

**УФИМСКИЙ
УНИВЕРСИТЕТ
НАУКИ И ТЕХНОЛОГИЙ**

Курс «Технологии
параллельного
программирования»

**Лабораторная работа
№2. Разработка и
отладка OpenMP-
программы вычисления
суммы ряда**

Юлдашев Артур Владимирович
art@ugatu.su

Спеле Владимир Владимирович
spele.vv@ugatu.su

Добровольцев Александр Сергеевич
dobrovolcev.as@ugatu.su

Сохатский Михаил Александрович
sohatskii.ma@ugatu.su

**Кафедра высокопроизводительных
вычислений и дифференциальных
уравнений (ВВиДУ)**

Цель работы

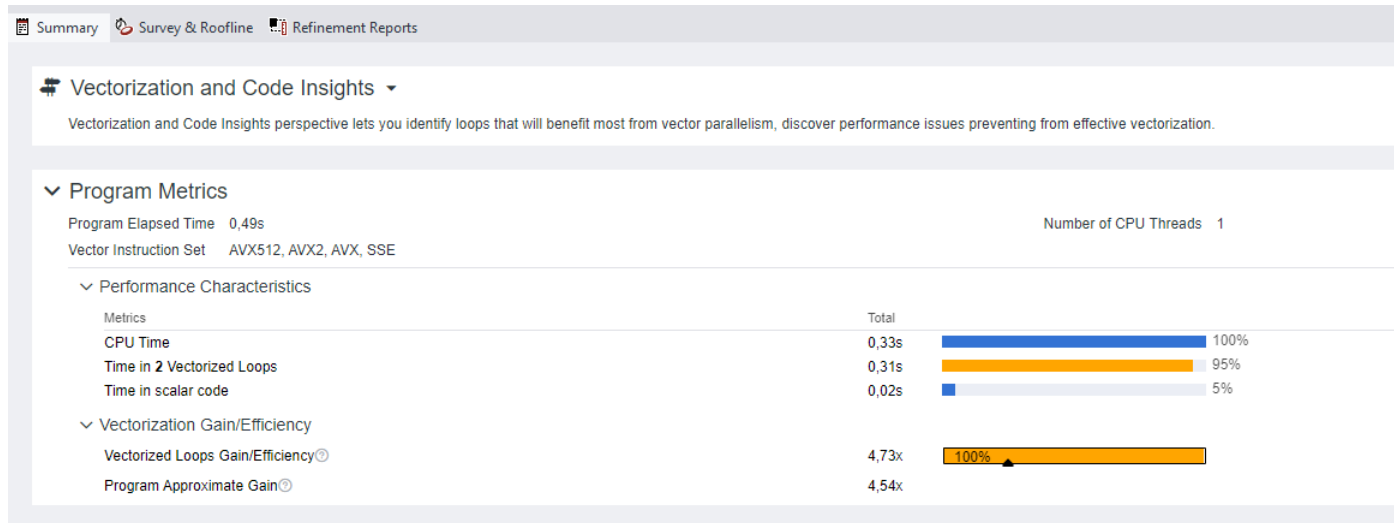
На примере задачи вычисления суммы ряда научиться разрабатывать простейшие параллельные программы средствами OpenMP, а также использовать инструменты для проверки корректности программ Intel Inspector и для профилирования производительности программ Intel VTune Profiler.

Задание

1. Используя последовательную программу с лучшими ключами оптимизации (без автораспараллеливания) из лаб. раб. №1 подобрать N , при котором программа работает ~ 30 с.
2. Оценить эффективность векторизации программы с помощью Intel Advisor и при необходимости обновить размерность N .
3. Выполнить распараллеливание последовательной программы путем включения в её код директив OpenMP. Проанализировать корректность распараллеливания с помощью Intel Inspector и при необходимости внести исправления.
4. Проанализировать производительность параллельной программы с помощью инструмента Intel VTune Profiler и при необходимости внести исправления.
5. Вычислить ускорение и эффективность отлаженной и оптимизированной параллельной программы, полученные данные занести в таблицу. Построить графики зависимостей ускорения и эффективности от числа потоков.

