The obtained results for the listed oscillatjry functions allowed us to create a theory of optimal integration of highly oscillatory functions both in classical formulation and for interpolation classes of functions.

Considerable attention is paid to the identification and refinement of a priori information about the integral function and its use for narrowing the usual (classical) classes of integral functions to interpolation classes [1]. The functions included in such (interpolation) classes do not differ in quadrature formulas (the approximate integral value will be the same for them all).

The second feature of the results is (in contrast to the results of all other authors) in the assumption of an approximate input of information about the integral function. Examining interpolation classes can increase the potential of quadrature formulas.

Computer technologies (CTs) of the integration of highly oscillatory functions with given accuracy are analyzed.

Key words: *quadrature formula, cap method, method of boundary functions, a priori information, Bessel functions, computer technology.*

Одержано 29.01.2019

УДК 004.056.55
DOI: 10.32626/2308-5878.2019-19.28-34

О. Г. Качко*, канд. техн. наук,
С. О. Кандій,** студент,
Є. В. Острянська,** студент

*АТ «Інститут інформаційних технологій», м. Харків,
**Харківський національний університет імені В. Н. Каразіна, м. Харків

ОПТИМІЗАЦІЯ ФУНКЦІЇ МНОЖЕННЯ ПОЛІНОМІВ ДЛЯ ЗВИЧАЙНОЇ ТА PRODUCT ФОРМИ ЗАДАННЯ ОДНОГО З ПОЛІНОМІВ

У роботі проведено дослідження та виконано розробку ефективного алгоритму множення тернарного полінома у кільці $\mathrm{Z}_3[x]\left(x^n - x - 1\right)$ з урахуванням його структури. Розглядаються варіанти для поліномів із звичайною структурою з фіксованою кількістю ненульових елементів «1» («–1») та у PRODUCT-формі, у якій поліном є результатом обчислення $F_1 * F_2 + F_3$, де $F_1, F_2, F_3 \in \mathrm{Z}_3[x]\left(x^n - x - 1\right)$ та мають відповідно d_1, d_2, d_3 елементів із значеннями «1» та «–1». Приводяться результати оптимізації за допомогою векторизованих наборів інструкцій (а саме, набір інструкцій AVX2), розпаралелювання та спеціальних засобів для мінімізації та компенсування використання не

вирівняної пам'яті. Критичний код написано на асемблері під мікропроцесорну архітектуру x86-64, яка є однією з найрозповсюдженіших на сьогоднішній день. Отримані часові показники оптимізованої реалізації алгоритму для наборів параметрів для 256, 384 та 512 біт класичної безпеки та зроблено порівняння ефективності з алгоритмом множення поліномів, що був запропонований у асиметричній постквантовій криптосистемі на алгебраїчних решітках NTRU Prime. Тестування здійснено на процесорі Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz на операційній системі Linux 4.15.0-44-generic #47-Ubuntu SMP x86-64. Результати, що отримані, на наш погляд — надзвичайно актуальні. Вони можуть бути корисними для криптологів та інших фахівців, що займаються розробкою нових, ефективних криптографічних алгоритмів та протоколів для постквантового періоду. Це пояснюється тим, що для класів криптосистем, у яких використовуються перетворення у кільцях поліномів, як базові операції, а саме операцій множення, займає найбільше часу та тому потребує значної оптимізації. Значна перевага розробленого алгоритму — можливість його розпаралелювання на багатопроцесорних системах, що є значною перевагою порівняно з алгоритмом, що представлені в NTRU Prime.

Ключові слова: *NTRU Prime, PRODUCT- форма , алгебраїчна решітка, кільце поліномів, множення поліномів, постквантова криптографія, тернарний поліном.*

Вступ. На сьогоднішній день квантові комп'ютери розвиваються з великою швидкістю — це значна загроза для існуючих асиметричних криптосистем. Один з найперспективніших напрямків — це криптографія на алгебраїчних решітках. В таких системах більшість обчислень зводиться до множення поліномів, один з яких є тернарний. Існує багато алгоритмів для множення поліномів, але ці алгоритми або не враховують спеціальну структуру поліномів [1], або не забезпечують константний час множення. У даній роботі запропоновано алгоритм, що використовує особливості тернарних поліномів, та забезпечує константний час. Функція множення поліномів обчислює $c = a * b$, де a — тернарний поліном, b — поліном у полі $Z_q[x](x^n - x - 1)$. Для завдання поліному a використовується 2 форми.

