УДК 004.315.7

[0000-0002-1596-4123] **А. О. Лавданський,** *канд. техн. наук, доцент,*
e-mail: a.lavdanskyi@chdtu.edu.ua
[0000-0002-2046-481X] **Е. В. Фауре,** *д-р техн. наук, професор,*
[0000-0003-3488-2753] **В. О. Щерба**
Черкаський державний технологічний університет
б-р Шевченка, 460, м. Черкаси, 18006, Україна

ПІДВИЩЕННЯ ШВИДКОСТІ ОПЕРАЦІЇ МНОЖЕННЯ ПЕРЕСТАНОВОК ЗА РАХУНОК ВИКОРИСТАННЯ SIMD ІНСТРУКЦІЙ

У роботі розроблено та досліджено алгоритми виконання множення перестановок за допомогою використання SIMD інструкцій сучасних процесорів. Виконано аналіз SIMD інструкцій, що можуть бути використані для виконання операцій над перестановками. Розроблені алгоритми базуються на використанні розширених інструкцій процесорів, що дають змогу виконувати операції над даними, представленими у векторному форматі. Практично визначено та досліджено переваги використання SIMD інструкцій для підвищення швидкості виконання операцій над перестановками. Проведено аналіз та порівняння швидкості виконання операцій над перестановками з та без використання SIMD інструкцій. Розроблені алгоритми можуть бути використані при реалізації методів, що базуються на великій кількості операцій множення перестановок, що дасть можливість значно підвищити швидкість їх виконання.

Ключові слова: *процесор, алгоритм, вектор, регістр, SSSE3, AVX2.*

Вступ. Сучасні процесори є досить потужними, кількість виконуваних ними операцій за секунду обчислюється мільярдами, а кількість обчислювальних ядер – десятками. Проте існують часто використовувані алгоритми (наприклад, функції хешування та шифрування, операції над зображеннями, операції над даними, що представлені у векторному вигляді, тощо), апаратна реалізація яких може значно пришвидшити та зменшити енергоспоживання під час виконання розрахунків [1]-[4]. Виробники процесорів реалізують такі апаратні блоки в своїх продуктах та додають нові інструкції у системи команд процесорів для їх зручного використання.

Значне прискорення операцій над даними може бути досягнуто за виконання операцій, вхідні та вихідні дані яких представлені у векторному форматі. У такому випадку операції можуть бути виконані паралельно над усіма елементами вектора одночасно. Такий принцип обчислень називають SIMD (Single Instruction Multiple Data) за класифікацією, що наведено у [5], [6]. SIMD інструкції різного роду реалізовані у багатьох архітектурах процесорів, зокрема для найбільш популярних x86 і x86-64 (MMX, 3DNow!, SSE, SSE2, SSE3, SSSE3, SSE4.x, AVX, AVX2, AVX512) та ARM (NEON). Повний список SIMD інструкцій, що реалізовані у процесорах Intel, наведений у [7].

Варто зазначити, що не тільки сучасні процесори можуть ефективно виконувати команди над даними, що представлені у векторному форматі. Сучасні відеоприскорювачі (відеокарти) можуть містити тисячі обчислювальних ядер (наприклад, до 10496 скалярних процесорів (ядер CUDA) у сучасних відеокартах компанії NVIDIA [8]), що можуть бути використані для прискорення операцій над даними, представленими у векторному форматі. Разом з тим, не всі комп'ютерні системи обладнані вбудованими або дискретними відеоприскорювачами необхідної обчислювальної потужності, а деякі взагалі їх позбавлені (наприклад, серверні системи). Саме тому задача розробки засобів прискорення операцій над перестановками за допомогою SIMD інструкцій сучасних процесорів є актуальною.

Зауважимо, що представити дані у векторному вигляді для їх обробки за допомогою SIMD інструкцій не завжди вдається. Якщо наступне значення операнду залежить від попереднього – паралельна обробка операндів неможлива. Проте операції над перестановками переважно виконуються одночасно над

DOI: 10.24025/2306-4412.3.2021.245347

усіма елементами перестановки, саме тому використання SIMD інструкцій є оптимальним для операцій над перестановками.

