**Київський національний університет імені Тараса Шевченка**

**факультет радіофізики, електроніки та комп’ютерних систем**

Лабораторна робота № 3

**Тема:** «Дослідження оптимізації коду з використанням векторних розширень CPU»

Роботу виконав

студент 3 курсу

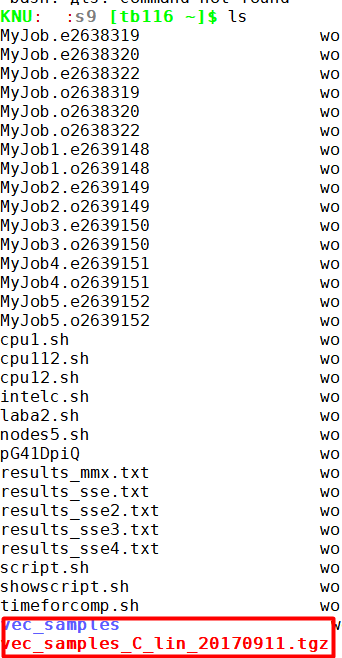
напряму Системний адміністратор

Апостолов Олександр Олексійович

Київ 2023

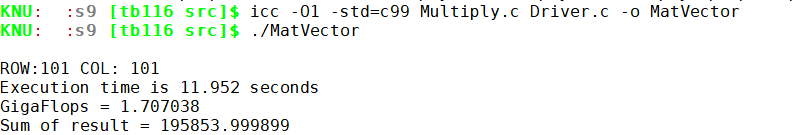
## Мета роботи: опанувати базові навички користування обчислювальним кластером, дослідити основні можливості оптимізації програмного коду з використанням векторних розширень CPU.

1. Отримав доступ на обчислювальний кластер для роботи з Intel Compiler
2. Завантажив файли Intel® C++ Compiler - Using Auto-Vectorization Tutorial

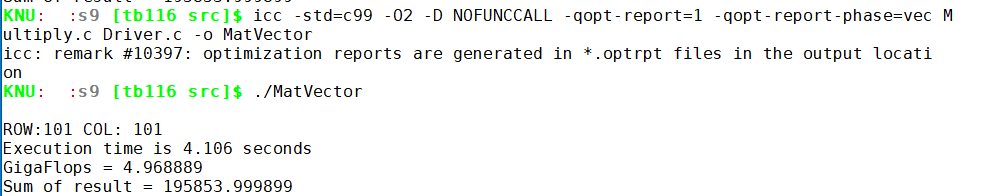


1. Використовуючи інструкції в readme.html ознайомився та виконав Tutorial на обчислювальному кластері

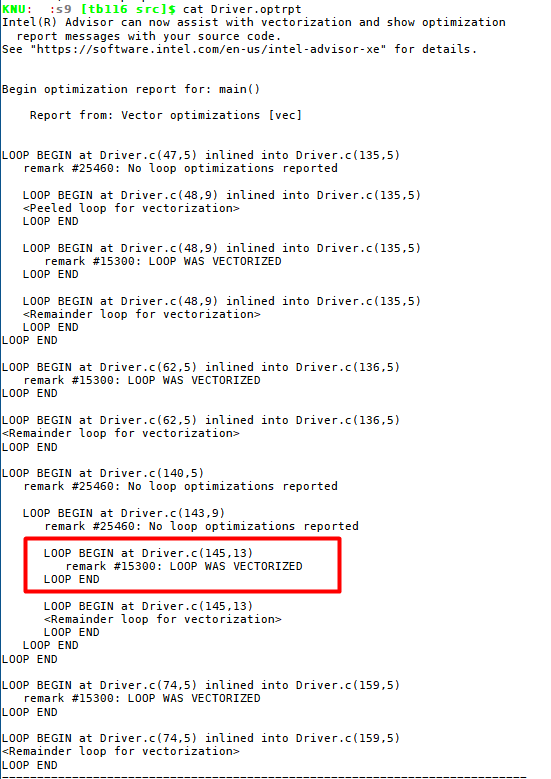
Пропишемо наступний код, для того, щоб розуміти базову "продуктивність" програми.



