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.

```
[tb289@plus7 ~]$ ls
V_Kravchenko
[tb289@plus7 ~]$ cd V_Kravchenko/
[tb289@plus7 V_Kravchenko]$ mkdir lab_3
[tb289@plus7 V_Kravchenko]$ cd lab_3/
[tb289@plus7 lab_3]$ wget https://software.intel.com/sites/default/files/vec_samples_C_lin_20170911.tgz
--2019-04-14 15:27:38--  https://software.intel.com/sites/default/files/vec_samples_C_lin_20170911.tgz
Resolving software.intel.com (software.intel.com)... 2.18.68.5
Connecting to software.intel.com (software.intel.com)|2.18.68.5|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 155309 (152K) [application/octet-stream]
Saving to: ‘vec_samples_C_lin_20170911.tgz’

100%[=============================================================================================================================>] 155,309     --.-K/s   in 0.07s

2019-04-14 15:27:38 (2.14 MB/s) - ‘vec_samples_C_lin_20170911.tgz’ saved [155309/155309]

[tb289@plus7 lab_3]$ ls
vec_samples_C_lin_20170911.tgz
```

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. 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

```
[tb289@plus7 lab_3]$ pwd
/home/grid/testbed/tb289/V_Kravchenko/lab_3
[tb289@plus7 lab_3]$ cd vec_samples/
[tb289@plus7 vec_samples]$ qsub -I -l nodes=1:ppn=1,walltime=00:30:00
qsub: waiting for job 2788799 to start
qsub: job 2788799 ready

autoscratch: creating directory '/mnt/work/tb289'
autoscratch: creating directory '/mnt/scratch/tb289'
KNU:  :s6 [tb289 ~]$ ml icc
```

➢ Establishing a Performance Baseline

```
KNU:  :s6 [tb289 vec_samples]$ icc -O1 -std=c99 src/Multiply.c src/Driver.c -o MatVector
KNU:  :s6 [tb289 vec_samples]$ ls
Makefile   build.bat    msvs2013  msvs2017     resources  tutorial              vec_samples_2015.sln
MatVector  license.txt  msvs2015  readme.html  src        vec_samples_2013.sln  vec_samples_2017.sln
KNU:  :s6 [tb289 vec_samples]$ ./MatVector

ROW:101 COL: 101
Execution time is 22.968 seconds
GigaFlops = 0.888289
Sum of result = 195853.999899
```

➢ Generating a Vectorization Report

```
KNU:  :s6 [tb289 vec_samples]$ icc -std=c99 -O2 -D NOFUNCCALL -qopt-report=1 -qopt-report-phase=vec src/Multiply.c src/Driver.c -o MatVector
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
KNU:  :s6 [tb289 vec_samples]$ ls
Driver.optrpt  MatVector       build.bat    msvs2013  msvs2017     resources  tutorial              vec_samples_2015.sln
Makefile       Multiply.optrpt  license.txt  msvs2015  readme.html  src        vec_samples_2013.sln  vec_samples_2017.sln
```

```
KNU:  :s6 [tb289 vec_samples]$ 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: Vector optimizations [vec]


LOOP BEGIN at src/Multiply.c(37,5)
   remark #25460: No loop optimizations reported

   LOOP BEGIN at src/Multiply.c(49,9)
      remark #25460: No loop optimizations reported
   LOOP END

   LOOP BEGIN at src/Multiply.c(49,9)
   <Remainder>
   LOOP END
LOOP END
================================================================================
```

```
KNU:  :s6 [tb289 vec_samples]$ ./MatVector

ROW:101 COL: 101
Execution time is 6.729 seconds
GigaFlops = 3.031770
Sum of result = 195853.999899
```

Помітно різницю у швидкості виконання за рахунок оптимізації (-O2)!

- Інші опції компіляції:

```
KNU:  :s6 [tb289 vec_samples]$ icc -std=c99 -O2 -D NOFUNCCALL -qopt-report-phase=vec,loop -qopt-report=2 src/Multiply.c src/Driver.c -o MatVector
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
KNU:  :s6 [tb289 vec_samples]$ 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 src/Multiply.c(37,5)
   remark #15541: outer loop was not auto-vectorized: consider using SIMD directive

   LOOP BEGIN at src/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 src/Multiply.c(49,9)
   <Remainder>
   LOOP END
LOOP END
--------------------------------------------------------------------------------
```

➢ Improving Performance by Pointer Disambiguation

```
KNU:  :s6 [tb289 vec_samples]$ icc -std=c99 -qopt-report=2 -qopt-report-phase=vec -D NOALIAS src/Multiply.c src/Driver.c -o MatVector
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
KNU:  :s6 [tb289 vec_samples]$ ./Ma
-bash: ./Ma: No such file or directory
KNU:  :s6 [tb289 vec_samples]$ ./MatVector

ROW:101 COL: 101
Execution time is 8.028 seconds
GigaFlops = 2.541508
Sum of result = 195853.999899
KNU:  :s6 [tb289 vec_samples]$ 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 src/Multiply.c(37,5)
   remark #15542: loop was not vectorized: inner loop was already vectorized

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

   LOOP BEGIN at src/Multiply.c(49,9)
      remark #15300: LOOP WAS VECTORIZED
   LOOP END

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

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

➢ Improving Performance by Aligning Data

```
KNU:  :s6 [tb289 vec_samples]$ icc -std=c99 -qopt-report=4 -qopt-report-phase=vec -D NOALIAS -D ALIGNED src/Multiply.c src/Driver.c -o MatVector
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
KNU:  :s6 [tb289 vec_samples]$ ./MatVector

ROW:101 COL: 102
Execution time is 7.974 seconds
GigaFlops = 2.558598
Sum of result = 195853.999899

KNU:  :s6 [tb289 vec_samples]$ 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 -D ALIGNED -o MatVector

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

    Report from: Vector optimizations [vec]