Мета та задачі дослідження. Метою роботи є підвищення продуктивності та зменшення енергоспоживання процесора під час виконання ним типових операцій над перестановками за рахунок створення алгоритмів для виконання операцій над перестановками з використанням SIMD інструкцій. Для виконання поставленої мети необхідно вирішити наступні задачі:

- виконати аналіз SIMD інструкцій, що можуть бути використані для виконання операцій над перестановками;
- виконати розробку алгоритмів для виконання операцій над перестановками з використанням SIMD інструкцій;
- виконати аналіз та порівняння швидкості виконання операцій над перестановками з та без використання SIMD інструкцій.

Виклад основного матеріалу. У цій роботі розглянемо реалізації прискорення операцій над перестановками за допомогою SIMD інструкцій архітектури x86-64. Над перестановками можуть виконуватися такі операції: множення перестановок, пошук зворотної перестановки, розкладання на добуток циклів, пошук порядку перестановки, визначення парності перестановки та інші. Зокрема, для трьохетапного криптографічного протоколу на основі перестановок [9] використовуються такі операції, як множення перестановок, пошук зворотної перестановки та розкладання перестановки на добуток циклів. Також перестановки використовуються у системах, що функціонують на основі факторіального кодування [10]-[13]. Прискорення зазначених операцій дасть змогу значно зменшити час, необхідний для виконання криптографічного перетворення повідомлень.

Однією з ключових операцій над перестановками є їх множення. Більшість операцій, що використовуються в трьохетапному криптографічному протоколі [9], є операції множення перестановок.

Добутком перестановок A та B довжини M є перестановка $C = A \cdot B$ така, що $C[i] = A[B[i]]$, де $A[i]$, $B[i]$, $C[i]$ – елементи перестановок, що знаходяться на позиції i, $i \in [0; M-1]$. Прискорення операцій множення перестановок дасть можливість значно пришвидшити реалізації алгоритмів, що використовують ці операції.

Для проведення дослідження виконаємо реалізацію алгоритму (назвемо його «еталонний алгоритм») множення перестановок за допомогою мови програмування C (лістинг 1) без використання SIMD інструкцій. Змінні *first* та *second* є масивами з вхідними перестановками, результат множення зберігається у масиві *result*. Розмірність вхідних та результуючого масиву (довжина перестановки) визначається константою *ARR_SIZE*.

```
for (int i = 0; i < ARR_SIZE; i++) {
    result[i] = first[second[i]];
}
```

Лістинг 1 – Програмний код виконання множення перестановок на мові програмування C без використання SIMD інструкцій

Під час використання SIMD інструкцій дані, над якими необхідно виконати операції, необхідно помістити у спеціальні регістри. Доступні різні розміри регістрів (64, 128, 256, 512 біт залежно від інструкції), що можуть вміщувати як декілька цілих чисел, так і декілька чисел з плаваючою крапкою. Оскільки перестановкою є послідовність невід'ємних цілих чисел діапазону $[0, M-1]$, а розмірність SIMD регістрів дорівнює 2^n, де $n \geq 6$ (може досягати значення 9 для інструкцій AVX512), будемо використовувати довжину перестановок $M=2^n$. Це дасть змогу упакувати декілька слів перестановки в стандартні регістри SIMD інструкцій для проведення операцій над ними. Наприклад, для $n=8$ до регістру довжиною 64 біт можна помістити 8 слів перестановки.

Для виконання операції множення перестановок зручно використовувати SIMD інструкції, що дають можливість виконувати перемішування слів у регістрі. Зокрема, інструкції pshufd (SSE2), pshufb (SSSE3), vpshufb (AVX2) та vpshufb (AVX512) дають змогу перемістити слово першого регістра

DOI: 10.24025/2306-4412.3.2021.245347

за індексом, що знаходиться у другому регістрі.