Задание

Используя последовательную программу с лучшими ключами оптимизации (без автораспараллеливания) из лабораторной работы №1 подобрать N, при котором программа работает ~30 сек. Зафиксировать значение суммы ряда (для дальнейшей оценки корректности).

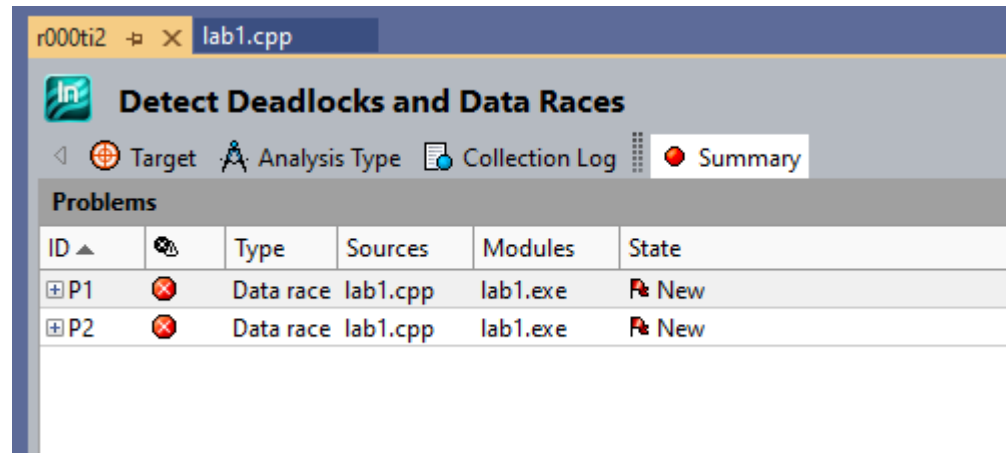


С помощью Intel Advisor оценить эффективность векторизации кода и вставить скриншот в отчет. При необходимости обновить N.

Задание

Провести распараллеливание с помощью директив OpenMP `#pragma omp parallel` и `#pragma omp for`

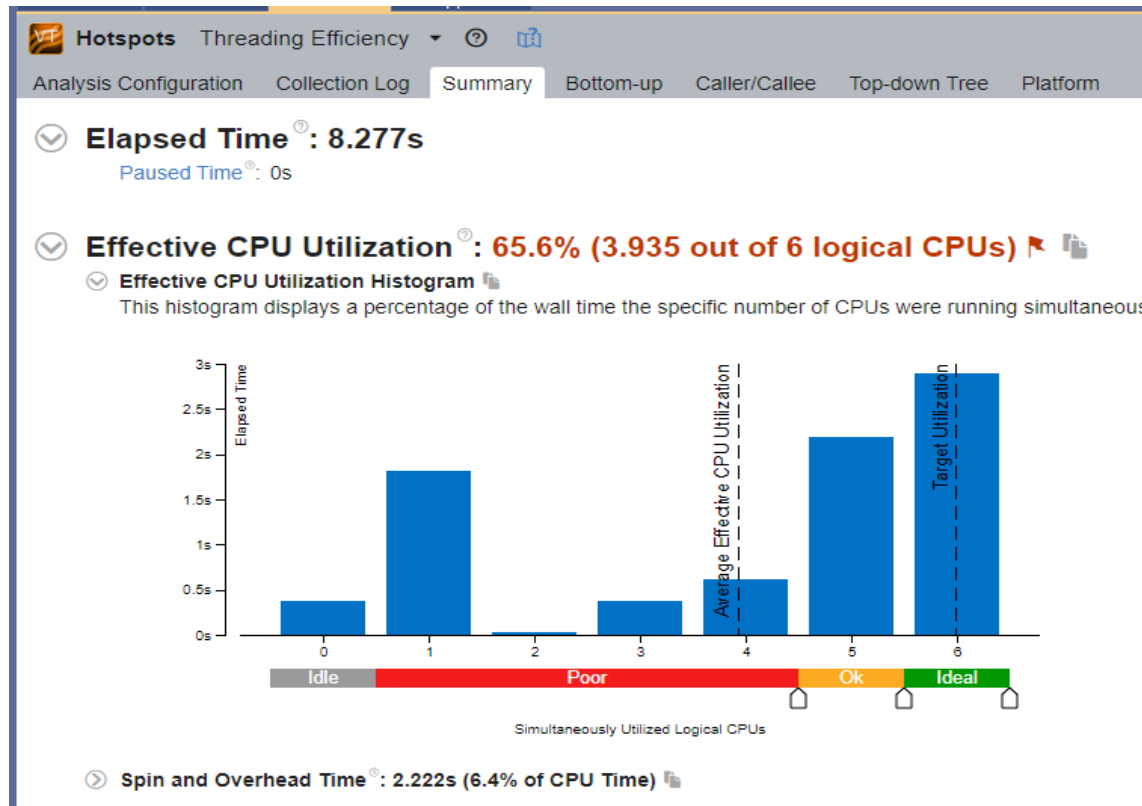
Проверить корректность параллельной программы с помощью инструмента Intel Inspector. Вставить в отчет скриншот.



