Київський національний університет імені Тараса Шевченка

Факультет радіофізики, електроніки та комп’ютерних систем

Навчальна дисципліна «Комп’ютерні системи»

**Звіт з лабораторної роботи №3**

на тему «**Дослідження оптимізації коду**

**з використанням векторних розширень CPU**»

Роботу виконав

Студент 3 курсу

КІ, група СА

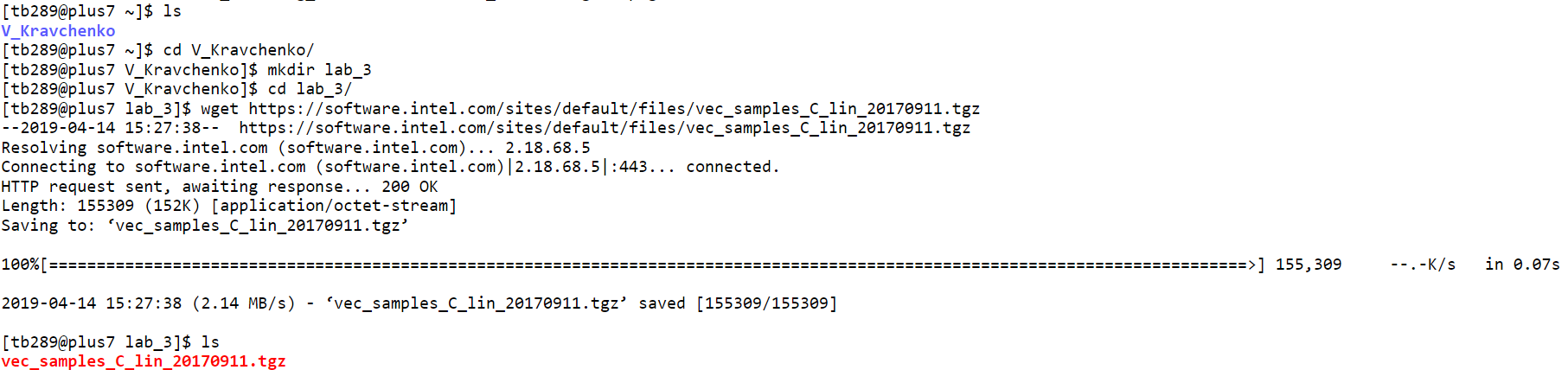
Кравченко В’ячеслав

Васильович

Київ 2019

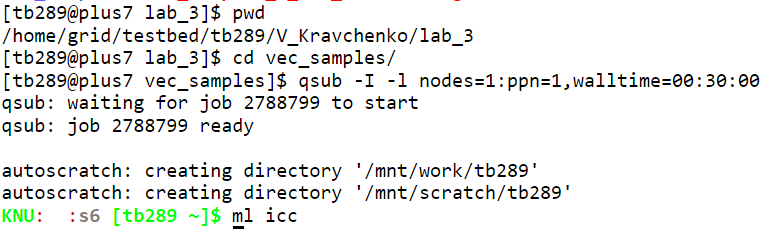
**Хід роботи**

1. Завантажте файли Intel® C++ Compiler - Using Auto-Vectorization Tutorial на свій комп’ютер та в домашню директорію користувача обчислювального кластеру.

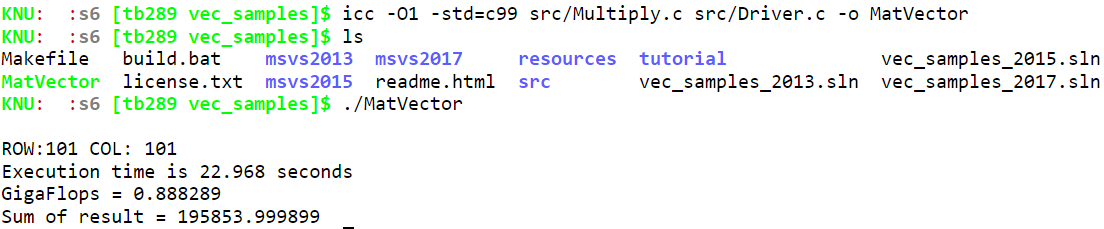


1. Використовуючи інструкції в readme.html ознайомтесь та виконайте Tutorial на обчислювальному кластері

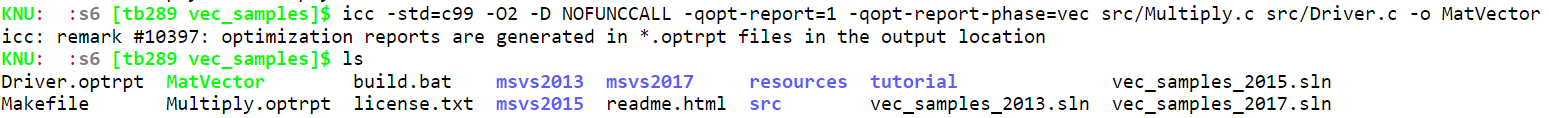
* Замість інструкцій в пункті “Setting the Environment Variables” завантажте оточення компілятора шляхом виконання команди: **ml icc**
* Виконуйте завдання на робочих вузлах кластеру замість вхідної ноди. Рекомендований варіант виконання роботи - використання інтерактивних задач в системі планування:
* [manf@plus7 ~]$ qsub -I -l nodes=1:ppn=1,walltime=00:30:00
* KNU:WN:s5 [manf ~]$ ml icc