Інструкція pshufw (SSE) дає змогу виконувати обробку чотирьох 16-розрядних слів перестановки, інструкція pshufd (SSE2) – чотирьох 32-розрядних слів, інструкція pshufb (SSSE3) – 16 або восьми 8-розрядних слів, інструкція vpshufd (AVX2) – восьми 32-розрядних слів, інструкція vpshufb (AVX2) – 32 8-розрядних слів, інструкція vpshufb (AVX512) – 64 8-розрядних слів, інструкція vpshufd (AVX512) – 16 32-розрядних слів. Зауважимо, що інструкції vpshufb (AVX2) та vpshufd (AVX2) виконують обробку даних у старших та молодших 128 бітах регістрів незалежно, тобто ці інструкції варто розглядати не як можливість обробки восьми 32-розрядних (або 32 8-розрядних) чисел, а як виконання обробки чотирьох 32-розрядних (або 16 8-розрядних) чисел одночасно [14].

Розглянемо сімейство інструкцій *pshuf* більш докладно. Залежно від інструкції вхідні, вихідні та регістри індексів можуть бути довжиною 64, 128, 256 та 512 біт. Тому для виконання множення перестановок необхідно упакувати слова перестановок у вхідні регістри. Це може бути виконано за допомогою спеціальних інструкцій сімейства *movdqa*, що дають змогу сформувати так званий вирівняний (aligned) формат регістра з даних з оперативної пам'яті. Також ці інструкції можуть бути використані для розпакування результуючого значення до оперативної пам'яті.

Для зручності проведення дослідження будемо використовувати функції-обгортки інструкцій на мові програмування С. Для інструкції pshufb (SSSE3) це

__m128i _mm_shuffle_epi8 (__m128i a, __m128i b).

Для проведення множення перестановок за допомогою інструкції pshufb (SSSE3) в регістри необхідно завантажити упаковані (послідовність із 16 слів перестановки розрядністю 8 біт кожна) перестановки. Це може бути виконано за допомогою інструкції movdqa (SSE2), що може бути викликана за допомогою функції

__m128i _mm_load_si128 (__m128i const mem_addr).*

Для отримання результату необхідно буде розпакувати дані, отримані у 128-бітному регістрі, виконаємо це за допомогою інструкції movdqa (SSE2), що може бути викликана за допомогою функції

void _mm_store_si128 (__m128i mem_addr, __m128i a).*

Програмний код для виконання множення перестановок за допомогою інструкції pshufb (SSSE3) наведено у лістингу 2. Варто зазначити, що розмірність вхідних та результуючого масиву повинна бути 16 слів (перестановка з 16 елементів, $M=2^4$).

```
__m128i f = _mm_load_si128((__m128i*) & first[0]);
__m128i s = _mm_load_si128((__m128i*) & second[0]);
m128i r = _mm_shuffle_epi8(f, s);
_mm_store_si128((__m128i*) & result[0], r);
```

Лістинг 2 – Програмний код для виконання множення перестановок за допомогою інструкції pshufb (SSSE3)

Для прикладу, на рисунку 1 зображено структурно схему виконання множення перестановок $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 3 & 2 & 6 & 5 & 4 & 0 & 7 \end{pmatrix}$ і $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 4 & 7 & 2 & 1 & 0 & 3 \end{pmatrix}$. Результатом такого множення буде перестановка $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 0 & 5 & 7 & 2 & 3 & 1 & 6 \end{pmatrix}$.

Наведена реалізація алгоритму множення перестановок з використанням SIMD інструкцій може бути використана для невеликих за розміром перестановок (16 або 32 слова). Проте часто виникає задача виконання множення перестановок значно більшої довжини (тисячі і більше елементів). У такому випадку використання сімейства інструкцій *pshuf* неможливе без додаткових модифікацій алгоритму.