Форма 1. a задається як поліном у полі $Z_3[x](x^n - x - 1)$ з фіксованою кількістю ненульових елементів;

Форма 2. a задається у формі $F_1 * F_2 + F_3$, де кожний з поліномів F_1, F_2, F_3 — це поліноми у полі, $Z_3[x](x^n - x - 1)$ містять однакову кількість «1» та «–1». Позначимо цю кількість відповідно d_1, d_2, d_3.

Множення для звичайної форми. У самому найпростішому «шкільному» методі послідовно виконується множення коефіцієнтів одного поліному на фіксований коефіцієнт іншого і накопичення суми для коефіцієнтів з однаковими номерами. Номер коефіцієнта визначається сумою номерів коефіцієнтів, які перемножаться. Формально це можна представити у вигляді псевдокоду 1.

Псевдокод 1

```
1.  for (i = 0; i < n; i++)
2.  for (j = 0; j < n; j++)
3.          c[i + j]+= a[i]∗b[j].
```

Оскільки основна кількість коефіцієнтів A — нульові елементи, то більшість операцій множення $a[i]$ на $b[j]$ не мають сенсу. Основна ідея нового алгоритму полягає у тому, щоб запам'ятати індекси ненульових елементів та замість множення $a[i]$ на $b[j]$ виконувати лише віднімання або додавання $b[j]$ в залежності від значення $a[i]$. Формально це можна представити у вигляді псевдокоду 2.

Псевдокод 2

```
1. c = 0;
2  for (i = 0; i < n; i++){
3.          if (a[i] == 1){
4.            for (j = 0; j < n; j++) {
5.              c[i+j] += b[j];
6.               if (c [i + j] >=q) c [i + j]-=q;
7.          }}else if (a[i] == -1){
8.               for (j = 0; j < n; j++){
9.                   c[i+j] -= b[j];
10.                  if (c [i + j] <0) c [i + j]+=q;
11.       }}}
```

Крім того, перед виконанням циклу усі елементи результату обнуляються, що гарантує їх наявність в кеші. При цьому загальний розмір не перевищує розміру кешу 1 рівня, тобто механізм витіснення за рахунок недостатнього розміру не буде спрацьовувати. Використання структури забезпечує неможливість перекриття адрес.

Передбачається використання SIMD операцій (AVX-2). Значення коефіцієнтів завжди цілі, тому використовується тип __m256i. Згідно значень параметрів [2] значення $c[i + j] < 2^{15} - 1$, тобто компоненти блоку 16 бітні знакові числа, а блок містить 16 таких компонентів і одночасно обробляється 16 коефіцієнтів.

Вищенаведений оптимізований варіант для псевдокоду передбачає:

- мінімізацію звернень до не вирівняної пам'яті (за рахунок перед обчислень) і компенсацію решти звернень за рахунок використання спеціальних операцій процесора;
- видалення операцій умовного переходу (за рахунок використання AVX-2 команд);
- мінімізацію промахів кешу (за рахунок обрання ефективної структури співмножників);
- паралельну обробку порцій коефіцієнтів. Кількість одиниць та мінус одиниць може не співпадати, тому порція для паралельного виконання містить і одиниці і мінус одиниці, в цьому разі їх кількість приблизно однакова для усіх порцій. Для збільшення навантаження на паралельну гілку кожна гілка приводить частковий результат не тільки по модулю q, а і по модулю p.

Результати тестування функції множення після оптимізації для набору параметрів з [2] наведені у табл. 1 (тактів).