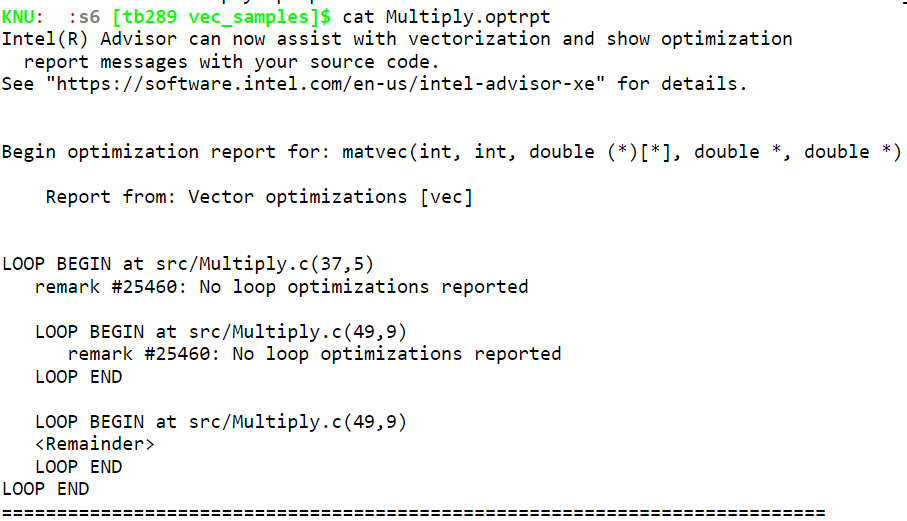


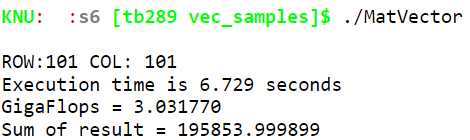
* Establishing a Performance Baseline



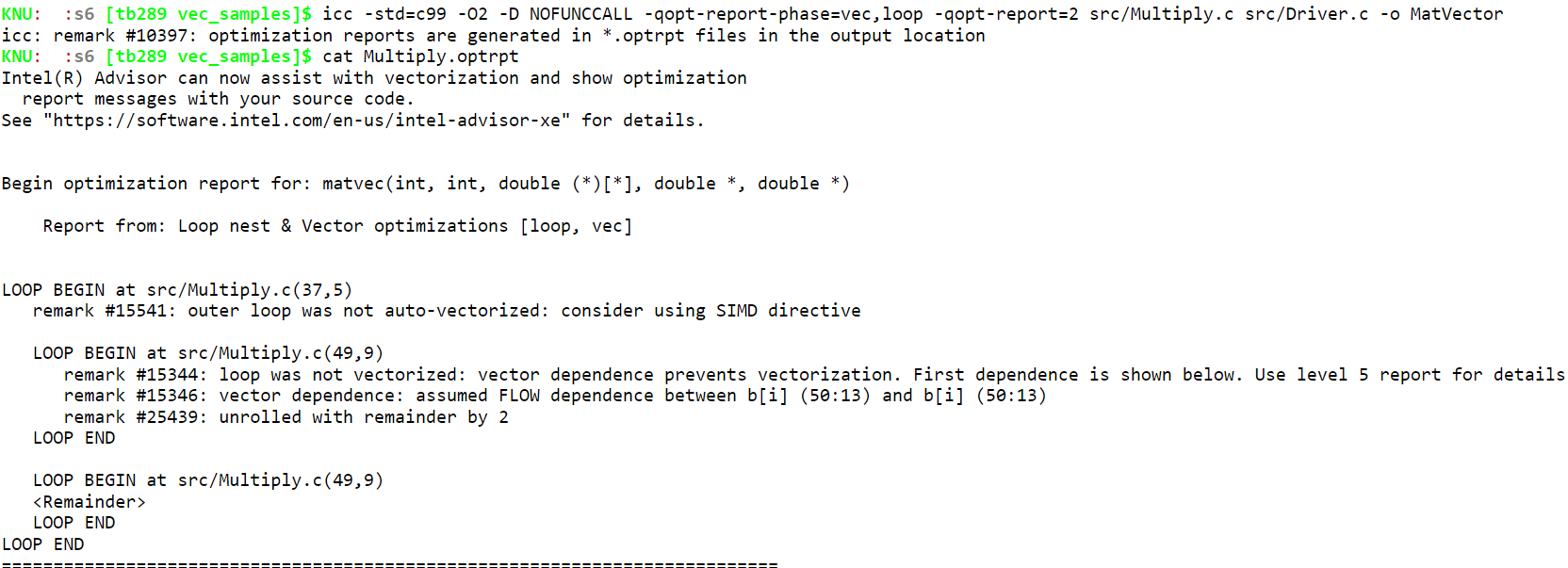
* Generating a Vectorization Report



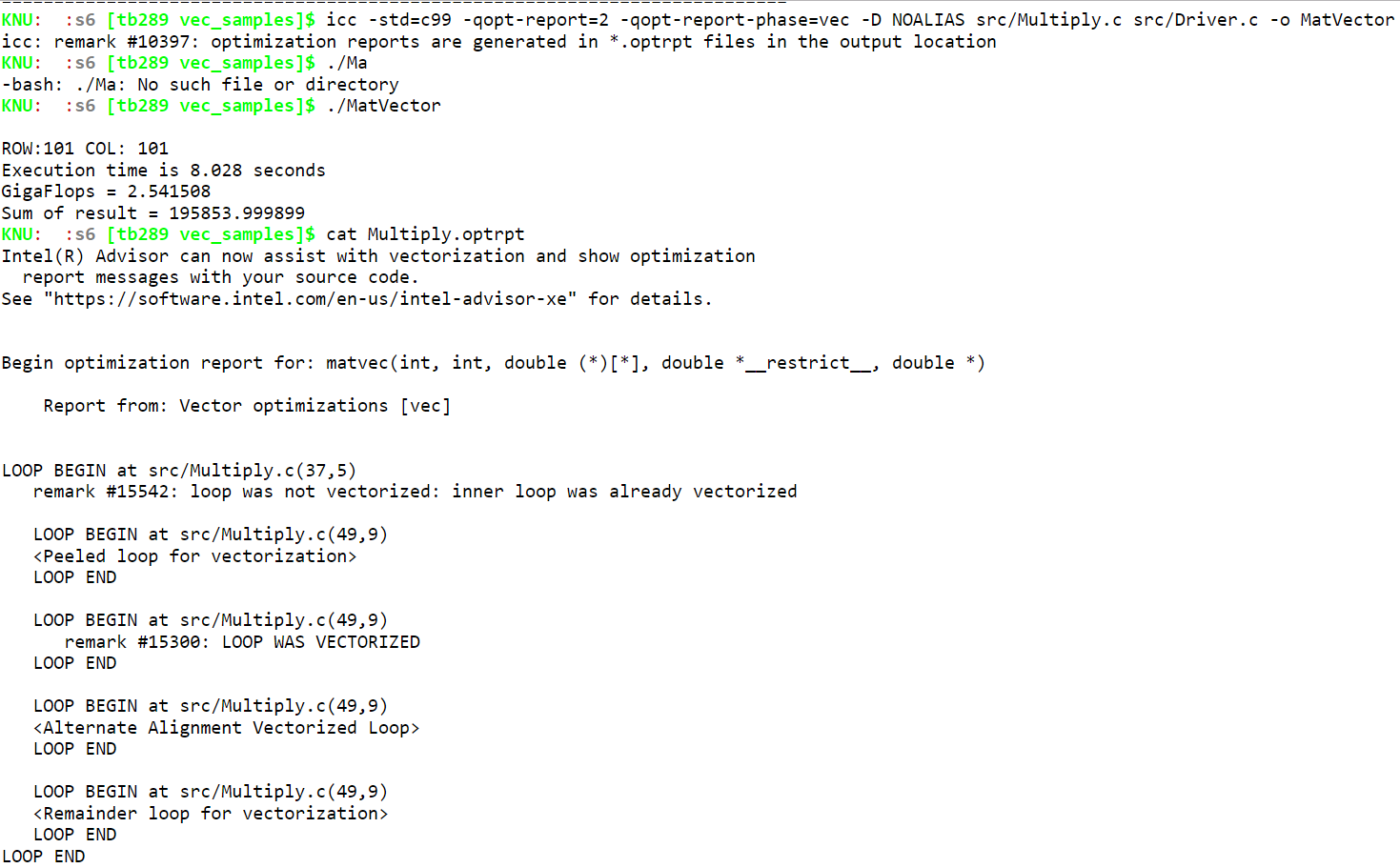


Помітно різницю у швидкості виконання за рахунок оптимізації (-О2)!

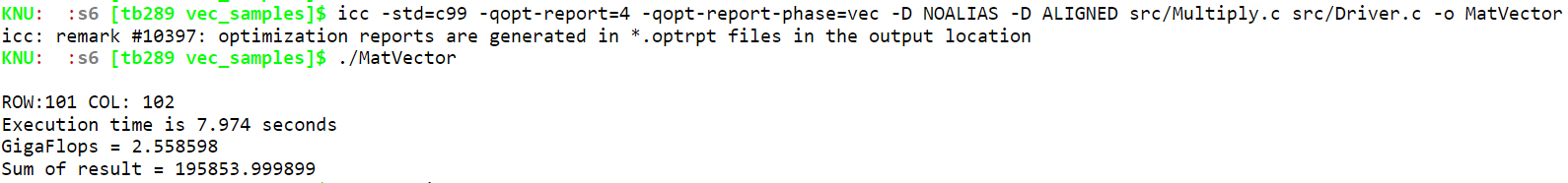
* Інші опції компіляції:

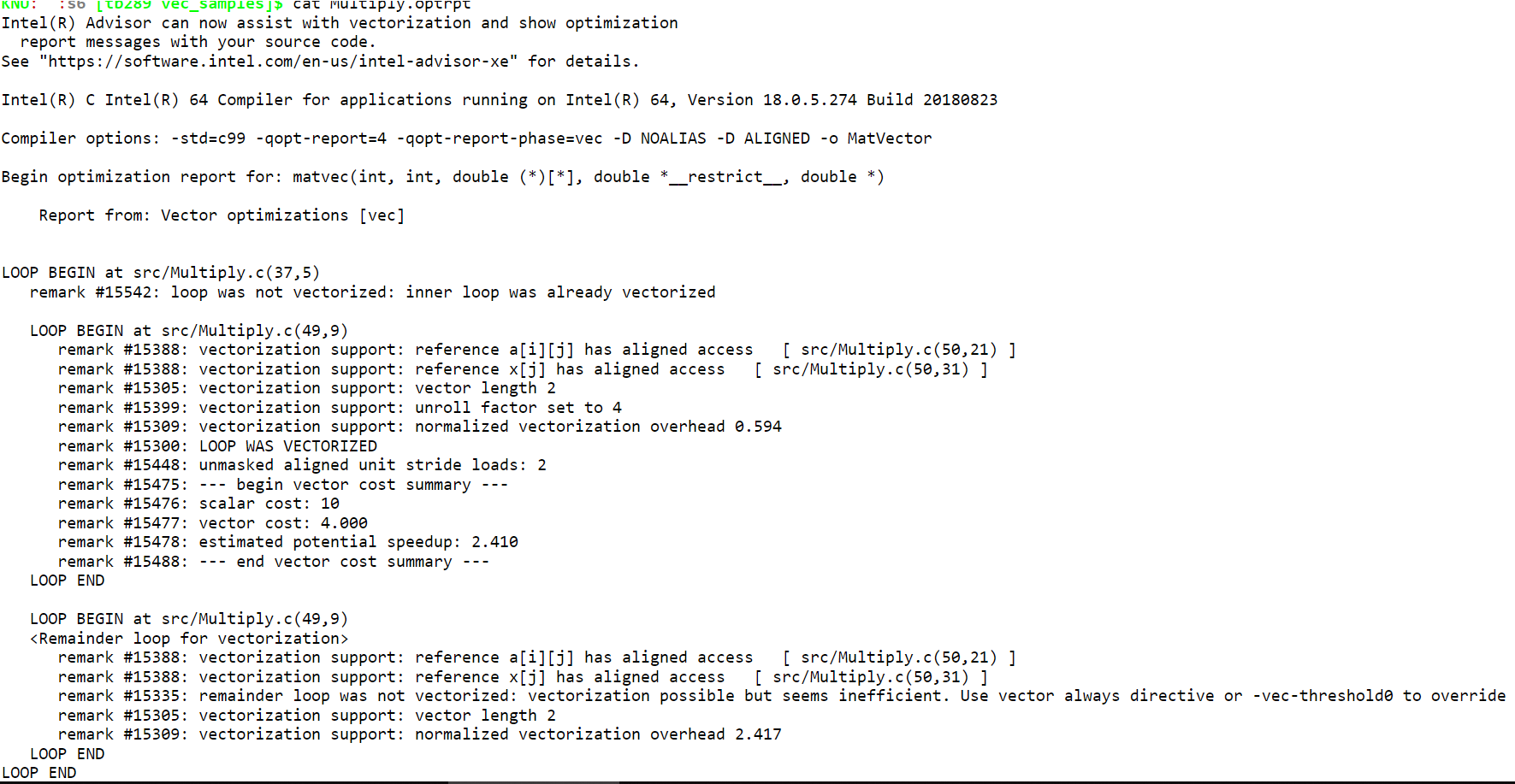


* Improving Performance by Pointer Disambiguation

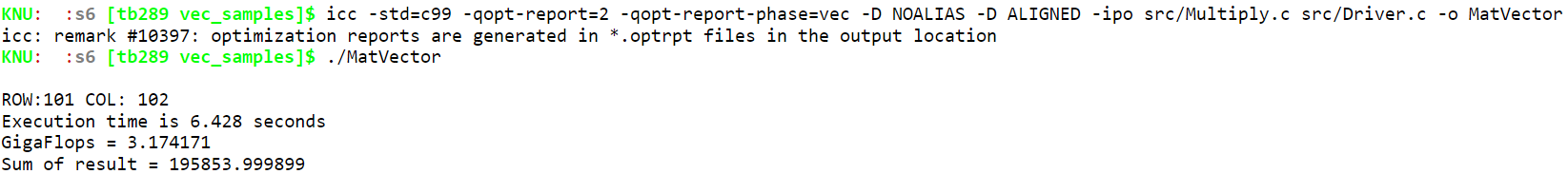


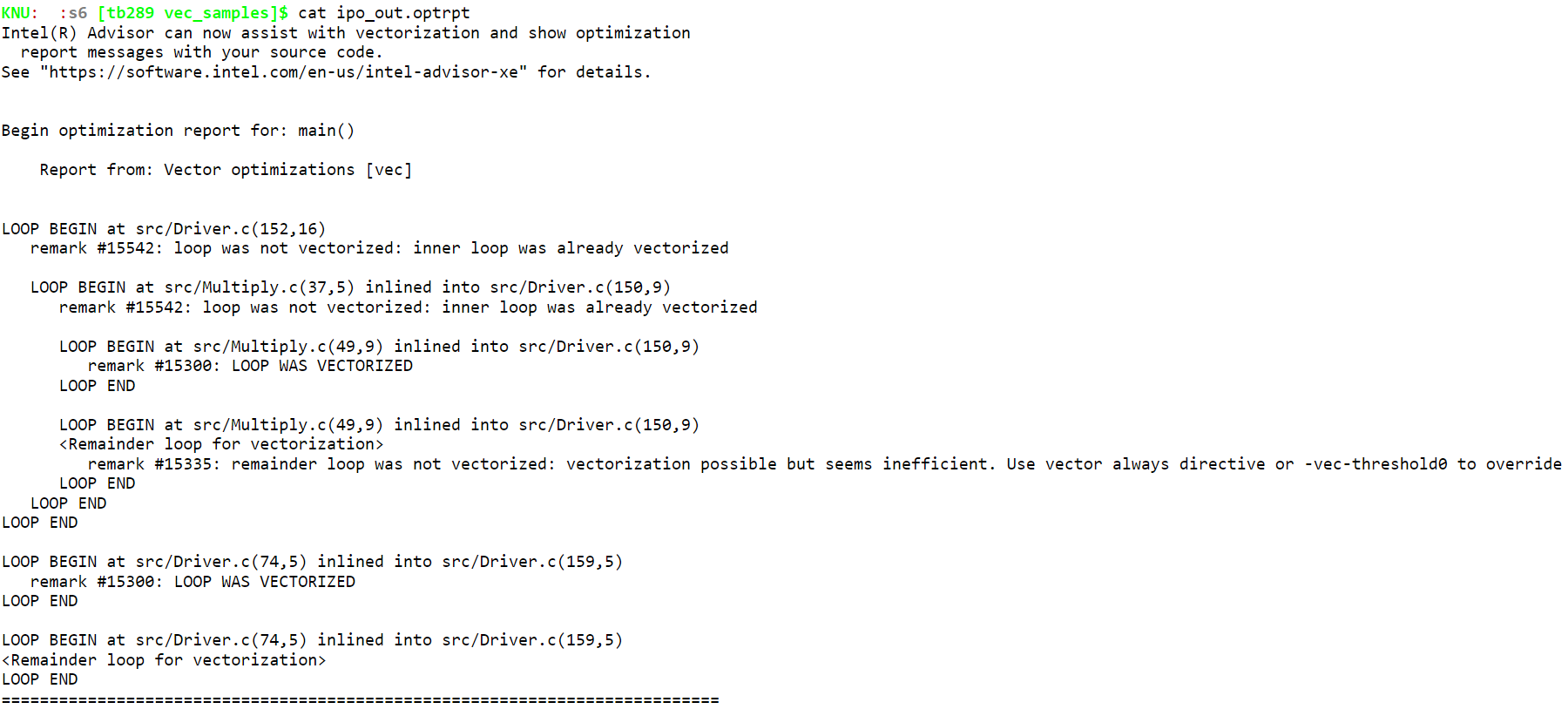
* Improving Performance by Aligning Data