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

   LOOP BEGIN at src/Multiply.c(49,9)
      remark #15388: vectorization support: reference a[i][j] has aligned access   [ src/Multiply.c(50,21) ]
      remark #15388: vectorization support: reference x[j] has aligned access   [ src/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 src/Multiply.c(49,9)
   <Remainder loop for vectorization>
      remark #15388: vectorization support: reference a[i][j] has aligned access   [ src/Multiply.c(50,21) ]
      remark #15388: vectorization support: reference x[j] has aligned access   [ src/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
```

➢ Improving Performance with Interprocedural Optimization

```
KNU:  :s6 [tb289 vec_samples]$ icc -std=c99 -qopt-report=2 -qopt-report-phase=vec -D NOALIAS -D ALIGNED -ipo src/Multiply.c src/Driver.c -o MatV
icc: remark #10397: optimization reports are generated in *.optrpt files in the output location
KNU:  :s6 [tb289 vec_samples]$ ./MatVector

ROW:101 COL: 102
Execution time is 6.428 seconds
GigaFlops = 3.174171
Sum of result = 195853.999899

KNU:  :s6 [tb289 vec_samples]$ cat ipo_out.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 src/Driver.c(152,16)
   remark #15542: loop was not vectorized: inner loop was already vectorized

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

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

      LOOP BEGIN at src/Multiply.c(49,9) inlined into src/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 src/Driver.c(74,5) inlined into src/Driver.c(159,5)
   remark #15300: LOOP WAS VECTORIZED
LOOP END

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

Begin optimization report for: init_matrix(int, int, double, double (*)[102])

    Report from: Vector optimizations [vec]


LOOP BEGIN at src/Driver.c(47,5)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at src/Driver.c(48,9)
      remark #15300: LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at src/Driver.c(48,9)
   <Remainder loop for vectorization>
   LOOP END
LOOP END

LOOP BEGIN at src/Driver.c(53,9)
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at src/Driver.c(53,9)
<Remainder loop for vectorization>
LOOP END
=================================================================

Begin optimization report for: init_array(int, double, double *)

    Report from: Vector optimizations [vec]


LOOP BEGIN at src/Driver.c(62,5)
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at src/Driver.c(62,5)
<Remainder loop for vectorization>
LOOP END
=================================================================
```

3. Оберіть будь-яку неінтрерактивну консольну програму мовою C/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**.

```c
//Fibonacci Series using Recursion
#include<stdio.h>
int fib(int n)
{
   if (n <= 1)
      return n;
   return fib(n-1) + fib(n-2);
}

int main ()
{
  int n = 42;
  printf("%d\n", fib(n));
  return 0;
}
```

3.1 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)

```
KNU:  :s1 [tb289 compiled]$ grep "flags" /proc/cpuinfo | uniq
flags           : 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 c
onstant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid
dca sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx lahf_lm epb tpr_shadow vnmi flexpriority ept vpid xsaveopt dtherm ida arat pln pts
```

- 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).

```
#!/bin/bash
    flags=`grep ^flags /proc/cpuinfo | uniq | cut -d':' -f2 | cut -d" " -f2- | tr _ . | tr "a-z" "A-Z"`
    ml icc
        for flag in $flags; do
            icc -O1 -x$flag fibo.cpp -o test.out 2> error
            if [ ! -s "error" ]; then #check if error occured - so the flag is compatible
                for o in {1..3}; do
                    echo "----------------------------"
                    echo "$flag with -O$o"
                    time `icc -O$o -x$flag fibo.cpp`
                    echo "*******************************"
                    echo "file executed for 5 times in"
                    time `for i in {1..5}; do ./a.out >/dev/null; done`
                done
            fi
        done;
```

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

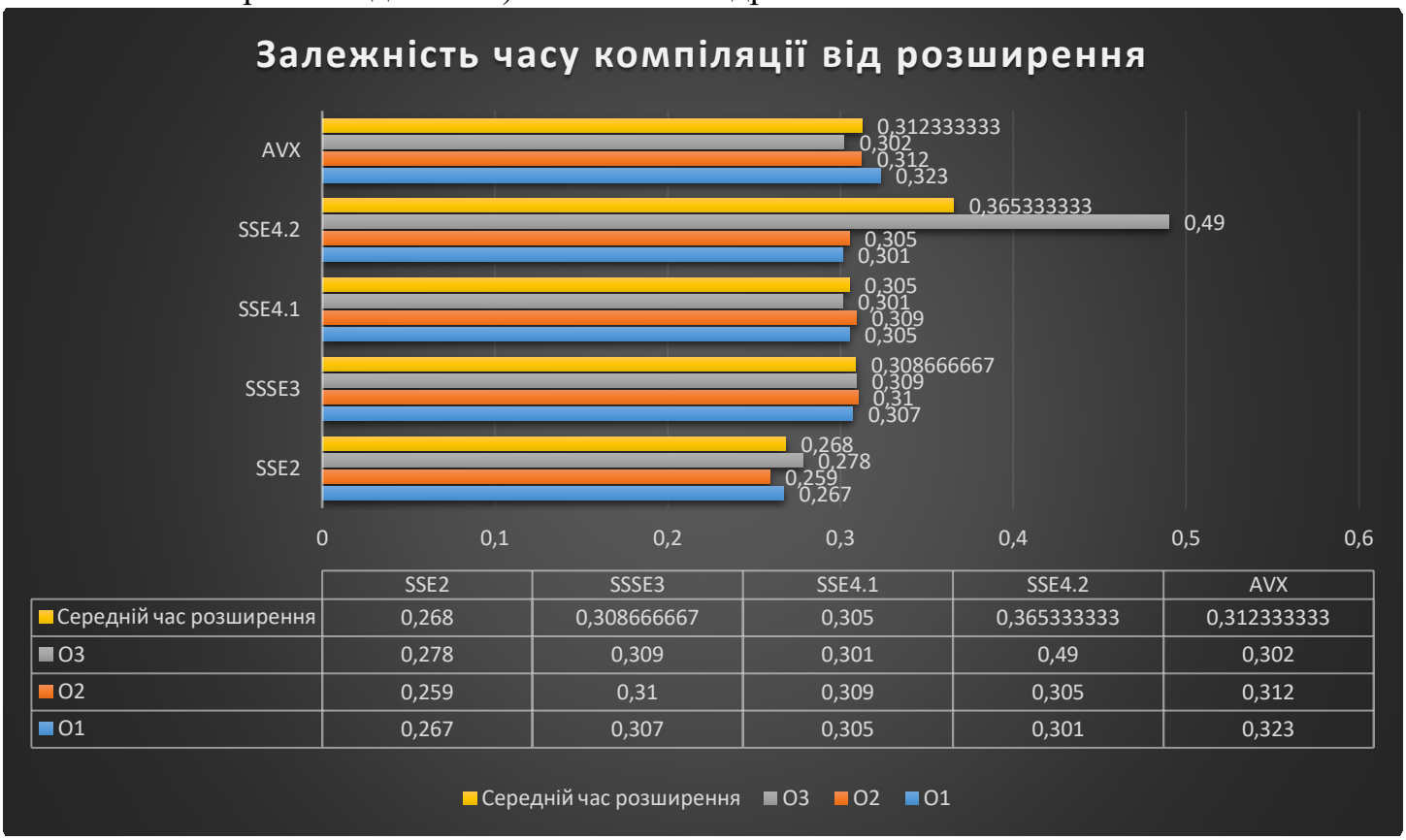
```
KNU:  :s1 [tb289 lab]$ qsub -N MorningJob -l nodes=1:ppn=1,walltime=00:30:00 script.bash
2789204
KNU:  :s1 [tb289 lab]$ qsub -N MorningJob -l nodes=1:ppn=1,walltime=00:30:00 script.bash
2789205
KNU:  :s1 [tb289 lab]$ qsub -N MorningJob -l nodes=1:ppn=1,walltime=00:30:00 script.bash
2789206
KNU:  :s1 [tb289 lab]$ qsub -N MorningJob -l nodes=1:ppn=1,walltime=00:30:00 script.bash
2789207
KNU:  :s1 [tb289 lab]$ qsub -N MorningJob -l nodes=1:ppn=1,walltime=00:30:00 script.bash
2789208
KNU:  :s1 [tb289 lab]$ qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2788982                   fK1e-6_J08_N500  kolezhuk        55:19:12 R mono_long
2789197                   STDIN            tb380           00:00:00 R interactive
2789203                   STDIN            tb289           00:00:00 R interactive
2789204                   MorningJob       tb289           00:00:00 C mono_short
2789205                   MorningJob       tb289                  0 Q mono_short
2789206                   MorningJob       tb289                  0 Q mono_short
2789207                   MorningJob       tb289                  0 Q mono_short
2789208                   MorningJob       tb289                  0 Q mono_short
```

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

