Parallel Small Polynomial Multiplication for Dilithium: A Faster Design and Implementation

Jieyu Zheng¹, Feng He¹, Shen Shiyu¹, Chenxi Xue¹, Yunlei Zhao¹
¹School of Computer Science, Fudan University
{jieyuzheng21,shenshiyu21,cxxue21}@m.fudan.edu.cn {fhe20,ylzhao}@fudan.edu.cn

Annual Computer Security Applications Conference (ACSAC)
December 5-9, 2022
Outline

• Introduction
• Motivation
• Small Polynomial Multiplication
• Experimental results
• Conclusion
Introduction

Dilithium

- One of third-round Signature finalists (The final Signature scheme to be standardized)
- Module-LWE and Module-SIS
- Small keys and signatures
- Operates in $\mathbb{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$
  - Allows efficient polynomial multiplication with NTT
- Parameters: $n = 256, q = 8380417$
Motivation

Dilithium Sign and Verify

Small Polynomial

- Coefficients are much smaller than $q$.
- Most coefficients are 0, few are $\pm 1$.

Small Polynomial Multiplication
Motivation

The previous technique to speed up polynomial multiplication is Number Theoretic Transform (NTT). Can we derive a polynomial multiplication for small polynomial?

The answer is “Yes!”
Small Polynomial Multiplication: Technique Overview

- Index-based polynomial multiplication
- Nonnegative Index-based polynomial multiplication
- Parallel index-based polynomial multiplication algorithm
Index-based Small Polynomial Multiplication

- For $u = \sum_{i=0}^{n-1} u_i \cdot x^i$, $0 \leq i \leq n - 2$ we have:

$$u_i = \sum_{j=0}^{i} c_j \cdot a_{i-j} - \sum_{j=i+1}^{n-1} c_j \cdot a_{n+i-j} = \sum_{j=0}^{i} c_j \cdot a_{i-j} + \sum_{j=i+1}^{n-1} c_j \cdot (-a_{n+i-j})$$

- For $i = n - 1$, we have $u_i = \sum_{j=0}^{n-1} c_j \cdot a_{i-j}$.

$c_j \cdot a_{i-j}$ and $c_j \cdot a_{n+i-j}$ can be replaced by $a_{i-j}$ and $a_{n+i-j}$ ($c_j = 1$), $-a_{i-j}$ and $-a_{n+i-j}$ ($c_j = -1$).
Index-based Small Polynomial Multiplication

Algorithm 6 An alternative index-based polynomial multiplication algorithm for computing $ca$

Input: $c = \sum_{i=0}^{n-1} c_i \cdot x^i \in B_r$, $a = \sum_{i=0}^{n-1} a_i \cdot x^i \in \mathcal{R}_q$

Output: $u = c \cdot a \in \mathcal{R}_q$

1: for $i \in \{0, 1, \cdots, n-1\}$ do
2: \hspace{1em} $w_i := 0$
3: \hspace{1em} $v_i := a_i$
4: \hspace{1em} $v_{i-n} := -a_i$
5: end for

6: for $i \in \{0, 1, \cdots, n-1\}$ do
7: \hspace{1em} if $c_i = 1$ then
8: \hspace{2em} for $j \in \{0, 1, \cdots, n-1\}$ do
9: \hspace{3em} $w_j := w_j + v_{j-i}$
10: \hspace{2em} end for
11: \hspace{1em} end if
12: \hspace{1em} if $c_i = -1$ then
13: \hspace{2em} for $j \in \{0, 1, \cdots, n-1\}$ do
14: \hspace{3em} $w_j := w_j - v_{j-i}$
15: \hspace{2em} end for
16: \hspace{1em} end if
17: end for

18: for $i \in \{0, 1, \cdots, n-1\}$ do
19: \hspace{1em} $u_i := w_i \pmod{q}$  \hspace{1em} $\Rightarrow u_i \in \mathbb{F}_q$
20: end for

21: $u := \sum_{i=0}^{n-1} u_i \cdot x^i$  \hspace{1em} $\Rightarrow u \in \mathcal{R}_q$

22: return $u$

Addition and subtraction replace multiplication
Nonnegative Small Polynomial Multiplication

- The above algorithm is not suitable for deriving parallel algorithm.

