Towards Closing the Performance Gap for Cryptographic Kernels Between CPUs and Specialized Hardware


Naifeng Zhang

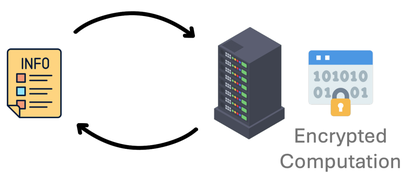

Sophia Fu


Franz Franchetti

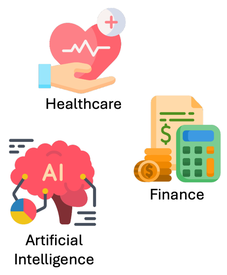
Carnegie Mellon University

I. MOTIVATION & BACKGROUND

Data Security Comes at a High Cost



Fully homomorphic encryption (FHE)



Cost: Prohibitive computational overhead

Operations with Large Integer Arithmetic

2,000-bit integer

29945058656012390289752201899548961002216938026898764670213921626494125002097316037309083423551230
90801402354741281525320130875756031797076046167772169925045685236286727961252942656722365196302071
16603324681021157993534366932467361582713800585917357036433047500540160547946659133404014756269446
50403492615533104586108008512036436642324307076750180870819991866737106993558380421464216659826172
30815293083164172838554002986172170145077240914636310277685115293248536940258787378968137743709119
53496219555899915324743599705479002269036442663447672197590995176588031386455684062123400247497680
58860138217379

32-bit 305,419,896

64-bit 17,920,781,457,621,970,241

128-bit 15,107,846,090,143,992,465,023,504,163,010,990,279

Residue Number System (RNS)

State-of-the-Art Solutions

Specialized hardware support on **application-specific integrated circuits (ASICs)**

Code generation-based approach on **GPUs**

- Multi-word modular arithmetic (MoMA)

Arbitrary precision **libraries** on **CPUs**

- GNU multiple precision (GMP) library
- OpenFHE MATHBACKEND

1,157x slowdown!

Q: HOW TO NARROW THE PERFORMANCE GAP?

II. APPROACH: SCALAR & SIMD

Part I: Modular Arithmetic

Math (over $\mathbb{Z}_q$)

$c = a + b \mod q$

$c = a - b \mod q$

$c = ab \mod q$

Algorithm

$$c = \begin{cases} a+b-q, & \text{if } (a+b) \geq q, \\ a+b, & \text{otherwise.} \end{cases}$$

$$c = \begin{cases} a-b+q, & \text{if } a < b, \\ a-b, & \text{otherwise.} \end{cases}$$

$c = ab - \lfloor ab\mu/2^k \rfloor q, \quad \mu = \lfloor 2^k/q \rfloor$

Barrett reduction

Part II: Double-Word Arithmetic

Double-word representation: $[x_0, x_1]_z = x_0 z + x_1 = x$ $\quad z = 2^{64}$

Examples

$[8,9]_{10} = 8 \cdot 10 + 9 = 89$

$[115292150460684697 5,18446744073709550897]_{2^{64}}$
$= 21267647932558653966460912964485512497$

Part I: Modular addition algorithm

$$c = \begin{cases} a+b-q, & \text{if } (a+b) > q, \\ a+b, & \text{otherwise.} \end{cases}$$

$a = [a_0, a_1]_z = a_0 z + a_1$

$b = [b_0, b_1]_z = b_0 z + b_1$

Part II: Double-Word addition

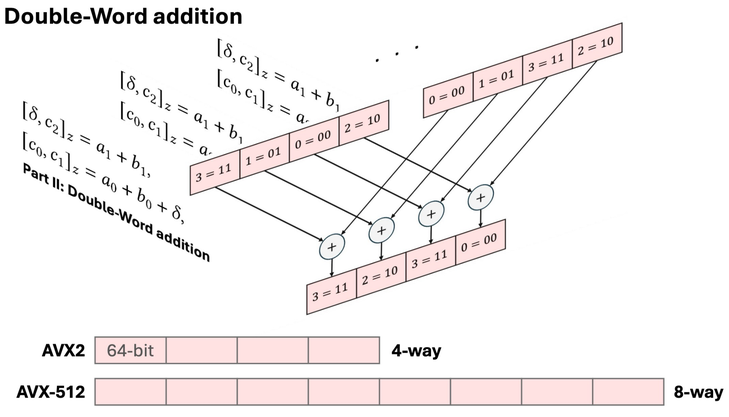
$[\delta, c_2]_z = a_1 + b_1,$

$[c_0, c_1]_z = a_0 + b_0 + \delta,$

where $c = [c_0, c_1, c_2]_z$ and $\delta \in \{0, 1\}$.