```
KNU:  :s1 [tb289 lab]$ qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2788982                   fK1e-6_J08_N500  kolezhuk        55:29:56 R mono_long
2789197                   STDIN            tb380           00:00:00 R interactive
2789203                   STDIN            tb289           00:00:00 R interactive
KNU:  :s1 [tb289 lab]$ ls
FinallyMyJob.e2789062  FinallyMyJob.e2789068  FinallyMyJob.o2789062  FinallyMyJob.o2789068  MorningJob.e2789204  MorningJob.o2789204  a.out       test.out
FinallyMyJob.e2789063  FinallyMyJob.e2789069  FinallyMyJob.o2789063  FinallyMyJob.o2789069  MorningJob.e2789205  MorningJob.o2789205  echo
FinallyMyJob.e2789064  FinallyMyJob.e2789070  FinallyMyJob.o2789064  FinallyMyJob.o2789070  MorningJob.e2789206  MorningJob.o2789206  error
FinallyMyJob.e2789065  FinallyMyJob.e2789071  FinallyMyJob.o2789065  FinallyMyJob.o2789071  MorningJob.e2789207  MorningJob.o2789207  fibo.cpp
FinallyMyJob.e2789066  FinallyMyJob.e2789072  FinallyMyJob.o2789066  FinallyMyJob.o2789072  MorningJob.e2789208  MorningJob.o2789208  script.bash
```

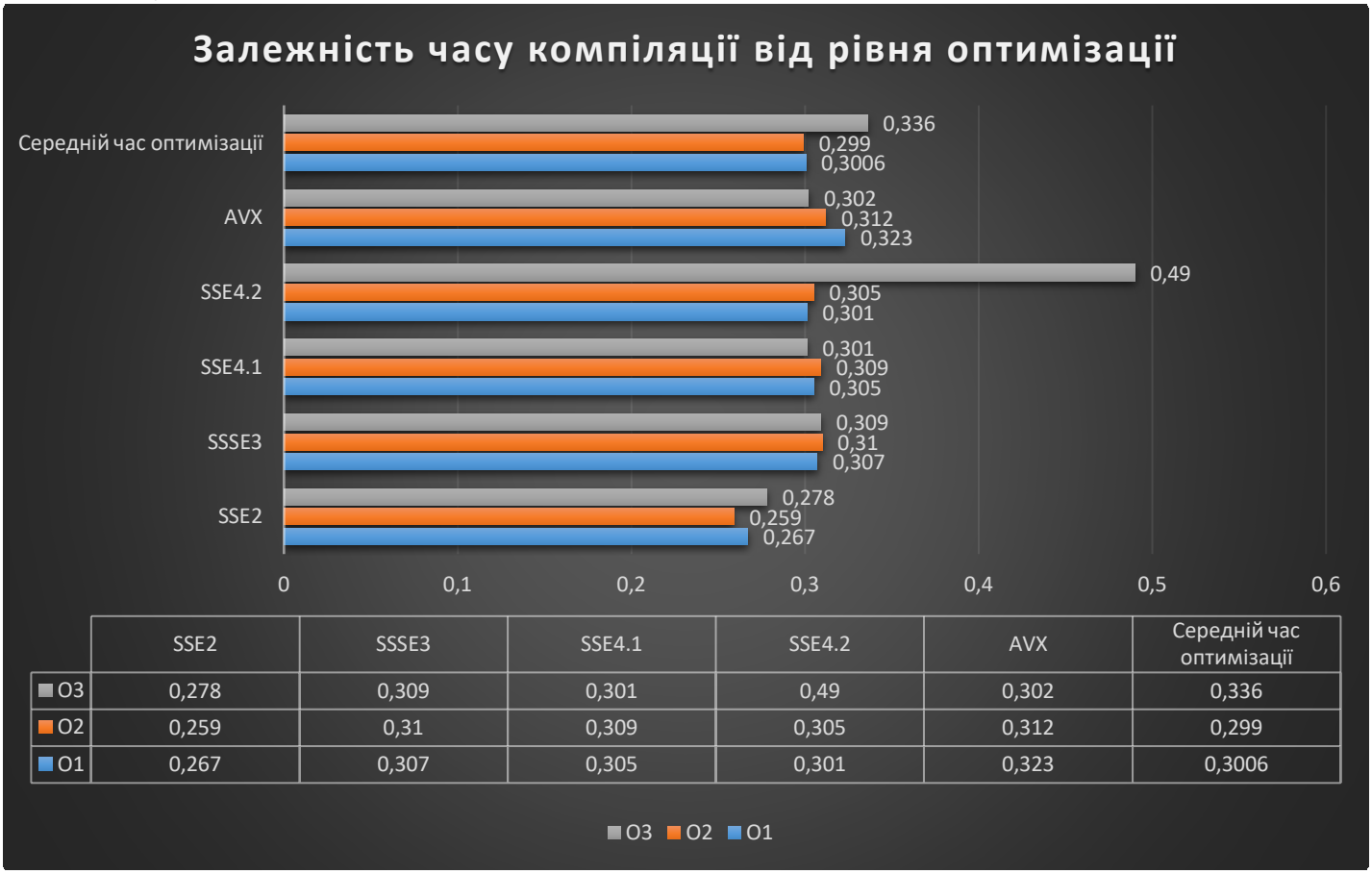
3.3 Побудуйте графіки залежності часу від різних варіантів компіляції.

