**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

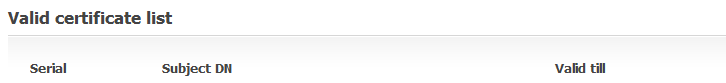
КІ-СА

Бондаренко Владислав

Київ 2020

**Хід роботи**

1. Отримайте доступ на обчислювальний кластер для роботи з Intel Compiler.





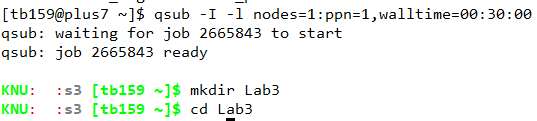
1. Завантажте файли Intel® C++ Compiler - Using Auto-Vectorization Tutorial (https://software.intel.com/en-us/product-code-samples?topic=20813) на свій комп’ютер та в домашню директорію користувача обчислювального кластеру.
2. Використовуючи інструкції в readme.html ознайомтесь та виконайте Tutorial на обчислювальному кластері

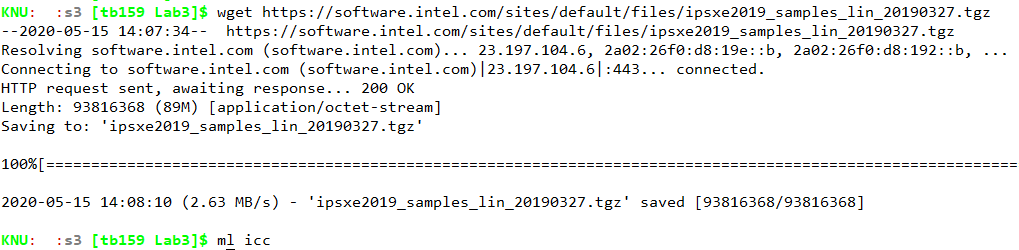
Замість інструкцій в пункті “Setting the Environment Variables” завантажте оточення компілятора шляхом виконання команди: ml icc

Виконуйте завдання на робочих вузлах кластеру замість вхідної ноди. По-перше процесори робочих вузлів мають набагато більше розширень. По-друге виконання компіляції та запуску на вхідній ноді заважає іншим користувачам, що призведе до блокування вашого аккаунту та автоматичного незарахування лабораторної роботи. Рекомендований варіант виконання роботи - використання інтерактивних задач в системі планування:

[manf@plus7 ~]$ qsub -I -l nodes=1:ppn=1,walltime=00:30:00

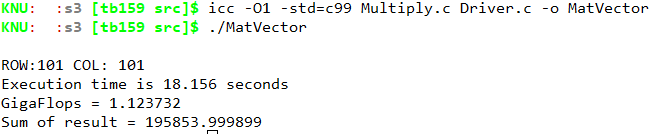
KNU:WN:s5 [manf ~]$ ml icc







To set a performance baseline for the improvements that follow in this tutorial, compile your sources from the src directory with these compiler options. Execute MatVector and record the execution time reported in the output. This is the baseline against which subsequent improvements will be measured.



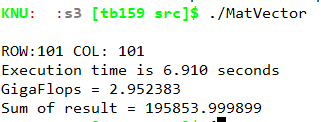
A vectorization report shows what loops in your code were vectorized and explains why other loops were not vectorized. To generate a vectorization report, use the qopt-report-phase=vec compiler options together with qopt-report=1 or qopt-report=2.

Together with qopt-report-phase=vec, qopt-report=1 generates a report with the loops in your code that were vectorized while qopt-report-phase=vec with qopt-report=2 generates a report with both the loops in your code that were vectorized and the reason that other loops were not vectorized.

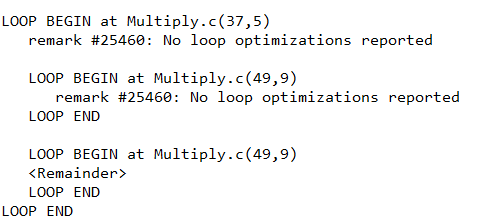
Because vectorization is turned off with the O1 option, the compiler does not generate a vectorization report. To generate a vectorization report, compile your project with the O2, qopt-report-phase=vec, qopt-report=1 options:



Recompile the program and then execute MatVector. Record the new execution time. The reduction in time is mostly due to auto-vectorization of the inner loop at line 145 noted in the vectorization report matvec.optrpt:



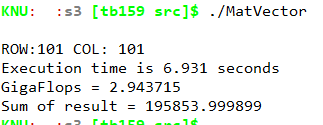
Multiply.optrpt:



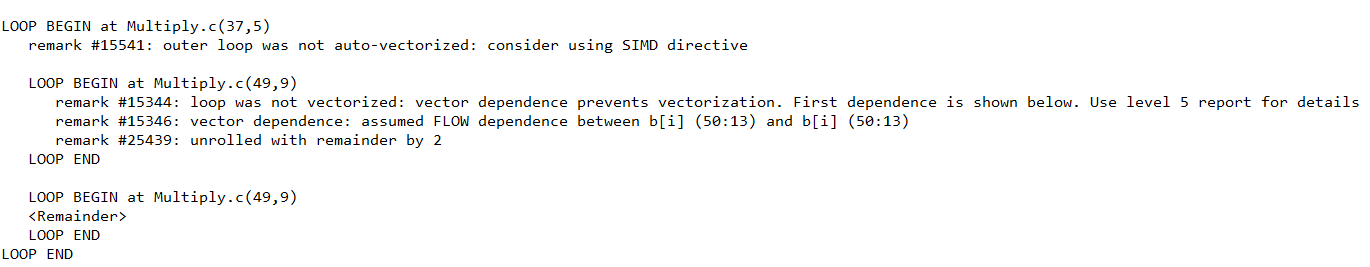
qopt-report=2 with qopt-report-phase=vec,loop returns a list that also includes loops that were not vectorized or multi-versioned, along with the reason that the compiler did not vectorize them or multi-version the loop.

Recompile your project with the qopt-report=2 and qopt-report-phase=vec,loop options.



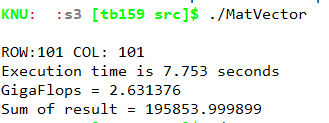


The vectorization report Multiply.optrpt indicates that the loop at line 37 in Multiply.c did not vectorize because it is not the innermost loop of the loop nest. Two versions of the innermost loop at line 49 were generated, and one version was vectorized.

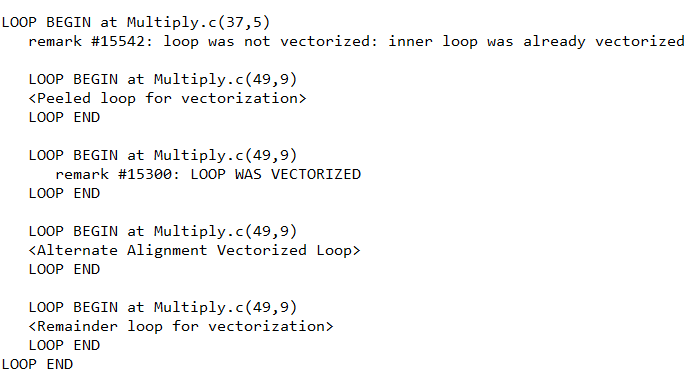


Remove the -D NOFUNCCALL to restore the call to matvec(), then add the -D NOALIAS option to the command line.



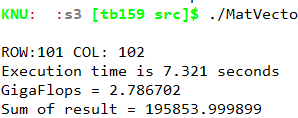


This conditional compilation replaces the loop in the main program with a function call. Execute MatVector and record the execution time reported in the output. Multiply.optrpt now shows:

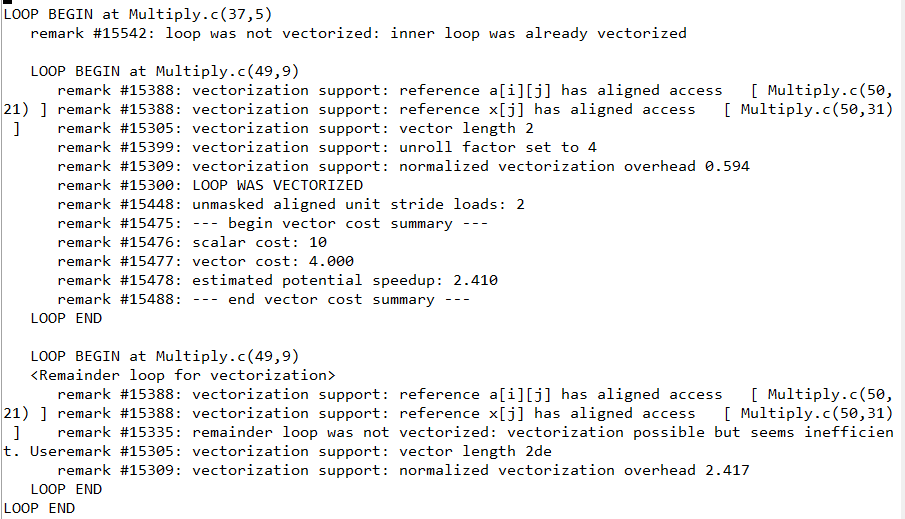


The vectorizer can generate faster code when operating on aligned data. In this activity you will improve performance by aligning the arrays a, b, and x in Driver.c on a 16-byte boundary so that the vectorizer can use aligned load instructions for all arrays rather than the slower unaligned load instructions and can avoid runtime tests of alignment. Using the ALIGNED macro will modify the declarations of a, b, and x in Driver.c using the aligned attribute keyword

Recompile the program after adding the ALIGNED macro to ensure consistently aligned data. Use -qopt-report=4 to see the change in aligned references.