Для виконання множення перестановок довільного розміру розглянемо використання інструкції vpgatherdd (AVX2). Обгортка цієї інструкції на мові програмування С має наступний вигляд:

DOI: 10.24025/2306-4412.3.2021.245347

__m256i _mm256_i32gather_epi32 (int const base_addr, __m256i vindex, const int scale)*

Інструкція vpgatherdd отримує 32-розрядні цілі числа з пам'яті за допомогою 32-розрядних індексів. 32-розрядні елементи завантажуються за адресами, що починаються з *base_addr*, і зміщуються кожним 32-розрядним елементом у *vindex* (кожний індекс масштабується за коефіцієнтом, вказаним у *scale*). Зібрані елементи об'єднуються в *dst* (повертаються функцією). Значення *scale* повинне бути 1, 2, 4 або 8 [7].

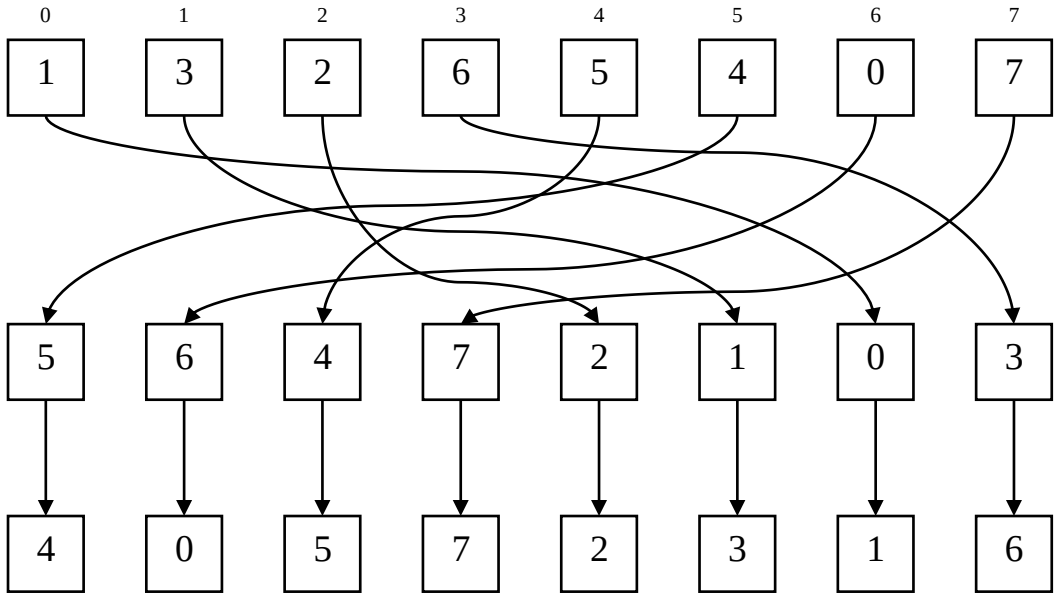

Рисунок 1 – Структурна схема виконання операції множення перестановок

Оскільки розмірність індексного регістру *vindex* становить 256 біт, а індекс для завантаження з пам'яті повинен бути 32-розрядним, обробка перестановки відбувається групами по 256/32=8 слів.

Варто зазначити, що інструкція vpgatherdd фактично виконує завантаження значень з оперативної пам'яті, яка, з точки зору процесора, є досить повільною (за час отримання даних проходять сотні тактів процесора). Тому, хоча й використання інструкції vpgatherdd є найпростішим, проте не є найефективнішим методом виконання множення перестановок.

```
for (int i = 0; i < ARR_SIZE; i += 8) {
    __m256i vindex = _mm256_load_si256((__m256i*) & second[i]);
    __m256i r = _mm256_i32gather_epi32((int*)&first[0], vindex, 4);
    _mm256_store_si256((__m256i*) & result[i], r);
}
```

Лістинг 3 – Програмний код для виконання множення перестановок за допомогою інструкції vpgatherdd (AVX2)

Результати досліджень. Для проведення порівняння та оцінки продуктивності алгоритмів з та без використання SIMD інструкцій будемо використовувати бібліотеку google/benchmark [15]. Вона дає можливість визначити та порівняти час, необхідний для проведення розрахунків, а також провести оцінку ступеня прискорення алгоритму, зокрема за використання SIMD інструкцій.