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



	SSE2	SSSE3	SSE4.1	SSE4.2	AVX
Середній час розширення	0,268	0,308666667	0,305	0,365333333	0,312333333
O3	0,278	0,309	0,301	0,49	0,302
O2	0,259	0,31	0,309	0,305	0,312
O1	0,267	0,307	0,305	0,301	0,323

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.



	SSE2	SSSE3	SSE4.1	SSE4.2	AVX	Середній час оптимізації
O3	0,278	0,309	0,301	0,49	0,302	0,336
O2	0,259	0,31	0,309	0,305	0,312	0,299
O1	0,267	0,307	0,305	0,301	0,323	0,3006

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

	SSE2	SSSE3	SSE4.1	SSE4.2	AVX
Середній час розширення	10,75266667	9,642333333	9,653333333	9,647333333	9,633
O3	11,31	9,643	9,653	9,653	9,652
O2	11,313	9,649	9,671	9,652	9,664
O1	9,635	9,635	9,636	9,637	9,583

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

	SSE2	SSSE3	SSE4.1	SSE4.2	AVX	Середній час оптимізації
O3	11,31	9,643	9,653	9,653	9,652	9,9822
O2	11,313	9,649	9,671	9,652	9,664	9,9898
O1	9,635	9,635	9,636	9,637	9,583	9,6252

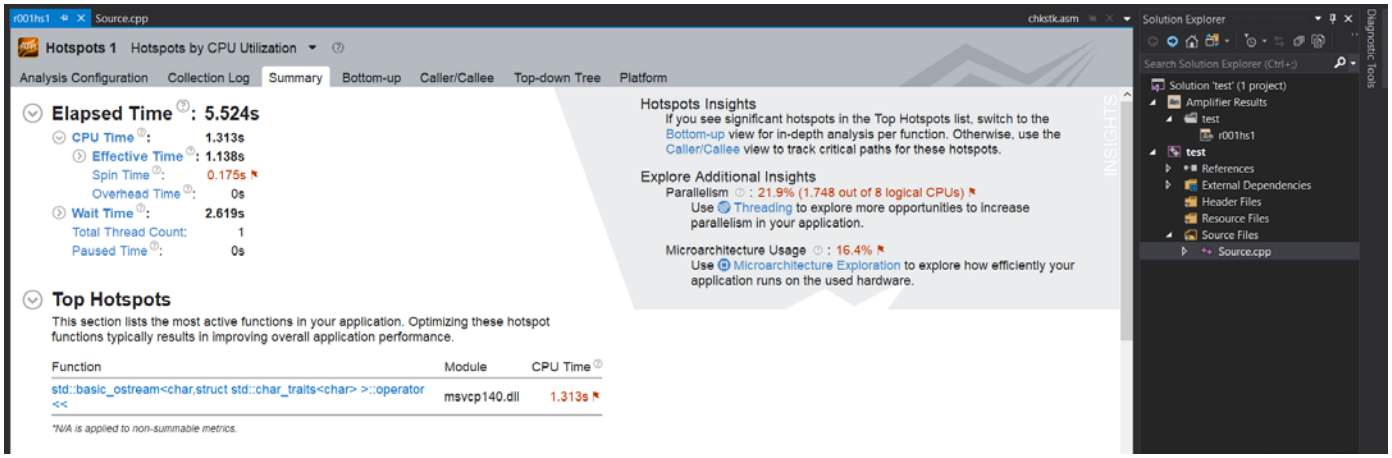
4. Оберіть будь-який зі створених вами програмних продуктів та виконайте його оптимізацію з використання Intel® Parallel Studio.

