Fast AVX-512 Implementation of the Optimal Ate Pairing on BLS12-381

Hao Cheng[1] Georgios Fotiadis[2,3] Johann Großschädl[3] Daniel Page[4]

[1]Shandong University [2]nil; Foundation [3]University of Luxembourg [4]University of Bristol

CHES 2025

Intel AVX-512IFMA

Integer Fused Multiply-Add (IFMA)

- SIMD on 512-bit vectors: 64-bit elements $\times$ 8 lanes
- Sub-extension of AVX-512: `vpmadd52luq` & `vpmadd52huq`
  1. $t_H \parallel t_L \leftarrow a \times b$ — 104-bit $\leftarrow$ 52-bit $\times$ 52-bit
  2. $r \leftarrow t_{[L|H]} + c$ — 64-bit $\leftarrow$ 52-bit $+$ 64-bit
- Intel to integrate 512-bit IFMA instructions to all future CPUs
  Intel AVX10 (Revision 3.0), March 2025

We target the Intel Ice Lake Core processor (Sunny Cove microarchitecture)

- Intel Core i3-1005G1 CPU clocked at 1.2 GHz
- IFMA instruction (512-bit `zmm` registers): CPI of 1, latency of 4 clock cycles

Pairings in the market

Popular pairing-based schemes

- BLS short signature scheme
- KZG polynomial commitments for zkSNARKs
- Groth16 ZK proof systems
- Advanced PKE schemes: IBE, ABE, WE, etc.

Focus on optimal ate pairing on BLS12-381

- Included in the IETF reference specification
- Used in Ethereum beacon chain protocol
- Used in Ethereum KZG commitments
- Zcash upgrades to BLS12-381

ethereum

Zcash

Optimal ate pairing on BLS12-381 curve

BLS12-381 curve

- Member of BLS12 family of curves with $k = 12$ and seed $z = -0\text{xd}201000000010000$
- Elliptic curve $E/\mathbb{F}_p : y^2 = x^3 + 4$, with 381-bit prime p
- Composite order: $\#E(\mathbb{F}_p) = h \cdot r$, with $\log_2(r) = 255$ bits
- Admits degree 6 twist $E'/\mathbb{F}_{p^2} \Longrightarrow$ efficient optimal ate pairing

Optimal ate pairing: two points $Q \in E'(\mathbb{F}_{p^2})$, $P \in E(\mathbb{F}_p)$ of order r

$$e(Q, P) \to f^{(p^{12}-1)/r} \in \mathbb{F}_{p^{12}}^*$$

- Miller loop + final exponentiation
- Arithmetic in $\mathbb{F}_{p^{12}}$: multiplication, squaring, cyclotomic squaring, sparse multiplication
- Tower extensions: $\mathbb{F}_p \to \mathbb{F}_{p^2} \to \mathbb{F}_{p^6} \to \mathbb{F}_{p^{12}}$, $\mathbb{F}_p \to \mathbb{F}_{p^2} \to \mathbb{F}_{p^4} \to \mathbb{F}_{p^{12}}$
- ECC arithmetic: point doubling/addition, line computation/evaluation

Optimal ate pairing on BLS12-381 curve

Algorithm: Miller Loop

Input: $|z| = (b_n, b_{n-1}, \ldots, b_0)_2$, P, Q.
Output: Miller function f.
$T \leftarrow Q$, $f \leftarrow 1$
for $i \leftarrow (n-1)$ **down to** 0 **do**
  $f \leftarrow f^2 \cdot \ell_{T,T}(P)$, $T \leftarrow [2]T$ /* Doubling step: point doubling, line comp/eval, Miller function update */
  if $b_i = 1$ **then**
    $f \leftarrow f \cdot \ell_{T,Q}(P)$, $T \leftarrow T + Q$ /* Addition step: point addition, line comp/eval, Miller function update */
return $f \leftarrow f^{-1}$ /* Inversion of Miller function: seed negative */