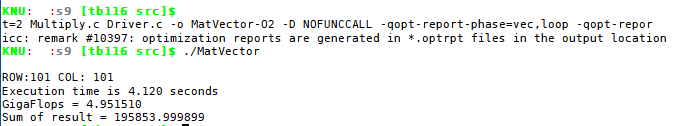
Ре компілював проект, оскільки векторизацію вимкнено в опції O1, компілятор не створює звіт про векторизацію. Щоб створити звіт векторизації, скомпілюємо свій проект із параметрами O2, qopt-report-phase=vec, qopt-report=1:



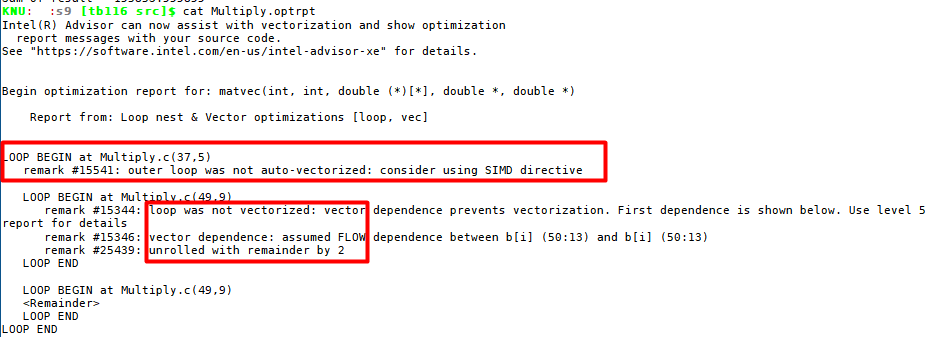
Скорочення часу здебільшого пов’язано з автоматичною векторизацією внутрішнього циклу в рядку 145



Ре компілював мій проект використовуючи qopt-report=2 and qopt-reportphase=vec,loop опції.

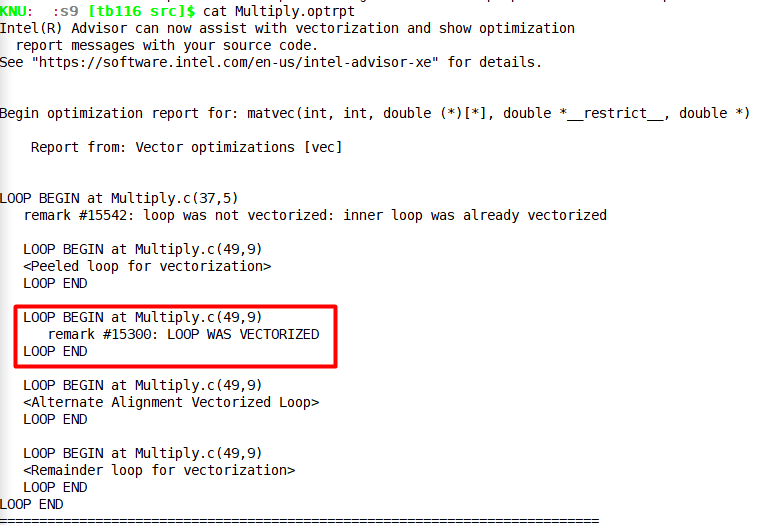


Отримуємо список, який також включає цикли, які не були векторизовані чи багатоверсійні, разом із причиною того, що компілятор не векторизував їх



Виконав покращення продуктивності шляхом усунення неоднозначності вказівника, вилучив -D NOFUNCCALL опцію, для відновлення виклику matvec(), а потім додав параметр -D NOALIAS до команди.

