WAHC 2021

Intel® HEXL: Accelerating Homomorphic Encryption with Intel AVX512-IFMA52

Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe Souza, Vinodh Gopal
Technical Lead, Intel Corporation

Agenda

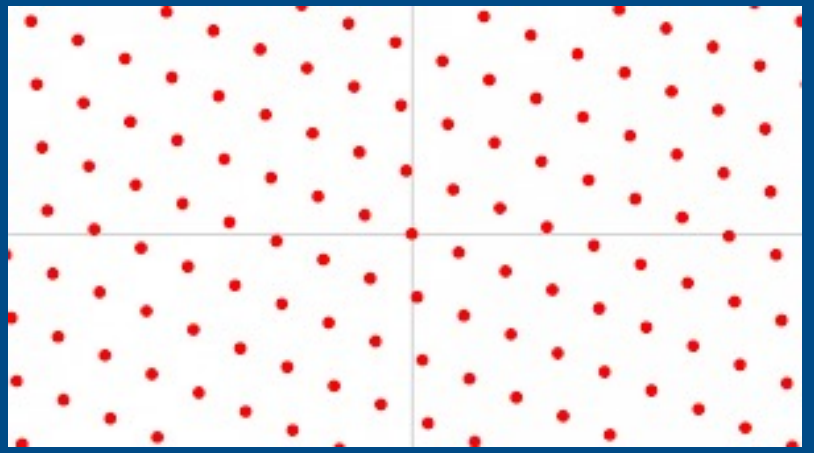
- Introduce Intel HEXL
  - Introduce Intel AVX512
- Describe HE kernel optimizations
- Show runtime speedup results

Intel HEXL- https://github.com/intel/hexl

- Homomorphic Encryption Acceleration Library
  - Open-source (Apache 2.0) C++ library
  - Uses Intel AVX512 to accelerate HE kernels
  - Adopted by leading HE libraries:
    - Microsoft SEAL
    - PALISADE
    - HElib
  - Supports Linux, Windows, Mac
  - Automatic detection of CPU features
    - Use AVX512-IFMA52 for best performance
  - Contributions welcome!

HE Background

- HE relies on lattice cryptography
  - Quantum-resistant!
  - Based on hardness of (ring) learning with errors
- HE schemes rely on modular arithmetic
  - $Z_q = \{0, 1, \dots, q-1\}$
  - HE polynomials live in quotient ring



https://www.esat.kuleuven.be/cosic/blog/introduction-to-lattices/

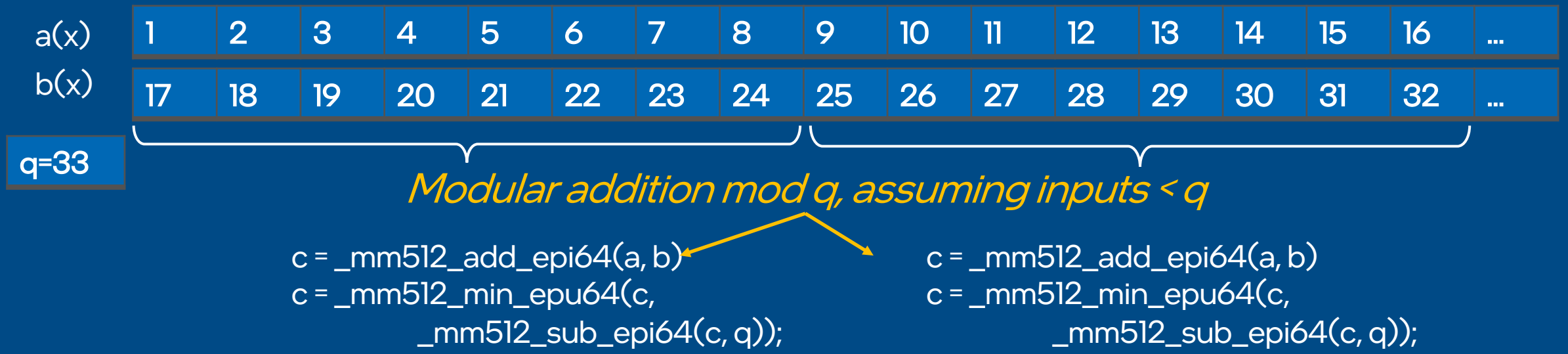
~100-1000 bits

$$R_q = \frac{Z_q[X]}{X^N+1} = \underbrace{(a_0 + a_1 x + a_2 x^2 + \dots + a_{N-1} x^{N-1})}_{\text{~4k to 32k coefficients}} \Rightarrow (a_0, a_1, a_2, \dots, a_{N-1}) = a(x)$$

  - Computing on these polynomials is slow!

