

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

```
KNU: :s9 [tb116 ~]$ ls
MyJob.e2638319      wo
MyJob.e2638320      wo
MyJob.e2638322      wo
MyJob.o2638319      wo
MyJob.o2638320      wo
MyJob.o2638322      wo
MyJob1.e2639148     wo
MyJob1.o2639148     wo
MyJob2.e2639149     wo
MyJob2.o2639149     wo
MyJob3.e2639150     wo
MyJob3.o2639150     wo
MyJob4.e2639151     wo
MyJob4.o2639151     wo
MyJob5.e2639152     wo
MyJob5.o2639152     wo
cpu1.sh            wo
cpu112.sh          wo
cpu12.sh           wo
intelc.sh           wo
laba2.sh            wo
nodes5.sh           wo
pG41DpiQ            wo
results_mmx.txt     wo
results_sse.txt     wo
results_sse2.txt    wo
results_sse3.txt    wo
results_sse4.txt    wo
script.sh           wo
showscript.sh       wo
timeforcomp.sh      wo
vec_samples
vec_samples_C_lin_20170911.tgz
```

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

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

```
KNU: :s9 [tb116 src]$ icc -O1 -std=c99 Multiply.c Driver.c -o MatVector
KNU: :s9 [tb116 src]$ ./MatVector
```

```
ROW:101 COL: 101
Execution time is 11.952 seconds
GigaFlops = 1.707038
Sum of result = 195853.999899
```

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:

```

KNU: :s9 [tb116 src]$ icc -std=c99 -O2 -D NOFUNCCALL -qopt-report=1 -qopt-report-phase=vec M
ultiply.c Driver.c -o MatVector
icc: remark #10397: optimization reports are generated in *.optrpt files in the output locati
on
KNU: :s9 [tb116 src]$ ./MatVector

ROW:101 COL: 101
Execution time is 4.106 seconds
GigaFlops = 4.968889
Sum of result = 195853.999899

```

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

```

KNU: :s9 [tb116 src]$ cat Driver.optrpt
Intel(R) Advisor can now assist with vectorization and show optimization
report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.

Begin optimization report for: main()

Report from: Vector optimizations [vec]

LOOP BEGIN at Driver.c(47,5) inlined into Driver.c(135,5)
remark #25460: No loop optimizations reported

LOOP BEGIN at Driver.c(48,9) inlined into Driver.c(135,5)
<Peeled loop for vectorization>
LOOP END

LOOP BEGIN at Driver.c(48,9) inlined into Driver.c(135,5)
remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at Driver.c(48,9) inlined into Driver.c(135,5)
<Remainder loop for vectorization>
LOOP END
LOOP END

LOOP BEGIN at Driver.c(62,5) inlined into Driver.c(136,5)
remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at Driver.c(62,5) inlined into Driver.c(136,5)
<Remainder loop for vectorization>
LOOP END

LOOP BEGIN at Driver.c(140,5)
remark #25460: No loop optimizations reported

LOOP BEGIN at Driver.c(143,9)
remark #25460: No loop optimizations reported

LOOP BEGIN at Driver.c(145,13)
remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at Driver.c(145,13)
<Remainder loop for vectorization>
LOOP END
LOOP END

LOOP BEGIN at Driver.c(74,5) inlined into Driver.c(159,5)
remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at Driver.c(74,5) inlined into Driver.c(159,5)
<Remainder loop for vectorization>
LOOP END

```

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

```
KNU: :s9 [tb116 src]$  
t=2 Multiply.c Driver.c -o MatVector-02 -D NOFUNCCALL -qopt-report-phase=vec,loop -qopt-repor  
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
```

