PQC Term Project

Group 7

R13921090 李臻茵 R13921065 陳俊丞

2025/06/09

Github Link

Outline

- Parameter Settings & System Architecture
- Initial Approach
  - Karatsuba Multiplication ~~& Modular Reduction for $2^{256}-1$~~
  - Testing Strategy & Debugging
  - Benchmarking & Performance Results
- Adjustment Attempt
  - Adjustment
  - Benchmarking & Performance Results

Parameter Settings

- **Operand size:**

  We perform **256-bit × 256-bit** multiplication.

- **Representation (Limbs = 8):**

  Each operand is represented using `uint32_t` **× 8** limbs (32 bits × 8 = 256 bits).

  The result after multiplication has **16 limbs** (512 bits) before reduction.

- ~~**Modulus:**~~

  ~~We reduce the result modulo:~~

  $$~~P = 2^{256} - 1~~$$

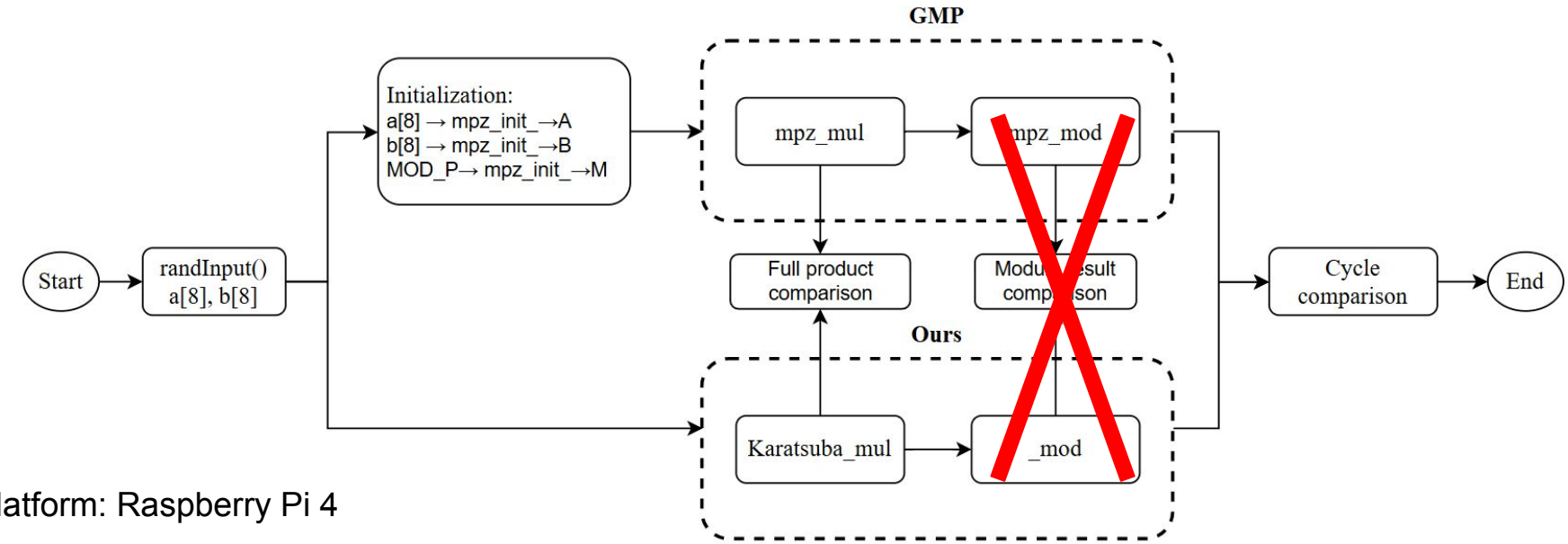
  ~~which allows fast modular reduction via wrap-around addition.~~

  ~~**MOD_P for GMP:**~~

  ~~The same modulus $2^{256} - 1$ is imported into GMP as:~~

```
~~uint32_t MOD_P[LIMBS] = { 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,~~
~~                          0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF };~~
```