Intel AVX512

- Intel Advanced Vector Extensions 512 (Intel AVX512)
  - SIMD instructions on 512-bit data
  - We use 8 64-bit integers
  - Apply element-wise, avoiding expensive permutations
  - E.g. addition:



Intel AVX512 MulHi

- Previously: 64-bit
  - Intel AVX512-DQ
  - 15 SIMD instructions using 32-bit arithmetic

```
inline __m512i _mm512_hexl_mulhi_epi<64>(__m512i x, __m512i y) {
  __m512i lo_mask = _mm512_set1_epi64(0x00000000ffffffff);
  // x0_lo, x0_hi, x1_lo, x1_hi, x2_lo, x2_hi, ...
  __m512i x_hi = _mm512_shuffle_epi32(x, (_MM_PERM_ENUM)0xB1);
  // y0_lo, y0_hi, y1_lo, y1_hi, y2_lo, y2_hi, ...
  __m512i y_hi = _mm512_shuffle_epi32(y, (_MM_PERM_ENUM)0xB1);
  __m512i z_lo_lo = _mm512_mul_epu32(x, y);        // x_lo * y_lo
  __m512i z_lo_hi = _mm512_mul_epu32(x, y_hi);     // x_lo * y_hi
  __m512i z_hi_lo = _mm512_mul_epu32(x_hi, y);     // x_hi * y_lo
  __m512i z_hi_hi = _mm512_mul_epu32(x_hi, y_hi);  // x_hi * y_hi
  __m512i z_lo_lo_shift = _mm512_srli_epi64(z_lo_lo, 32);
  __m512i sum_tmp = _mm512_add_epi64(z_lo_hi, z_lo_lo_shift);
  __m512i sum_lo = _mm512_and_si512(sum_tmp, lo_mask);
  __m512i sum_mid = _mm512_srli_epi64(sum_tmp, 32);
  __m512i sum_mid2 = _mm512_add_epi64(z_hi_lo, sum_lo);
  __m512i sum_mid2_hi = _mm512_srli_epi64(sum_mid2, 32);
  __m512i sum_hi = _mm512_add_epi64(z_hi_hi, sum_mid);
  return _mm512_add_epi64(sum_hi, sum_mid2_hi);
}
```

- Today: 52-bit
  - Intel AVX512-IFMA52
  - A single instruction!

```
inline __m512i _mm512_hexl_mulhi_epi<52>(__m512i x, __m512i y) {
  __m512i zero = _mm512_set1_epi64(0);
  return _mm512_madd52hi_epu64(zero, x, y);
}
```

  - Caveat: inputs must be less than 52 bits
    - Choose RNS moduli < 52 bits

$$q = q_1 \cdot q_2 \cdot \ldots \cdot q_L$$

$< 2^{52}$

    - May reduce precision

Accelerating HE with Intel AVX512 – Modular multiplication

- Multi-word *d* computed via
  - MulLo (52 or 64)
  - MulHi (52 or 64)
- Choose
  - L = 52+ceil(log2(q)) or
  - L = 63 + ceil(log2(q))
  - Ensures c3 is single-word
  - Ensures c3 = MulHi(c1, k)

Algorithm 1: Barrett's Algorithm

Input: $n < 2^N, d < 2^D, \kappa = \left\lfloor \frac{2^L}{n} \right\rfloor$ where $N \leq D \leq L$