Оскільки SIMD інструкції реалізуються фізично на кристалі процесора, переваги від використання таких інструкцій можуть відрі-

DOI: 10.24025/2306-4412.3.2021.245347

знятися залежно від виробника. Тому для проведення досліджень використано дві платформи різних виробників.

Тестування проводилося з використанням процесорів Intel та AMD у таких конфігураціях:

- процесор INTEL Core i5 10400 (модулі пам'яті – DDR4 32GB (2x16GB) 2666 MHz);
- процесор AMD Ryzen 5 3600 (модулі пам'яті – DDR4 32GB (2x16GB) 3200 MHz).

Зауважимо, що метою дослідження не є порівняння продуктивності апаратного забезпечення різних виробників. Основною задачею є дослідження переваг використання SIMD інструкцій за допомогою конкретного апаратного забезпечення.

Результати порівняння еталонного алгоритму (розмір перестановки *ARR_SIZE* 16 слів) з алгоритмом з використанням SIMD інструкцій (лістинг 2) наведено на рисунку 2. У процесі тестування процесори працювали у режимі максимальної частоти.

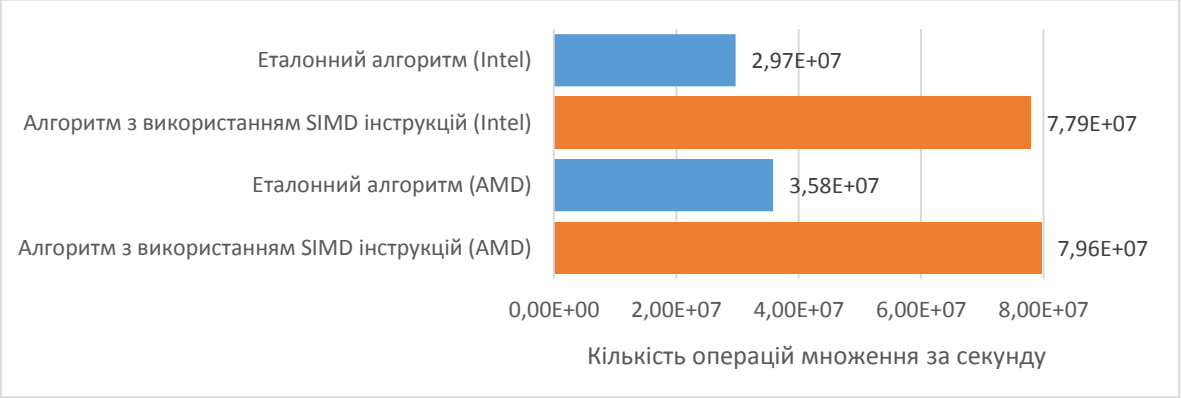

Рисунок 2 – Порівняння продуктивності еталонного алгоритму та алгоритму з використанням SIMD інструкцій

Відповідно до рисунка 2 використання інструкції pshufb (SSSE3) дає змогу прискорити виконання операцій множення перестановок у 2,2 разу для процесора AMD Ryzen 5 3600 та у 2,6 разу для процесора INTEL Core i5 10400.

Програмний код, наведений у лістингу 3, може бути використаний для виконання множення перестановок для будь-якого значення *M*, що кратне 8. Для значення *M*, не кратного 8, необхідно доповнити перестановку до довжини, кратної 8, словами, значення яких відповідає індексу слова в перестановці. Після виконання множення ці додаткові слова необхідно відкинути.

Проведемо порівняння алгоритму множення перестановок, програмну реалізацію якого наведено в лістингу 3, з еталонною реалізацією для різних значень *M*. Результати дослідження зведено до таблиць 1 і 2.