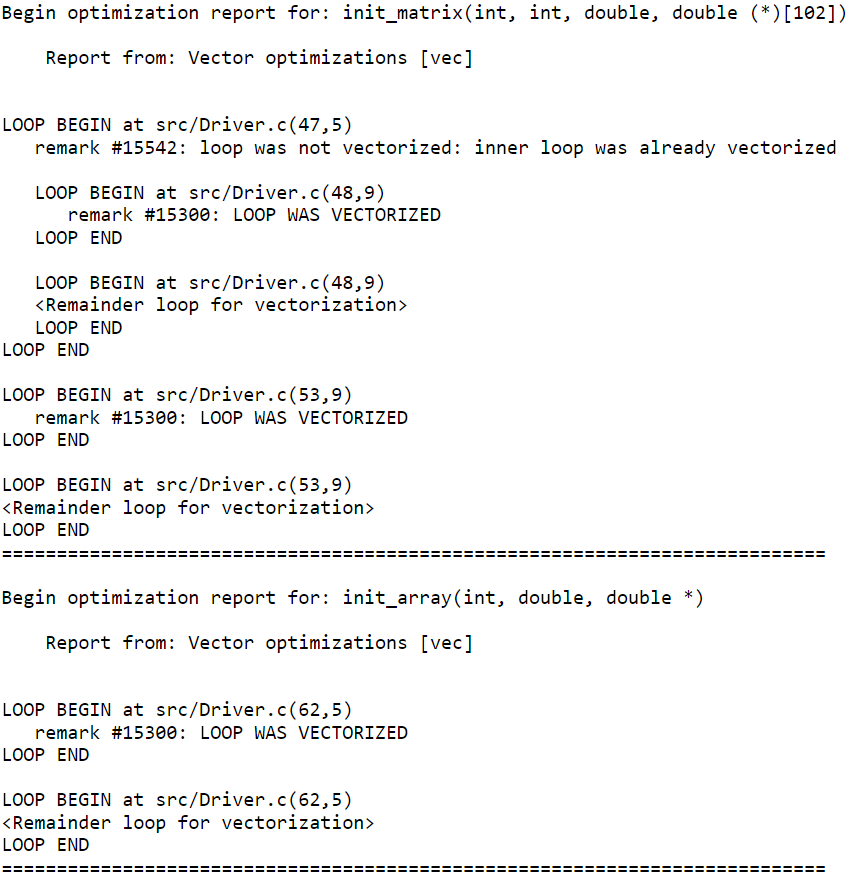




* Improving Performance with Interprocedural Optimization



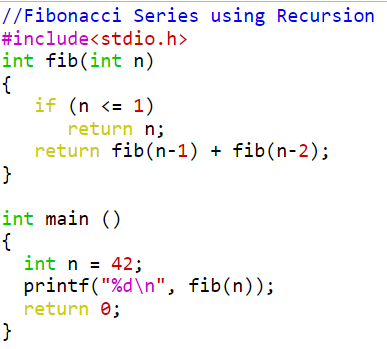




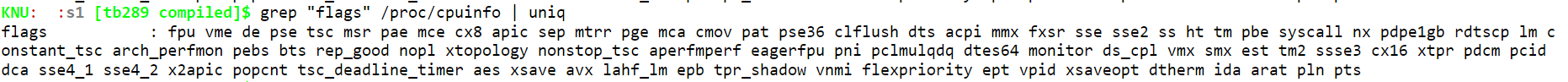
1. Оберіть будь-яку неінтрерактивну консольну програму мовою С/C++ (унікальну в межах групи, в гуглі більше ніж 50 програм).

Я обрав обрахунок Послідовності Фібоначчі, використовуючи рекурсію.

Якщо Ви запитаєте, для скількох чисел обраховується послідовність, то відповідь така ж, як і на «питання про життя, Всесвіт і взагалі» - тобто **42**.

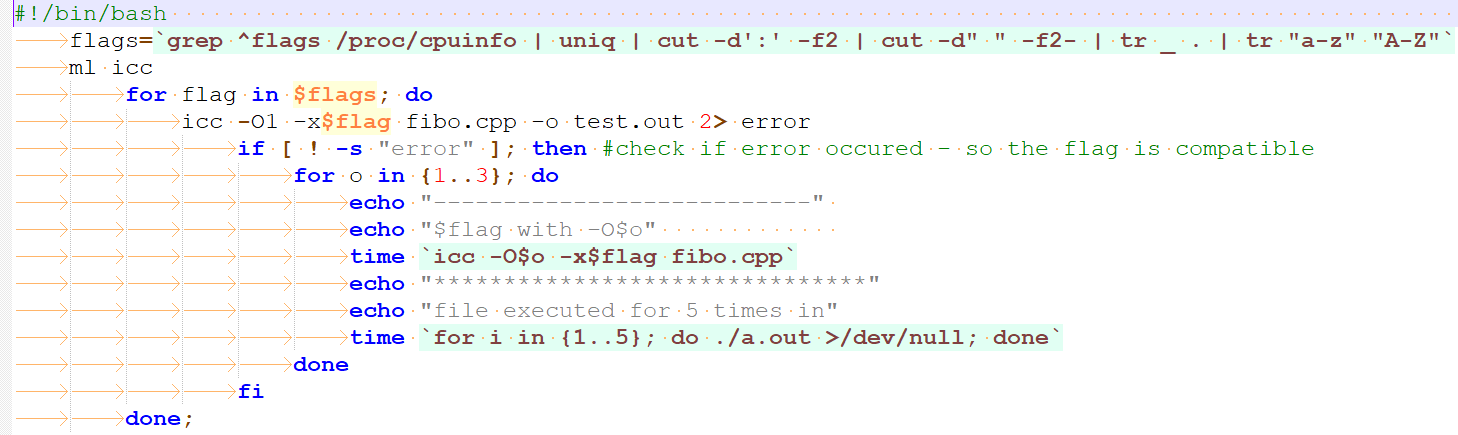


* 1. Напишіть сценарій, що:
  + Компілює програму з різними оптимізаціями (-O) та виміряйте час її роботи. Якщо час досить малий - вимірюйте час роботи 1000 (чи 1000000) запусків алгоритму в циклі. Час роботи можна виміряти утилітою time.
  + Отримує перелік всіх розширень процесору що підтримуються (тут просто усі)

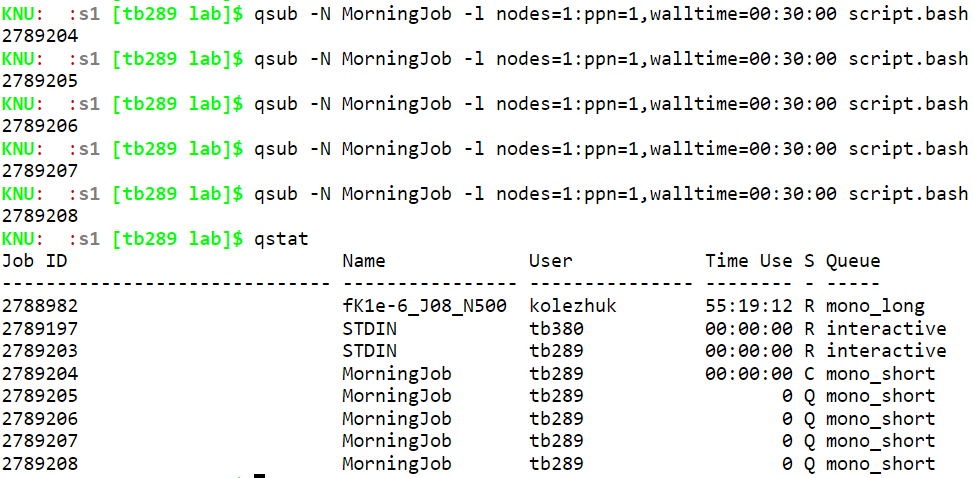


* + Для кожного розширення компілює Intel-компілятором окремий варіант оптимізованого коду (наприклад -x SSE2)
  + Вимірює час виконання кожного варіанта оптимізованої програми