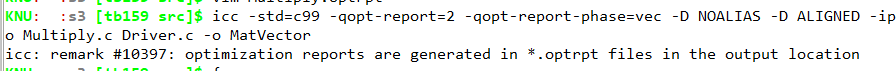


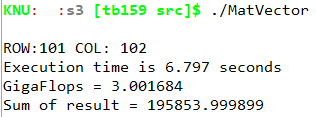
Multiply.optrpt after adding -D ALIGNED shows:



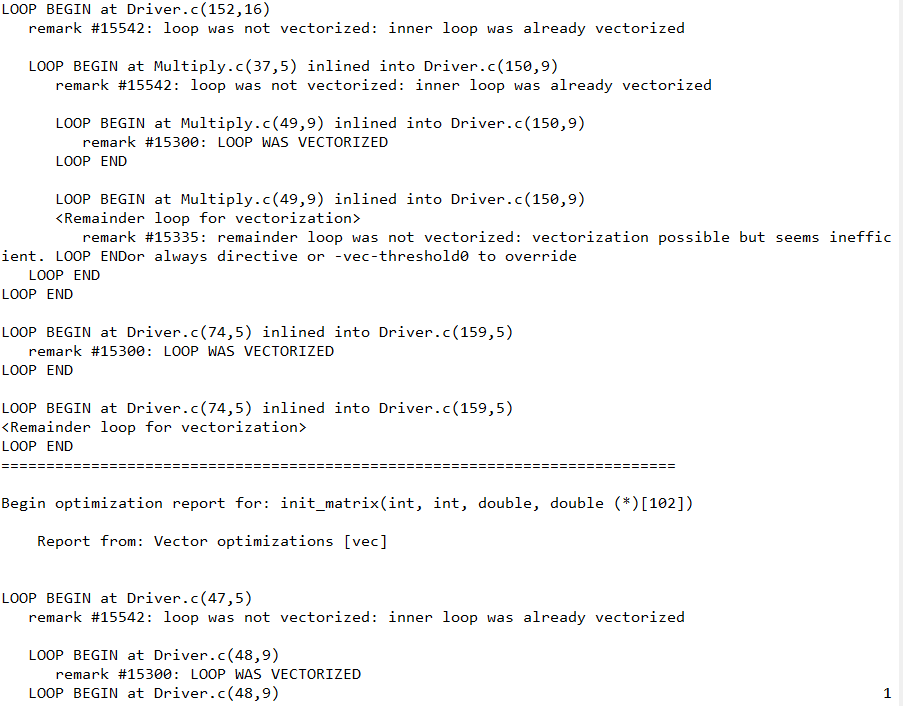
The compiler may be able to perform additional optimizations if it is able to optimize across source line boundaries. These may include, but are not limited to, function inlining. This is enabled with the -ipo option.

Recompile the program using the -ipo option to enable interprocedural optimization.





Note that the vectorization messages now appear at the point of inlining in Driver.c (line 150) and this is found in the file ipo\_out.optrpt.



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

Була написана програма мовою C++, в якій послідовно заповнюються 2 двовимірні масиви. За алгоритмами, у яких заповнення кожного наступного елементу масиву залежить від значення індексів вимірів, та деяких дробових чисел. Здається, що заповнення масивів можна легко розпаралелити, перевіримо це практично.

<https://github.com/Lrazerz/CompSystemsLabs/blob/master/Lab3/autoVectorization.cpp>

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

Для кожного розширення компілює Intel-компілятором окремий варіант оптимізованого коду (наприклад -x SSE2)

Вимірює час виконання кожного варіанта оптимізованої програми

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

[manf@plus7 ~]$ qsub -N MyJob -l nodes=1:ppn=1,walltime=00:30:00 script.sh

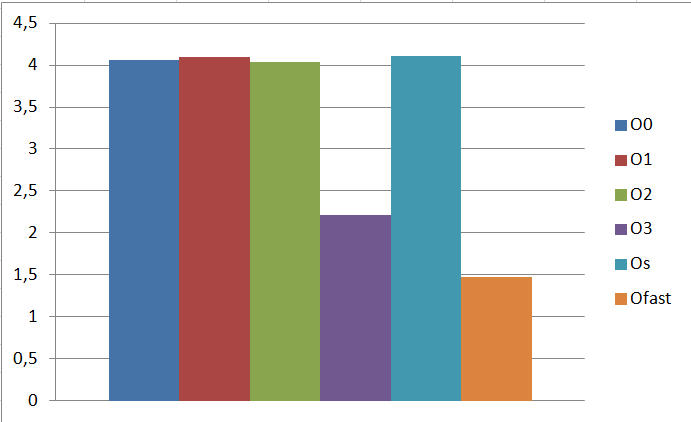
Побудуйте графіки залежності часу від різних варіантів компіляції.