- Make algorithm nonnegative.

- $U$ is upper bound of coefficients.

**Algorithm 7** An index-based polynomial multiplication algorithm with translations

```
Input: $c = \sum_{i=0}^{n-1} c_i \cdot x^i \in B_r$, $a = \sum_{i=0}^{n-1} a_i \cdot x^i \in R_q$
Output: $u = c \cdot a \in R_q$
1: for $i \in \{0, 1, \cdots, n-1\}$ do
2: \hspace{1em} $\omega_i := 0$
3: \hspace{1em} $v_i := U + a_i$
4: \hspace{1em} $v_{i-n} := U - a_i$
5: end for
6: for $i \in \{0, 1, \cdots, n-1\}$ do
7: \hspace{1em} if $c_i = 1$ then
8: \hspace{2em} for $j \in \{0, 1, \cdots, n-1\}$ do
9: \hspace{3em} $\omega_j := \omega_j + v_{j-i}$
10: \hspace{2em} end for
11: \hspace{1em} end if
12: \hspace{1em} if $c_i = -1$ then
13: \hspace{2em} for $j \in \{0, 1, \cdots, n-1\}$ do
14: \hspace{3em} $\omega_j := \omega_j + (2U - v_{j-i})$
15: \hspace{2em} end for
16: \hspace{1em} end if
17: end for
18: for $i \in \{0, 1, \cdots, n-1\}$ do
19: \hspace{1em} $u_i := \omega_i - \tau U \ (mod \ q)$ \hspace{1em} \(\triangleright\ u_i \in \mathbb{F}_q\)
20: end for
21: $u := \sum_{i=0}^{n-1} u_i \cdot x^i$ \hspace{1em} \(\triangleright\ u \in R_q\)
22: return $u$
```
Parallel Small Polynomial Multiplication

- Compute $c \cdot \vec{a}$
  - $c \in B_{\tau}, c_i \in \{-1,0,1\}$
  - $\vec{a} = [a^{(0)}, \ldots, a^{(r-1)}]^T \in R_q^r$ is a polynomial vector, there exists an constant $U$ that $\|a^{(j)}\|_\infty \leq U$, $\forall j \in \{0,1,\ldots,r-1\}$, $r$ is the number of polynomial that a word (64bit) can pack.
Algorithm 9 A parallel index-based polynomial multiplication algorithm with translations

Input: \((\mathbf{c}, \mathbf{a})\), where
- \(\mathbf{c} = \sum_{i=0}^{n-1} c_i \cdot x^i \in B_r\);
- \(\mathbf{a} = \{a^{(j)}\} \in R_q^r\);
- Every \(a^{(j)} = \sum_{i=0}^{n-1} a_i^{(j)} \cdot x^i \in R_q\);
- Every \(a_i^{(j)} \in \{-U, \ldots, U\}\)

Output: \(\overrightarrow{\mathbf{u}} = \left[u^{(0)}, \ldots, u^{(r-1)}\right]^T \in R_q^r\), where
- \(u^{(j)} = c \cdot a^{(j)} \in R_q\);

