International Institute of Information Technology
Bangalore

VLSI READING ELECTIVE

HARDWARE ACCELERATION OF NTT USING PYNQ-Z2

B Sathiya Naraayanan
(IMT2020534)

December 11, 2023


iit-b
ज्ञानमुत्तमम्

INTRODUCTION

Homomorphic encryption is the conversion of data into ciphertext that can be analyzed and worked with as if it were still in its original form.Homomorphic encryption enables complex mathematical operations to be performed on encrypted data without compromising the encryption.The data in a homomorphic encryption scheme retains the same structure, identical mathematical operations will provide equivalent results regardless of whether the action is performed on encrypted or decrypted data.

The homomorphic encryption involves polynomial multiplication.This step consumes lot of resources and takes majority of the execution time.We perform NTT to save the resources used and execution time.We perform NTT on two polynomials that should be multiplied and The output of forward transform(NTT) can be element wise multiplied to get output.The inverse NTT of the obtained output will give the polynomial multiplication.

NTT algorithm reduces the computation time of polynomial multiplication from O(N^2) to O(NlogN).Microsoft SEAL algorithm uses 54% of its computation time to perform NTT/iNTT and IEEE HPCA'19 algorithm uses 34% of its computation time. We can perform NTT/iNTT in hardware to accelerate the NTT/iNTT process.

ALGORITHM

The NTT algorithm uses N inputs and gives N outputs.We gave the prime-modulo and the twiddle factor as input as well.The following alogithm was implemented in verilog and packed into IP using AXI4-lite protocool.A new project was made and this IP was put into a block design and connected to the Zynq processing system. The block design and bitstream file were exported and put onto the PYNQ-Z2 board.

$$\tilde{a}_i \;=\; \sum_{j=0}^{n-1} a_j \omega^{ij} \bmod q \;=\; f(\omega^i)$$

Figure 1: NTT Algorithm

IMPLEMENTATION

The proposed algorithm was implemented in verilog and packed into and IP.
The IP was connected with zynq processing system.

The NTT was implemented for 8 inputs and 16 inputs.The input and output bitwidth of 4,8,16 and 32 bits were implemented as well. The verilog implementation of NTT for 16 inputs and each input 8 bits wide is as follows.

```
//////////////////////////////////////////////////////////////////////////////
module ntt(input[0:7] a1,
           input[0:7] a2,
           input[0:7] a3,
           input[0:7] a4,
           input[0:7] a5,
           input[0:7] a6,
           input[0:7] a7,
           input[0:7] a8,
           input[0:7] a9,
           input[0:7] a10,
           input[0:7] a11,
           input[0:7] a12,
           input[0:7] a13,
           input[0:7] a14,
           input[0:7] a15,
           input[0:7] a16,
           input [0:7] q,input [0:3] w,
            output reg [0:7] ntt_out1,
            output reg [0:7] ntt_out2,
            output reg [0:7] ntt_out3,
            output reg [0:7] ntt_out4,
            output reg [0:7] ntt_out5,
            output reg [0:7] ntt_out6,
            output reg [0:7] ntt_out7,
            output reg [0:7] ntt_out8,
            output reg [0:7] ntt_out9,
            output reg [0:7] ntt_out10,
            output reg [0:7] ntt_out11,
            output reg [0:7] ntt_out12,
            output reg [0:7] ntt_out13,
            output reg [0:7] ntt_out14,
            output reg [0:7] ntt_out15,
            output reg [0:7] ntt_out16
            );
    integer i,j,n=16;
    reg [0:7] sum=0;
    wire [0:7] in[0:15];
    reg [0:7] out[0:15];
    assign in[0]=a1;
    assign in[1]=a2;
    assign in[2]=a3;
```

```
    assign in[3]=a4;
    assign in[4]=a5;
    assign in[5]=a6;
    assign in[6]=a7;
    assign in[7]=a8;
    assign in[8]=a9;
    assign in[9]=a10;
    assign in[10]=a11;
    assign in[11]=a12;
    assign in[12]=a13;
    assign in[13]=a14;
    assign in[14]=a15;
    assign in[15]=a16;
    always@(*)begin
        for(i=0;i<n;i=i+1) begin
       sum=in[0];
            for(j=1;j<n;j=j+1) begin
                sum=sum+in[j]*((w**(i*j))%q);
            end
            out[i]=sum%q;
        end
    ntt_out1=out[0];
    ntt_out2=out[1];
    ntt_out3=out[2];
    ntt_out4=out[3];
    ntt_out5=out[4];
    ntt_out6=out[5];
    ntt_out7=out[6];
    ntt_out8=out[7];
    ntt_out9=out[8];
    ntt_out10=out[9];
    ntt_out11=out[10];
    ntt_out12=out[11];
    ntt_out13=out[12];
    ntt_out14=out[13];
    ntt_out15=out[14];
    ntt_out16=out[15];
    end
endmodule

//////////////////////////////////////////////////////////////////////////////
```