При наличии ошибок исправить их и проверить корректность исправленной программы в Intel Inspector. Вставить в отчет соответствующий скриншот и краткое описание исправленных ошибок.

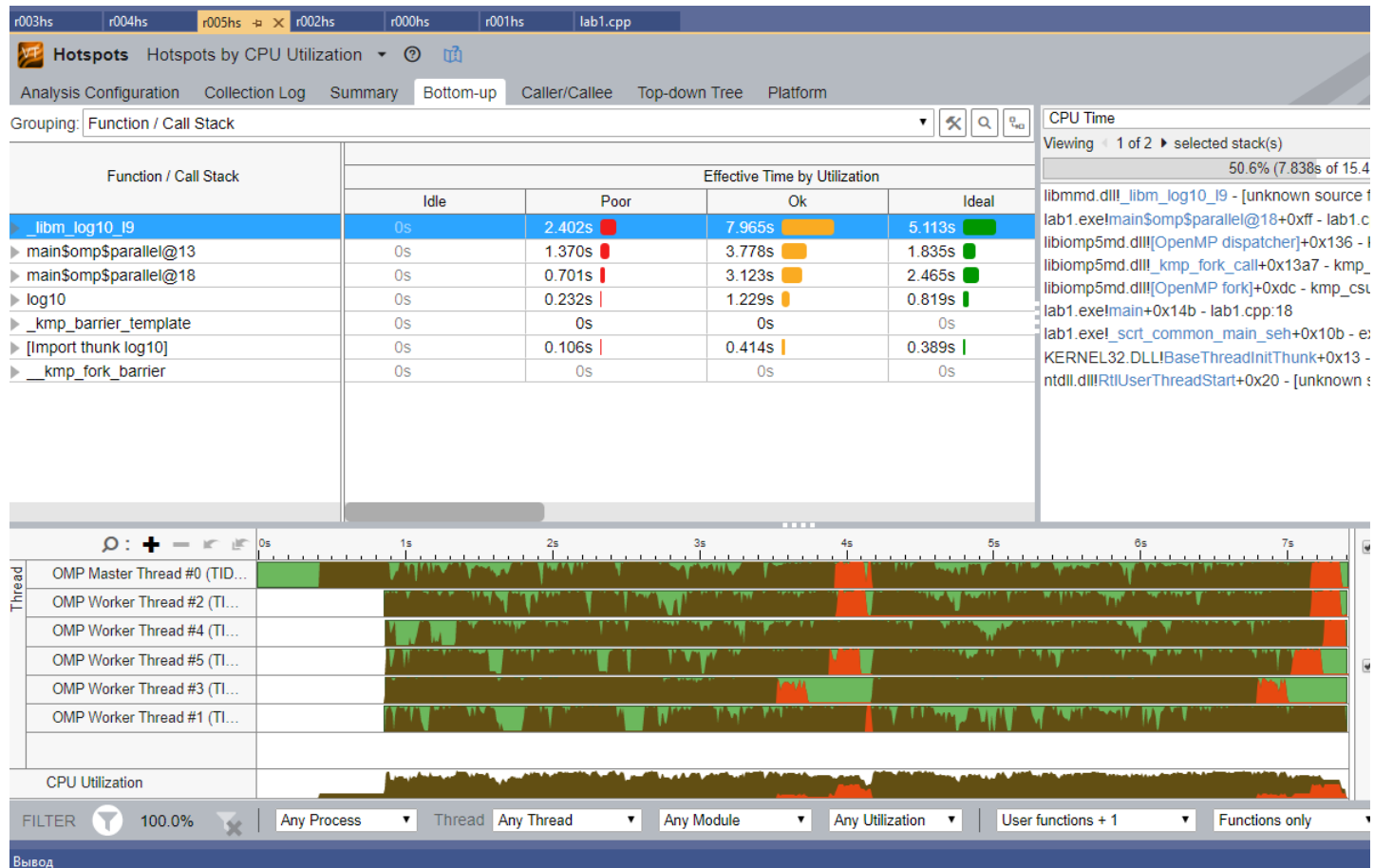
Задание

Снять профиль параллельной программы с помощью инструмента Intel VTune Profiler. Вставить в отчет скриншот эффективности загрузки CPU.



Задание

Вставить в отчет скриншот детальной загрузки каждого ядра CPU.



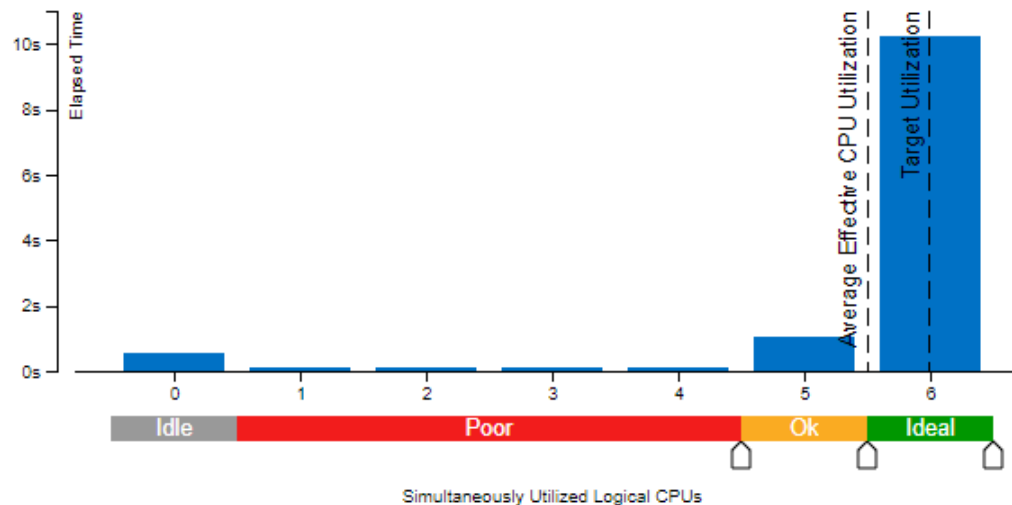
Задание

С использованием спецификатора **schedule** директивы **#pragma omp for** оптимизировать загрузку CPU (при необходимости).

⌵ **Effective CPU Utilization[®] : 91.8% (5.506 out of 6 logical CPUs)**

⌵ **Effective CPU Utilization Histogram**

This histogram displays a percentage of the wall time the specific number of CPUs were running simultai



Вставить в отчет скриншоты с профилями оптимизированной программы.

Задание

Замерить время работы программы на 1, 2 ... p ядрах, где p – максимальное число ядер на вашем ПК.

Кол-во потоков	Время работы
1	
2	
...	
p	

Вычислить ускорение и эффективность параллельной программы, полученные данные занести в таблицу.

Кол-во потоков	Ускорение	Эффективность
1		
2		
...		
p		

По данным таблицы построить графики зависимостей ускорение и эффективности от числа потоков.