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

```
KNU: :s9 [tb116 src]$ cat Multiply.optrpt  
Intel(R) Advisor can now assist with vectorization and show optimization  
report messages with your source code.  
See "https://software.intel.com/en-us/intel-advisor-xe" for details.  
  
Begin optimization report for: matvec(int, int, double (*)[*], double *, double *)  
  
Report from: Loop nest & Vector optimizations [loop, vec]  
  
LOOP BEGIN at Multiply.c(37,5)  
remark #15541: outer loop was not auto-vectorized: consider using SIMD directive  
  
LOOP BEGIN at Multiply.c(49,9)  
remark #15344: loop was not vectorized: vector dependence prevents vectorization. First dependence is shown below. Use level 5  
report for details  
remark #15346: vector dependence: assumed FLOW dependence between b[i] (50:13) and b[i] (50:13)  
remark #25439: unrolled with remainder by 2  
LOOP END  
  
LOOP BEGIN at Multiply.c(49,9)  
<Remainder>  
LOOP END  
LOOP END
```

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

```
icc -std=c99 -qopt-report=2 -qopt-report-phase=vec -D  
NOALIAS Multiply.c Driver.c -o MatVector
```

```
KNU: :s9 [tb116 src]$ cat Multiply.optrpt  
Intel(R) Advisor can now assist with vectorization and show optimization  
report messages with your source code.  
See "https://software.intel.com/en-us/intel-advisor-xe" for details.  
  
Begin optimization report for: matvec(int, int, double (*)[*], double *__restrict__, double *)  
  
Report from: Vector optimizations [vec]  
  
LOOP BEGIN at Multiply.c(37,5)  
remark #15542: loop was not vectorized: inner loop was already vectorized  
  
LOOP BEGIN at Multiply.c(49,9)  
<Peeled loop for vectorization>  
LOOP END  
  
LOOP BEGIN at Multiply.c(49,9)  
remark #15300: LOOP WAS VECTORIZED  
LOOP END  
  
LOOP BEGIN at Multiply.c(49,9)  
<Alternate Alignment Vectorized Loop>  
LOOP END  
  
LOOP BEGIN at Multiply.c(49,9)  
<Remainder loop for vectorization>  
LOOP END  
LOOP END  
=====
```

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
```

```
GNU: :s9 [tb116 src]$ cat Multiply.optrpt  
Intel(R) Advisor can now assist with vectorization and show optimization  
report messages with your source code.  
See "https://software.intel.com/en-us/intel-advisor-xe" for details.  
  
Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64, Version 18.0.5.274 Build 201808  
23  
  
Compiler options: -std=c99 -qopt-report=4 -qopt-report-phase=vec -D NOALIAS -D ALIGNED -o MatVector  
  
Begin optimization report for: matvec(int, int, double (*)[*], double *__restrict__, double *)  
  
Report from: Vector optimizations [vec]  
  
LOOP BEGIN at Multiply.c(37,5)  
remark #15542: loop was not vectorized: inner loop was already vectorized  
  
LOOP BEGIN at Multiply.c(49,9)  
remark #15388: vectorization support: reference a[i][j] has aligned access [ Multiply.c(50,21) ]  
remark #15388: vectorization support: reference x[j] has aligned access [ Multiply.c(50,31) ]  
remark #15305: vectorization support: vector length 2  
remark #15399: vectorization support: unroll factor set to 4  
remark #15309: vectorization support: normalized vectorization overhead 0.594  
remark #15300: LOOP WAS VECTORIZED  
remark #15448: unmasked aligned unit stride loads: 2  
remark #15475: --- begin vector cost summary ---  
remark #15476: scalar cost: 10  
remark #15477: vector cost: 4.000  
remark #15478: estimated potential speedup: 2.410  
remark #15488: --- end vector cost summary ---  
LOOP END  
  
