**ФГБОУ ВО   
Уфимский университет науки и технологий**

**Кафедра ВМиК**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Разработка и отладка OpenMP-программы вычисления суммы ряда

**ОТЧЕТ**

**по лабораторной работе**

**по** Технологиям параллельного программирования

(*наименование дисциплины*)

|  |
| --- |
| Лабораторная работа 2 (Вар. 25) |
| (обозначение документа) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа |  |  | Фамилия, И., О. | Подпись | Дата | Оценка |
| МО-325Б |  |
|  |  |
| Студент | | | Шарыгин М.С. |  |  |  |
| Преподаватель | | | Спеле В.В. |  |  |  |
| Принял | | |  |  |  |  |

**Уфа 2025 г****.**

Содержание

[1 Цель работы 3](#_Toc193368106)

[2 Практическая часть 4](#_Toc193368107)

[2.1 Подбор лучшего «N» с лучшим ключом 4](#_Toc193368108)

[2.2 Оценка эффективности векторизации 4](#_Toc193368109)

[2.3 Распараллеливание последовательной программы 5](#_Toc193368110)

[2.4 Анализ производительности 5](#_Toc193368111)

[2.5 Ускорение и эффективность 7](#_Toc193368112)

[3 Вывод 9](#_Toc193368113)

[4 Приложение 10](#_Toc193368114)

[4.1 Код программы 10](#_Toc193368115)

# 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».

# Практическая часть

## Подбор лучшего «N» с лучшим ключом

В первой лабораторной работе наилучший результат был получен при использовании «Intel Clang/LLVM C/C++» с ключами «/O3» и «/QxHost». Теперь подберем параметр «N» такой, чтобы выполнение программы занимало примерно 30 секунд. Методом подбора получили, что «N» равен 15000000000 (рисунок 2.1).



Рисунок 2.1 – Результат при подобранном «N»

## Оценка эффективности векторизации

При небольшом «N» эффективность векторизации составляет 62%, но при его [«N»] увеличении векторизация не происходит, как показано на рисунке 2.2 и рисунке 2.3.

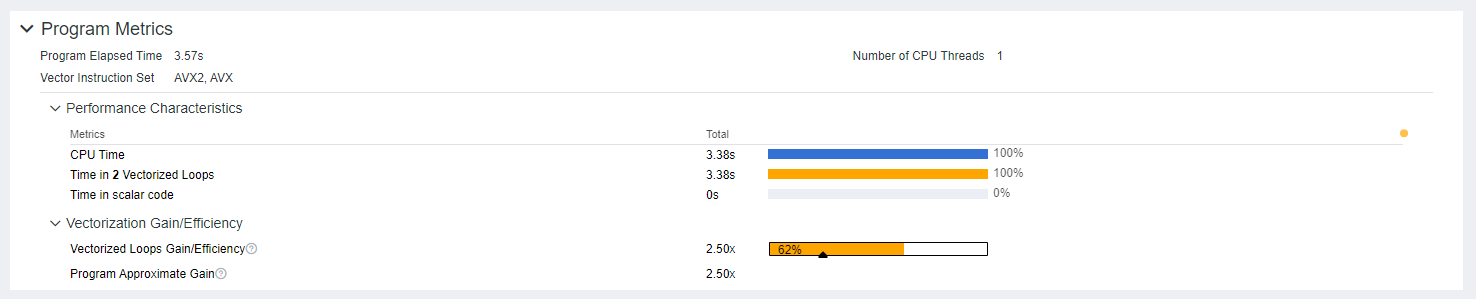


Рисунок 2.2 – Векторизация при небольшом «N»

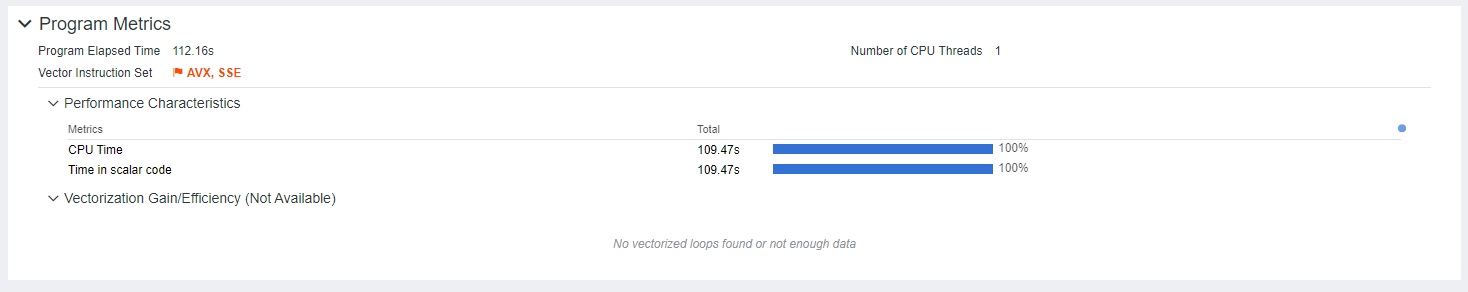


Рисунок 2.3 – Векторизация при большом «N»

## Распараллеливание последовательной программы

После добавления в код программы директив «#pragma omp parallel» и «#pragma omp for» вычисления стали некорректными (рисунок 2.4).

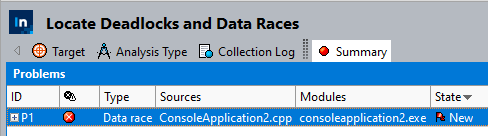


Рисунок 2.4 – Найденные проблемы

Для исправления проблемы добавим в код «reduction(+:sum)». После применения команды переменная «sum» создается для каждого потока, вследствие чего каждый поток обращается к отдельной переменной, что исключает проблему одновременного обращения к одной переменной. На рисунке 2.5 показано, что проблемы была исправлена.