FPGA AND PYNQ-Z2 RESULTS

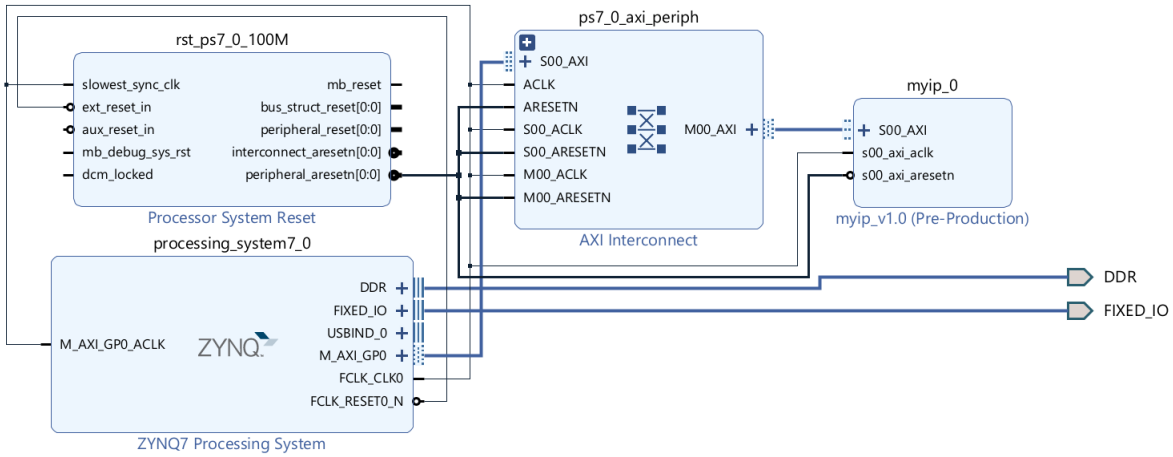

Figure 2: Block Design

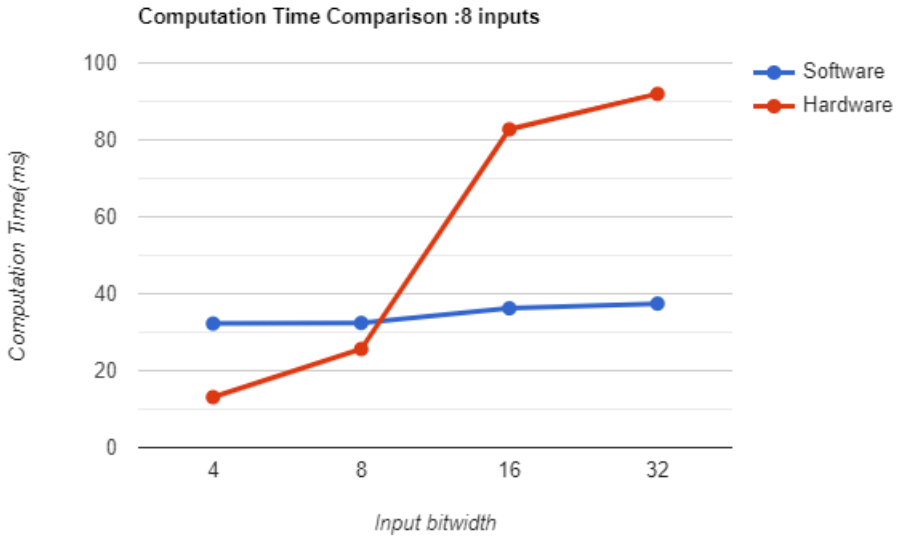

Figure 3: Software vs Hardware

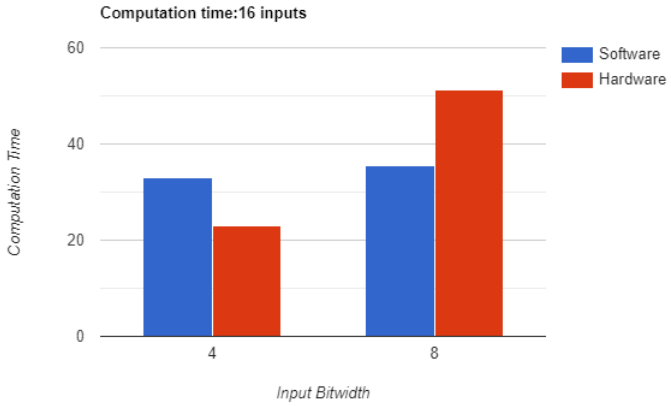

Figure 4: Software vs Hardware

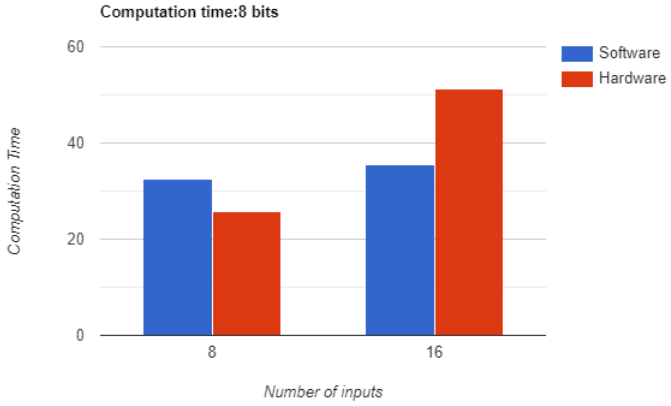

Figure 5: Software vs Hardware

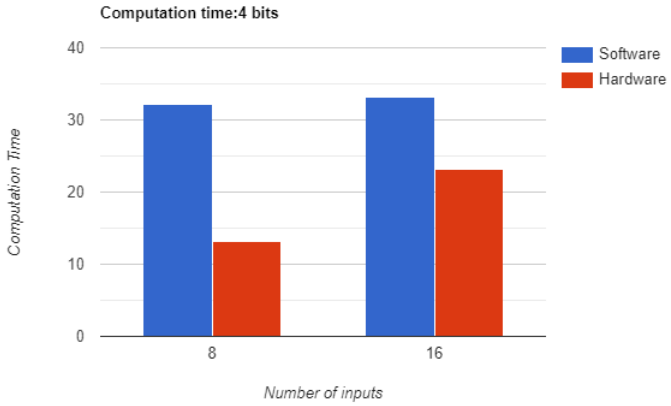

Figure 6: Software vs Hardware

```
The computation time was observed 10 times and the average is as follows

8 inputs 4 bits  sw 32.24   Hw 13.11
8 inputs 8 bits  sw 32.412  Hw 25.635
8 inputs 16 bits sw 36.22   Hw 82.74
8 inputs 32 bits sw 37.44   Hw 91.95

16 inputs 4 bits sw 33.12  Hw 23.10
16 inputs 8 bits sw 35.42  Hw 51.24
```

Video demonstration

Youtube link

References

- youtube1
- youtube2
- A Flexible and Scalable NTT Hardware: Applications from Homomorphically Encrypted Deep Learning to Post-Quantum Cryptography Ahmet Can Mert†‡, Emre Karabulut‡, Erdinc¸ Ozt ¨ urk ¨ †, Erkay Savas¸ †, Michela Becchi‡, Aydin Aysu‡ †Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey ‡Department of Electrical and Computer Engineering, North Carolina State University, NC, USA
- Verification of an Optimized NTT Algorithm Jorge A. Navas, Bruno Dutertre, Ian A. Mason SRI Internationa
- Design and Implementation of a Fast and Scalable NTT-Based Polynomial Multiplier Architecture Ahmet Can Mert1 , Erdinc Ozturk1 , and Erkay Savas1 Sabanci University Orta Mahalle, 34956 Tuzla, Istanbul, Turkey