Таблиця 1 – **Результати виконання множення перестановок довільної довжини для процесора Intel Core i5 10400**

Алгоритм	*M*	Кількість операцій множення за секунду (середнє з трьох спроб)	Прискорення
Еталонний алгоритм	16	$28.3320 * 10^6$	1.00
	32	$13.6533 * 10^6$	1.00
	128	$4.34424 * 10^6$	1.00
	512	$1.14688 * 10^6$	1.00
	4096	$146.286 * 10^3$	1.00
Алгоритм з використанням інструкції vpgatherdd (AVX2)	16	$49.0120 * 10^6$	1.72
	32	$35.3975 * 10^6$	2.59
	128	$10.8196 * 10^6$	2.49
	512	$2.89909 * 10^6$	2.52
	4096	$366.649 * 10^3$	2.50

DOI: 10.24025/2306-4412.3.2021.245347

Таблиця 2 – **Результати виконання множення перестановок довільної довжини для процесора AMD Ryzen 5 3600**

Алгоритм	*M*	Кількість операцій множення за секунду (середнє з трьох спроб)	Прискорення
Еталонний алгоритм	16	$35.3975 * 10^6$	1.00
	32	$21.4933 * 10^6$	1.00
	128	$6.37156 * 10^6$	1.00
	512	$1.70667 * 10^6$	1.00
	4096	$219.709 * 10^3$	1.00
Алгоритм з використанням інструкції vpgatherdd (AVX2)	16	$38.3316 * 10^6$	1.08
	32	$26.0655 * 10^6$	1.21
	128	$7.58519 * 10^6$	1.19
	512	$2.04800 * 10^6$	1.19
	4096	$254.862 * 10^3$	1.15

Обговорення результатів. Наведені реалізації алгоритмів множення перестановок з використанням SIMD інструкцій можуть бути використані для прискорення операцій над перестановками, зокрема в трьохетапному криптографічному протоколі на основі перестановок [9]. Використання SIMD інструкцій дає можливість прискорити операції множення перестановок до 2,6 разу (залежно від використаного апаратного забезпечення).

Варто зазначити, що залежно від реалізації апаратного забезпечення, результати використання SIMD інструкцій можуть значно відрізнятися. Зокрема, відповідно до таблиць 1 та 2 використання інструкції vpgatherdd (AVX2) на процесорі INTEL Core i5 10400 дає можливість отримати прискорення в середньому 2,5 разу. У той же час використання тієї самої інструкції процесора AMD Ryzen 5 3600 дає можливість досягти прискорення лише в 1,2 разу. Використання інших моделей процесорів також приведе до різних результатів.

Виконання операції множення перестановок за допомогою інструкції pshufb (SSSE3) для розміру перестановки 16 слів дає змогу досягти прискорення у 2,2 разу для процесора AMD Ryzen 5 3600 та у 2,6 разу для процесора INTEL Core i5 10400 відносно еталонного алгоритму.

Висновки. У роботі детально досліджено питання швидкодії однієї з найтиповіших процесорних операцій над перестановками – їх множення.

Наукова новизна роботи полягає в запропонованому підході до апаратної реалізації множення перестановок, який за рахунок застосування SIMD інструкцій дає можливість зменшити час виконання цієї операції і, як наслідок, підвищити продуктивність і зменшити енергоспоживання процесора.

Практичну цінність роботи обумовлюють розроблені алгоритми множення перестановок, ефективність яких підтверджено порівнянням з класичними процедурами множення.

Разом з тим, перспективним є напрям пошуку методів виконання множення перестановок довільної довжини з використанням SIMD інструкцій сімейства *pshuf*, що дасть змогу досягти суттєвого приросту продуктивності, порівнюючи з алгоритмом, що використовує інструкції vpgatherdd (AVX2). Також використання відеоприскорювачів теоретично дасть можливість значно прискорити виконання операцій над перестановками за рахунок виконання обчислень паралельно за допомогою багатьох обчислювальних блоків відеокарт. Ці питання є предметом подальших досліджень.

Подяка. Роботу виконано в рамках науково-технічної (експериментальної) розробки молодих вчених «Розробка мобільної системи захищеного інформаційного обміну для військових і цивільних підрозділів державних структур» (№ ДР 0120U102607), а також стипендіальної роботи «Розробка методів і протоколів інтегрованого захисту інформації на основі факторіального кодування даних для мобільної системи інформаційного обміну» (№ ДР 0121U112454) в рамках іменної стипендії Верховної Ради України для молодих учених – докторів наук (Постанова Верховної Ради України від 14.07.2021 № 1641-IX).