```
#include <iostream>
using namespace std;
int main(void) {
    int table[100][100] = { };
    int *pointer = &table[0][0];
    for (int i = 1; i <= 100; i++)
    {
        for (int j = 1; j <= 100; j++)
        {
            *pointer = i * j;
            pointer++;
        }
    }

    for (int i = 0; i < 100; i++)
    {
        for (int j = 0; j < 100; j++)
        {
            cout.width(5);
            cout << table[i][j];
        }
        cout << endl;
    }
    return 1;
}
```

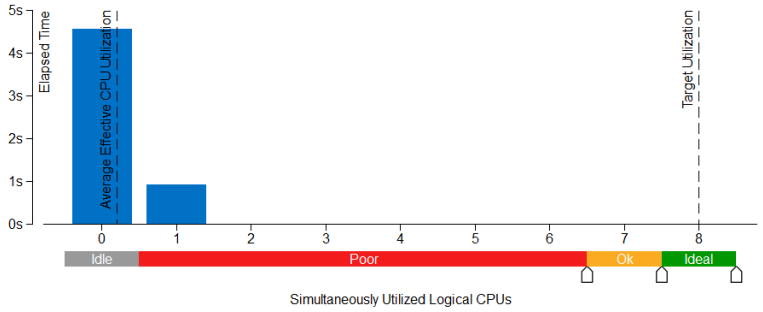
