УДК 004.056

ГІБРІДНА ОПТІМІЗАЦІЯ АЛГОРИТМУ ХЕЩУВАННЯ DJB2 З ВИКОРИСТАННЯМ ІНСТРУКЦІЙ AVX2

Олександр Рисований, Олена Литовська

НТУ «харківський політехнічний інститут», м. Харків, Україна

Алгоритм хешування DJB2 є одним з кращих послідовних алгоритмів для хешування коротких рядків (наприклад, імен змінних або ключів у словниках). В асемблері він реалізується вкрай ефективно через зсуви (shl), що швидше за звичайне множення.

У найпростішому випадку хеш-функція послідовно обробляє символи або байти вхідного повідомлення та на кожному кроці змінює потокове значення хешу [1].

Призначається початкове значення хешу

hash = 5381

Для кожного байта (символу) рядка виконується операція:

hash = hash * 33 + c

Множення на 33 є характерною особливістю djb2. У низькорівневій реалізації множення можна представити через операції зсуву та додавання:

hash × 33 = hash × (32 + 1) = (hash << 5) + hash.

Тобто замість звичайного множення може використовуватися послідовність операцій зсуву та додавання. Наприклад, концептуально операцію можна представити як:

shl hash, 5

add hash, old_hash

add hash, c

або організувати обчислення таким чином, щоб початкове значення хешу не втрачалося. Такий підхід добре відповідає особливостям низькорівневого програмування і дозволяє зменшити обчислювальні витрати окремої ітерації.

Однак алгоритм djb2 має принципову особливість — послідовну залежність між ітераціями. Значення нового хешу залежить від результату попередньої ітерації. Якщо позначити значення хешу після обробки i-го байта як H_i, то:

H_(i+1) = 33 × H_i + c_i.

Отже, значення H_(i+1) неможливо отримати до завершення обчислення H_i. Саме ця властивість суттєво обмежує можливість прямої векторизації основного циклу djb2.

Тому просте замінювання послідовних операцій на VPADD, VPXOR або інші AVX2-інструкції не дає еквівалентної реалізації. Якщо незалежно обробляти всі 32 байти одного блока, не враховуючи залежність між значеннями хешу, результат уже не буде відповідати оригінальному алгоритму djb2.

У запропонованому підході використовується інша стратегія: математичне визначення djb2 залишається незмінним, а AVX2 застосовується до тих операцій, які дійсно можуть бути виконані паралельно. Зокрема, SIMD-інструкції використовуються для швидкого пошуку кінця нуль-термінованого рядка блоками по 32 байти.

Для цього застосовуються 256-бітні регістри YMM. Один регістр YMM може містити:

256 / 8 = 32 байти.

Таким чином, замість посимвольної перевірки:

```
cmp byte ptr [rcx+rdx],0
inc rdx
jmp ...
```

можна одночасно перевірити 32 байти.

Основна частина AVX2-обробки має вигляд:

```
vpxor ymm0,ymm0,ymm0
vmovdqu ymm1,ymmword ptr [r8+r9]
vpcmpeqb ymm2,ymm1,ymm0
vpmovmskb eax,ymm2
```

Спочатку за допомогою:

vpxor ymm0,ymm0,ymm0

формується нульовий вектор. У результаті всі 32 байти YMM0 дорівнюють нулю.

Далі виконується:

vmovdqu ymm1,ymmword ptr [r8+r9]

Ця інструкція завантажує з пам'яті 32 байти вхідного рядка до регістра YMM1. Оскільки використовується невирівняне завантаження VMOVDQU, адреса джерела не повинна бути обов'язково вирівняна на межу 32 байтів.

Таким чином, одним завантаженням отримується:

$32 \times 8 = 256$ біт.

Наступна інструкція:

vpcmpeqb ymm2,ymm1,ymm0

порівнює байти YMM1 з нульовими байтами YMM0. Для кожної позиції формується результат порівняння. Якщо відповідний байт дорівнює нулю, у відповідному байтовому елементі результату встановлюються одиниці.

Після цього:

vpmovmskb eax,ymm2

перетворює старші біти 32 байтових елементів YMM2 у 32-бітну маску в регістрі EAX. Кожен біт маски відповідає одному байту вихідного блока.

Якщо в оброблюваному блоці немає нульового байта, маска буде дорівнювати нулю. У такому випадку можна перейти до наступних 32 байтів:

r9 = r9 + 32

і повторити процедуру.