LOOP BEGIN at Multiply.c(49,9)  
<Remainder loop for vectorization>  
remark #15388: vectorization support: reference a[i][j] has aligned access [ Multiply.c(50,21) ]  
remark #15388: vectorization support: reference x[j] has aligned access [ Multiply.c(50,31) ]  
remark #15335: remainder loop was not vectorized: vectorization possible but seems inefficient. Use  
vector always directive or -vec-threshold0 to override  
remark #15305: vectorization support: vector length 2  
remark #15309: vectorization support: normalized vectorization overhead 2.417  
LOOP END  
LOOP END
```

```

KNU: :s9 [tb116 src]$ icc -std=c99 -qopt-report=4 -qopt-report-phase=vec -D NOALIAS Multiply.c Driver.c -o MatVec
or
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
KNU: :s9 [tb116 src]$ cat Multiply.optrpt
Intel(R) Advisor can now assist with vectorization and show optimization
report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.

```

Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64, Version 18.0.5.274 Build 20180823

Compiler options: -std=c99 -qopt-report=4 -qopt-report-phase=vec -D NOALIAS -o MatVector

Begin optimization report for: matvec(int, int, double (*)(*), double *__restrict__, double *)

Report from: Vector optimizations [vec]

```

LOOP BEGIN at Multiply.c(37,5)
  remark #15542: loop was not vectorized: inner loop was already vectorized

  LOOP BEGIN at Multiply.c(49,9)
  <Peeled loop for vectorization>
  LOOP END

  LOOP BEGIN at Multiply.c(49,9)
  remark #15388: vectorization support: reference a[i][j] has aligned access [ Multiply.c(50,21) ]
  remark #15388: vectorization support: reference x[j] has aligned access [ Multiply.c(50,31) ]
  remark #15305: vectorization support: vector length 2
  remark #15399: vectorization support: unroll factor set to 4
  remark #15309: vectorization support: normalized vectorization overhead 1.031
  remark #15300: LOOP WAS VECTORIZED
  remark #15442: entire loop may be executed in remainder
  remark #15448: unmasked aligned unit stride loads: 2
  remark #15475: --- begin vector cost summary ---
  remark #15476: scalar cost: 10
  remark #15477: vector cost: 4.000
  remark #15478: estimated potential speedup: 2.380
  remark #15488: --- end vector cost summary ---
  LOOP END

  LOOP BEGIN at Multiply.c(49,9)
  <Alternate Alignment Vectorized Loop>
  LOOP END

  LOOP BEGIN at Multiply.c(49,9)
  <Remainder loop for vectorization>
  LOOP END
LOOP END

```

```

KNU: :s9 [tb116 src]$ ./MatVector

```

```

ROW:101 COL: 102
Execution time is 4.270 seconds
GigaFlops = 4.777800
Sum of result = 195853.999899

```

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

```

KNU: :s9 [tb116 src]$ icc -std=c99 -qopt-report=2 -qopt-report-phase=vec -D NOALIAS -D ALIGNED -ipo Multiply.c Dr
ver.c -o MatVector

```

```

KNU: :s9 [tb116 src]$ cat ipo_out.optirpt
Intel(R) Advisor can now assist with vectorization and show optimization
report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.

Begin optimization report for: main()

  Report from: Vector optimizations [vec]

LOOP BEGIN at Driver.c(152,16)
  remark #15542: loop was not vectorized: inner loop was already vectorized

  LOOP BEGIN at Multiply.c(37,5) inlined into Driver.c(150,9)
    remark #15542: loop was not vectorized: inner loop was already vectorized

    LOOP BEGIN at Multiply.c(49,9) inlined into Driver.c(150,9)
      remark #15300: LOOP WAS VECTORIZED
    LOOP END

    LOOP BEGIN at Multiply.c(49,9) inlined into Driver.c(150,9)
      <Remainder loop for vectorization>
      remark #15335: remainder loop was not vectorized: vectorization possible but seems inefficient. Use vector
always directive or -vec-threshold0 to override
    LOOP END
  LOOP END
LOOP END