Анализ полученных результатов

Определение. Отношение времени выполнения параллельной программы на одном процессоре (ядре) T_1^* ко времени выполнения параллельной программы на p процессорах (ядер) T_p называется **ускорением** при использовании p процессоров:

$$S_p^* = \frac{T_1^*}{T_p}$$

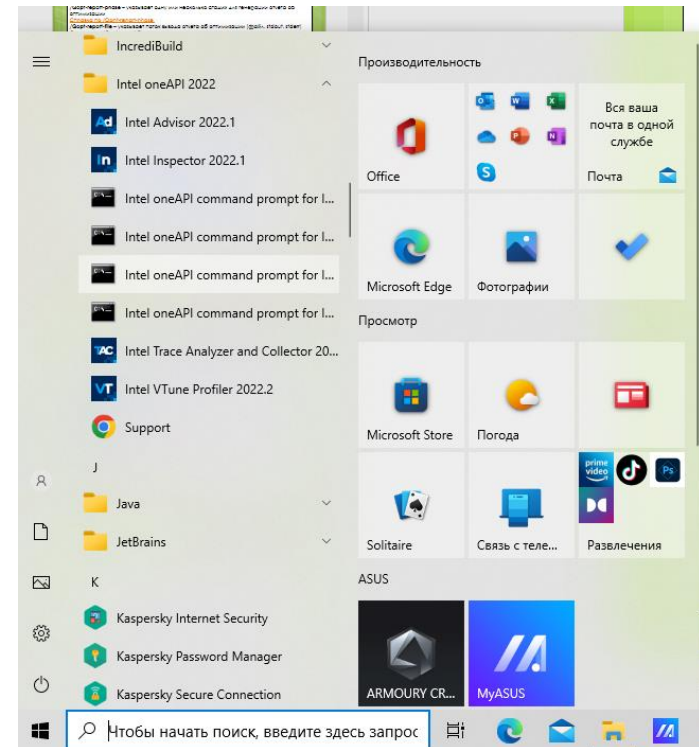
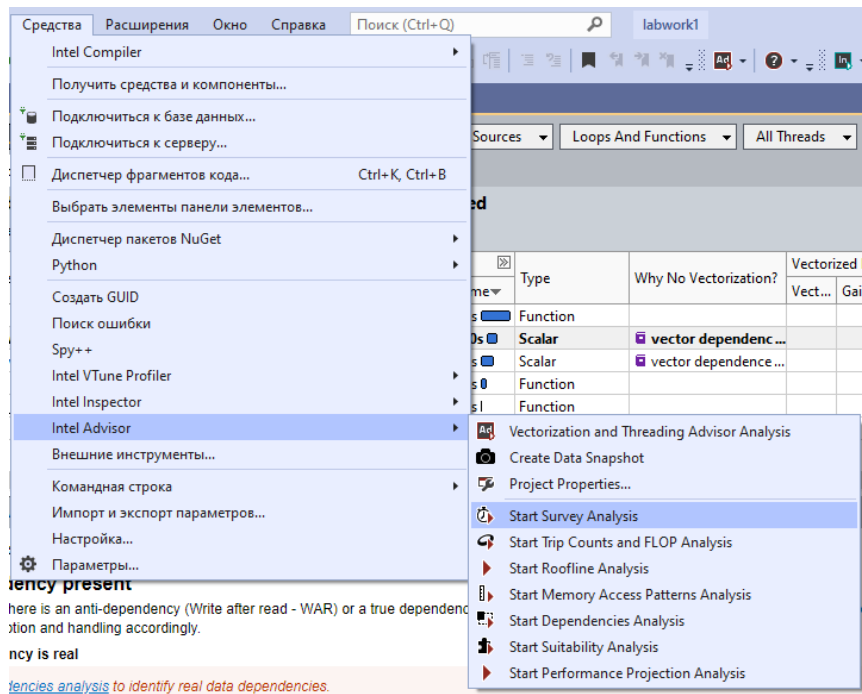
Определение. Отношение ускорения S_p^* к количеству процессоров (ядер) p называется **эффективностью** при использовании p процессоров (ядер):

$$E_p^* = \frac{S_p^*}{p}$$

Требования к оформлению отчета

- В отчет по проделанной работе включить:
 - 1) описание используемых компиляторов;
 - 2) характеристики центрального процессора;
 - 3) скриншоты проверки корректности вычислений при различных размерностях;
 - 4) скриншоты из Intel Inspector и Intel VTune Profiler;
 - 5) заполненные таблицы;
 - 6) графики ускорения и эффективности;
 - 7) ВЫВОД.

