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
  - $\mathbb{Z}_q = \{0, 1, ..., q - 1\}$
  - HE polynomials live in quotient ring
    - $R_q = \frac{\mathbb{Z}_q[x]}{x^{N+1}} = (a_0 + a_1 x + a_2 x^2 + ... + a_{N-1} x^{N-1}) \Rightarrow (a_0, a_1, a_2, ..., a_{N-1}) = a(x)$
  - $\sim 100$-$1000$ bits
  - $\sim 4k$ to $32k$ coefficients
- 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:

```
  c = _mm512_add_epi64(a, b)
  c = _mm512_min_epu64(c, _mm512_sub_epi64(c, q));
```

Modular addition mod q, assuming inputs < q

```
c = _mm512_add_epi64(a, b)
c = _mm512_min_epu64(c, _mm512_sub_epi64(c, q));
```
Intel AVX512 MulHi

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

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

```c
inline __m512i _mm512_hexl_mulhi_epi64(__m512i x, __m512i y) {
    __m512i lo_mask = _mm512_set1_epi64(0x0000000000000000);
    // x0_lo, x0_hi, x1_lo, x1_hi, x2_lo, x2_hi, ...
    __m512i x_hi = _mm512_shuffle_epi32(x, (MM_PERM_ENUM)0x81);
    // y0_lo, y0_hi, y1_lo, y1_hi, y2_lo, y2_hi, ...
    __m512i y_hi = _mm512_shuffle_epi32(y, (MM_PERM_ENUM)0x81);
    __m512i z_lo_lo = _mm512_mulu32(x, y);  // x_lo * y_lo
    __m512i z_lo_hi = _mm512_mulu32(x, y_hi); // x_lo * y_hi
    __m512i z_hi_lo = _mm512_mulu32(x_hi, y);  // x_hi * y_lo
    __m512i z_hi_hi = _mm512_mulu32(x_hi, y_hi); // x_hi * y_hi
    __m512i z_lo_shift = _mm512_srlx_epi64(z_lo_lo, 32);
    __m512i sum_temp = _mm512_add_epi64(z_lo_hi, z_lo_lo_shift);
    __m512i sum_lo = _mm512_and_epi64(sum_temp, lo_mask);
    __m512i sum_mid = _mm512_srlx_epi64(sum_temp, 32);
    __m512i sum_mid2 = _mm512_add_epi64(z_hi_lo, sum_lo);
    __m512i sum_mid2_hi = _mm512_srlx_epi64(sum_mid2, 32);
    __m512i sum_hi = _mm512_add_epi64(z_hi_hi, sum_mid);
    return _mm512_add_epi64(sum_hi, sum_mid2_hi);
}
```

- 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
  - $L = 52 + \lceil \log_2(q) \rceil$ or
  - $L = 63 + \lceil \log_2(q) \rceil$
  - Ensures $c_3$ is single-word
  - Ensures $c_3 = \text{MulHi}(c_1, k)$

### Algorithm 1: Barrett’s Algorithm

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

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.     $c_4 \leftarrow c_4 - n$;
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

<table>
<thead>
<tr>
<th>Implementation</th>
<th>1024</th>
<th>4096</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native C++</td>
<td>9.08</td>
<td>38.8</td>
<td>177</td>
</tr>
<tr>
<td>NFLlib[1]</td>
<td>4.82</td>
<td>21.1</td>
<td>97.8</td>
</tr>
<tr>
<td>Intel AVX512-DQ</td>
<td>3.26</td>
<td>13.4</td>
<td>62.3</td>
</tr>
<tr>
<td>NTL[22]</td>
<td>2.44</td>
<td>8.48</td>
<td>40.2</td>
</tr>
<tr>
<td>Intel AVX512-IFMA52</td>
<td>1.25</td>
<td>5.81</td>
<td>33.1</td>
</tr>
</tbody>
</table>

(b) Inverse NTT

<table>
<thead>
<tr>
<th>Implementation</th>
<th>1024</th>
<th>4096</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native C++</td>
<td>8.25</td>
<td>37.8</td>
<td>174</td>
</tr>
<tr>
<td>NFLlib[1]</td>
<td>6.07</td>
<td>26.7</td>
<td>124</td>
</tr>
<tr>
<td>Intel AVX512-DQ</td>
<td>3.16</td>
<td>14.6</td>
<td>68.2</td>
</tr>
<tr>
<td>NTL[22]</td>
<td>2.12</td>
<td>9.05</td>
<td>42.4</td>
</tr>
<tr>
<td>Intel AVX512-IFMA52</td>
<td>1.23</td>
<td>5.72</td>
<td>32.4</td>
</tr>
</tbody>
</table>

(c) Element-wise vector-vector modular multiplication

<table>
<thead>
<tr>
<th>Implementation</th>
<th>1024</th>
<th>4096</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native C++ Int</td>
<td>1.51</td>
<td>5.71</td>
<td>23.6</td>
</tr>
<tr>
<td>Intel AVX512-DQ Int</td>
<td>0.982</td>
<td>3.43</td>
<td>12.3</td>
</tr>
<tr>
<td>Intel AVX512-DQ Float</td>
<td>0.251</td>
<td>1.08</td>
<td>5.26</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Implementation</th>
<th>1024</th>
<th>4096</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native C++</td>
<td>0.53</td>
<td>2.11</td>
<td>9.01</td>
</tr>
<tr>
<td>Intel AVX512-DQ</td>
<td>0.53</td>
<td>2.11</td>
<td>9.01</td>
</tr>
<tr>
<td>Intel AVX512-IFMA52</td>
<td>0.302</td>
<td>1.20</td>
<td>5.08</td>
</tr>
</tbody>
</table>

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

SEAL Micro-benchmarks on Intel Xeon Platinum 8360Y CPU
Single-core, single-thread

Normalized Performance (higher is better)

CKKS Encode  CKKS Decrypt  CKKS Decode  CKKS Add  CKKS Multiply  CKKS MultiplyRelin  CKKS Rescale  CKKS Rotate  BFV Encrypt  BFV Decrypt  BFV Add  BFV Multiply  BFV MultiplyRelin  BFV Rotate  Forward NTT  Inverse NTT

See backup for workloads and configurations. Results may vary
PALISADE Integration

PALISADE Micro-benchmarks on Intel Xeon Platinum 8360Y CPU
Single-core, single-thread

Normalized Performance (higher is better)

- CKKS Encode
- CKKS Encrypt
- CKKS Multiply
- CKKS MulRelinRescl
- CKKS Rotate
- BFV Encode
- BFV Encrypt
- BFV Decrypt
- BFV Decode
- BFV Multiply
- BFV Rotate
- Forward NTT
- Inverse NTT

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

© Notice: Copyright © 2021, Intel Corporation. All Rights Reserved.

Intel TM Notice: Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

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