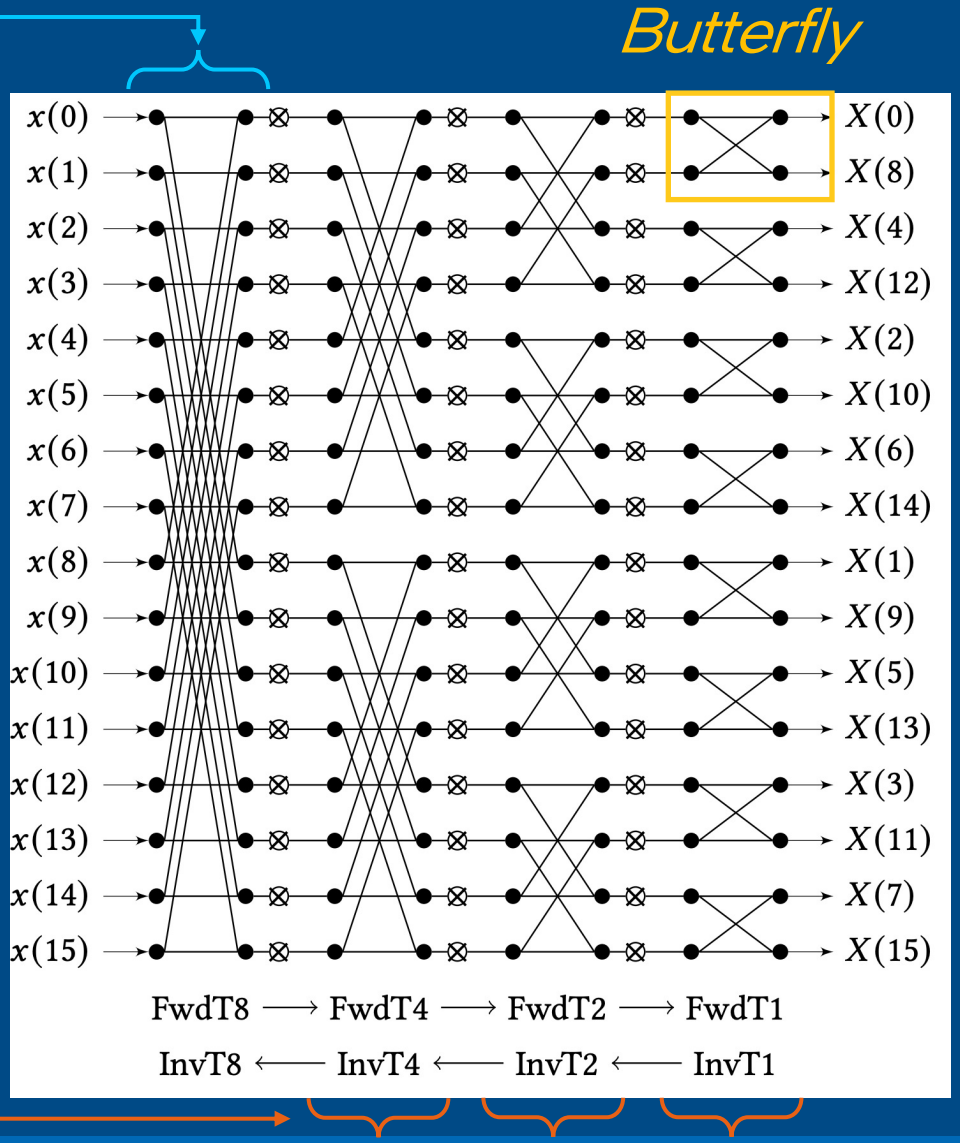
Output: $c = d \bmod n$

1 $c_1 \leftarrow d \gg (N-1)$;
2 $c_2 \leftarrow c_1 \kappa$;
3 $c_3 \leftarrow c_2 \gg (L-N+1)$;
4 $c_4 \leftarrow d - nc_3$;
5 **while** $c_4 \geq n$ **do**
6 $\quad c_4 \leftarrow c_4 - n$;
7 **end while**
8 **return** c_4

https://hal.archives-ouvertes.fr/hal-01215845/document

Accelerating HE with Intel AVX512 - NTT

- Core operation:
  - Radix-2 Cooley-Tukey / Gentleman-Sande Harvey butterfly
  - (X + WY, X – WY)
  - (X+Y, W(X-Y))
- 8 sequential butterflies – AVX512 easy
- 4/2/1 sequential butterflies
  - 4: Permute, 8-lane butterfly, permute
  - 2: Permute, 8-lane butterfly, permute
  - 1: Permute, 8-lane butterfly, permute
  - Optimizations:
    - Fuse sequential permutations
    - Skip final permutation?
      - Non-standard output order



HEXL Kernels Speedup

Table 1: Single-threaded, single-core Intel HEXL kernel runtimes in microseconds on a 50-bit modulus.