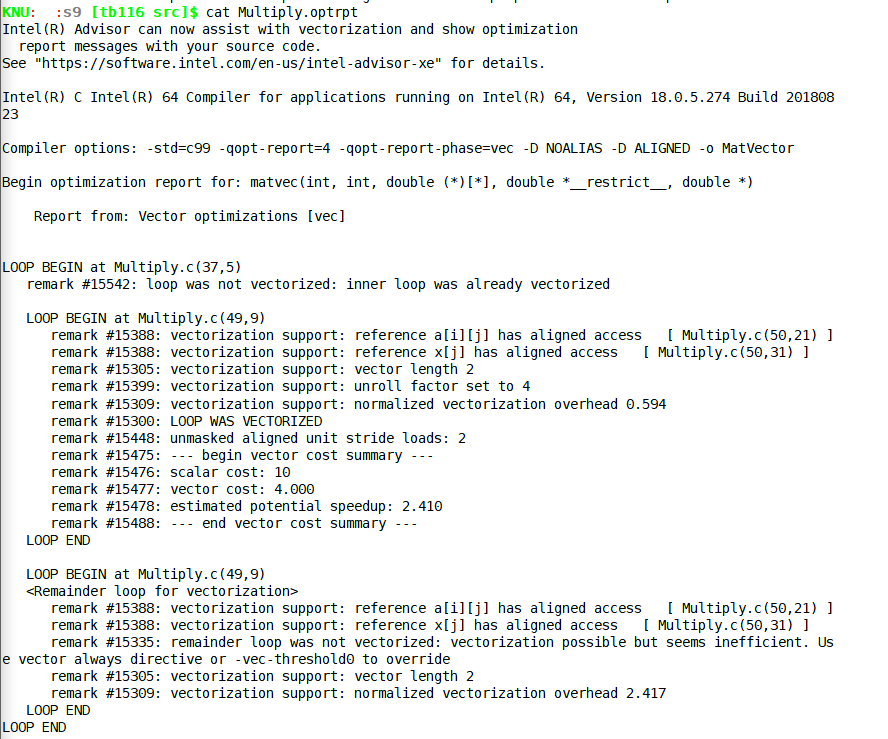


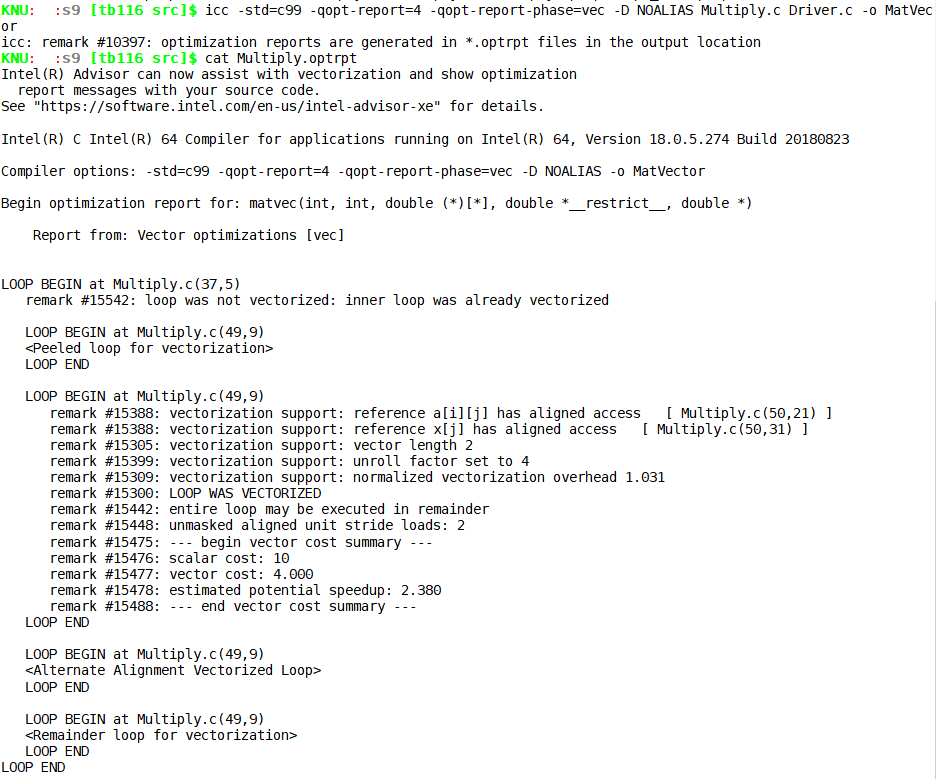


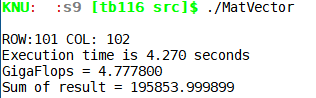
Далі виконаємо покращення продуктивності шляхом вирівнювання даних, перекомпілював програму після додавання макросу ALIGNED(щоб забезпечити стабільне вирівнювання даних). Використав -qopt-report=4, щоб побачити зміни у вирівняних посиланнях. (1 – NO -D ALIGNED option)