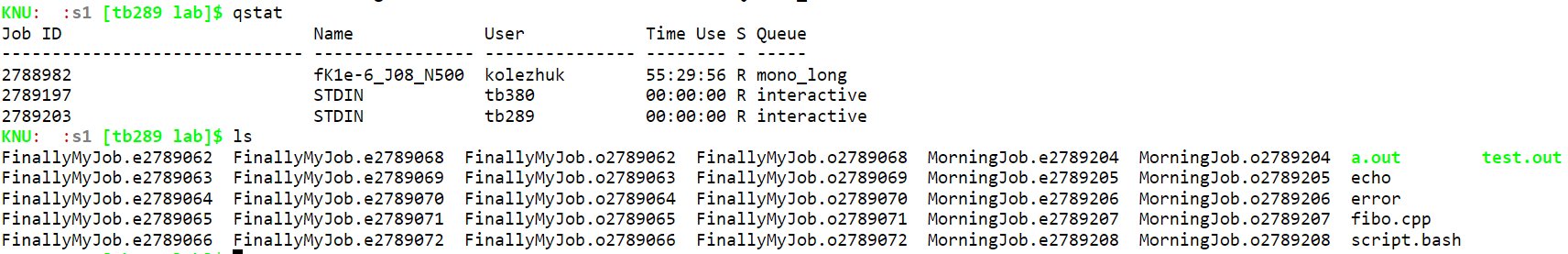
Мій скрипт знаходить всі наявні розширення процесора, але виводить інформацію лише про сумісні з компілятором (не з’являються помилки).



3.2 Запустіть задачу в планувальник обчислювального кластеру 5 разів (для статистики на різних нодах)



От список файлів, включно з файлами з попереднього «прогону» скрипта:



3.3 Побудуйте графіки залежності часу від різних варіантів компіляції.

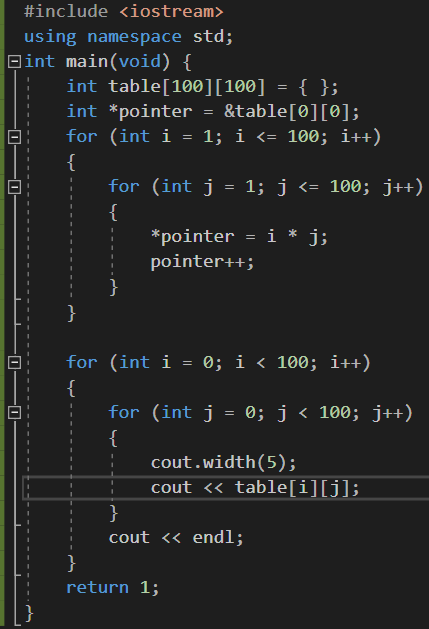
Згідно першого графіку найефективнішим виявився SSE2 (при рівні оптимізації 2, який являється рекомендованим) з помітним відривом.

Також нижче наведено графік залежностей від рівня оптимізації, де лідером виявився 2 рівень (рекомендований). Поряд знаходиться 1 рівень, який насправді є швидким, але виконує мало оптимізації.

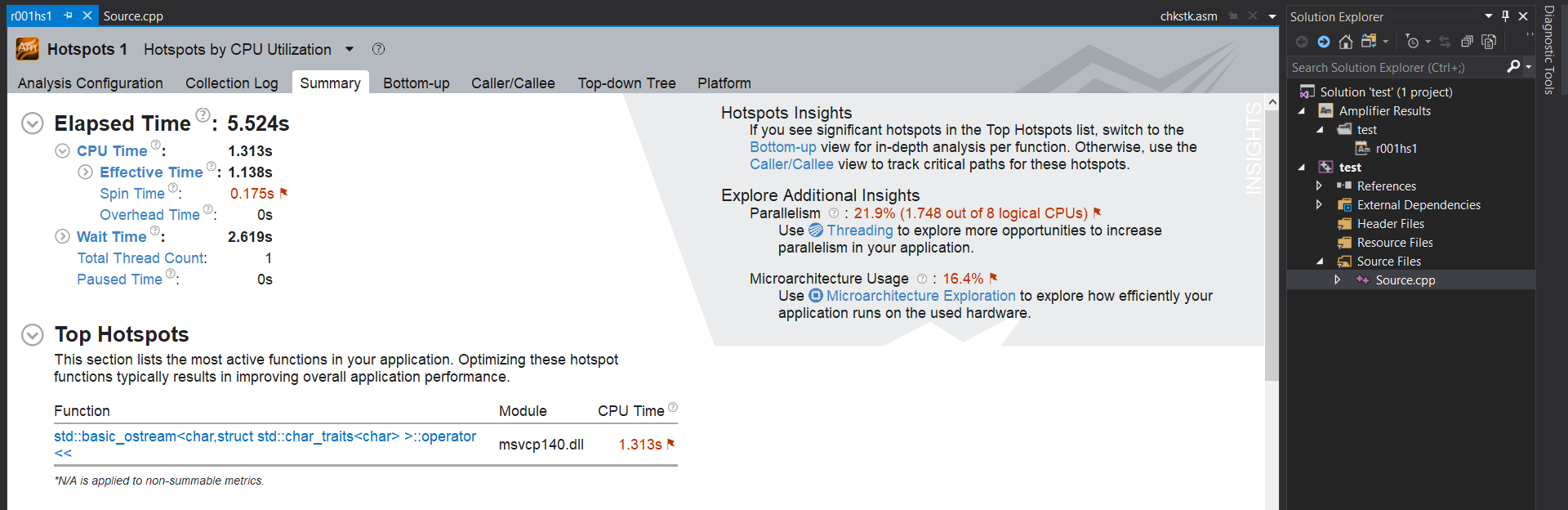
Наступні графіки демонструють залежності часу виконання програми (5 разів) від розширення та рівня оптимізації.

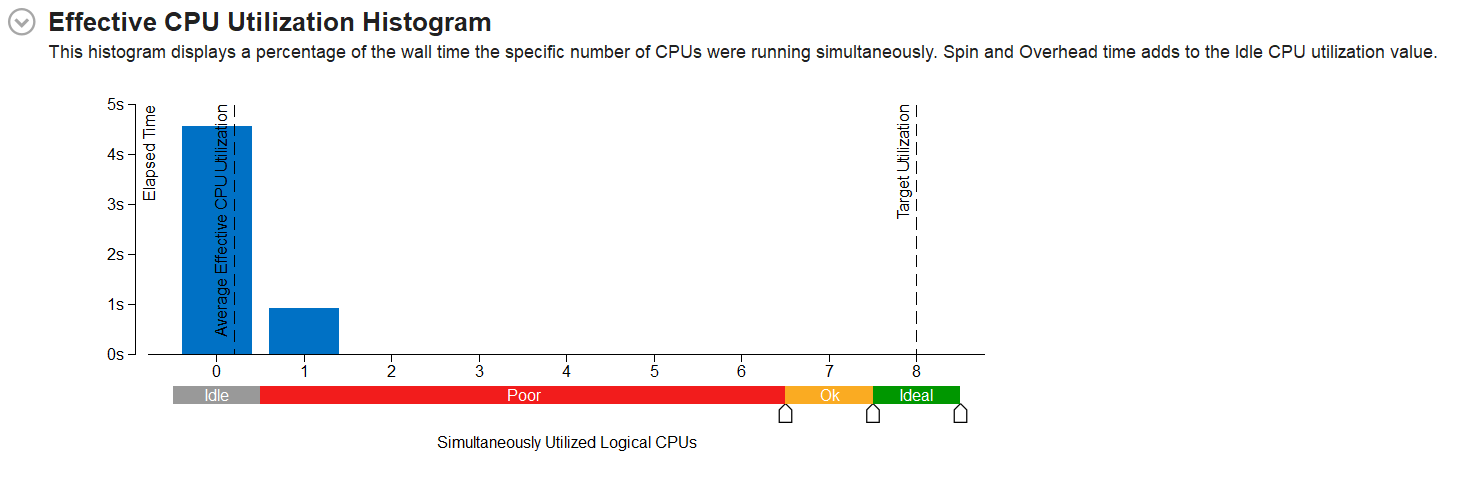
Згідно графіків, хоч код при розширенні SSE2 найшвидше компілювався, але виконується найповільніше.