Запуск Intel Advisor



Справка по Intel Advisor

Intel Advisor

The screenshot displays the Intel Advisor interface with the following components:

- Analysis Workflow:** Includes buttons for Accuracy (Low, Medium, High, Custom) and Overhead, and checkboxes for Analysis Types (Survey, Characterization, Trip Counts, FLOP, Callstacks, Memory Access Patterns, Dependencies, Reduction Detection).
- Performance Issues:** A table listing issues such as "1 Data type conversions present" and "3 Unoptimized floating point operation processing possible".
- Source Code:** A view of the source code for `Source.cpp:12 main`, showing a loop that calculates the sum of a series.
- Table:** A table showing the results of the analysis, including CPU Time, Type, Why No Vectorization?, Vectorized Loops, and Instruction Set Analysis.

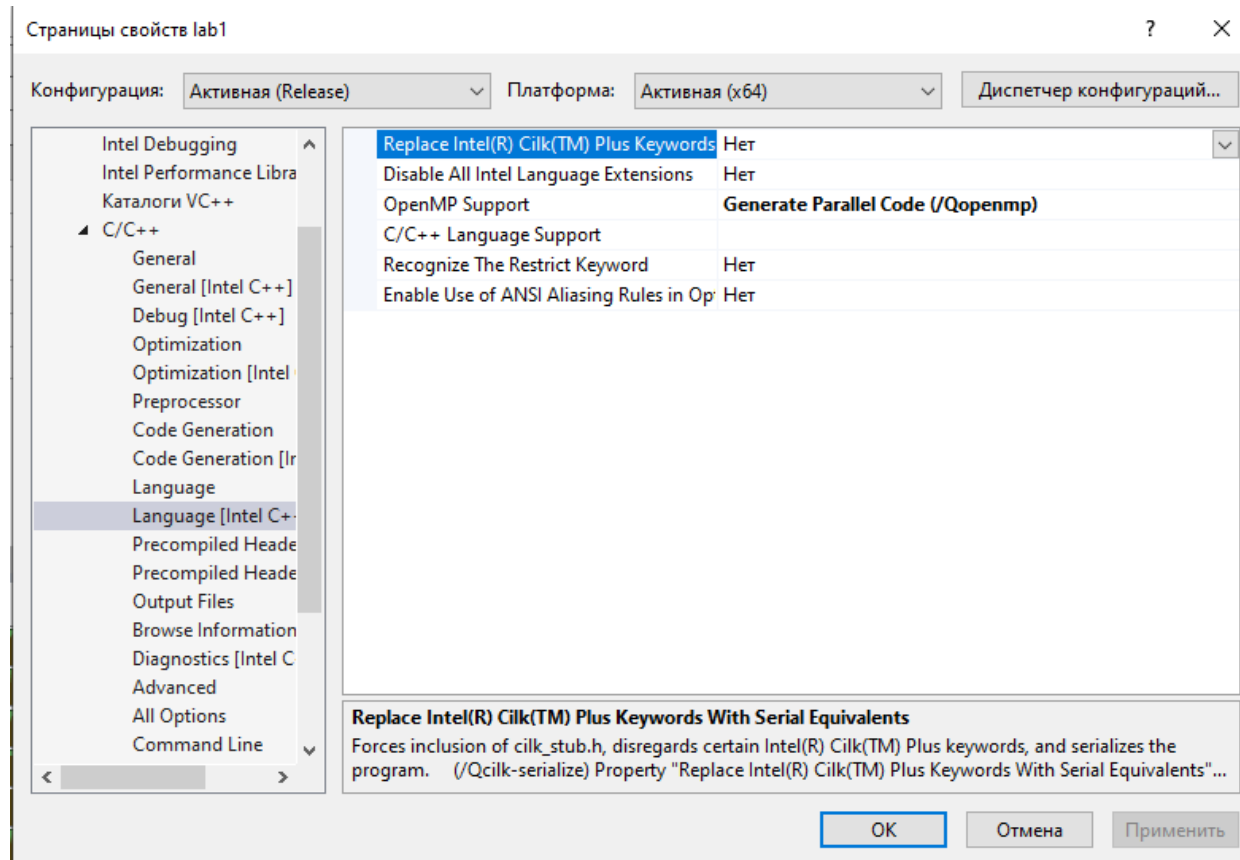
Function Call Sites and Loops	Performance Issues	CPU Time	Type	Why No Vectorization?	Vectorized Loops	Instruction Set Analysis
		Total Time	Self Time		Vect... Efficiency Gain... VL (...)	Traits
<code>_svm_log104_I9</code>	<input type="checkbox"/> 1 Data type conversions present	0,144s	0,144s	Vector Function	AVX2	Appr. Reciprocals(AVX...
<code>[loop in main at Source.cpp:12]</code>	<input type="checkbox"/> 3 Unoptimized floating point operation processing possible	0,156s	0,084s	Vectorized (B...	AVX	Divisions; Type Conv...
<code>[loop in main at Source.cpp:16]</code>	<input type="checkbox"/> 3 Unoptimized floating point operation processing possible	0,158s	0,072s	Vectorized (Bo...	AVX	Divisions; Type Conve...
<code>[Import thunk _svm_log104_I9]</code>		0,014s	0,014s	Vector Function		
<code>invoke_main</code>		0,314s	0,000s	Inlined Function		
<code>_scr_common_main_seh</code>		0,331s	0,000s	Function		
<code>main</code>	<input type="checkbox"/> 1 Data type conversions present	0,314s	0,000s	Function		Divisions; Extracts; Ty...

```
1 #include <iostream>
2 #define _USE_MATH_DEFINES
3 #include <math.h>
4 #include <time.h>
5 using namespace std;
6
7 int main()
8 {
9     int N = 210000000;
10    double start_time = clock();
11    double sum = 0;
12    for (int i = 1; i < N; i+=2)
13    {
14        sum += -1. / (i - log10(i));
15    }
16    for (int i = 2; i < N; i+=2)
17    {
18        sum += 1. / (i - log10(i));
19    }
20    double end_time = clock();
21    cout << "TIME = " << (end_time - start_time) / CLK_TCK << endl;
```