Final exponentiation: compute $f^{(p^{12}-1)/r}$

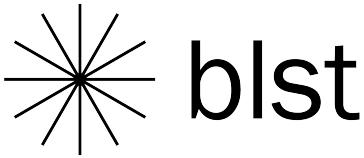
- Split computation into easy part and hard part

$$(p^{12}-1)/r \;=\; \underbrace{(p^6-1)(p^2+1)}_{\text{easy part}}\underbrace{(p^4-p^2+1)/r}_{\text{hard part}}$$

- Most efficient formula for computing the hard part

$$3\cdot(p^4-p^2+1)/r \;=\; (z-1)^2\cdot(z+p)\cdot(z^2+p^2-1)+3$$

Baseline implementation

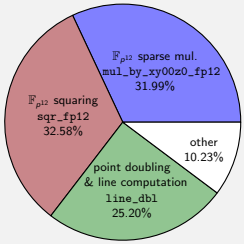


The `blst` (pronounced "blast") library

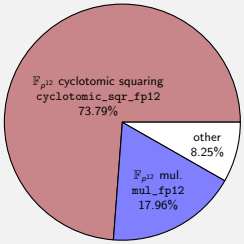
- Multilingual BLS12-381 signature library
- Currently integrated in Ethereum
- Baseline: x64 assembly impl. of the optimal ate pairing

Target operations (based on profiling)

1. $\mathbb{F}_{p^{12}}$ cyclotomic squaring
2. $\mathbb{F}_{p^{12}}$ multiplication
3. $\mathbb{F}_{p^{12}}$ sparse multiplication
4. $\mathbb{F}_{p^{12}}$ squaring
5. point doubling & line computation
6. point addition & line computation



Miller loop



Final exponentiation

Another scalar implementation

Longa's implementation (CHES 2023)

- Based on the `RELIC` library
- Slightly faster than `blst` (1.07×)
- Using the compressed cyclotomic squaring in $\mathbb{F}_{p^{12}}$

$\mathbb{F}_{p^{12}}$ cyclotomic squaring: $\gamma = \alpha^2 = (a + bw + cw^2)^2 = A + Bw + Cw^2$

- Uncompressed (Granger-Scott): $A = 3a^2 - 2\bar{a}$, $B = 3c^2v + 2\bar{b}$, $C = 3b^2 - 2\bar{c}$
- Compressed (Aranha et al.): $B = 3c^2v + 2\bar{b} = B_0 + B_1 v$, $C = 3b^2 - 2\bar{c}$
  - For consecutive squarings in computing $g^{|z|}$
  - Parallel decompressions (requiring only one $\mathbb{F}_{p^2}$ inversion)
  - Different formulas depending on whether $B_0 \neq 0$ or $B_0 = 0$ (negligible probability)
  - `blst` becomes 1.08× after using it

Vectorization: $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ operations

- Based on our previous work `avxsike` (CHES 2022)
- Necessary modifications: 48-bit-per-limb representation, prime p of BLS12-381, etc.
- (8×1)-way and (4×2)-way $\mathbb{F}_p$ implementations
- $(4 \times 2 \times 1)$-way, $(2 \times 4 \times 1)$-way, $(2 \times 2 \times 2)$-way, and $(1 \times 4 \times 2)$-way $\mathbb{F}_{p^2}$ implementations
- Example: $\mathbb{F}_{p^2}$ multiplication $r = r_0 + r_1 u = (a_0 + a_1 u) \cdot (b_0 + b_1 u) = (a_0 b_0 - a_1 b_1) + (a_0 b_1 + a_1 b_0) u$

2-way vectorized $\mathbb{F}_{p^2}$ multiplication

1	$tt_0 \leftarrow a_0 \times b_1$	$ss_0 \leftarrow a_0 \times b_0$
2	$tt_1 \leftarrow a_1 \times b_0$	$ss_1 \leftarrow a_1 \times b_1$
3	$r_1 \leftarrow tt_0 + tt_1$	$r_0 \leftarrow ss_0 - ss_1$