System Architecture



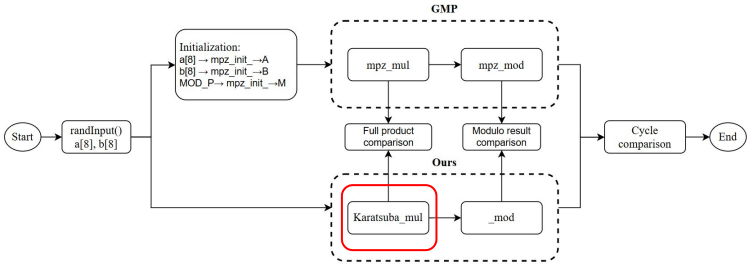
Platform: Raspberry Pi 4

Programming language: C language

Cycle count measured over:

- 500 total tests
- 50 warm-up rounds + 300 iterations per test

Karatsuba Multiplication



- **Input operands**:

  Two 256-bit numbers, each split as

  $$a = a_0 \,||\, a_1, \quad b = b_0 \,||\, b_1$$

  , where each part is 128 bits (4 limbs × 32 bits).

- **Algorithm steps**:

  a. Compute:

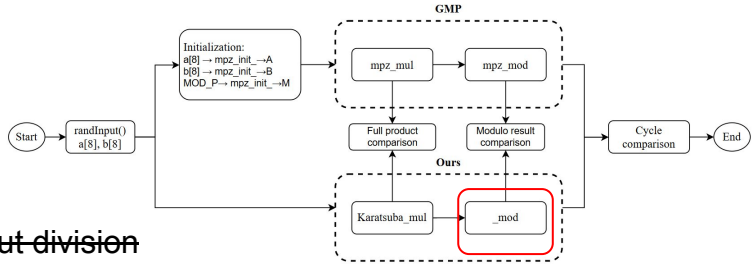
  $$z_0 = a_0 . \, b_0$$
  $$z_2 = a_1 . \, b_1$$
  $$z_1 = (a_0 + a_1)(b_0 + b_1) - z_0 - z_2$$

  b. Combine: result = $z_0 + (z_1 << 128) + (z_2 << 256)$

- **Implementation note**:

  We used schoolbook multiplication for 128×128-bit multiplication and a custom `add_n` and `sub_n` for combining intermediate terms.

Modular Reduction for $2^{256}-1$



- **Why this modulus**:

  2^k-1 primes enable fast reduction via *wrap-around addition* without division

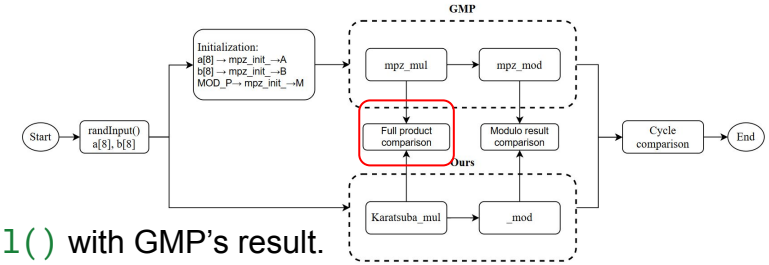
- **Algorithm steps**:

  a. Let the raw product be 512 bits = `x[16]` (`uint32_t`):
  b. Add the upper 256 bits into the lower 256 bits:

```
uint64_t carry = 0;
for (int i = 0; i < 8; ++i) {
    uint64_t sum = (uint64_t)x[i] + x[i + 8] + carry;
    x[i] = (uint32_t)sum;
    carry = sum >> 32;
}
```

  c. Carry propagation

Testing Strategy (1/2)



- Full Product Check

  Compare the full 512-bit product from our `karatsuba_512_mul()` with GMP's result.

  a. Convert our result into GMP format:

     mpz_import(RAW_KARA, 16, -1, sizeof(uint32_t), 0, 0, raw_result);

  b. Compute the GMP reference result:

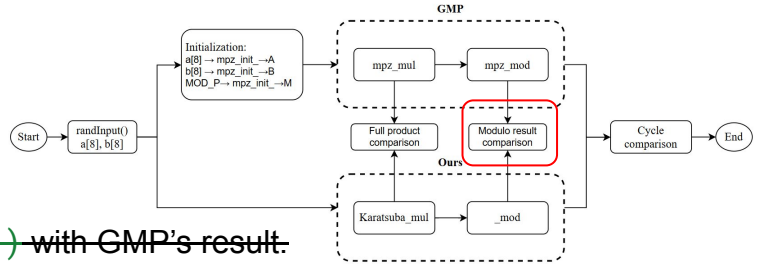
     mpz_mul(RAW_GMP, A, B);

  c. Compare both:

     mpz_cmp(RAW_KARA, RAW_GMP);

```
GMP FullProd : 657cee705877848b37824a79d431c2c5377303a0d69c76a01507751aff242a70294c6974f35813a129577308ecd51a98da716a9143d942fcd1a5556bae56af60
Raw Karatsuba: 657cee705877848b37824a79d431c2c5377303a0d69c76a01507751aff242a70294c6974f35813a129577308ecd51a98da716a9143d942fcd1a5556bae56af60
Full product matches: Karatsuba is correct

GMP mod P: 8ec957e54bcf982c60d9bd82c106dd5e11e46e321a75b99ce6acca86ad7ad9d0
Our modP: 8ec957e54bcf982c60d9bd82c106dd5e11e46e321a75b99ce6acca86ad7ad9d0
Mod result matches: mod_2to256_minus1 is correct
```

Testing Strategy (2/2)



- ~~Modular Reduction Check~~

  ~~Compare the 256-bit reduction from our `mod_2to256_minus1()` with GMP's result.~~

  a. ~~After reduction with `mod_2to256_minus1()`, we compare it against GMP:~~

     ~~**mpz_mod(R, R, M);** where `M = 2^256 − 1`.~~

  b. ~~Our reduced result (`result[8]`) is imported:~~

     ~~**mpz_import(TMP, 8, -1, sizeof(uint32_t), 0, 0, result);**~~

  c. ~~Compare both:~~

     ~~**mpz_cmp(TMP, R);**~~

```
GMP FullProd : 657cee705877848b37824a79d431c2c5377303a0d69c76a01507751aff242a70294c6974f35813a129577308ecd51a98da716a9143d942fcd1a5556bae56af60
Raw Karatsuba: 657cee705877848b37824a79d431c2c5377303a0d69c76a01507751aff242a70294c6974f35813a129577308ecd51a98da716a9143d942fcd1a5556bae56af60
Full product matches: Karatsuba is correct

GMP mod P: 8ec957e54bcf982c60d9bd82c106dd5e11e46e321a75b99ce6acca86ad7ad9d0
Our modP: 8ec957e54bcf982c60d9bd82c106dd5e11e46e321a75b99ce6acca86ad7ad9d0
Mod result matches: mod_2to256_minus1 is correct
```

Adjustment Attempt:Optimizing 4×4 Multiplication

- Karatsuba algorithm repeatedly invokes two 4×4 multiplications:

$$z_0 = a_0 . b_0, \qquad z_2 = a_1 . b_1$$

- Attempt to rewrite them in **aarch64 assembly** for performance gain

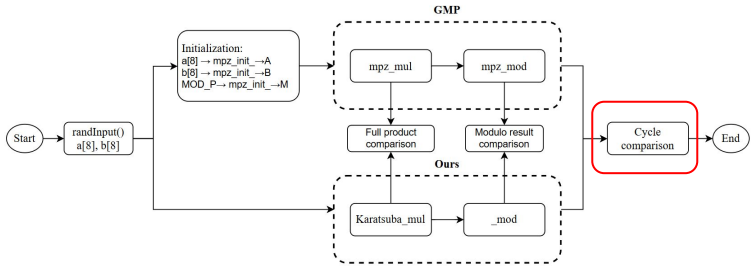
```c
static void schoolbook_256_mul(const uint32_t *a, const uint32_t *b, uint32_t *res)
{
    memset(res, 0, 8 * sizeof(uint32_t));
    for (int i = 0; i < 4; i++)
    {
        uint64_t carry = 0;
        for (int j = 0; j < 4; j++)
        {
            uint64_t sum = (uint64_t)a[i] * b[j] + res[i + j] + carry;
            res[i + j] = (uint32_t)sum;
            carry = sum >> 32;
        }
        int k = i + 4;
        while (carry)
        {
            uint64_t sum = (uint64_t)res[k] + carry;
            res[k++] = (uint32_t)sum;
            carry = sum >> 32;
        }
    }
}
```

```asm
outer_loop:
    cmp     w19, #4
    bge     done

    ldr     w3, [x0, w19, UXTW #2]   // a[i]
    mov     x20, #0                  // carry = 0

    // j = 0
    ldr     w4, [x1]                 // b[0]
    add     w5, w19, #0
    add     x5, x2, w5, UXTW #2
    ldr     w6, [x5]
    umull   x7, w3, w4
    add     x7, x7, x6, UXTW
    adds    x7, x7, x20
    str     w7, [x5]
    lsr     x20, x7, #32

    // j = 1, 2, 3
    ...

    // propagate carry
    add     w5, w19, #4

carry_loop:
    cmp     x20, #0
    beq     next_i

    add     x6, x2, w5, UXTW #2
    ldr     w7, [x6]
    add     x7, x20, x7, UXTW
    str     w7, [x6]
    lsr     x20, x7, #32
    add     w5, w5, #1
    b       carry_loop
```