Справка по Vectorization Advisor

Поддержка OpenMP

Включение поддержки OpenMP в проекте на компиляторе Intel.



Пример последовательной программы

```
1  #include <iostream>
2  #define _USE_MATH_DEFINES
3  #include <math.h>
4  #include <time.h>
5
6  using namespace std;
7  int main()
8  {
9      const long long N = 21000000000;
10     double start_time = clock();
11     double sum = 0.0;
12
13
14
15     for (long long i = 1; i < N; i += 2)
16     {
17         sum -= 1 / (i - log10(i));
18     }
19
20     for (long long i = 2; i < N; i += 2)
21     {
22         sum += 1 / (i - log10(i));
23     }
24
25     double end_time = clock();
26     cout << "time = " << (end_time - start_time) / CLK_TCK << endl;
27     cout << "SUM = " << sum << endl;
28     return 0;
29 }
```

Наивное распараллеливание программы

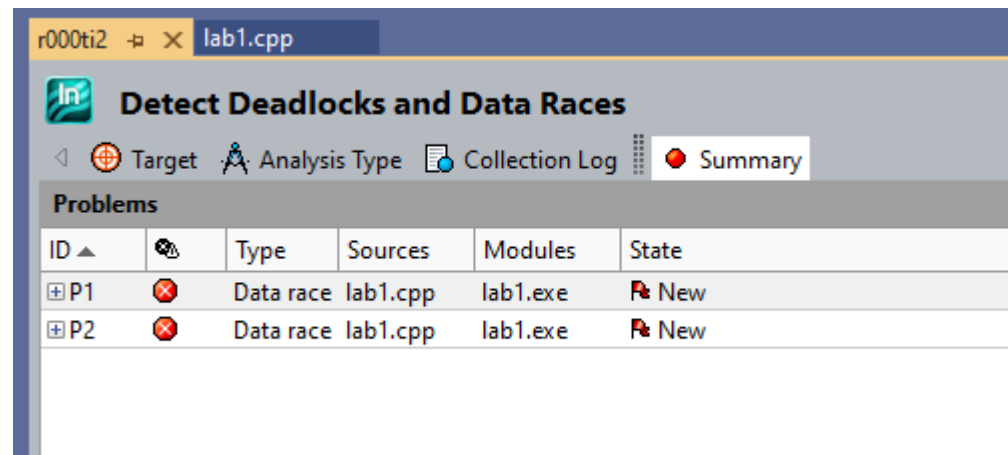
```
1  #include <iostream>
2  #define _USE_MATH_DEFINES
3  #include <math.h>
4  #include <time.h>
5  #include <omp.h>
6  using namespace std;
7  int main()
8  {
9      const long long N = 21000000000;
10     double start_time = omp_get_wtime();
11     double sum = 0.0;
12     #pragma omp parallel num_threads(6)
13     {
14         #pragma omp for
15         for (long long i = 1; i < N; i += 2)
16         {
17             sum -= 1 / (i - log10(i));
18         }
19         #pragma omp for
20         for (long long i = 2; i < N; i += 2)
21         {
22             sum += 1 / (i - log10(i));
23         }
24     }
25     double end_time = omp_get_wtime();
26     cout << "time = " << (end_time - start_time) << endl;
27     cout << "SUM = " << sum << endl;
28     return 0;
29 }
```