Якщо ж маска ненульова, це означає, що в поточному 32-байтовому блоці знайдено завершальний нульовий байт. Його точна позиція може бути визначена за допомогою інструкції TZCNT, яка знаходить позицію наймолодшого встановленого біта.

Такий підхід дозволяє значно скоротити кількість ітерацій під час пошуку кінця довгого рядка. Наприклад, для рядка довжиною приблизно 320 байтів скалярна реалізація виконає близько 320 перевірок байтів, тоді як AVX2-варіант працює приблизно з десятьма 32-байтовими блоками. При цьому одна AVX2-операція порівнює відразу 32 байти.

У низькорівневій реалізації множення на 33 може бути замінене зсувом і додаванням. Наприклад:

hash × 33 =

(hash << 5) + hash

після чого до отриманого значення додається черговий байт.

Важливо, що AVX2 у цьому випадку не змінює порядок обчислення хешу. Якщо вхідний рядок містить байти:

c0 c1 c2 c3 ...

то операції виконуються саме послідовно:

$H1 = 33 \times H0 + c0$

$H2 = 33 \times H1 + c1$

$H3 = 33 \times H2 + c2$

Це забезпечує повну відповідність результату модернізованої реалізації класичному алгоритму djb2.

Таким чином, запропонована оптимізація є не повною векторизацією хеш-функції, а гібридною SIMD-оптимізацією. Векторна частина виконує операції, які не мають залежності між сусідніми байтами, а скалярна частина зберігає послідовну математичну структуру djb2.

Такий принцип особливо важливий під час оптимізації алгоритмів хешування. Необхідно спочатку визначити залежності між операціями. Якщо кожна наступна операція використовує результат попередньої, повна SIMD-векторизація може бути неможливою або потребуватиме зміни структури алгоритму.

Іншим можливим підходом є використання декількох незалежних станів хешу. У такому випадку різні блоки даних можуть оброблятися паралельно, після чого результати комбінуються. Проте для djb2 це вже потребує математичного перетворення алгоритму, оскільки звичайний послідовний вираз:

H = (((5381 × 33 + c0) × 33 + c1) × 33 + c2) ...

не дозволяє довільно розбивати повідомлення на незалежні частини без урахування степенів множника 33.

Тому в межах запропонованої модернізації доцільно залишити обчислення djb2 у його початковому вигляді, а AVX2 застосовувати для прискорення пошуку довжини рядка та інших незалежних операцій попередньої обробки.

Під час реалізації необхідно враховувати також можливість виходу за межі доступної області пам'яті. Завантаження 32 байтів одним VMOVDQU повинно бути організоване таким чином, щоб читання не виходило за допустимі межі доступного буфера. Для нуль-термінованих рядків це питання особливо важливе, оскільки фізична довжина виділеної області пам'яті може бути меншою за 32 байти.

Необхідно також правильно обробляти ситуацію, коли довжина рядка не кратна 32. Наприклад, якщо рядок має 70 байтів, можна обробити перші 64 байти двома AVX2-блоками, а останні 6 байтів обробити скалярним кодом. Інший варіант — обробити блок, у якому міститься нульовий байт, повністю та використати отриману маску для визначення його точної позиції.

Окремої уваги потребує питання розрядності результату. У класичній реалізації djb2 часто використовується 32-бітне або 64-бітне беззнакове значення. При цьому переповнення розглядається відповідно до вибраної розрядності.

Це важливо під час порівняння різних реалізацій. Якщо скалярна та AVX2-версія використовують різну розрядність або різні правила обробки переповнення, результати можуть відрізнятися навіть у випадку однакової вхідної послідовності.

Для перевірки коректності модернізованої реалізації необхідно використовувати однаковий набір тестових рядків. Доцільно перевірити порожній рядок, короткі рядки, рядки довжиною менше 32 байтів, рядки довжиною рівно 32 байти, а також довші повідомлення, які складаються з декількох AVX2-блоків.

Для запропонованої реалізації основна ідея полягає у використанні AVX2 для одночасної перевірки 32 байтів:

```
vpxor ymm0,ymm0,ymm0
vmovdqu ymm1,ymmword ptr [r8+r9]
vpcmpeqb ymm2,ymm1,ymm0
vpmovmskb eax,ymm2
```

а потім – у використанні скалярного алгоритму djb2 для безпосереднього формування хеш-значення.