Performance Results(Pure multiplication)

Performance Summary

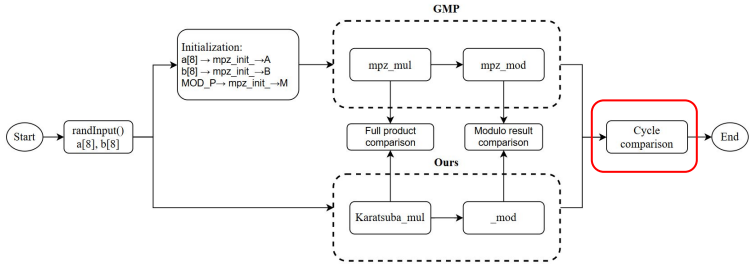
Implementation	Median Cycles	Speedup
GMP	741	1×
Karatsuba-C	419	**1.77× faster**
Karatsuba-Assembly 4x4	578	**1.28× faster**



Clock time comparison

Trial	Ⅰ			Ⅱ			Ⅲ			Avg.
Time used (ns)	Z_0	Z_2	Sum	Z_0	Z_2	Sum	Z_0	Z_2	Sum	
Karatsuba-C	519	167	686	148	93	241	463	111	574	500.3
Karatsuba-Assembly 4x4	407	185	592	556	185	741	500	204	704	679

Performance Results

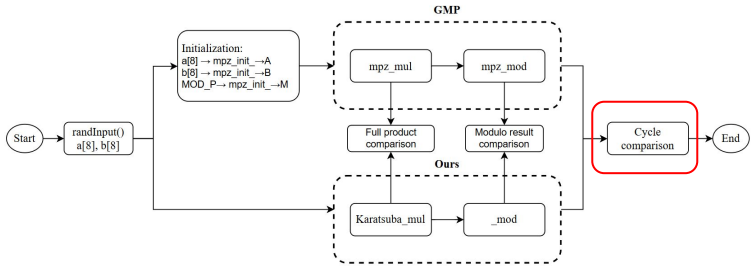



```
========= mpz_mul ==========
   mpz_mul cycles = 741
           percentile      1     10     20     30     40     50     60     70     80     90     99
   mpz_mul percentiles:    735    735    740    741    741    741    741    741    741    741    749

========= your_mul (Karatsuba with pure C) ==========
   Our_mul cycles = 419
           percentile      1     10     20     30     40     50     60     70     80     90     99
   Our_mul percentiles:    418    418    419    419    419    419    419    420    420    420    422

[Result] mpz / karatsuba_C: 1.766317
========= your_mul (Karatsuba with partial assembly) ==========
   Our_mul cycles = 578
           percentile      1     10     20     30     40     50     60     70     80     90     99
   Our_mul percentiles:    578    578    578    578    578    578    578    578    578    578    580

[Result] mpz / karatsuba_asm: 1.281684

====
GMP FullProd : 7060d822b10d9ba338791266d3ad8aa8e46ed9a1661a3e1d9b84444fbfc8bbbf8047fa98002d6f6de77cc6af66a0cecf109e31ee8cb00b91739901d4bbf6dc60
Raw Karatsuba: 7060d822b10d9ba338791266d3ad8aa8e46ed9a1661a3e1d9b84444fbfc8bbbf8047fa98002d6f6de77cc6af66a0cecf109e31ee8cb00b91739901d4bbf6dc60
Full product matches: Karatsuba is correct
```

