Accelerating NTRU based Homomorphic Encryption using GPUs

Wei Dai, Yarkın Doröz, Berk Sunar

Worcester Polytechnic Institute
100 Institute Road, Worcester, MA, USA, 01609
Outline:

- Motivations
- NTRU
- GPU NTRU
- Evaluation Results
Motivation

- Homomorphic Encryption is not efficient
- Speedup computations with GPUs
NTRU

- Keys, cipher texts are polynomials
  - \( n \): polynomial degree
  - \( q \): prime modulus
  - \( l \): length of \( q \)
- Operations are performed in \( \mathbb{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1 \rangle \)
- Previously, implemented with NTL Library in C++
GPU NTRU

• Core problem: *polynomial multiplications*

• How to create parallelism?
  ─ Chinese Remainder Theorem (CRT)
  ─ Number Theory Transform (NTT)

• Fast algorithms

• Memory access
GPU NTRU: Multiplication

- AES: 32768 degree, 1271-bit coefficients
- PRINCE: 16384 degree, 575-bit coefficients
- The Strassen’s NTT based integer multiplication algorithm

**Algorithm 1 Polynomial Multiplication**

**Input:** Polynomials $a$, $b$ with $(n, \log(q))$  
**Output:** Polynomial $c$ with $(2n, \log(nq^2))$

1. $\{a_i\} \xrightarrow{\text{CRT}(a)} \{b_i\} \xrightarrow{\text{CRT}(b)}$
2. $\{A_i\} = \text{NTT}(\{a_i\}), \{B_i\} = \text{NTT}(\{b_i\})$
3. $\{C_i\} = \{A_i\} \cdot \{B_i\}$
4. $\{c_i\} = \text{INTT}(\{C_i\})$
5. $\{c_i\} \xrightarrow{\text{ICRT}(\{c_i\})}$
GPU NTRU: CRT

\[ \text{CRT: } x \longrightarrow \{x \mod p_0, x \mod p_1, \ldots, x \mod p_{l-1}\} \]

- Reduce coefficient size
  - 1271-bit \(\longrightarrow\) 42-bit
  - 575-bit \(\longrightarrow\) 25-bit

- Size and number of \(p_i\) are decided automatically
  - in different circuit levels
  - only according to \(n\) and \(q\)
  - as level increases, computation goes faster

- Rules will be explained later
**GPU NTRU: ICRT**

\[
\text{ICRT: } x = \sum_{i=0}^{l-1} \left( \frac{M}{p_i} \right) \cdot \left( \left( \frac{M}{p_i} \right)^{-1} \cdot x_i \mod p_i \right) \mod M
\]

\[
M = \prod_{i=0}^{l} p_i
\]

- Modified ICRT scheme
  - avoid large integer multiplication
  - avoid large integer modular reduction
- NVIDIA GPU Constant memory
GPU NTRU: NTT

- Emmart and Weems’ approach
- 2n-point coefficient-wise NTT (padding with 0)
- Four-step Cooley-Tukey NTT iterations:
**GPU NTRU: NTT**

- Over the finite field
  \[
  \mathbb{Z}/P\mathbb{Z} \quad P = 0xffffffff00000001
  \]

- Prime numbers:
  \[
  P > n \cdot p_i^2 \\
  \prod_{i=0}^{l-1} p_i > n \cdot q^2
  \]

- Memory arrangement:
  - coalesced global memory access
  - shared memory as buffers
  - registers for arithmetic operations
GPU NTRU: Relinearization

- Input: cipher text \( c(x) \), evaluation keys \( \{E K_i(x)\} \)
- Take the \( i \)-th bit of coefficients in \( c(x) \)
- Binary polynomials \( \tilde{c}_i(x) \)
- Output:
  \[
  \tilde{c}(x) = \sum_{i=0}^{l-1} \tilde{c}_i(x) \cdot E K_i(x)
  \]
- Thousands of polynomial multiplications
GPU NTRU: Relinearization