Був написаний скрипт, який замірює час виконання циклу з 100 запусків виконання оптимізованої програми.

<https://github.com/Lrazerz/CompSystemsLabs/blob/master/Lab3/script.sh>

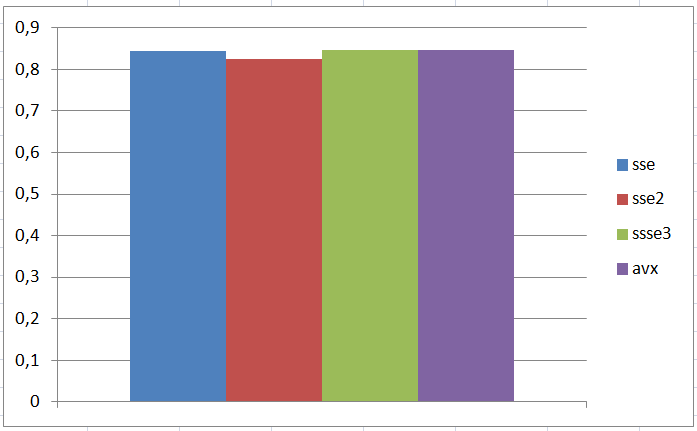
Час виконання програми скомпільованої з різними прапорцями.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер запуску \ прапорець | O0 | O1 | O2 | O3 | Os | Ofast |
| 1 | 4.060 | 4.100 | 4.037 | 2.230 | 4.110 | 1.489 |
| 2 | 4.057 | 4.106 | 4.051 | 2.185 | 4.076 | 1.441 |
| 3 | 4.081 | 4.117 | 4.081 | 2.201 | 4.088 | 1.455 |
| 4 | 4.075 | 4.103 | 4.068 | 2.200 | 4.064 | 1.494 |
| 5 | 4.125 | 4.146 | 4.110 | 2.244 | 4.131 | 1.501 |
| average | 4.080 | 4.114 | 4.070 | 2.212 | 4.094 | 1.476 |



Результат з -Оfast прапорцем і різними розширеннями:

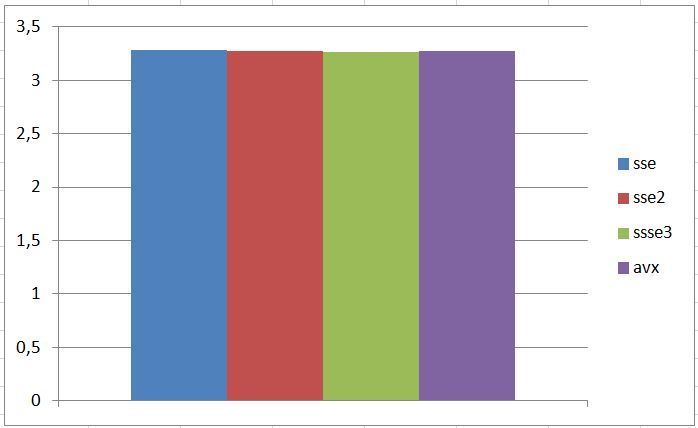
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер запуску \ розширення | sse | sse2 | ssse3 | avx |
| 1 | 0.852 | 0.852 | 0.852 | 0.853 |
| 2 | 0.827 | 0.723 | 0.837 | 0.838 |
| 3 | 0.822 | 0.836 | 0.834 | 0.834 |
| 4 | 0.855 | 0.857 | 0.853 | 0.856 |
| 5 | 0.857 | 0.856 | 0.859 | 0.856 |
| average | 0.843 | 0.825 | 0.847 | 0.847 |



В даному випадку за допомогою розширень вдалося зекономити ще ~43% часу виконання програми.

Result with -О1 flag and different extensions:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер запуску \ розширення | sse | sse2 | ssse3 | avx |
| 1 | 3.318 | 3.284 | 3.283 | 3.283 |
| 2 | 3.243 | 3.249 | 3.229 | 3.255 |
| 3 | 3.254 | 3.252 | 3.243 | 3.249 |
| 4 | 3.293 | 3.299 | 3.288 | 3.291 |
| 5 | 3.297 | 3.293 | 3.297 | 3.294 |
| average | 3.281 | 3.275 | 3.268 | 3.274 |



В даному випадку за допомогою розширень вдалося зекономити ще ~20% часу виконання програми.

**Висновок:** В даній лабораторній роботі було ознайомлено з автоматичною векторизацією в паралельних обчисленнях. Було проведено оптимізацію коду з використанням векторних розширень CPU компілятора Intel на прикладі програми, яка виконує заповнення масивів.