(a) Forward NTT

Implementation	N / Speedup					
	1024		4096		16384	
Native C++	9.08	1.0x	38.8	1.0x	177	1.0x
NFLlib[1]	4.82	1.8x	21.1	1.8x	97.8	1.8x
Intel AVX512-DQ	3.26	2.7x	13.4	2.8x	62.3	2.8x
NTL[22]	2.44	3.7x	8.48	4.5x	40.2	4.3x
Intel AVX512-IFMA52	1.25	7.2x	5.81	6.6x	33.1	5.3x

(b) Inverse NTT

Implementation	N / Speedup					
	1024		4096		16384	
Native C++	8.25	1.0x	37.8	1.0x	174	1.0x
NFLlib[1]	6.07	1.3x	26.7	1.4x	124	1.4x
Intel AVX512-DQ	3.16	2.6x	14.6	2.5x	68.2	2.5x
NTL[22]	2.12	3.8x	9.05	4.1x	42.4	4.1x
Intel AVX512-IFMA52	1.23	6.7x	5.72	6.6x	32.4	5.3x

(c) Element-wise vector-vector modular multiplication

Implementation	N / Speedup					
	1024		4096		16384	
Native C++ Int	1.51	1.0x	5.71	1.0x	23.6	1.0x
Intel AVX512-DQ Int	0.982	1.5x	3.43	1.6x	12.3	1.9x
Intel AVX512-DQ Float	0.251	6.0x	1.08	5.2x	4.58	5.1x

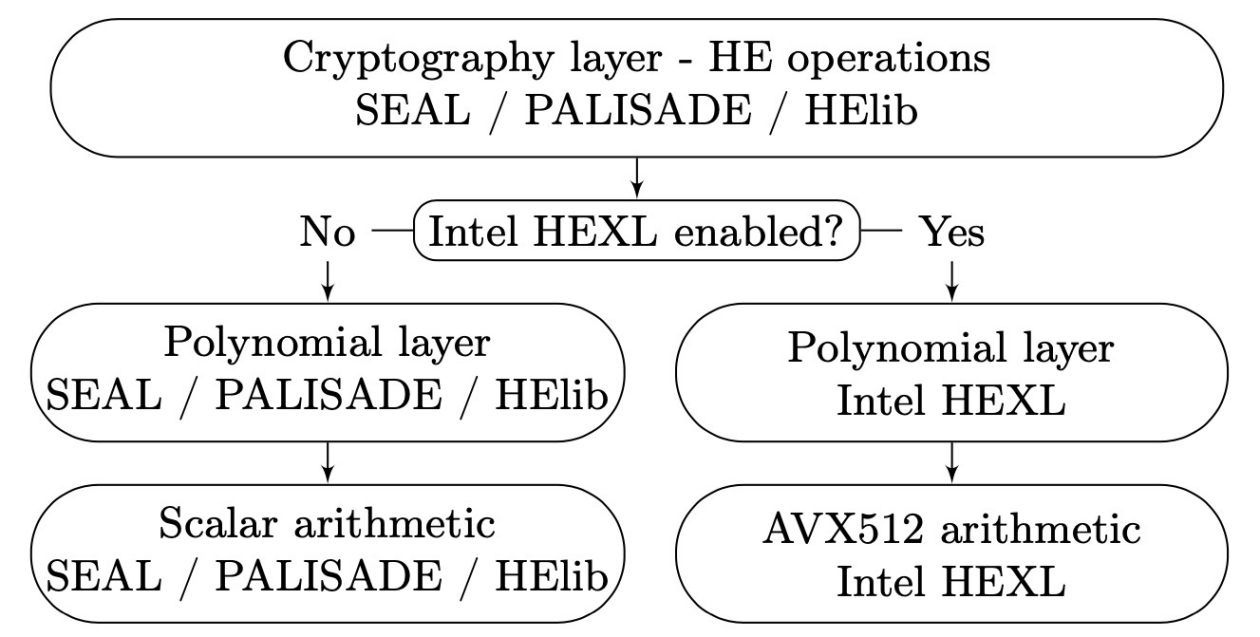
(d) Element-wise vector-scalar-vector modular multiply-add