Використав наступний код:

icc -std=c99 -qopt-report=4 -qopt-report-phase=vec -D NOALIAS -D ALIGNED Multiply.c Driver.c -o MatVector

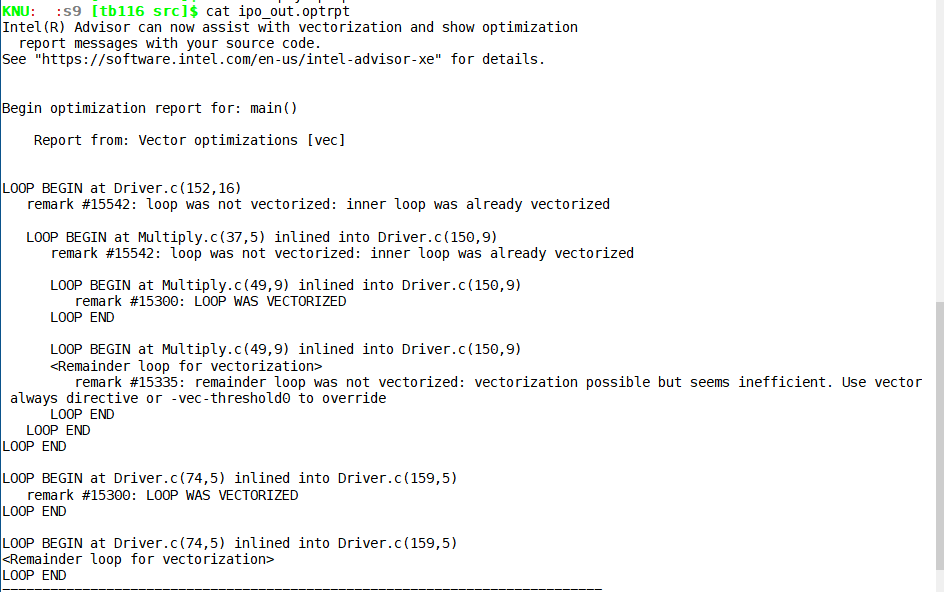


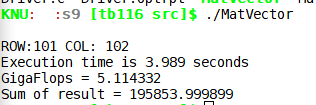




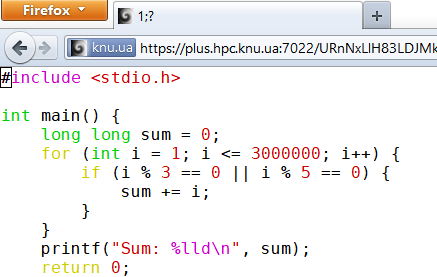
Рекомпілюємо програму за допомогою параметра -ipo(за допомогою нього можемо увімкнути міжпроцедурну оптимізацію)