DOI: 10.24025/2306-4412.3.2021.245347

Список використаних джерел

[1] W. Mula, and D. Lemire, "Faster Base64 encoding and decoding using AVX2 instructions", *ACM Transactions on the Web (TWEB)*, vol. 12, no. 3, pp. 1-26, 2018.

[2] A. Faz-Hernández, and J. López, "Fast implementation of Curve25519 using AVX2", in *International Conference on Cryptology and Information Security in Latin America*, 2015, pp. 329-345.

[3] A. Lemmetti, A. Koivula, M. Viitanen, J. Vanne, and T. D. Hämäläinen, "AVX2-optimized Kvazaar HEVC intra encoder", in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 549-553.

[4] W. Mula, N. Kurz, and D. Lemire, "Faster population counts using AVX2 instructions", *The Computer Journal*, vol. 61, no. 1, pp. 111-120, 2018.

[5] M. J. Flynn, "Very high-speed computing systems", *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901-1909, 1966.

[6] M. J. Flynn, "Some computer organizations and their effectiveness", *IEEE Trans. Comput.*, vol. 21, no. 9, pp. 948-960, 1972.

[7] "Intel® Intrinsics Guide". [Online]. Available at: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html.

[8] "GeForce RTX 3090 Graphics Card | NVIDIA". [Online]. Available at: https://www.nvidia.com/en-eu/geforce/graphics-cards/30-series/rtx-3090/.

[9] A. Shcherba, E. Faure, and O. Lavdanska, "Three-pass cryptographic protocol based on permutations", in *2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT)*, 2020, pp. 281-284.

[10] Е. В. Фауре, О. О. Харін, та А. О. Лавданський, "Оцінка властивостей синтезованих на основі теорії решіток сигнально-кодових конструкцій для нероздільних факторіальних кодів", *Вісник Черкаського державного технологічного університету*, № 3, с. 40-47, 2020.

[11] Е. В. Фауре, В. В. Швидкий, А. І. Щерба, О. О. Харін, та Б. А. Ступка, "Метод циклової синхронізації на основі перестановок", *Вісник Черкаського державного технологічного університету*, № 4, с. 67-76, 2020.

[12] E. V. Faure, V. V. Shvydkyi, A. O. Lavdanskyi, and O. O. Kharin, "Methods of factorial coding of speech signals", *Radio Electronics, Computer Science, Control*, no. 4, pp. 186-198, Nov. 2019.

[13] E. Faure, A. Shcherba, Y. Vasiliu, and A. Fesenko, "Cryptographic key exchange method for data factorial coding", vol. 2654, p. 643, Aug. 2020.

[14] "Programming using AVX2. Permutations". [Online]. Available at: https://software.intel.com/content/www/us/en/develop/blogs/programming-using-avx2-permutations.html.

[15] "google/benchmark: A microbenchmark support library". [Online]. Available at: https://github.com/google/benchmark.

References

[1] W. Mula, and D. Lemire, "Faster Base64 encoding and decoding using AVX2 instructions", *ACM Transactions on the Web (TWEB)*, vol. 12, no. 3, pp. 1-26, 2018.

[2] A. Faz-Hernández, and J. López, "Fast implementation of Curve25519 using AVX2", in *International Conference on Cryptology and Information Security in Latin America*, 2015, pp. 329-345.

[3] A. Lemmetti, A. Koivula, M. Viitanen, J. Vanne, and T. D. Hämäläinen, "AVX2-optimized Kvazaar HEVC intra encoder", in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 549-553.

[4] W. Mula, N. Kurz, and D. Lemire, "Faster population counts using AVX2 instructions", *The Computer Journal*, vol. 61, no. 1, pp. 111-120, 2018.

[5] M. J. Flynn, "Very high-speed computing systems", *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901-1909, 1966.

[6] M. J. Flynn, "Some computer organizations and their effectiveness", *IEEE Trans. Comput.*, vol. 21, no. 9, pp. 948-960, 1972.