Performance Results(including mod)

Performance Summary

Implementation	Median Cycles	Speedup
GMP	1185	1×
Karatsuba	443	**2.67× faster**



Percentile Breakdown

Percentile	1	10	20	50	90	99
GMP	1174	1179	1180	1185	1185	1198
Karatsuba	443	443	443	443	443	445

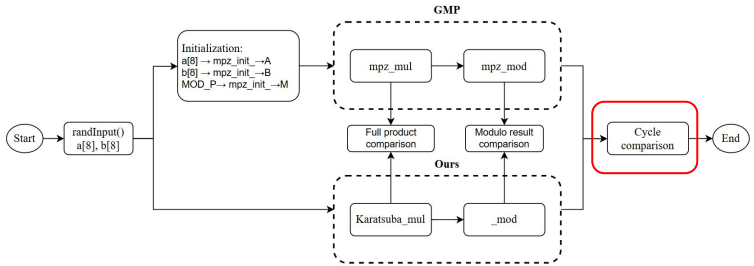
Correctness Verification

- **Full Product** : Karatsuba output = GMP output
- **Modulo Result :** Karatsuba mod 2^{256}-1 = GMP mod

Performance Results(including mod)

Performance Summary

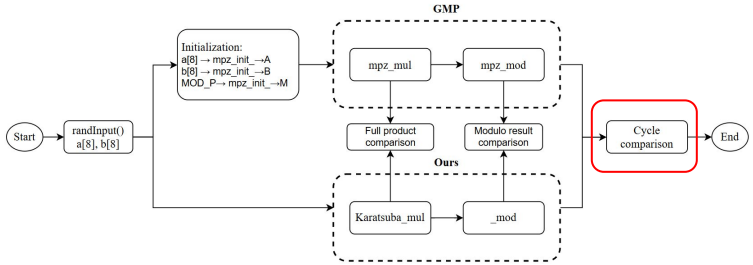
Implementation	Median Cycles	Speedup
GMP	1185	1×
Karatsuba-C	443	**2.67× faster**
Karatsuba-Assembly 4x4	611	**1.94× faster**



Clock time comparison

Trial	Ⅰ			Ⅱ			Ⅲ			Avg.
Time used (ns)	Z_0	Z_2	Sum	Z_0	Z_2	Sum	Z_0	Z_2	Sum	
Karatsuba-C	519	167	686	148	93	241	463	111	574	500.3
Karatsuba-Assembly 4x4	407	185	592	556	185	741	500	204	704	679

Performance Results




```
========= mpz_mul ==========
   mpz_mul cycles = 1185
          percentile      1     10     20     30     40     50     60     70     80     90     99
   mpz_mul percentiles:   1174   1179   1180   1180   1185   1185   1185   1185   1185   1185   1198

========= your_mul (Karatsuba with pure C) ==========
   Our_mul cycles = 443
          percentile      1     10     20     30     40     50     60     70     80     90     99
   Our_mul percentiles:    443    443    443    443    443    443    443    443    443    443    445

========= your_mul (Karatsuba with partial assembly) ==========
   Our_mul cycles = 611
          percentile      1     10     20     30     40     50     60     70     80     90     99
   Our_mul percentiles:    610    610    610    610    611    611    611    611    611    611    613

GMP FullProd : 6e115495b62dbcce0a7c38042dc74fb08da7f261f8de8d683287b0ed80fc26aefe5296b4d41463c1ea50b21a1447b65175947be599be118e0e20f27ded5a5b3f
Raw Karatsuba: 6e115495b62dbcce0a7c38042dc74fb08da7f261f8de8d683287b0ed80fc26aefe5296b4d41463c1ea50b21a1447b65175947be599be118e0e20f27ded5a5b3f
Full product matches: Karatsuba is correct

GMP mod P: 6c63eb4a8a42208ff4ccea1e420f0602033c6e47929c9ef640a8a36b6e5681ee
Our mod P: 6c63eb4a8a42208ff4ccea1e420f0602033c6e47929c9ef640a8a36b6e5681ee
Mod result matches: mod_2to256_minus1 is correct
```

Thank you for your attention :)