Для визначення ефективності AVX2-реалізації необхідно порівнювати її з базовою скалярною версією алгоритму.

Висновки. У роботі розглянуто можливість модернізації алгоритму хешування djb2 за допомогою SIMD-інструкцій набору AVX2. Показано, що використання 256-бітних регістрів YMM дозволяє виконувати однакові операції одночасно над декількома елементами даних і тим самим зменшувати кількість послідовних машинних операцій.

Запропонований підхід передбачає використання AVX2 у тій частині обробки, де існує реальний паралелізм, а саме під час пошуку нульового байта та визначення довжини вхідного рядка. Використання інструкцій VMOVDQU, VPCMPEQB, VPMOVMSKB та TZCNT дозволяє обробляти до 32 байтів за один векторний блок.

Важливою особливістю модернізованої реалізації є комбінування AVX2 та скалярного коду.

Ефективність модернізації необхідно визначати експериментально шляхом порівняння часу виконання базової та AVX2-реалізацій на однакових наборах даних. Основним показником може

бути коефіцієнт прискорення, який визначається відношенням часу скалярного виконання до часу AVX2-виконання.

Таким чином, використання AVX2 є перспективним напрямом оптимізації програмних реалізацій алгоритмів хешування, особливо під час обробки великих обсягів інформації. Водночас векторизація повинна розглядатися саме як метод підвищення продуктивності, а не як спосіб покращення криптографічних властивостей хеш-функції.

Поєднання оптимізації алгоритму, низькорівневої реалізації MASM64 та експериментального вимірювання продуктивності дозволяє отримати об'єктивну оцінку ефективності запропонованої модернізації. Результати такого дослідження також демонструють, що ефективне використання SIMD-інструкцій потребує не механічної заміни скалярних команд на векторні, а попереднього аналізу алгоритмічних залежностей та пошуку ділянок, які можуть бути виконані паралельно.

Список використаних джерел

1. Рисований О.М. Реверсне програмування. Хеш-функції та їх використання. Середовище програмування masm64: навчальний посібник для студентів спеціальності F7 "Комп'ютерна інженерія" всіх форм навчання [електронне видання] / О.М. Рисований. – Харків: НТУ «ХПІ», 2026. – 342 с. https://repository.kpi.kharkov.ua/handle/KhPI-Press/103824.

УДК 378.147.091.31:004.77

ОРГАНІЗАЦІЯ САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ ЗА ДОПОМОГОЮ ПЛАТФОРМИ MOODLE У МИКОЛАЇВСЬКОМУ НАЦІОНАЛЬНОМУ АГРАРНОМУ УНІВЕРСИТЕТІ

Микола Алєксєєв

Миколаївський національний аграрний університет, м. Миколаїв, Україна

Сучасний етап розвитку вищої аграрної освіти визначається активною цифровою трансформацією освітнього середовища та вимушеним переходом до змішаних і дистанційних форм навчання внаслідок безпекових викликів і військово-політичної ситуації в південному регіоні України [4]. Відповідно до нормативно-правових засад організації освітнього процесу в Миколаївському національному аграрному університеті (МНАУ), самостійна робота студентів (СРС) є основною формою засвоєння навчального матеріалу у час, вільний від обов'язкових аудиторних занять [3]. Обсяг СРС становить від 1/3 до 2/3 від загальної кількості навчальних годин за дисципліною [3].

Ефективним інструментом забезпечення та методичного супроводу СРС в МНАУ є система управління навчанням (LMS) Moodle, яка реалізує філософію педагогіки соціального конструкціонізму та забезпечує взаємодію суб'єктів освітнього процесу [1].

Таблиця 1. Фактичні показники активності здобувачів вищої освіти у LMS Moodle (за останні 365 днів)

№	Показник / Змінна	Позначення	Фактичне значення (БД moode)
1.	Загальна чисельність активних здобувачів вищо	Nст	12 769 осіб
2.	Кількість поданих індивідуальних завдань	Nз	33 604 од.
3.	Кількість виконаних тестових сесій	Nт	47 638 од.
4.	Сумарна кількість інтерактивних дій СРС	Nдій	81 242 од.
5.	Індекс цифровізації СРС	Iсрс	6,36 дій/студ.