4-way vectorized $\mathbb{F}_{p^2}$ multiplication

1	$tt_0 \leftarrow a_0 \times b_1$	$ss_0 \leftarrow a_0 \times b_0$	$tt_1 \leftarrow a_1 \times b_0$	$ss_1 \leftarrow a_1 \times b_1$
2	$r_1 \leftarrow tt_0 + tt_1$	$r_0 \leftarrow ss_0 - ss_1$		

Vectorization: $\mathbb{F}_{p^{12}}$ cyclotomic squaring (`cyclotomic_sqr_fp12`)

$$A = 3a^2 - 2\bar{a}, \quad B = 3c^2 v + 2\bar{b}, \quad C = 3b^2 - 2\bar{c}$$

```
/**
 * Input:  a   = a1 | a0             in (2x2x2)-way
 * Input:  bc  = c1 | c0 | b1 | b0   in (4x2x1)-way
 * Output: ra  = A1 | A0             in (2x2x2)-way
 * Output: rbc = C1 | C0 | B1 | B0   in (4x2x1)-way
 */

/* Compute A in (1x2x2x2)-way */
sqr_fp4_1x2x2x2w(ta, a);          //         t1 |         t0
as_fp2_2x2x2w(ra, ta, a);         //     t1+a1 |     t0-a0
add_fp2_2x2x2w(ra, ra, ra);       // 2*(t1+a1) | 2*(t0-a0)
add_fp2_2x2x2w(ra, ra, ta);       // 3*t1+2*a1 | 3*t0-2*a0

/* Compute B and C simultaneously in (2x2x2x1w)-way */
sqr_fp4_2x2x2x1w(tbc, bc);        //         t5 |         t4 |         t3 |               t2
mul_by_up1_fp2_4x2x1w(tmp, tbc); //  t5*(u+1) |  t4*(u+1) |  t3*(u+1) |        t2*(u+1)
// Transformation to make tbc =             t3 |         t2 |         t4 |        t5*(u+1)
assa_fp2_4x2x1w(rbc, tbc, bc);    //     t3+c1 |     t2-c0 |     t4-b1 |     t5*(u+1)+b0
add_fp2_4x2x1w(rbc, rbc, rbc);    // 2*(t3+c1) | 2*(t2-c0) | 2*(t4-b1) | 2*(t5*(u+1)+b0)
add_fp2_4x2x1w(rbc, rbc, tbc);    // 3*t3+2*c1 | 3*t2-2*c0 | 3*t4-2*b1 | 3*t5*(u+1)+2*b0
```

Vectorization: $\mathbb{F}_{p^4}$ squaring

"1-way + 2-way" vectorized $\mathbb{F}_{p^4}$ squaring

1	$tt_0 \leftarrow a_1^2$	$ss_0 \leftarrow a_0^2$
2	$tt_0 \leftarrow tt_0 \cdot (u+1)$	
3	$tt_1 \leftarrow a_0 \cdot a_1$	
4	$tt_1 \leftarrow tt_1 + tt_1$	$ss_0 \leftarrow ss_0 + tt_0$
5	$r_1 \leftarrow tt_1 \bmod p$	$r_0 \leftarrow ss_0 \bmod p$

$$r = r_0 + r_1 v = (a_0 + a_1 v)^2 = (a_0^2 + a_1^2(u+1)) + 2a_0 a_1 v$$

`sqr_fp4_2x2x2x1w` $\leftarrow (4 \times 2 \times 1)$-way $\mathbb{F}_{p^2}$ squaring + $(2 \times 4 \times 1)$-way $\mathbb{F}_{p^2}$ multiplication

`sqr_fp4_1x2x2x2w` $\leftarrow (2 \times 2 \times 2)$-way $\mathbb{F}_{p^2}$ squaring + $(1 \times 4 \times 2)$-way $\mathbb{F}_{p^2}$ multiplication