Implementation	N / Speedup					
	1024		4096		16384	
Native C++	0.53	1.0x	2.11	1.0x	9.01	1.0x
Intel AVX512-DQ	0.53	1.0x	2.11	1.0x	9.01	1.0x
Intel AVX512-IFMA52	0.302	1.7x	1.20	1.7x	5.08	1.7x

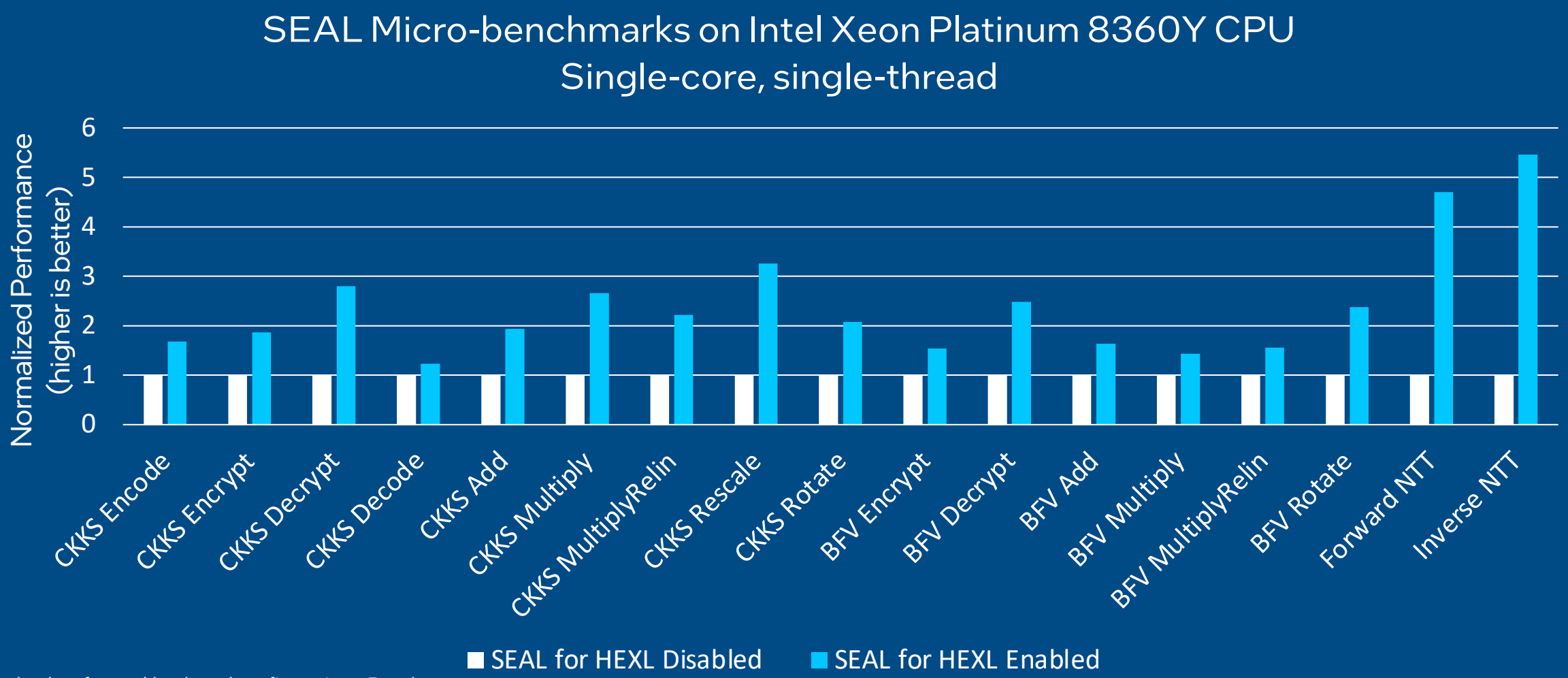
See backup for workloads and configurations. Results may vary

HE Library Integration

- HEXL intercepts HE libraries at polynomial layer
  - Simple - small integration area
  - Flexible – accelerates several HE schemes
  - Performant – most runtime spent at polynomial layer