- Evaluation keys are stored in NTT domain
- Computations are mainly in NTT domain

Algorithm 2 Relinearization

**Input:** Polynomial $c$ with $(n, \log(q))$

**Output:** Polynomial $d$ with $(2n, \log(nq\log(q)))$

1. $\{\widetilde{C}_\tau\} = \text{NTT}(\{\widetilde{c}_\tau\})$
2. for $i = 0 \ldots, l - 1$ do
3. load $EK_{i,0}, EK_{i,1}, \cdots, EK_{i,[\log(q)]-1}$
4. $\{D_i\} = \{\sum_{\tau=0}^{[\log(q)]-1} \widetilde{C}_\tau \cdot EK_{i,\tau}\}$
5. end for
6. $\{d_i\} = \text{INTT}(\{D_i\})$
7. $d = \text{ICRT}(\{d_i\})$
• EKs are huge (23 GB)
• Memory copy takes most of time
• Page-locked host memory
• Prime numbers:
  \[ P > \lceil \log(q) \rceil \cdot n \cdot p_i \]
  \[ \prod_{i=0}^{l-1} p_i > \lceil \log(q) \rceil \cdot n \cdot q \]
GPU NTRU

- NTL Data → 1-D arrays
- Coefficient and polynomial reductions
Implementation

Implementation Parameters

<table>
<thead>
<tr>
<th></th>
<th>AES</th>
<th>PRINCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Levels</td>
<td>40</td>
<td>24</td>
</tr>
<tr>
<td>Polynomial Size</td>
<td>(32768, 1271)</td>
<td>(16384, 575)</td>
</tr>
<tr>
<td>Maximum Size of Evaluation Keys</td>
<td>23 GBytes</td>
<td>2 GBytes</td>
</tr>
</tbody>
</table>

Server specs:

• Intel Xeon E5-2609
  @2.5 GHz, 64 GB (1 thread)

• NVIDIA GeForce GTX690
  @915 MHz, 3072 CUDA cores, 4 GB (1536 cores, 2 GB)
### GPU NTRU

#### TABLE II. Timing comparison between the CPU and GPU implementations for the operations.

<table>
<thead>
<tr>
<th></th>
<th>Prince</th>
<th></th>
<th>AES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPU</td>
<td>CPU</td>
<td>SPEEDUP</td>
<td>GPU</td>
</tr>
<tr>
<td>Multiplication</td>
<td>0.063</td>
<td>0.18</td>
<td>×2.8</td>
<td>0.34</td>
</tr>
<tr>
<td>Relinearization</td>
<td>0.89</td>
<td>10.9</td>
<td>×12.2</td>
<td>8.97</td>
</tr>
</tbody>
</table>

#### TABLE III. Performance comparison of Prince and AES implementations.

<table>
<thead>
<tr>
<th></th>
<th>TOTAL TIME</th>
<th>#BLOCKS</th>
<th>PER BLOCK</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD Xeon [9]</td>
<td>36 h</td>
<td>54</td>
<td>2400 sec</td>
<td>×1</td>
</tr>
<tr>
<td>Byte Xeon [9]</td>
<td>65 h</td>
<td>720</td>
<td>300 sec</td>
<td>×8</td>
</tr>
<tr>
<td>NTRU Xeon [10]</td>
<td>31 h</td>
<td>2048</td>
<td>55 sec</td>
<td>×43</td>
</tr>
<tr>
<td>Ours (GPU)</td>
<td>4.15 h</td>
<td>2048</td>
<td>7.3 sec</td>
<td>×328</td>
</tr>
<tr>
<td>Prince</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prince [32]</td>
<td>57 min</td>
<td>1024</td>
<td>3.3 sec</td>
<td>×1</td>
</tr>
<tr>
<td>Ours (GPU)</td>
<td>22 min</td>
<td>1024</td>
<td>1.28 sec</td>
<td>×2.57</td>
</tr>
</tbody>
</table>
Questions?
Thank you.