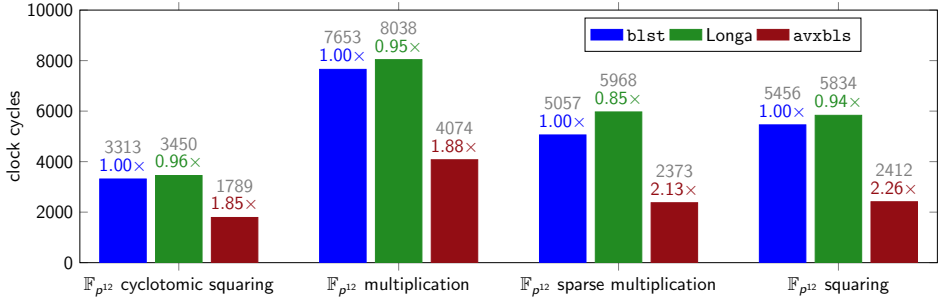
Vectorization: $\mathbb{F}_{p^{12}}$ multiplication

$$
\begin{aligned}
\gamma &= \alpha \cdot \beta = (a + a'w) \cdot (b + b'w) \\
&= (ab + a'b'v) + (ab' + a'b)w \\
&= (\underline{ab} + a'b'v) + ((a + a')(b + b') - ab - a'b')w \\
&\quad \Downarrow \\
r &= a \cdot b = (a_0 + a_1 v + a_2 v^2) \cdot (b_0 + b_1 v + b_2 v^2) \\
&= r_0 + r_1 v + r_2 v^2 \\
r_0 &= ((a_1 + a_2)(b_1 + b_2) - a_1 b_1 - a_2 b_2)(u + 1) + a_0 b_0 \\
r_1 &= (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1 + a_2 b_2 (u + 1) \\
r_2 &= (a_0 + a_2)(b_0 + b_2) - a_0 b_0 - a_2 b_2 + a_1 b_1
\end{aligned}
$$

2-way vectorized $\mathbb{F}_{p^6}$ multiplication

1	$t_0 \leftarrow b_1 + b_2$	$s_0 \leftarrow a_1 + a_2$
2	$t_1 \leftarrow b_0 + b_1$	$s_1 \leftarrow a_0 + a_1$
3	$t_2 \leftarrow b_0 + b_2$	$s_2 \leftarrow a_0 + a_2$
4	$tt_0 \leftarrow a_1 \cdot b_1$	$ss_0 \leftarrow a_0 \cdot b_0$
5	$tt_1 \leftarrow a_2 \cdot b_2$	$ss_1 \leftarrow s_0 \cdot t_0$
6	$tt_2 \leftarrow s_1 \cdot t_1$	$ss_2 \leftarrow s_2 \cdot t_2$
7	$tt_2 \leftarrow tt_2 - tt_0$	$ss_1 \leftarrow ss_1 - tt_0$
8	$ss_2 \leftarrow ss_2 - tt_1$	$ss_1 \leftarrow ss_1 - tt_1$
9	$tt_2 \leftarrow tt_2 - ss_0$	$ss_2 \leftarrow ss_2 - ss_0$
10	$tt_1 \leftarrow tt_1 \cdot (u + 1)$	$ss_1 \leftarrow ss_1 \cdot (u + 1)$
11	$r_1 \leftarrow tt_1 + tt_2$	$r_0 \leftarrow ss_0 + ss_1$
12	$r_2 \leftarrow ss_2 + tt_0$	

Benchmark: $\mathbb{F}_{p^{12}}$ operations



- Our vectorized software library `avxbls`
- Speed-up factors ranging from 1.85 to 2.26
- "1-way + 2-way" hybrid vectorization used in $\mathbb{F}_{p^{12}}$ cyclotomic squaring and $\mathbb{F}_{p^{12}}$ multiplication
- Some subroutines still non-vectorized due to code dependency (e.g., $\mathbb{F}_{p^2}$ inversion)