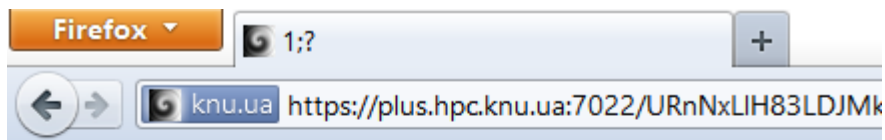
LOOP BEGIN at Driver.c(74,5) inlined into Driver.c(159,5)
  remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at Driver.c(74,5) inlined into Driver.c(159,5)
<Remainder loop for vectorization>
LOOP END
-----
Driver.c Driver.optirpt IntelAdvisor
KNU: :s9 [tb116 src]$ ./MatVector

Row:101 COL: 102
Execution time is 3.989 seconds
GigaFlops = 5.114332
Sum of result = 195853.999899

```

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



```
#include <stdio.h>
```

```
int main() {  
    long long sum = 0;  
    for (int i = 1; i <= 3000000; i++) {  
        if (i % 3 == 0 || i % 5 == 0) {  
            sum += i;  
        }  
    }  
    printf("Sum: %lld\n", sum);  
    return 0;  
}
```

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.

```
KNU: :s7 [tb116 ~]$ ls  
vec_samples vec_samples_C_lin_20170911.tgz work1.cpp  
KNU: :s7 [tb116 ~]$
```

Запустимо програму без оптимізації 10000 разів та продемонструю час виконання.

```
KNU: :s9 [tb116 ~]$ time for i in {1..10000}; do ./work1; done  
real    2m49.801s  
user    2m44.369s  
sys     0m5.081s
```

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

```
KNU: :s9 [tb116 ~]$ gcc -O1 -o work1 work1.cpp  
KNU: :s9 [tb116 ~]$ time for i in {1..100}; do ./work1; done  
real    0m19.677s  
user    0m14.397s  
sys     0m4.956s
```

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

```
KNU: :s9 [tb116 ~]$ gcc -O2 -o work1 work1.cpp  
KNU: :s9 [tb116 ~]$ time for i in {1..100}; do ./work1; done
```



```
real    0m7.366s
user    0m3.840s
sys     0m3.498s
```

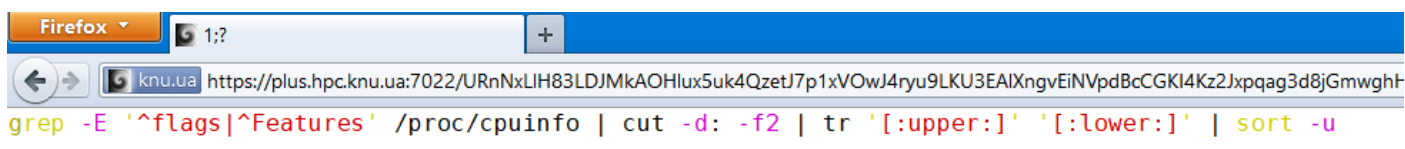
Запустимо програму з оптимізацією O3 10000 разів та продемонструю час виконання.

```
KNU: :s9 [tb116 ~]$ gcc -O3 -o work1 work1.cpp
KNU: :s9 [tb116 ~]$ time for i in {1..100}; do ./work1; done