Таблиця 1

Результати тестування множення для звичайної форми

K	**Linux**
256 (N = 761; q = 4591; T = 143)	7564
384 (N = 1031; q = 8297; T = 172)	11864
512 (N = 1301; q = 10427; T = 217)	16347

Для порівняння, час виконання функції множення з роботи [1] для того ж процесору, тих же режимів компіляції і для тих же ключів дорівнює 12321 тактів (K = 256, N = 761, q = 4591, t = 143). Прискорення в порівнянні з [1] складає 38.6 %

Множення для PRODUCT-форми. Для забезпечення ефективного використання кешу 1 рівня ключ задається як структура:

```
struct{
    unsigned short ones1[ d1 ], minusones1[ d1 ];
    unsigned short ones2[ d2 ], minusones2[ d2 ];
    unsigned short ones3[ d3 ], minusones3[ d3 ];
}
```

де d_1, d_2, d_3 — кількість 1 та –1 в F_1, F_2, F_3 відповідно; ones1, ones2, ones3 — масив з індексами відповідних коефіцієнтів, які дорівнюють 1; minusones1, minusones2, minusones3 — масиви з індексами відповідних коефіцієнтів, які дорівнюють –1.

Є 2 варіанти обчислення:

1) спочатку обчислити $F = F_1 * F_2 \bmod p + F_3$, а потім $F * h$;

2) спочатку обчислити $F' = \left(\left(F_1 * h\right) \bmod q\right) \bmod p$. Далі обчислити $F'' = \left(\left(F' * F_2\right) \bmod q\right) \bmod p$, а потім $F''' = F'' + \left(\left(\left(F_3 * h\right) \bmod q\right) \bmod p\right)$. Еквівалентність $F = F'''$ забезпечується на етапі генерації ключів.

Перевірка показала, що відсоток ключів, для яких ця умова не виконується, дуже незначний. Результати експериментальної перевірки для 100000 ключів наведені в табл. 2.

Аналіз показав, що перший варіант не дозволяє використовувати операції додавання та віднімання замість операції множення, тому що коефіцієнти обчисленого поліному можуть відрізнятися від 1, –1. Другий варіант дозволяє це зробити, тому далі будемо використовувати другий варіант.

Таблиця 2

Кількість відбракованих ключів (всього 100000 ключів)

K	**Кількість**	**%**
256 (N = 787; q = 7307; D_1 = 12; D_2 = 12; D_3 = 15)	102	0.1
384 (N = 1019; q = 8867; D_1 = 13; D_2 = 13; D_3 = 31)	60	0,06
512 (N = 1301; q = 11959; D_1 = 15; D_2 = 15; D_3 = 48)	50	0,05

Для обчислення часткових добутків F_1', F_2', $F_3 * h$ використовуються засоби оптимізації, які наведені вище. Для цього варіанту кількість одиниць та мінус одиниць у кожній з трьох функцій співпадає, крім того, кількість ненульових елементів у кожній з функцій значно менше, ніж при звичайному завданні, тому принцип розподілу обчислень між паралельними гілками відрізняється від попереднього.

Аналіз параметрів (див. табл. 3) показує що кількість ненульових елементів $D_1 + D_2$ в більшості випадків не перевищує D_3, більше того F_1 та F_2 пов'язані між собою знаком множення.

Пропонується паралельно обчислювати:

$$\left(\left(\left(\left(\left(F_1 * h\right) \bmod q\right) \bmod p\right) * F_2\right) \bmod q\right) \bmod p\Big)$$

та

$$\left(\left(\left(F_3 * h\right) \bmod q\right) \bmod p\right).$$

Значення D_1, D_2, D_3 — параметри алгоритму, тому різниця в часі для паралельних гілок не залежить від місця ненульових коефіцієнтів.

Пам'ять під внутрішні значення подвійної довжини може бути локальною, що забезпечує використання свого кешу для їх зберігання.

Результати тестування функції множення для product-форми після оптимізації для набору параметрів з [2] наведені у табл. 3 (тактів).