Vectorization: point doubling & line computation (`line_dbl`)

Doubling step: point doubling + line computation + line evaluation + update of the Miller function

$$R = (X_R, Y_R, Z_R) = \left(9X_T^4 - 8X_T Y_T^2,\ 3X_T^2(12X_T Y_T^2 - 9X_T^4) - 8Y_T^4,\ 2Y_T Z_T\right)$$

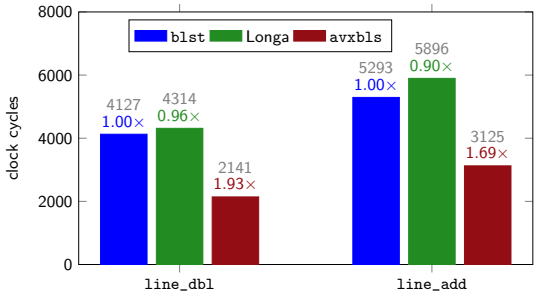
$$\ell_{T,T} = (\ell_0, \ell_1, \ell_2) = \left(6X_T^3 - 4Y_T^2,\ 3X_T^2 Z_T^2,\ 2Y_T Z_T^3\right)$$

`line_dbl`

- Scalar: $3M_2 + 8S_2 + 21A_2$
- 2-way vectorization: $2M_2^2 + 4S_2^2 + 11A_2^2$
- 4-way vectorization: $1M_2^4 + 2S_2^4 + 9A_2^4$ ✔

`line_add`

- Scalar: $9M_2 + 4S_2 + 16A_2$
- 2-way vectorization: $6M_2^2 + 1S_2^2 + 8A_2^2$ ✔



Vectorization: point doubling & line computation (`line_dbl`)

2-way vectorized `line_dbl`

1	$B \leftarrow Y_T^2$	$A \leftarrow X_T^2$
2	$t_0 \leftarrow X_T + B$	$s_0 \leftarrow A + A$
3	$t_0 \leftarrow t_0^2$	$C \leftarrow B^2$
4	$t_0 \leftarrow t_0 - A$	$E \leftarrow s_0 + A$
5	$t_0 \leftarrow t_0 - C$	$s_0 \leftarrow E + X_T$
6	$ZZ \leftarrow Z_T^2$	$F \leftarrow E^2$
7	$D \leftarrow t_0 + t_0$	$s_1 \leftarrow Y_T + Z_T$
8	$s_0 \leftarrow s_0^2$	$s_1 \leftarrow s_1^2$
9	$t_0 \leftarrow F - D$	$s_1 \leftarrow s_1 - B$
10	$X_R \leftarrow t_0 - D$	$s_0 \leftarrow s_0 - A$
11	$Z_R \leftarrow s_1 - ZZ$	$s_2 \leftarrow C + C$
12	$s_2 \leftarrow s_2 + s_2$	$s_1 \leftarrow B + B$
13	$s_2 \leftarrow s_2 + s_2$	$s_1 \leftarrow s_1 + s_1$
14	$t_0 \leftarrow D - X_R$	$s_0 \leftarrow s_0 - F$
15	$t_0 \leftarrow t_0 \cdot E$	$\ell_1 \leftarrow ZZ \cdot E$
16	$Y_R \leftarrow t_0 - s_2$	$\ell_0 \leftarrow s_0 - s_1$
17	$\ell_2 \leftarrow Z_R \cdot ZZ$	