real    0m7.393s
user    0m3.892s
sys     0m3.475s
```

- сценарій, що отримує перелік всіх розширень процесору що підтримуються

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

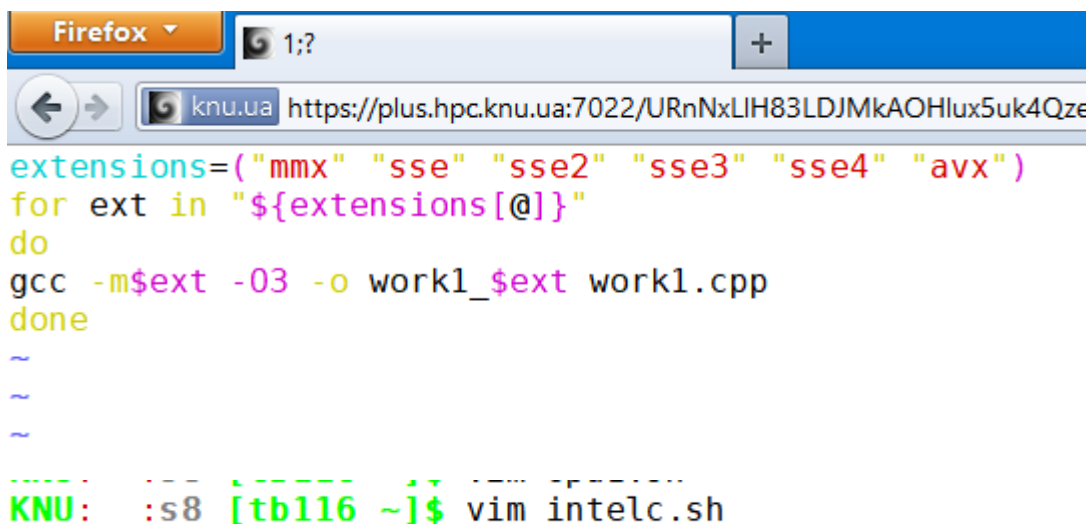


```
Firefox 1:2
knu.ua https://plus.hpc.knu.ua:7022/URnNxLIH83LDJMkAOHlux5uk4QzetJ7p1xVOwJ4ryu9LKU3EAlXngvEiNVpdBcCGKI4Kz2Jxpqag3d8jGmwghT
grep -E '^flags|^Features' /proc/cpuinfo | cut -d: -f2 | tr '[:upper:]' '[:lower:]' | sort -u
~
```

Отримуємо вивід з розширеннями процесору.

```
KNU: :s8 [tb116 ~]$ bash cpu1.sh
fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx
fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts r
ep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vm
x smx est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer ae
s xsave avx lahf_lm epb ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid xsaveopt d
therm ida arat pln pts md_clear spec_ctrl intel_stibp flush_lld
```

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



```
Firefox 1:2
knu.ua https://plus.hpc.knu.ua:7022/URnNxLIH83LDJMkAOHlux5uk4Qze
extensions=("mmx" "sse" "sse2" "sse3" "sse4" "avx")
for ext in "${extensions[@]}"
do
gcc -m$ext -O3 -o work1_$ext work1.cpp
done
~
~
~
KNU: :s8 [tb116 ~]$ vim intelc.sh
```

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:

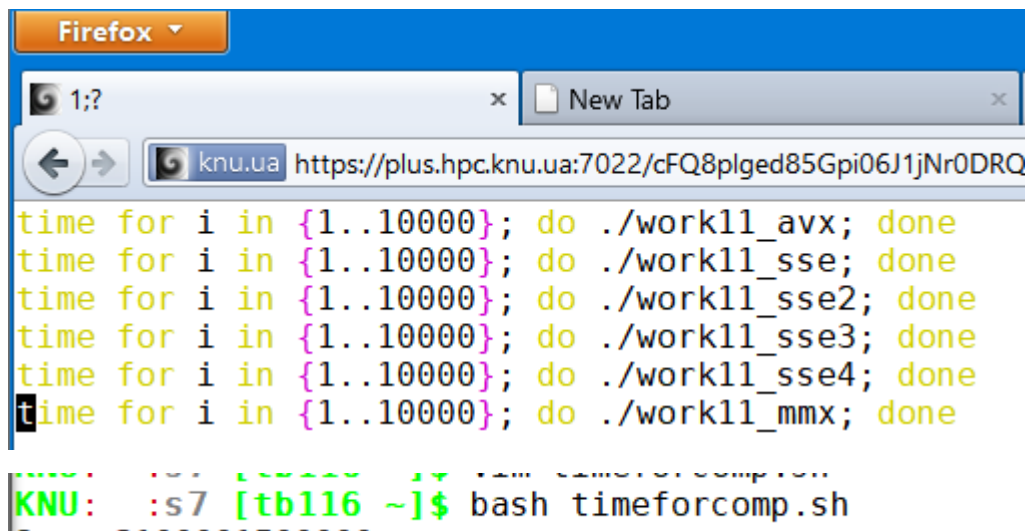
```