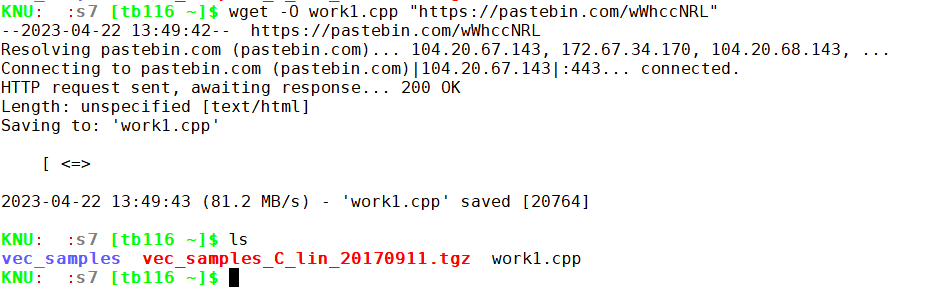


4. Оберіть будь-яку неінтрерактивну консольну програму мовою С/C++

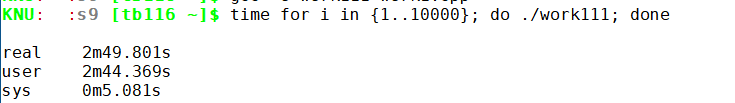


Створимо наступну програму, яка перебирає всі числа від 1 до 3,000,000 за допомогою циклу for та перевіряє, чи кожне число кратне 3 або 5. Якщо так, додає їх між собою.

Запишемо програму через редактор vim, командою vim work1.cpp.

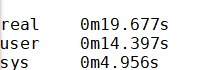


Запустимо програму без оптимізації 10000 разів та продемонструю час виконання.



Запустимо програму з оптимізацією O1 10000 разів та продемонструю час виконання.





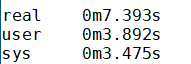
Запустимо програму з оптимізацією O2 10000 разів та продемонструю час виконання.





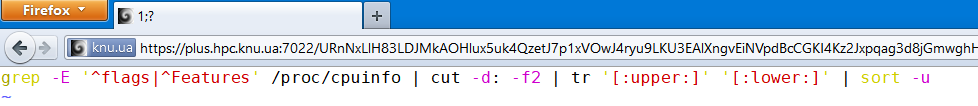
Запустимо програму з оптимізацією O3 10000 разів та продемонструю час виконання.



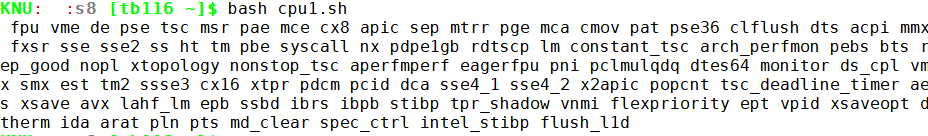


- сценарій, що отримує перелік всіх розширень процесору що підтримуються

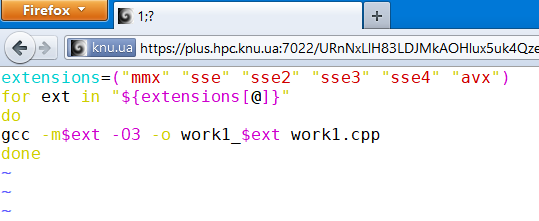
За допомогою утиліти grep сценарій проходиться по флагам, та виводить їх.



Отримуємо вивід з розширеннями процесору.

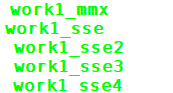


-створимо сценарій, що для кожного розширення компілює Intel-компілятором окремий варіант оптимізованого коду (наприклад -x SSE2)



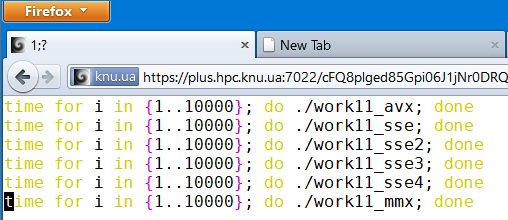


Для кожного розширення, що записані в масиві, скрипт викликає компілятор gcc з параметрами -m та назвою розширення та оптимізацією -O3 та проводить компіляцію. Після запуску коду отримуємо такі файли:



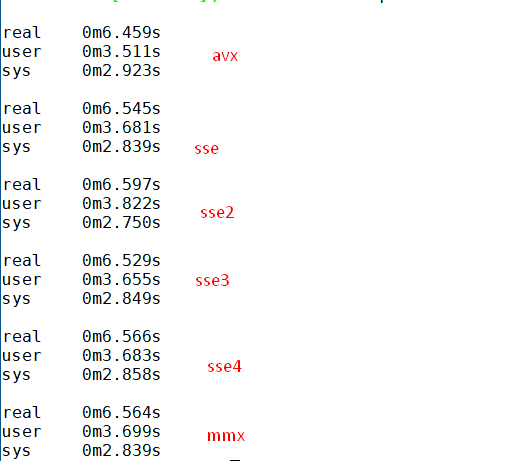


-Напишемо сценарій, що вимірює час виконання кожного варіанта оптимізованої програми.