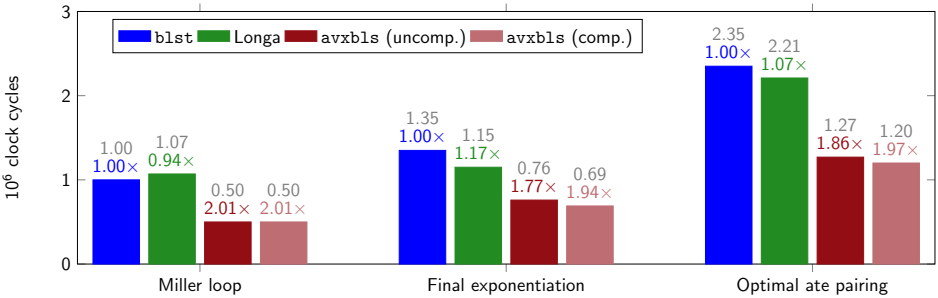
4-way vectorized `line_dbl`

1	$t_0 \leftarrow Y_T + Z_T$			
2	$B \leftarrow Y_T^2$	$A \leftarrow X_T^2$	$ZZ \leftarrow Z_T^2$	$s_0 \leftarrow t_0^2$
3	$t_1 \leftarrow X_T + B$	$s_1 \leftarrow A + A$	$t_2 \leftarrow B + B$	$s_2 \leftarrow A + X_T$
4	$t_0 \leftarrow s_1 + s_2$	$E \leftarrow s_1 + A$	$t_2 \leftarrow t_2 + t_2$	
5	$t_1 \leftarrow t_1^2$	$C \leftarrow B^2$	$t_0 \leftarrow t_0^2$	$F \leftarrow E^2$
6	$t_1 \leftarrow t_1 + t_1$	$s_2 \leftarrow C + C$	$t_0 \leftarrow t_0 - A$	$s_0 \leftarrow s_0 - B$
7	$t_1 \leftarrow t_1 - s_2$	$s_2 \leftarrow s_2 + s_2$	$t_0 \leftarrow t_0 - F$	$Z_R \leftarrow s_0 - ZZ$
8	$D \leftarrow t_1 - s_1$	$s_2 \leftarrow s_2 + s_2$	$\ell_0 \leftarrow t_0 - t_2$	
9	$t_0 \leftarrow F - D$	$s_0 \leftarrow D - F$	$t_1 \leftarrow D + D$	
10	$X_R \leftarrow t_0 - D$	$s_0 \leftarrow s_0 + t_1$		
11	$s_0 \leftarrow s_0 \cdot E$	$\ell_1 \leftarrow ZZ \cdot E$	$\ell_2 \leftarrow Z_R \cdot ZZ$	
12	$Y_R \leftarrow s_0 - s_2$			

$$2M_2^2 + 4S_2^2 \rightarrow 1M_2^4 + 2S_2^4$$

(4×2)-way $\mathbb{F}_p$ impl. $\rightarrow$ (8×1)-way $\mathbb{F}_p$ impl.

Benchmark: pairing



- AVX-512 vectorization offers significant performance gain
- 1.86× speed-up for the optimal ate pairing on BLS12-381
- Speed-up further improved to 1.97× with the use of $\mathbb{F}_{p^{12}}$ compressed cyclotomic squaring

Comparison: avxbls vs avxsike

avxsike library

- AVX-512IFMA implementations of SIKE (isogeny-based)
- Developed also by us, benchmarked on the same CPU
- Higher speed-ups (e.g., 3.21× for Encaps and 2.45× for Decaps)

Why

- $\mathbb{F}_{p^{12}}$ tower extension needs more field reductions compared to $\mathbb{F}_{p^2}$ arithmetic
- SIKE allows better parallelization at higher layers
  - For example, $(8 \times 1 \times 1 \times 1)$-way vectorization for 3-isogeny evaluations
  - 8-way parallelization only possible at the $\mathbb{F}_p$ layer in avxbls

Concluding remarks

Summary

- AVX-512 is very useful for the pairing computation
- 8 parallel lanes seem to be an ideal setup
- For example, 16 lanes would result in many lanes being idle
- Our vectorization strategies can be adapted to other vector extensions
- Source code: `https://github.com/hchengv/avxbls`

Future work

- To look at the complete BLS signature
- To research optimal "vectorization plus multi-processing" strategies

Thank you for your attention!