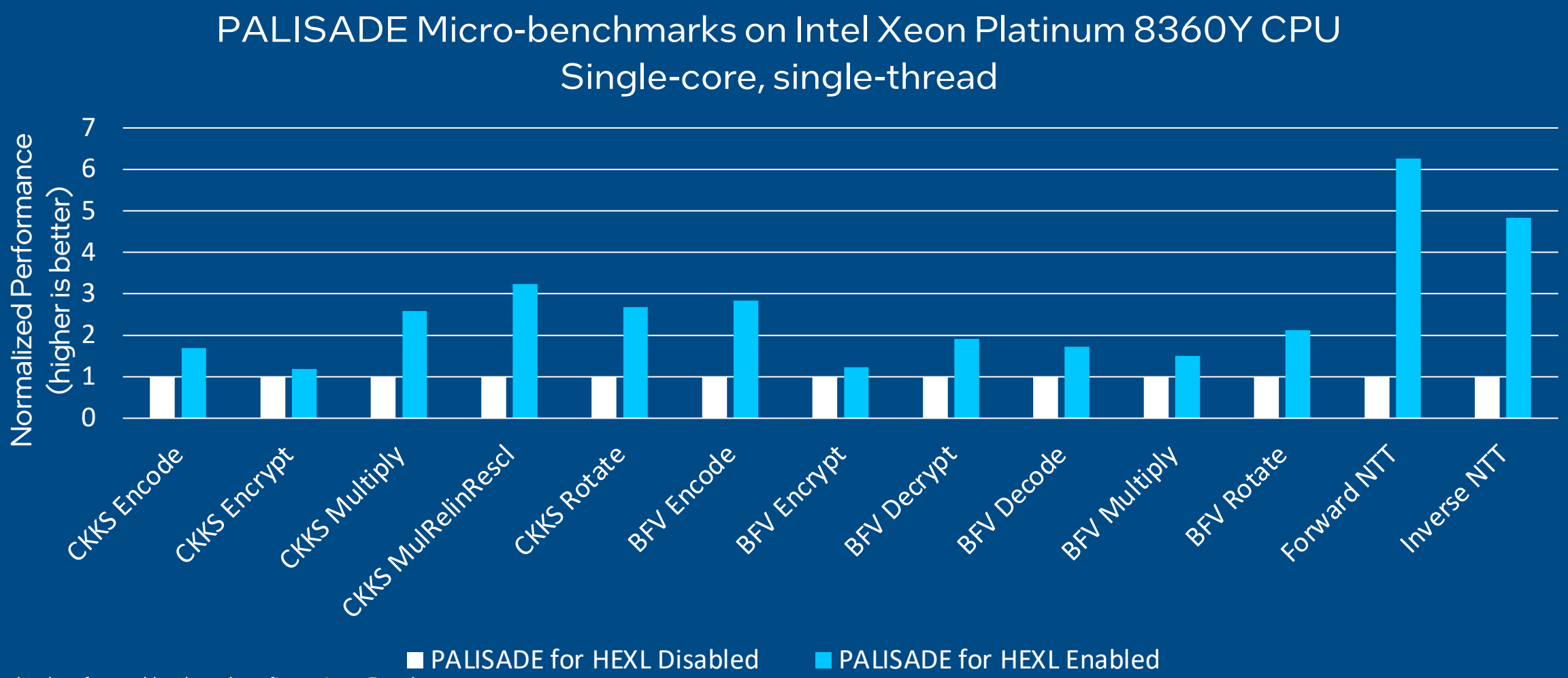


SEAL Integration



See backup for workloads and configurations. Results may vary

PALISADE Integration



See backup for workloads and configurations. Results may vary

Next steps

- Higher-radix NTT?
- More HE operations?
- More HE libraries?
- HW acceleration?
- Thank you
  - Wei Dai, Kim Laine – Microsoft
  - Kurt Rohloff, Yuriy Polyakov – Duality Technologies
  - Flavio Bergamaschi - Intel

Notices & Disclaimers

Performance/Benchmarking Disclaimer: Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Intel technologies may require enabled hardware, software or service activation. Your results may vary.

No product or component can be absolutely secure.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates.

https://github.com/intel/hexl

he_toolkit_support@intel.com


intel