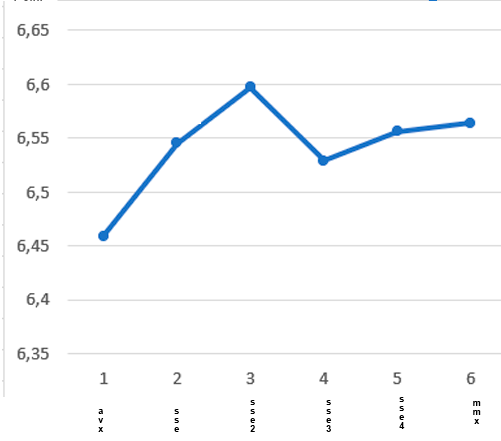


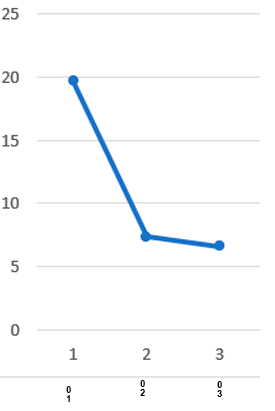


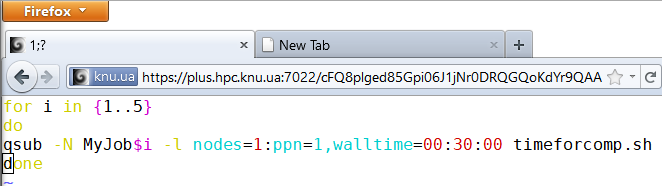
Було запущено программу 10000 разів, щоб наглядніше було видно час.

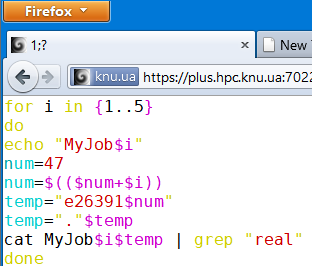


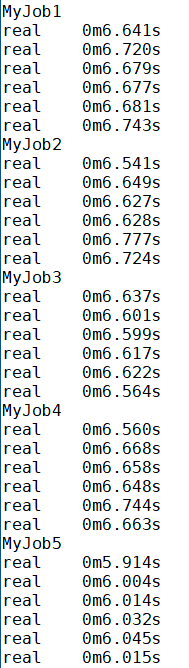
Побудуємо графіки залежності.

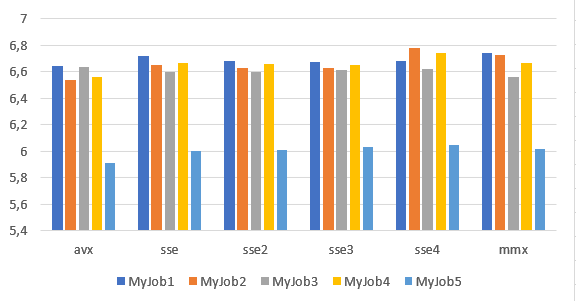












**Висновок: Виконуючи лабораторну роботу, опанував базовими навичками користування обчислювальним кластером, дослідив основні можливості оптимізації програмного коду з використанням векторних розширень CPU. Дослідив та порівняв векторні розширення CPU - MMX, SSE, AVX. Порівняв їх між собою. За результатами виконаної роботи побудував графік залежності, з отриманого графіку зробив висновок про загальну ефективність методу векторизації і різних систем команд відповідно.**