Для тестування я використав програму, яка заповнює та виводить матрицю 100 на 100 значеннями від 1 до 10000.

Нижче ось такі результати до оптимізації:



Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.



Collection and Platform Info

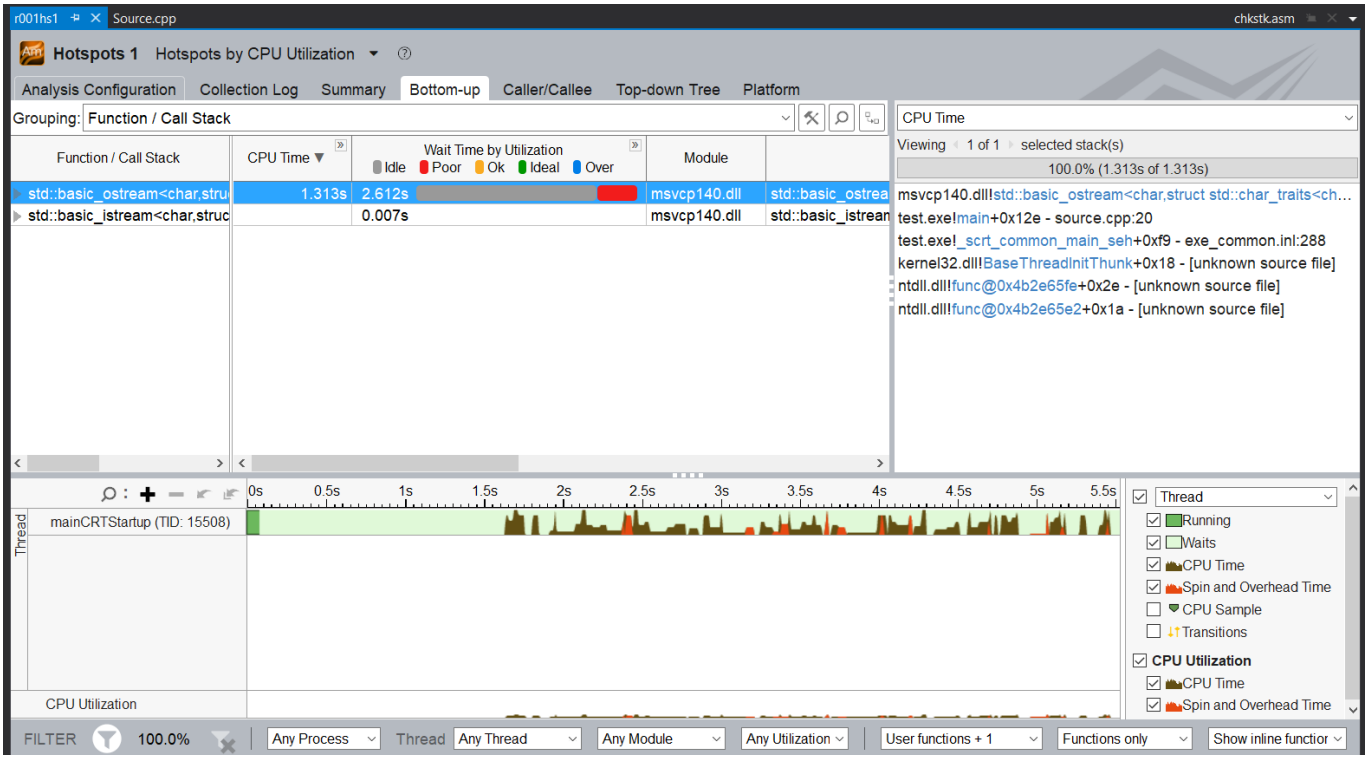
This section provides information about this collection, including result set size and collection platform data.

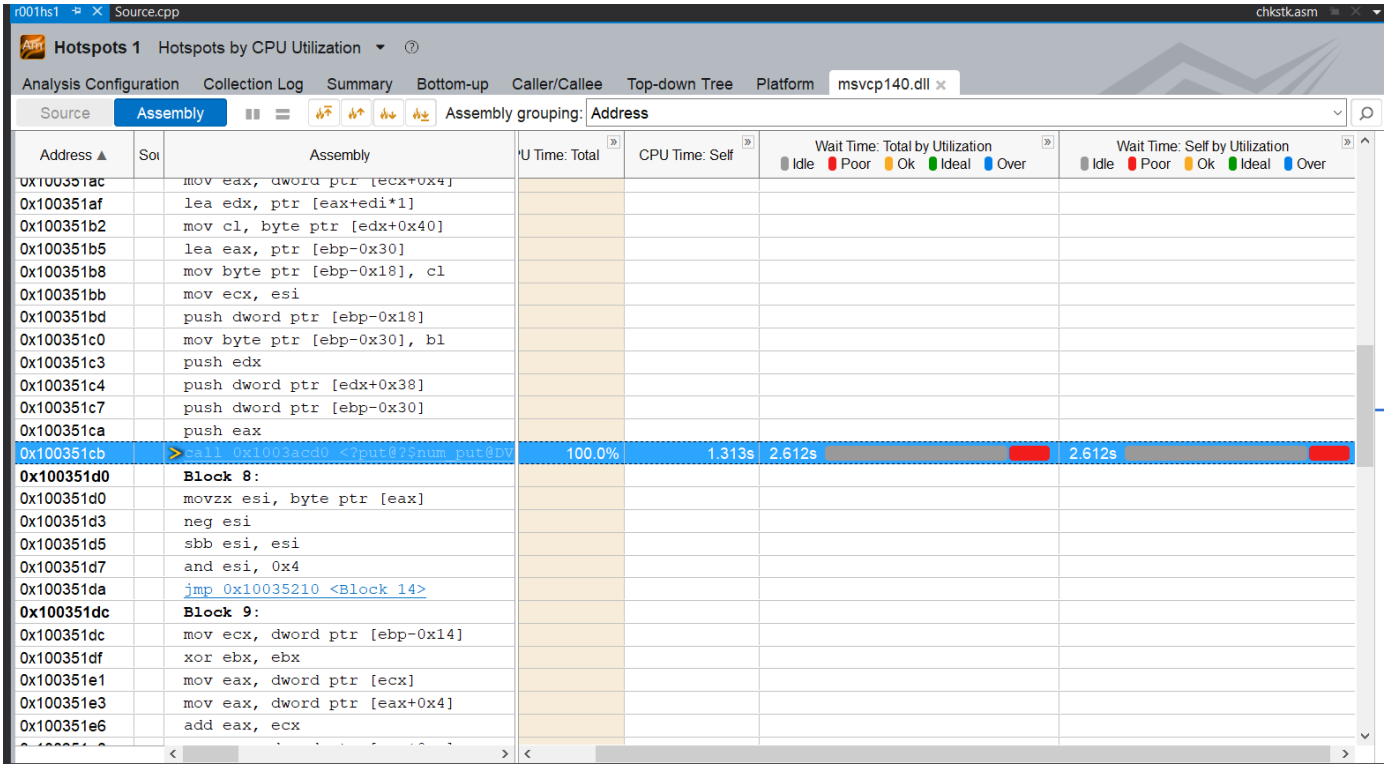
Application Command Line:	C:\Univ\test\Release\test.exe
