

Перевірка векторизації інструкцій

Богдюк М.О., науковий керівник Коваленко О.Є.

Використання векторних інструкцій (таких, як SSE, AVX, Neon, надалі узагальнено «SIMD») є ключовим для підвищення продуктивності сучасних програм, особливо в обчислювально інтенсивних задачах, оскільки дозволяє обробляти відразу кілька даних за один такт процесора. Ручне формування SIMD-інструкцій є складним і схильним до помилок, тому розробники покладаються на механізми автоматичної векторизації компілятора [1]. Використовуючи певні вказівки [2], можна скоригувати код високого рівня так, щоб оптимізатор міг виконати векторизацію і при цьому зберегти читабельність коду.

Однак, навіть після первинної перевірки скомпільованого і оптимізованого бінарного коду на наявність SIMD-інструкцій, векторизація може бути втрачена внаслідок:

- подальших змін у коді,
- використання іншого компілятора,
- регресії в оптимізаторі.

Це створює потребу в інструменті, який би автоматично перевіряв факт векторизації конкретних рядків коду на етапі CI/CD (Continuous Integration / Delivery).

Метою даного дослідження є аналіз існуючих інструментів для перевірки векторизації та виявлення їхніх обмежень, а також обґрунтування необхідності створення нового інструменту, який би забезпечував автоматичну перевірку наявності SIMD-інструкцій у конкретних рядках коду в процесі безперервної інтеграції.

Методика. Проведено порівняльний аналіз існуючих інструментів за наступними критеріями:

1. Автоматизованість: можливість використання в CI/CD процесах, тобто автоматичного запуску без необхідності ручного втручання.
2. Деталізація: здатність аналізувати векторизацію на рівні окремих рядків коду.
3. Специфікація: можливість задати як конкретно очікувану інструкцію, так і набір інструкцій (чи ширину вектора)
4. Підтримка inline-функцій: здатність виявляти векторизацію у вбудованих функціях.
5. Стабільність результатів аналізу.

Огляд інструментів. Директива компілятора «`#pragma loop`» дозволяє вказати на необхідність векторизувати цикл, а також ширину вектора (кількість значень для одночасної обробки), і не можна вказати конкретну очікувану інструкцію. При використанні спеціальних директив чи синтаксичних конструкцій, оптимізатор може сигналізувати про невдачу векторизації. Проте, у ряді випадків відповідне попередження може не з'явитися, що призводить до непередбачуваної генерації неоптимізованого коду. Для прикладу, через помилки кодування, оптимізатор не видає попередження при обробці даного C++ фрагменту:

```
unsigned char is_zero[8];                // масив не ініціалізований
#pragma clang loop vectorize(enable)
for (int i = 0; i < 8; i++) {
    is_zero[i] |= cursor[i] ? 0 : 0xFF;   // «|=» замість «=»
}
if (*((unsigned long long*)is_zero)) { ... }
```

Компілятори мають режим виводу інструкцій асемблера (для clang це аргумент «`--assemble/-S`»). Нині розповсюджені підходи зводяться до ручного аналізу дизасемблера

на рівні функцій (наприклад, пошук `vroundsdb` чи інших AVX-опкодів через `grep`). В результаті, у файлі можна знайти функцію, в якій мала б відбутись оптимізація, і перевірити наявність інструкцій. Однак, цей інструмент не дозволяє відстежити інструкції у вбудованих функціях (`inline`). В результаті, при агресивному вбудовуванні інструкції можуть бути приховані в багатьох контекстах, що робить ручний аналіз трудомістким і не повноцінним.

Утиліта `llvm-mca` (LLVM Machine Code Analyzer) надає детальний аналіз продуктивності машинного коду, включаючи інформацію про використання ресурсів процесора: затримки, швидкість, кількість мікрооперацій, факт завантаження чи запису в пам'ять. Надзвичайно корисна при першому написанні критичного коду.

`Godbolt.org` також є альтернативою для ручного аналізу швидкодії, це онлайн-сервіс, який дозволяє аналізувати вивід компілятора та оптимізатора - як «`clang -S`», але ізольовано для кожної функції, без службового `asm` коду. Хоча він зручний для ручного аналізу, його неможливо інтегрувати в автоматизовані CI/CD процеси.

Профайлери (такі, як `perf`, Visual Studio Profiling Tools, Intel SDE) дозволяють аналізувати виконання програми на рівні інструкцій: скільки разів була викликана інструкція, як довго зайняло її виконання, а також низькорівневі параметри виконання: очікування конвеєра, кількість звернень до пам'яті (пропусків кешу). Проте вони вимагають виконання програми, що ускладнює їх використання для статичного аналізу в CI/CD, тому більш застосовні до модульних тестів швидкодії.

В табл. 1 наведено порівняння оглянутих інструментів за зазначеними на початку параметрами:

Таблиця 1. Порівняння інструментів для перевірки векторизації

Засіб	Автоматизованість	Рівень деталізації	Специфікація	inline	Стабільність
<code>#pragma loop</code>	+	3 = циклу	2 сімейство	+	-
<code>clang -S</code>	+	1 = модуля	2 інструкцію	-	+
<code>llvm-mca</code>	+	4 = інструкції	3 повна	-	+
<code>Godbolt</code>	-	2 = функції	2 інструкцію	+	+
Профайлер	unit test	4 = інструкції	2 інструкцію	+	+

Висновок. Відсутність чіткого інструментального рішення для верифікації того, які саме інструкції потрапили до фінального бінарного коду, може призводити до невідповідностей між очікуваною та реальною продуктивністю. Існуючі інструменти мають обмеження щодо автоматизації, деталізації та інтеграції в CI/CD процеси. Це підкреслює необхідність розробки нового інструменту, який би забезпечував автоматичну перевірку наявності SIMD-інструкцій у конкретних рядках коду, підтримував аналіз інлайн-функцій та легко інтегрувався в процеси безперервної інтеграції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- [1] Feng, Jing Ge, He, Ye Ping, Tao, Qiu Ming, Evaluation of Compilers' Capability of Automatic Vectorization Based on Source Code Analysis, *Scientific Programming*, 2021, 3264624, 15 pages, 2021. <https://doi.org/10.1155/2021/3264624>
- [2] Intel Corporation. Use automatic vectorization (DPC++/C++ Compiler Developer Guide and Reference). 2023. URL: <https://www.intel.com/content/www/us/en/docs/dpcpp-cpp-compiler/developer-guide-reference/2023-1/use-automatic-vectorization.html>

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**



**ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ
АСПЕКТИ РОЗРОБКИ
КОМП'ЮТЕРНИХ СИСТЕМ**

**ЗБІРНИК НАУКОВИХ ПРАЦЬ ЗА МАТЕРІАЛАМИ
VII ВСЕУКРАЇНСЬКОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ
СТУДЕНТІВ І АСПІРАНТІВ
*24 квітня 2025 року***

Київ 2025

УДК 004

Відповідальна за випуск: М.І. Лендел

Збірник наукових праць за матеріалами VII Всеукраїнської науково-практичної конференції студентів і аспірантів «ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ РОЗРОБКИ КОМП'ЮТЕРНИХ СИСТЕМ '2025», 24 квітня 2025 року, НУБіП України, Київ. – 452 с. (електронне видання)

Відповідальність за зміст публікацій несуть автори.

Передрук матеріалів, а також використання їх будь-якій формі допускається лише з дозволу авторів