work1_mmx
work1_sse
work1_sse2
work1_sse3
work1_sse4

work1_avx

```

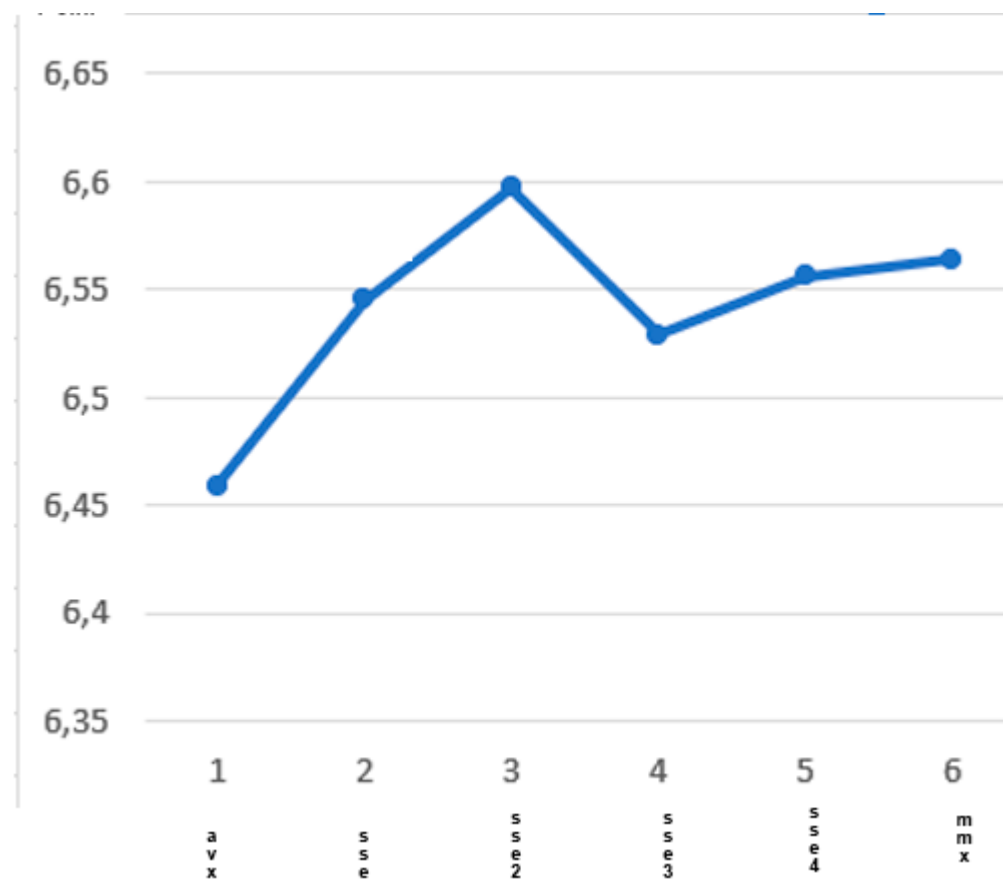
-Напишемо сценарій, що вимірює час виконання кожного варіанта оптимізованої програми.



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

real	0m6.459s	
user	0m3.511s	avx
sys	0m2.923s	
real	0m6.545s	
user	0m3.681s	sse
sys	0m2.839s	
real	0m6.597s	
user	0m3.822s	sse2
sys	0m2.750s	
real	0m6.529s	
user	0m3.655s	sse3
sys	0m2.849s	
real	0m6.566s	
user	0m3.683s	sse4
sys	0m2.858s	
real	0m6.564s	
user	0m3.699s	mmx
sys	0m2.839s	

Побудуємо графіки залежності.



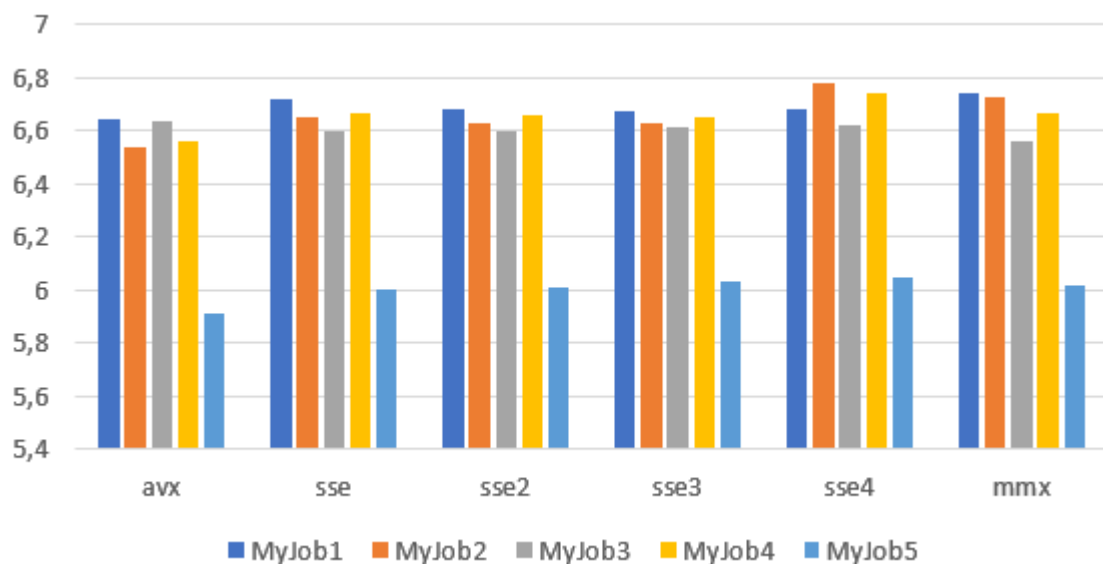
```
Firefox ▾  
1;? x New Tab  
knu.ua https://plus.hpc.knu.ua:7022/cFQ8plged85Gpi06J1jNr0DRQGQoKdYr9QAA ☆ ↻  
for i in {1..5}  
do  
qsub -N MyJob$i -l nodes=1:ppn=1,walltime=00:30:00 timeforcomp.sh  
done  
~
```

```
Firefox ▾  
1;? x New  
knu.ua https://plus.hpc.knu.ua:7022  
for i in {1..5}  
do  
echo "MyJob$i"  
num=47  
num=$(( $num+$i ))  
temp="e26391$num"  
temp=". "$temp  
cat MyJob$i$temp | grep "real"  
done
```

```

MyJob1
real    0m6.641s
real    0m6.720s
real    0m6.679s
real    0m6.677s
real    0m6.681s
real    0m6.743s
MyJob2
real    0m6.541s
real    0m6.649s
real    0m6.627s
real    0m6.628s
real    0m6.777s
real    0m6.724s
MyJob3
real    0m6.637s
real    0m6.601s
real    0m6.599s
real    0m6.617s
real    0m6.622s
real    0m6.564s
MyJob4
real    0m6.560s
real    0m6.668s
real    0m6.658s
real    0m6.648s
real    0m6.744s
real    0m6.663s
MyJob5
real    0m5.914s
real    0m6.004s
real    0m6.014s
real    0m6.032s
real    0m6.045s
real    0m6.015s

```



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.