020	462/297/3	263	17024
Our work	8	xc7z020	783/432/5	202	8512
Our work	16	xc7z020	1407/704/9	157	4256
Wang el al.[8]	2	xc7z020	364/120/2	196	66432
Wang el al.[8]	4	xc7z020	476/163/3	196	33280
Wang el al.[8]	8	xc7z020	699/241/7	196	16672
Wang el al.[8]	16	xc7z020	1114/384/13	155	8352

Implementation Results for LAC128

- Cycles:
  - After using Dual-For-Loop-Parallel (DFLP) technique, the number of cycles is halved compared to the previous work.



Design	p	Devices	LUTs/FFs/ BRAM18	Freq MHz	Cycles
Our work	2	xc7z020	328/230/2	263	34048
Our work	4	xc7z020	462/297/3	263	17024
Our work	8	xc7z020	783/432/5	202	8512
Our work	16	xc7z020	1407/704/9	157	4256
Wang el al.[8]	2	xc7z020	364/120/2	196	66432
Wang el al.[8]	4	xc7z020	476/163/3	196	33280
Wang el al.[8]	8	xc7z020	699/241/7	196	16672
Wang el al.[8]	16	xc7z020	1114/384/13	155	8352

Conclusion

- An Efficient and Scalable Sparse Polynomial Multiplication Accelerator for LAC on FPGA
  - Better parallel design: Dual-For-Loop-Parallel (DFLP) technique.
  - A new modular reduction for the modulus $q = 251$.
  - Higher frequency through optimization of the pipeline design.
  - Scalable design can achieve various performance-area trade-offs.
- The clock cycle is halved and the frequency is increased with a small resources cost.

Thanks

Thanks for listening