# Прискорення операції множення поліномів (Цифровий підпис Dilithium)

Цифровий підпис Dilithium – фіналіст 2 раунду конкурсу NIST постквантових криптографічних алгоритмів( <https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions>)

Особлива увага при оптимізації наділяється операції множення поліномів.

Для множення поліномів необхідно виконати наступні функції:

Пряме NTT перетворення;

По компонентне множення;

Зворотне NTT перетворення

Автори Dilithium для реалізації цих функцій застосовували AVX-2 операції та мову асемблер.

Ми застосовували AVX-512 операції та мову C.

Застосування мови С дозволяє використовувати один і той же код для застосування в процесорах з різною архітектурою та для різних операційних систем.

Результати нашої реалізації наведені в таблиці для процесору Intel (R) Core (TM) i9-7960X CPU @ 2.80GHz, 2808 MHz cores, 16 logical processors: 32.

Компиляторы:

Microsoft Visual Studio Community 2019, Version 16.6.2, VisualStudio.16.Release / 16.6.2 + 30204.135, / o2 optimization flags

GCC 9.3, флаги оптимізації: -О3 –march=native,-mtune=native

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | Linux,  Dilithium (мова асемблер, GCC) | Linux  (мова с, GCC) | Windows 10, мова С,  msvc | Прискорення  Linux  (2/3) | Прискорення  (Windows)  (2/4) |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Пряме Ntt перетворення  Покомпонентне множення  Зворотне Ntt преобразование | 987  135  972 | 663  81  677 | 727  87  741 | 1.48  1.6  1.43 | 1.35  1.51  1.31 |

В таблиці наведені результати виміру кількості тактів для кожної функції. Для виміру кількості тактів застосовувалась команда процесора rdtsc або відповідна функція.

В другій колонці таблиці наведені результати для авторської реалізації (розробники Dilithium), які отримані з застосуванням ключів компілятора GCC, які застосовували автори (Linux).

В третій колонці наведені результати нашої реалізації для тих же ключів компілятору GCC, які застосовувалися для отримання попередніх результатів (Linux)

В четвертій колонці наведені результати нашої реалізації з застосуванням компілятору msvc

В наступних колонках наведено прискорення для нашої реалізації по відношенню до авторської реалізації.

Висновки:

1 За рахунок застосування 512 бітних операцій та спеціальних алгоритмів перемішування досягнуто суттєве прискорення незалежно від платформ

2 Застосування мови С замість мови асемблер дозволяє застосовувати крос платформений код

3 Як допоміжний результат наші експерименти показали, що застосування мови асемблер замість мови С не дозволяє отримати прискорення при умові обрання режимів компіляції з повною оптимізацією