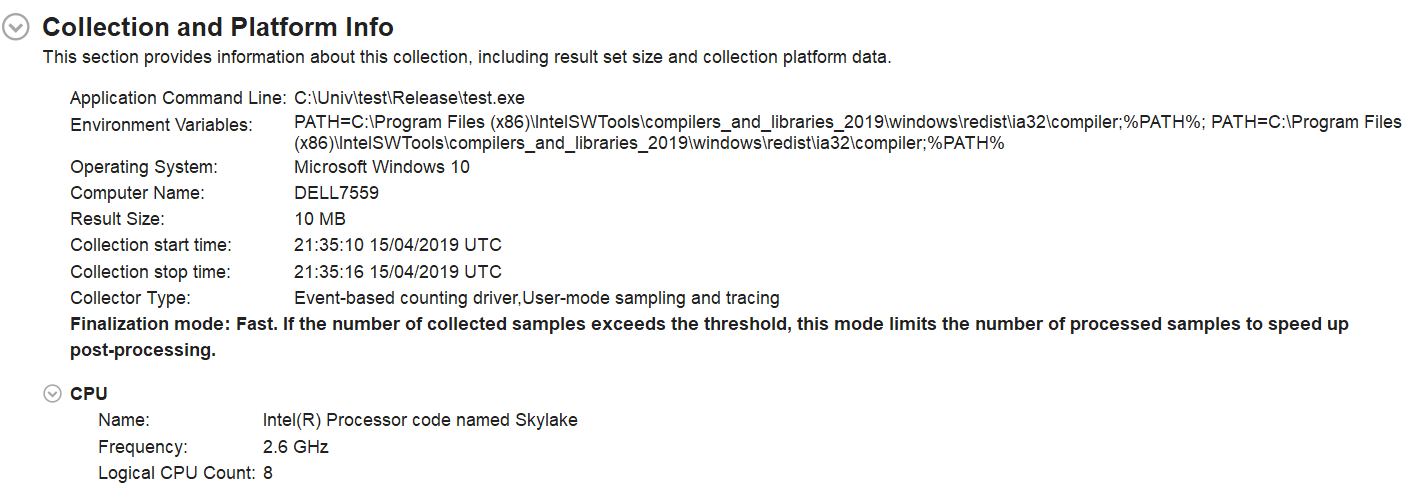
1. Оберіть будь-який зі створених вами програмних продуктів та виконайте його оптимізацію з використання Intel® Parallel Studio.

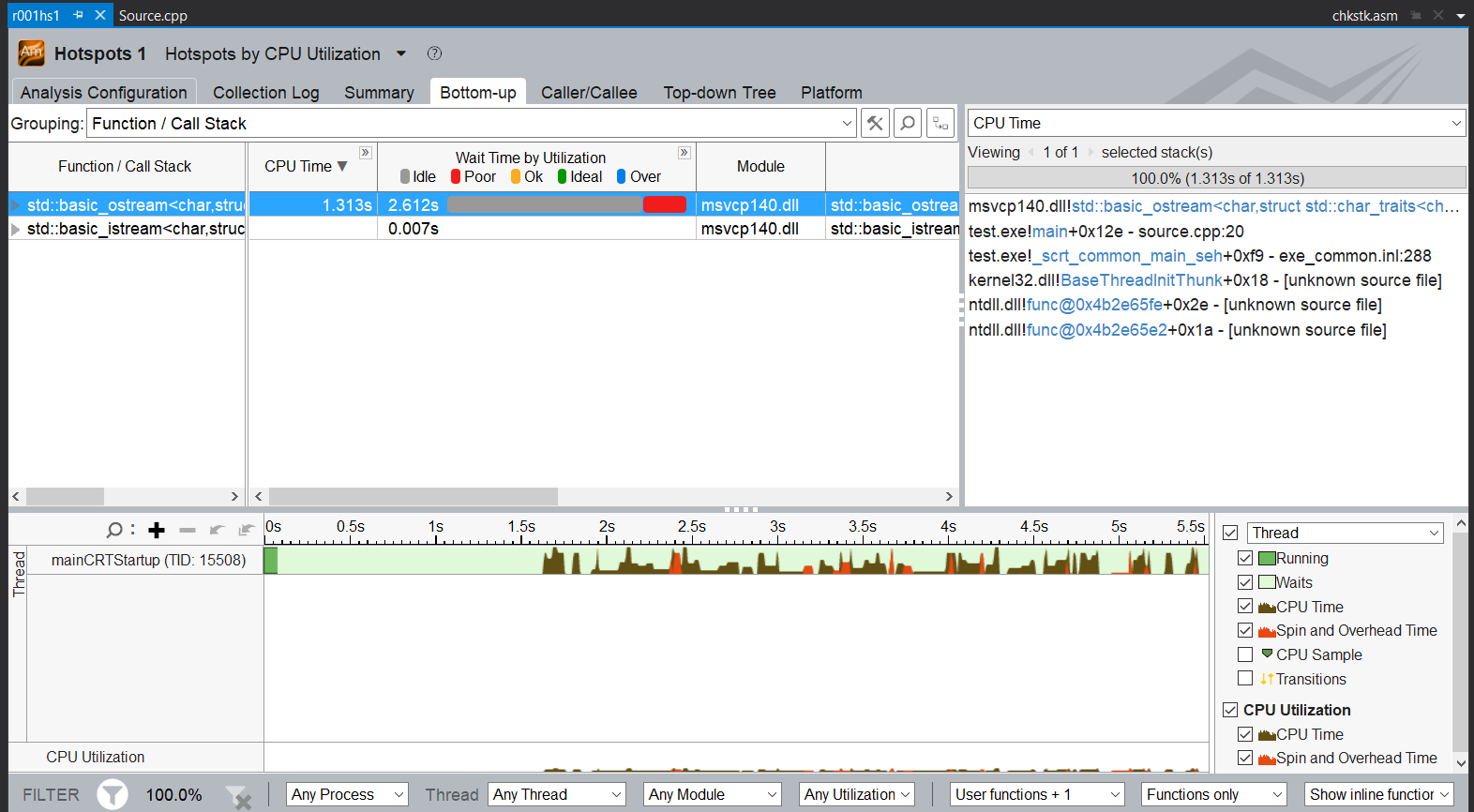
Для тестування я використав програму, яка заповнює та виводить матрицю 100 на 100 значеннями від 1 до 10000.