\begin{algorithm}
1: for \(i \in \{0, 1, \ldots, n - 1\}\) do
2: \(w_i := 0\)
3: \(u_i := 0\)
4: \(u_{i-n} := 0\)
5: for \(j = 0\) to \(r - 1\) do
6: \(v_i := u_i \cdot M + \left(U + a_i^{(j)}\right)\)
7: \(v_{i-n} := u_{i-n} \cdot M + \left(U - a_i^{(j)}\right)\)
8: end for
9: end for
10: \(\gamma := 2U \cdot \frac{M^r-1}{M-1}\)
11: for \(i \in \{0, 1, \ldots, n - 1\}\) do
12: if \(c_i = 1\) then
13: for \(j \in \{0, 1, \ldots, n - 1\}\) do
14: \(w_j := w_j + u_{j-i}\)
15: end for
16: end if
17: if \(c_i = -1\) then
18: for \(j \in \{0, 1, \ldots, n - 1\}\) do
19: \(w_j := w_j + (\gamma - u_{j-i})\)
20: end for
21: end if
22: end for
23: for \(i \in \{0, 1, \ldots, n - 1\}\) do
24: \(t := w_i\)
25: for \(j = 0\) to \(r - 1\) do
26: \(u_i^{(r-1-j)} := (t \mod M) - tU \mod q\)
27: \(t := \lfloor t/M \rfloor\)
28: end for
29: end for
30: for \(j \in \{0, 1, \ldots, r - 1\}\) do
31: \(u^{(j)} := \sum_{i=0}^{n-1} u_i^{(j)} \cdot x^i\)
32: end for
33: \(\overrightarrow{\mathbf{u}} := \left[u^{(0)}, \ldots, u^{(r-1)}\right]^T\)
34: return \(\overrightarrow{\mathbf{u}}\)
\end{algorithm}

Pack vector coefficients

\(\gamma \in \mathbb{Z}^{>0}\)
Experimental Results

For Dilithium-2, we achieve **18%** speed-up in Sign, **19%** in Verify.
For Dilithium-3, we achieve **30%** speed-up in Sign, **7%** in Verify.
For Dilithium-5, we achieve **27%** speed-up in Sign, **3%** speed-up in Verify.

Table 5: Reference Implementation Comparative Results in cycles.
Experimental Results

For Dilithium-2, we achieve 64% speed-up in Sign, 50% in Verify.

For Dilithium-3, we achieve 60% speed-up in Sign, 32% in Verify.
Experimental Results

Table 7: Arm neon Implementation Comparative Results in cpycles.

<table>
<thead>
<tr>
<th></th>
<th>[4]</th>
<th>Our work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign in Dilithium-2</td>
<td>3327206</td>
<td>2934124</td>
</tr>
<tr>
<td>Verification in Dilithium-2</td>
<td>1191080</td>
<td>1231796</td>
</tr>
<tr>
<td>Sign in Dilithium-3</td>
<td>5321618</td>
<td>4927678</td>
</tr>
<tr>
<td>Verification in Dilithium-3</td>
<td>2018945</td>
<td>2073518</td>
</tr>
</tbody>
</table>

- Compared with the state-of-art implementation.
- For Dilithium-2, we achieve 13.4% speed-up in Sign.
- For Dilithium-3, we achieve 8% speed-up in Sign.

Experimental Results

- Polynomial vector multiplication in Dilithium-3

<table>
<thead>
<tr>
<th></th>
<th>Our Algorithm</th>
<th>NTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_\mathbf{s} &amp; c_\mathbf{e}) in Dilithium-3</td>
<td>8477</td>
<td>73924</td>
</tr>
<tr>
<td>(c_\mathbf{t}_0) in Dilithium-3</td>
<td>14132</td>
<td>88198</td>
</tr>
<tr>
<td>(c_\mathbf{t}_1) in Dilithium-3</td>
<td>11760</td>
<td>90992</td>
</tr>
</tbody>
</table>

- Compared with the NTT technique implementation.
- For \(c_\mathbf{s} \& c_\mathbf{e}\), we achieve 88% speed-up.
- For \(c_\mathbf{t}_0\), we achieve 84% speed-up.
- For \(c_\mathbf{t}_1\), we achieve 87% speed-up.
Conclusion

- We exhibit a small polynomial multiplication parallel algorithm.
- We complete the C reference implementation.
- We improve the algorithm by Neon vector extension on the Cortex-A72 platform.
- Our Arm Neon implementation of Dilithium achieves a new record of fast Dilithium implementation.
Parallel Small Polynomial Multiplication for Dilithium: A Faster Design and Implementation

THANKS!

Questions?