Таблиця 3

Результати тестування множення для PRODUCT — форми

K	**Linux**
256 (N = 787; q = 7307; D_1 = 12; D_2 = 12; D_3 = 15)	6499
384 (N = 1019; q = 8867; D_1 = 13; D_2 = 13; D_3 = 31)	8468
512 (N = 1301; q = 11959; D_1 = 15; D_2 = 15; D_3 = 48)	14718

Для порівняння, час виконання функції rq_mult [1] для того ж процесору і для найближчих параметрів (K = 256, N = 761, q = 4591, t = 143) дорівнює 12321 тактів. Прискорення в порівнянні з [1] практично в 2 рази. Використання multiply форми замість звичайної забезпечує прискорення.

Висновки. Визнано, що одним з перспективних напрямків у пост квантовій криптографії є криптографія в кільцях поліномів (на алгебраїчних решітках). Реалізації таких криптосистем вимагають ефективних алгоритмів множення поліномів. Запропоновані алгоритми враховують особливості структури тернарних поліномів у таких NTRU — подібних системах та забезпечують константність часу виконання операції множення. Для реалізації використовувалися кеш ефективні структури даних, AVX2 інструкції, розпаралелювання та спеціальні засоби для мінімізації та компенсування використання не вирівняної пам'яті. Критичний код написано на асемблері. Показано, що він працює вдвічі швидше за алгоритм, запропонований у [1] як для звичайної форми, так і для product-форми. Для звичайної форми відсоток прискорення — 38.6%, а для product-форми в 2 рази.

Список використаних джерел:

1. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja lange, and Christine van Vredendaal. NTRU Prime. URL: https://ntruprime.cr.yp.to/ntruprime-20160511.pdf.
2. Gorbenko I. D., Alekseychuk A. N., Kachko O. H., Yesina M. V., Bobukh V. A., Kandyi S. O., Ponomar V. A. Calculation of general parameters form NTRU Prime Ukraine of 6-7 levels of stability. *Radiotekhnika* : All-Ukr. Sci. Indep. Mag. Kharkiv : KNURE. 2019. № 195. P. 17–25.
3. Agner Fog. Optimizing subroutines in assembly language. URL: https://www.agner.org/optimize/ optimizing_assembly.pdf.

OPTIMIZATION OF THE MULTIPLY FUNCTION OF POLYNOMIALS FOR GENERAL AND PRODUCT FORMS OF THE REPRESENTATION OF ONE POLYNOMIAL

The research was carried out and the development of an effective practical algorithm for multiplying ternary polynomials in a ring $Z_3[x]\left(x^n - x - 1\right)$ was performed taking into account their structure. Variants for polynomials

with a normal structure with a fixed number of nonzero elements and in the PRODUCT-form in which the polynomial is the result of the calculation $F_1 * F_2 + F_3$, where $F_1, F_2, F_3 \in \mathrm{Z}_3[x]\left(x^n - x - 1\right)$ and have d_1, d_2, d_3 elements with values «1» and «–1» respectively, were considered. The results of optimization are given using vectorized instructions (AVX2 instructions), parallelization and special tools to minimize and compensate for the use of unbalanced memory. The critical code was written on the assembler under the microprocessor architecture x86-64, which is one of the most widely spread for today. Optimized version's time values for the parameter sets for 256, 384 and 512 bit classical of security were obtained and a comparison of the efficiency with the polynomial multiplication algorithm, which was proposed in an asymmetric post-quantum cryptosystem on algebraic NTRU Prime grids, was proposed. Testing was done on the Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz processor and on operating system Linux 4.15.0-44-generic #47-Ubuntu SMP x86-64. The results obtained in this paper are extremely relevant. They can be useful for cryptologists and other professionals involved in the development of new, effective cryptographic scheme and protocols for the post-quantum period, because for cryptosystem classes that use the modification in polynomial rings as basic operations, multiplication operation is operation that takes most of the time and requires significant optimization. The biggest advantage of the developed algorithm is the possibility of its parallelization on multiprocessor systems, which is a significant advantage over the algorithm presented in NTRU Prime.

Key words: *NTRU, multiplication of polynomials, ternary polynomial, PRODUCT-form, algebraic lattice, post quantum.*

Одержано 12.02.2019