Intel Inspector

Переходим в Средства/Intel Inspector/Threading Error Analysis.

ИЛИ

Меню Пуск -> Intel OneAPI 2022 -> Intel Inspector.

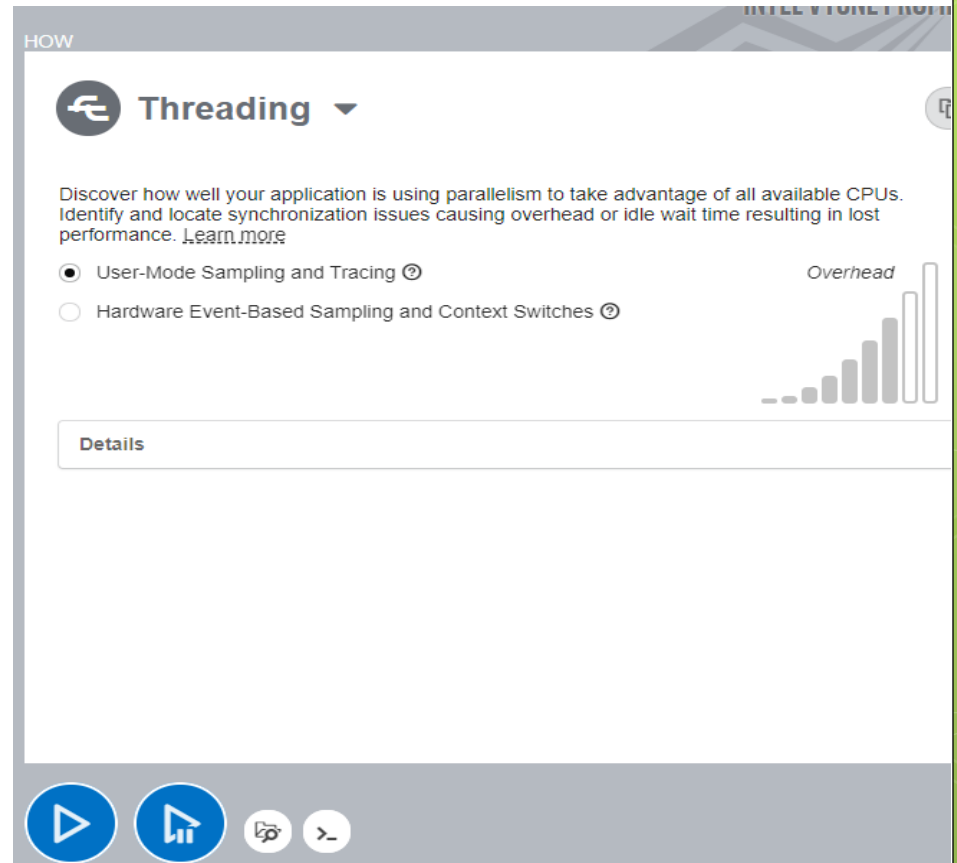


[Справка по Intel Inspector](#)

Intel VTune Profiler

Переходим в Средства/Intel VTune Profiler/ Open VTune Profiler. Далее Configure Analysis. Вместо Hotspots выбираем Threading. Запускаем анализ.

▶ Configure Analysis...



[Справка по Intel VTune Profiler](#)