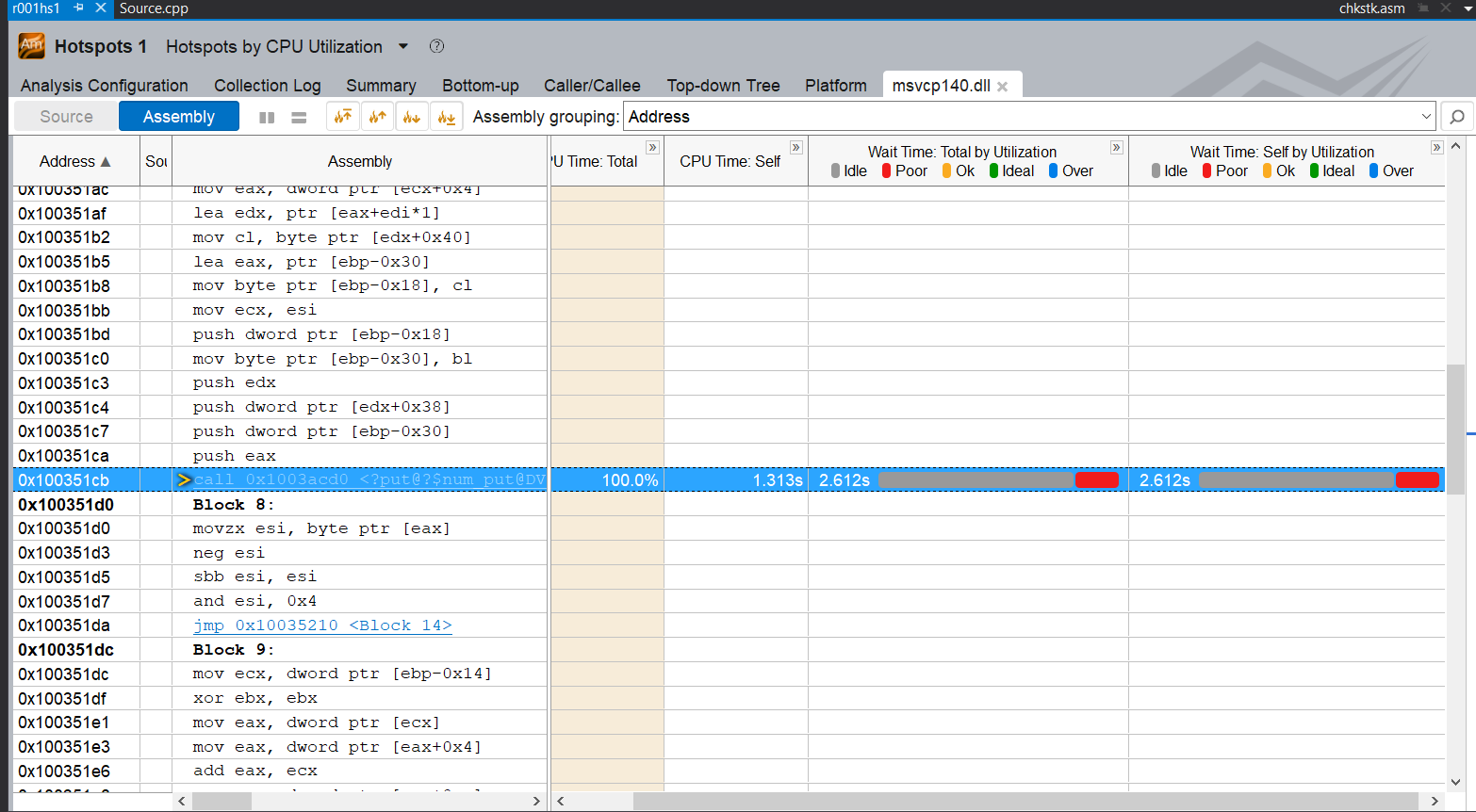
Нижче ось такі результати до оптимізації:



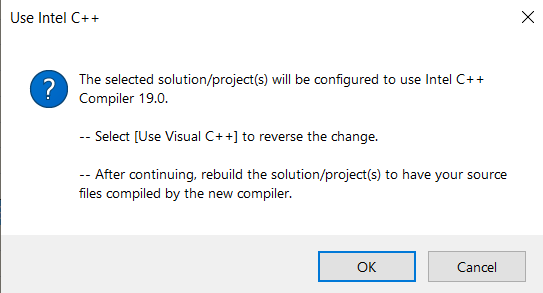


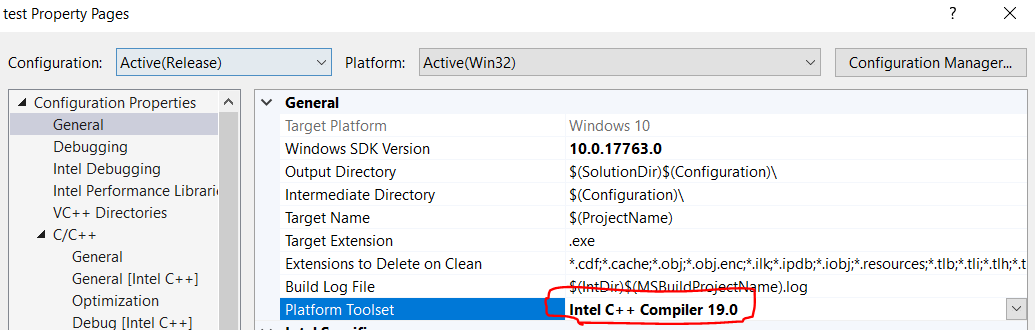


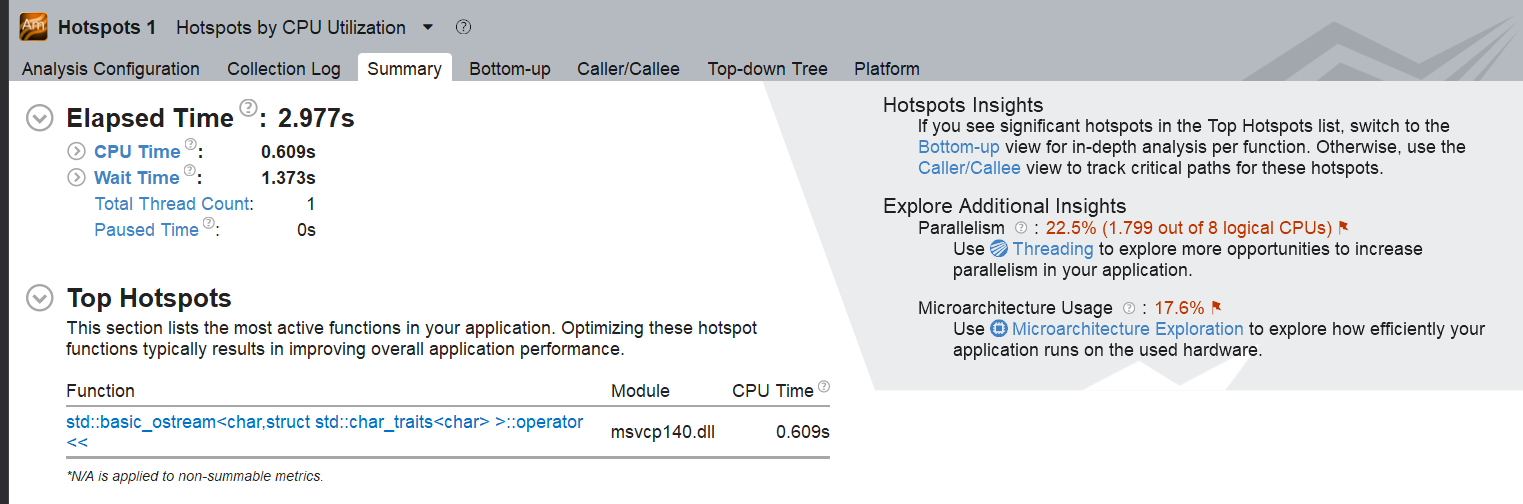
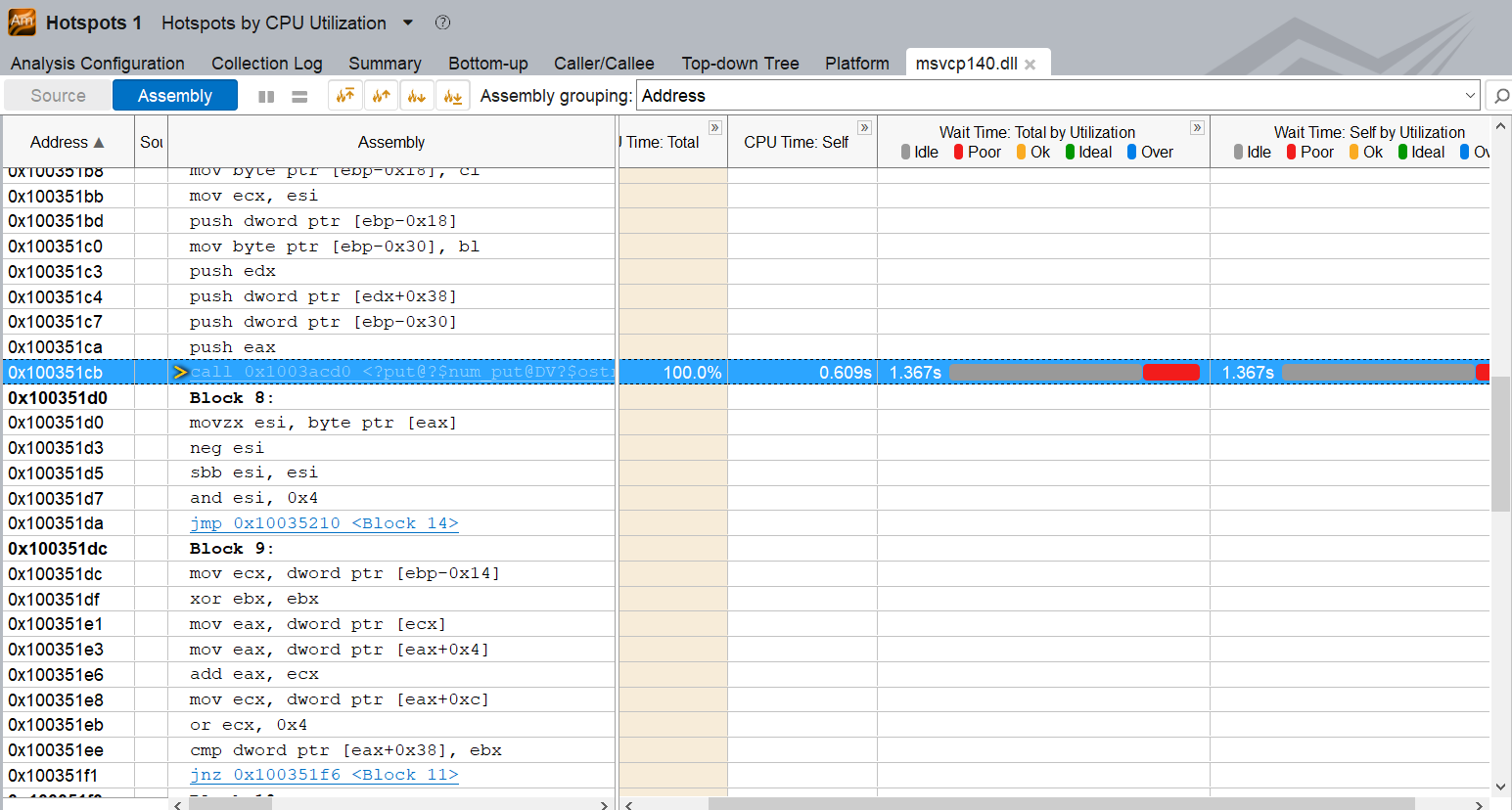




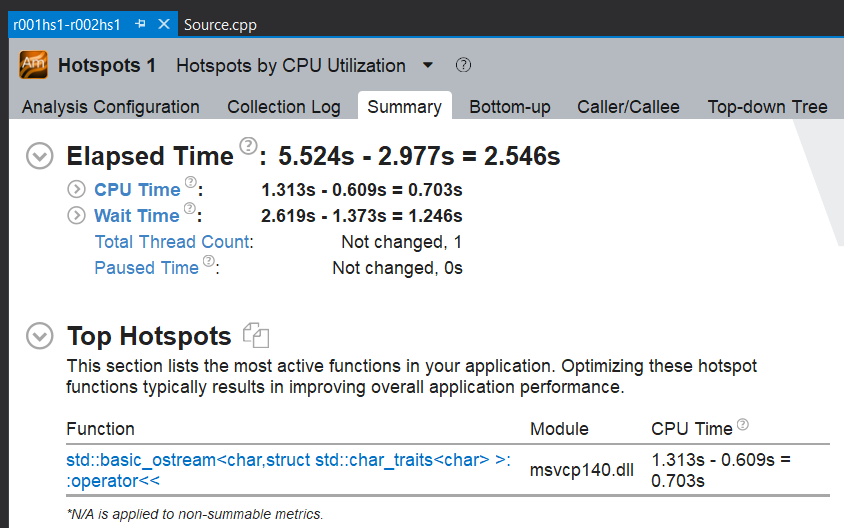
Застосовую новий компілятор:

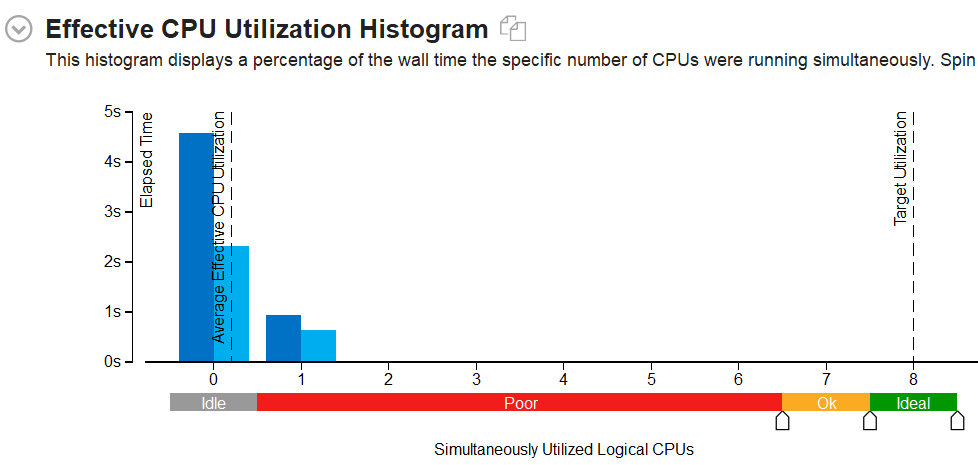




Порівнюю результати:





Отже, можна помітити досить суттєвий приріст продуктивності у **1.86 разів**.

**ВИСНОВОК**

У ході виконання лабораторної роботи було досліджено різні способи компіляції та оптимізації програми за допомогою різних векторних розширень CPU. Ознайомлено з основами роботи на обчислювальному кластері університету, роботою Intel C++ Compiler. Виявлено різницю у часі компіляції\виконання відповідно до розширення та рівня оптимізації.

Проведено оптимізацію продуктивності, використовуючи Intel C++ Compiler 19 версії у моїй програмі та упевнився в ефективності цього методу з суттєвим покращенням продуктивності.

Код програм та звіт містяться у репозиторії за [цим посиланням (натисніть мене).](https://github.com/Edocksil/Computer_Systems/tree/master/lab_3)