[7] "Intel® Intrinsics Guide". [Online]. Available at: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html.

[8] "GeForce RTX 3090 Graphics Card | NVIDIA". [Online]. Available at: https://www.nvidia.com/en-eu/geforce/graphics-cards/30-series/rtx-3090/.

[9] A. Shcherba, E. Faure, and O. Lavdanska, "Three-pass cryptographic protocol based on permutations", in *2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT)*, 2020, pp. 281-284.

DOI: 10.24025/2306-4412.3.2021.245347

[10] E. V. Faure, O. O. Kharin, and A. O. Lavdanskyi, "Evaluation of properties of signal-code structures synthesized on the basis of lattice theory for inseparable factorial codes", *Visnyk Cherkaskogo derzhavnogo tehnologichnogo universitetu*, no. 3, pp. 40-47, 2020 [in Ukrainian].

[11] E. V. Faure, V. V. Shvydkyi, A. I. Shcherba, O. O. Kharin, and B. A. Stupka, "Method of cyclic synchronization based on permutations", *Visnyk Cherkaskogo derzhavnogo tehnologichnogo universitetu*, no. 4, pp. 67-76, 2020 [in Ukrainian].

[12] E. V. Faure, V. V. Shvydkyi, A. O. Lavdanskyi, and O. O. Kharin, "Methods of factorial coding of speech signals", *Radio Electronics, Computer Science, Control*, no. 4, pp. 186-198, Nov. 2019.

[13] E. Faure, A. Shcherba, Y. Vasiliu, and A. Fesenko, "Cryptographic key exchange method for data factorial coding", vol. 2654, p. 643, Aug. 2020.

[14] "Programming using AVX2. Permutations". [Online]. Available at: https://software.intel.com/content/www/us/en/develop/blogs/programming-using-avx2-permutations.html.

[15] "google/benchmark: A microbenchmark support library". [Online]. Available at: https://github.com/google/benchmark.

A. O. Lavdanskyi, *Ph. D., Associate Professor,*
e-mail: a.lavdanskyi@chdtu.edu.ua
E. V. Faure, *Dr. Sc., Professor,*
V. O. Shcherba
Cherkasy State Technological University
Shevchenko blvd, 460, Cherkasy, 18006, Ukraine

INCREASING THE SPEED OF THE PERMUTATION MULTIPLICATION OPERATION DUE TO USE OF SIMD INSTRUCTIONS

Algorithms for performing permutation multiplication using SIMD (Single Instruction Multiple Data) instructions of modern processors are developed and investigated in the article. The purpose of the article is to increase the performance and reduce the power consumption of the processor during the executing of typical permutation operations by creating algorithms for performing permutation operations using SIMD instructions.

The scientific novelty of the article lies in the proposed approach to the hardware implementation of permutation multiplication, which through the use of SIMD instructions can reduce the execution time of this operation and, consequently, increase performance and reduce CPU power consumption. The practical value of the article is determined by the developed algorithms for permutation multiplication, the effectiveness of which is confirmed by comparison with classical multiplication procedures.

An analysis of SIMD instructions that can be used to perform permutation operations is performed. The developed algorithms are based on the use of advanced processor instructions that allow you to perform operations on data presented in vector format. The advantages of using SIMD instructions to increase the speed of permutation operations are practically identified and investigated. The analysis and comparison of the speed of permutation operations with and without the use of SIMD instructions are performed.

Described implementations of permutation multiplication algorithms using SIMD instructions can be used to accelerate permutation operations, in particular in the three-pass cryptographic protocol based on permutations. The use of SIMD instructions makes it possible to speed up permutation multiplication operations up to 2.6 times (depending on the hardware). The developed algorithms can be used in the implementation of methods based on a large number of permutation multiplication operations, which will significantly increase the speed of their execution.

Keywords: *processor, algorithm, vector, register, SSSE3, AVX2.*

Стаття надійшла 27.09.2021
Прийнято 15.10.2021

DOI: 10.24025/2306-4412.3.2021.245347