Environment Variables:	PATH=C:\Program Files (x86)\IntelSWTools\compilers_and_libraries_2019\windows\redist\ia32\compiler;%PATH%; PATH=C:\Program Files (x86)\IntelSWTools\compilers_and_libraries_2019\windows\redist\ia32\compiler;%PATH%
Operating System:	Microsoft Windows 10
Computer Name:	DELL7559
Result Size:	10 MB
Collection start time:	21:35:10 15/04/2019 UTC
Collection stop time:	21:35:16 15/04/2019 UTC
Collector Type:	Event-based counting driver,User-mode sampling and tracing

Finalization mode: Fast. If the number of collected samples exceeds the threshold, this mode limits the number of processed samples to speed up post-processing.

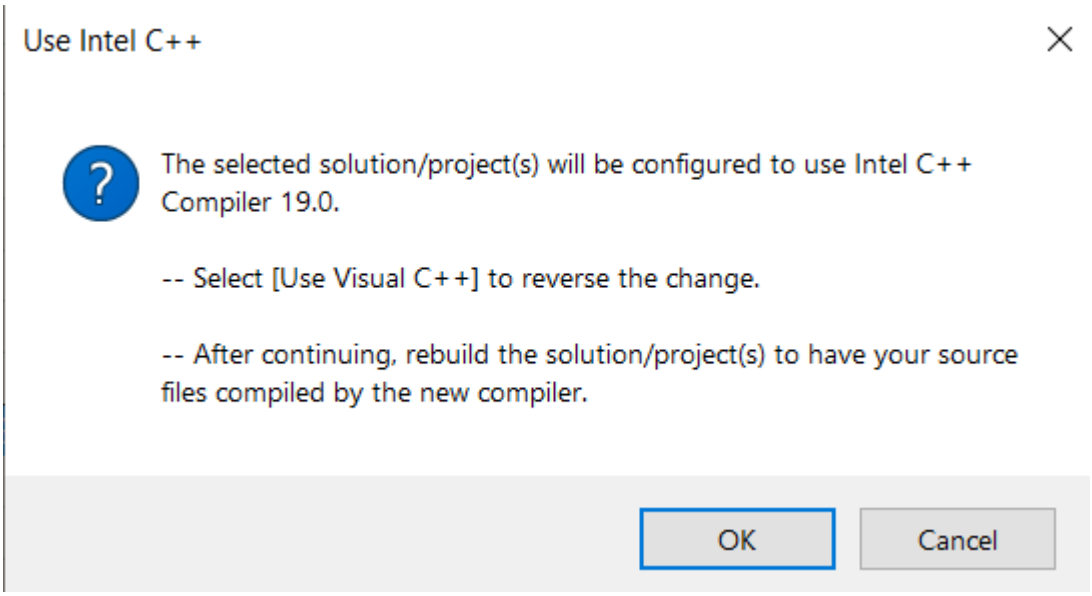
CPU

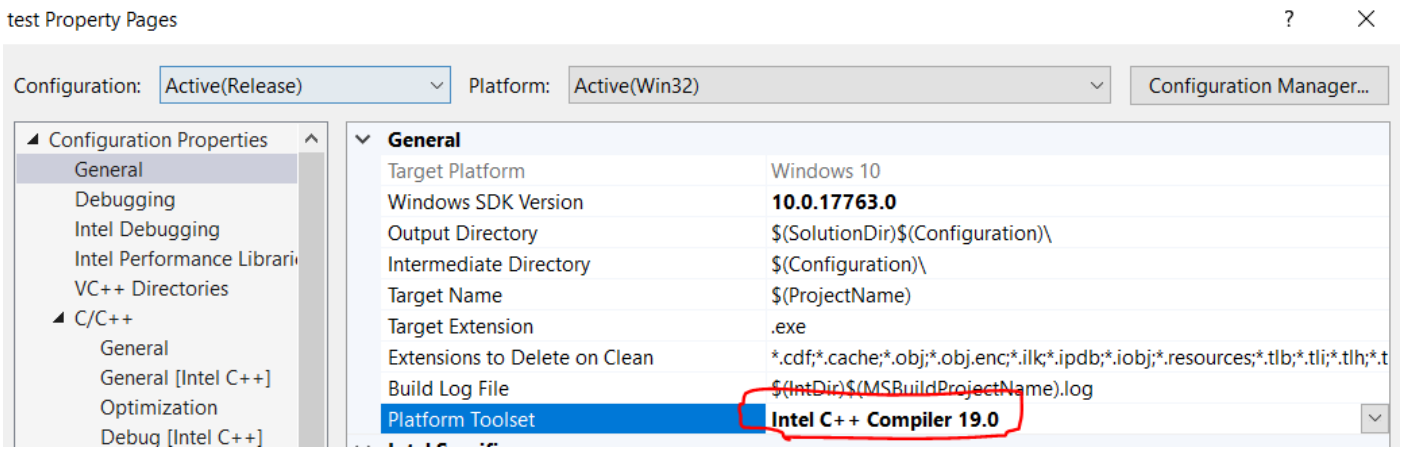
Name:	Intel(R) Processor code named Skylake
Frequency:	2.6 GHz
Logical CPU Count:	8

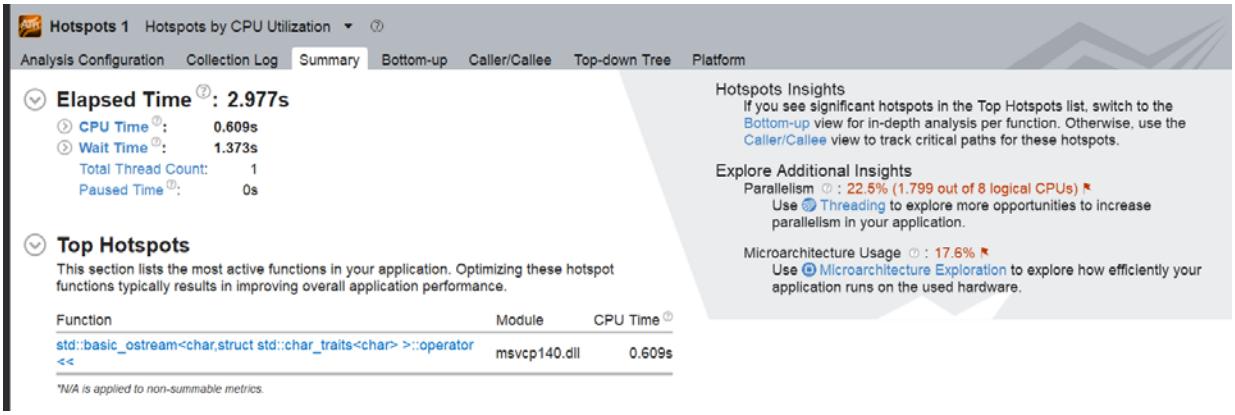


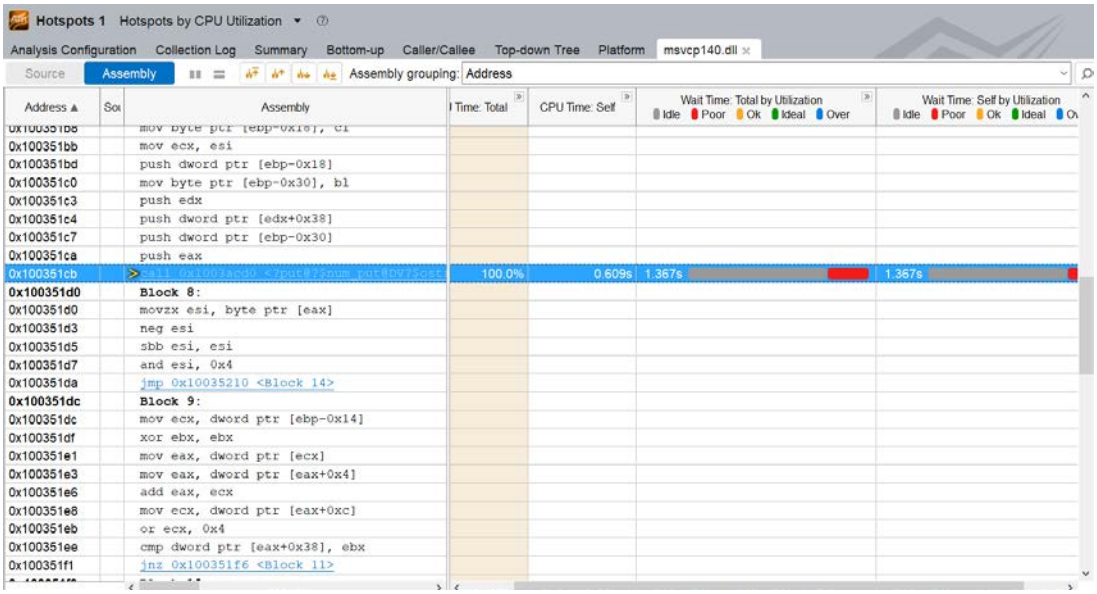


Застосовую новий компілятор:

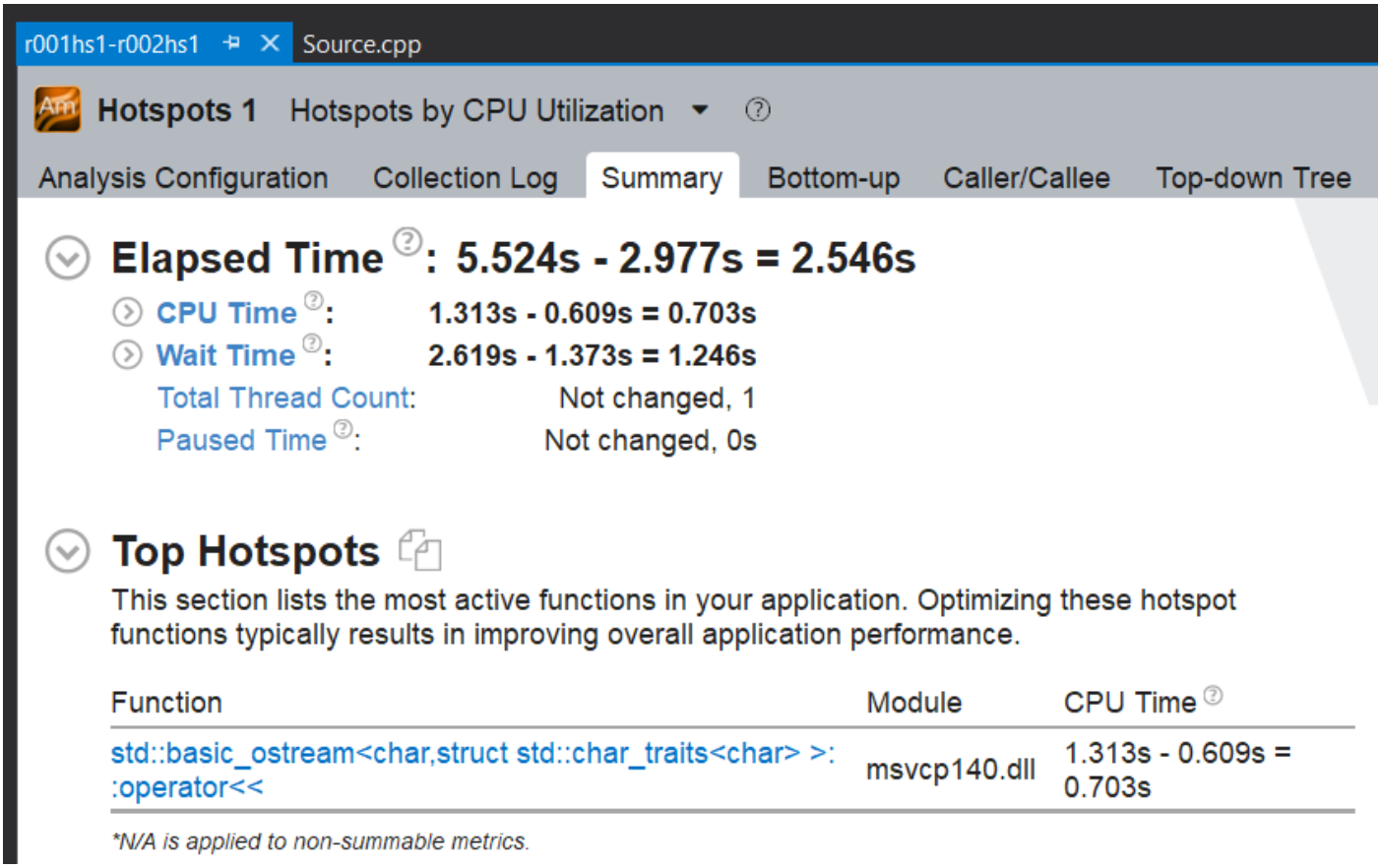


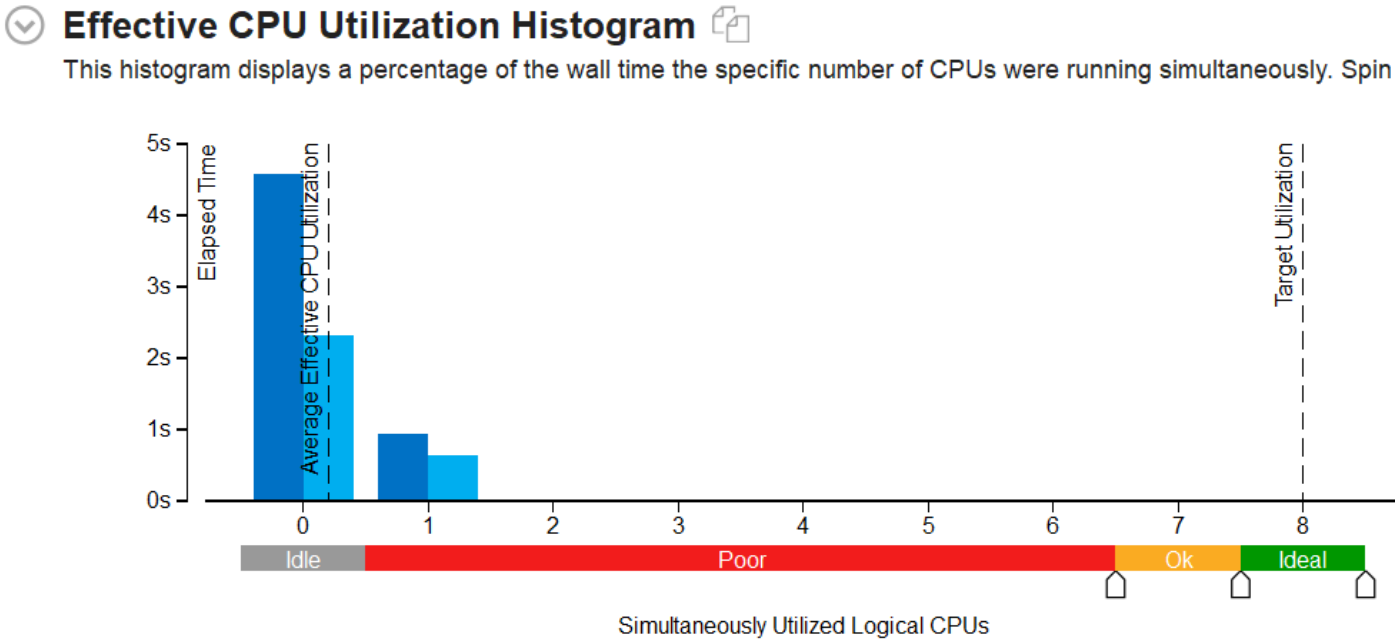






Порівнюю результати:





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).