Part III: Single Instruction, Multiple Data

SIMD Double-Word addition



III. APPROACH: MULTI-WORD EXTENSION (MQX)

Instruction I: SIMD Addition with Carry

```
__m512i _mm512_adc_epi64(__m512i a, __m512i b, __mmask8 ci, __mmask8* co)
```

Per-lane 64-bit addition with carry-in and outputting both the addition result and carry-out

Similar instructions

ISA	Instruction
x86	ADC
LRBni	vadcpi
KNC	__m512i _mm512_adc_epi32 (__m512i v2, __mmask16 k2, __m512i v3, __mmask16* k2_res)

Instruction II: SIMD Subtraction with Borrow

```
__m512i _mm512_sbb_epi64(__m512i a, __m512i b, __mmask8 bi, __mmask8* bo)
```

Per-lane 64-bit subtraction with borrow-in and outputting the subtraction result with borrow-out

Similar instructions

ISA	Instruction
x86	SBB
LRBni	vsbbpi
KNC	__m512i _mm512_sbb_epi32 (__m512i v2, __mmask16 k, __m512i v3, __mmask16* borrow)

Instruction III: SIMD Widening Multiplication

```
void _mm512_mul_epi64(__m512i* ch, __m512i* cl, __m512i a, __m512i b)
```

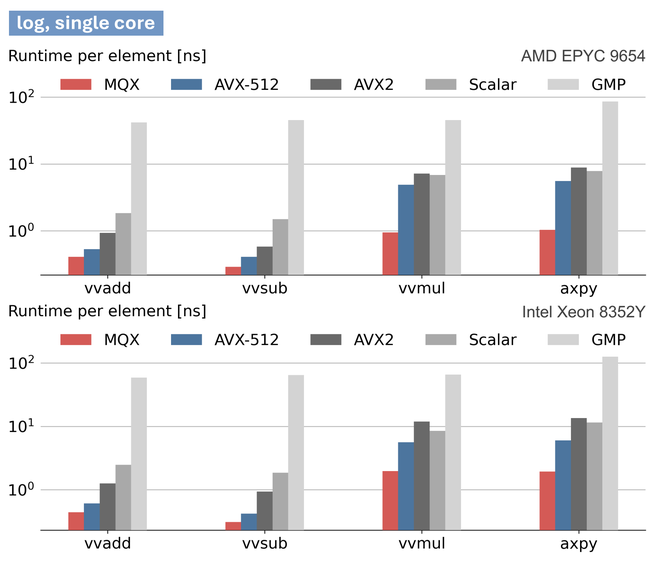
Per each SIMD lane, multiplying two 64-bit words and storing the high part of the result in one 64-bit word and the low part in another

Similar instructions

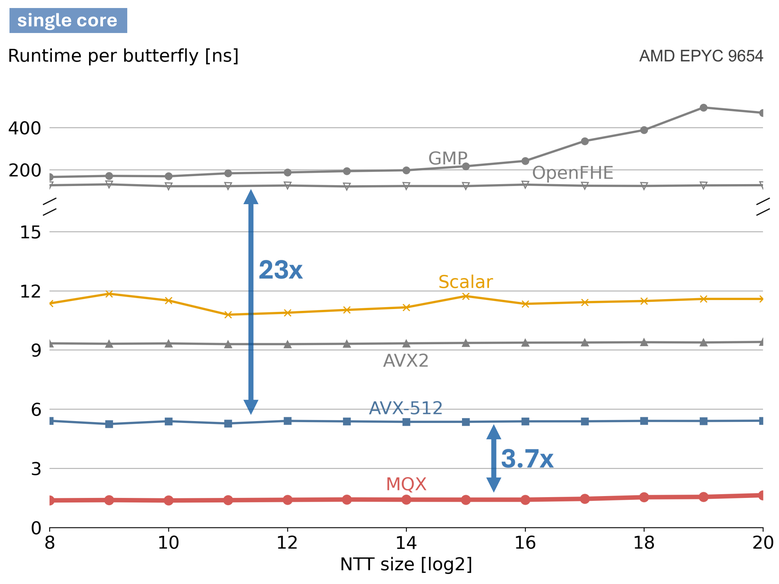
ISA	Instruction
x86	MUL
LRBni	vmulhpi
KNC	__m512i _mm512_mulhi_epi32 (__m512i a, __m512i b)
AVX-512	__m512i _mm512_mul_epi32 (__m512i a, __m512i b)

IV. RESULTS

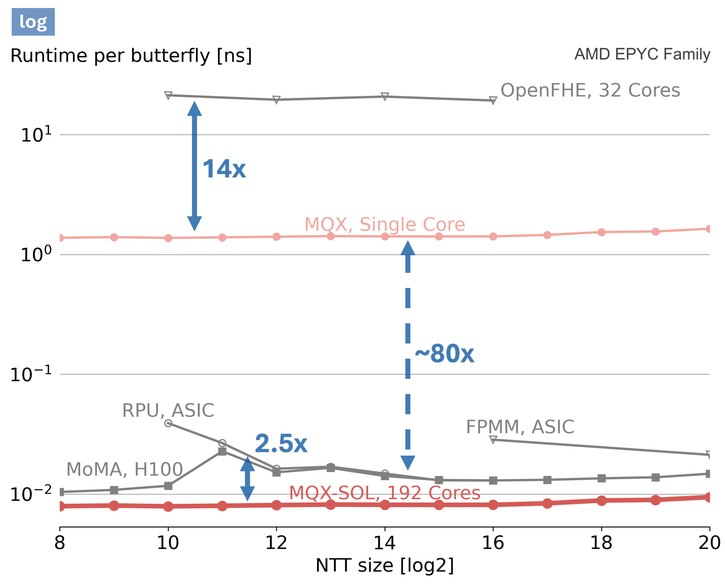
Kernel I: BLAS Operations



Kernel II: Number Theoretic Transform



Speed-of-Light Analysis



MQX ENABLES CPUS TO APPROACH ASIC-LEVEL PERFORMANCE

Code available at github.com/naifeng/benchntt
Reach us at naifengz@cmu.edu

Artifacts Evaluated Functional · Results Reproduced V1.1