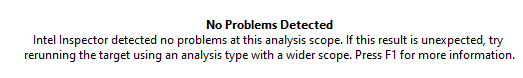


Рисунок 2.5 – Исправление проблемы

## Анализ производительности

Теперь снимем профиль параллельной программы с помощью инструмента «Intel VTune Profiler» (рисунок 2.6, рисунок 2.7).

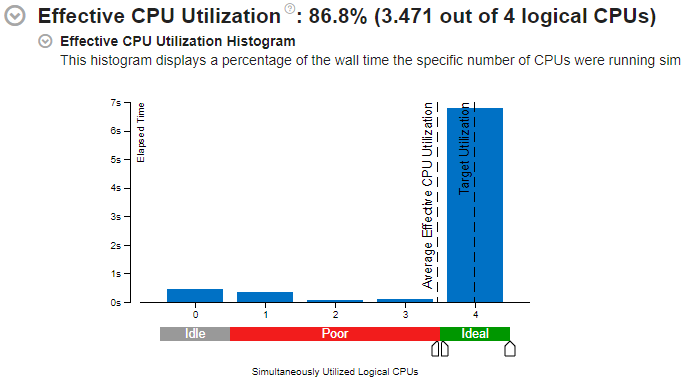


Рисунок 2.6 – Профиль параллельной программы

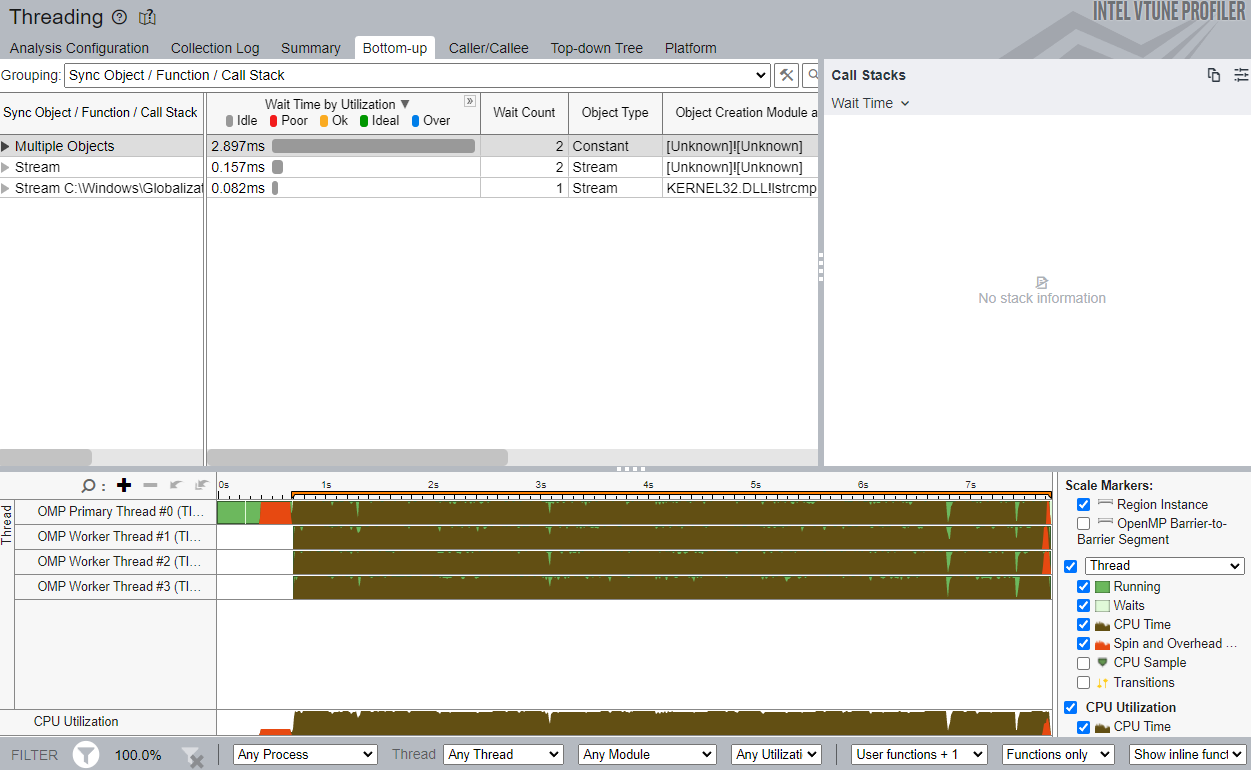


Рисунок 2.7 – Детальная загрузка каждого ядра

Используем спецификатор «schedule» директивы «#pragma omp for», чтобы оптимизировать загрузку «CPU» и довести эффективность до 90% (рисунок 2.8, рисунок 2.9).

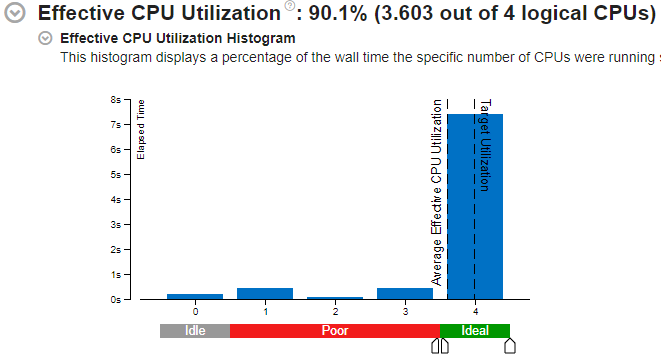


Рисунок 2.8 - Профиль параллельной программы с использованием «schedule»

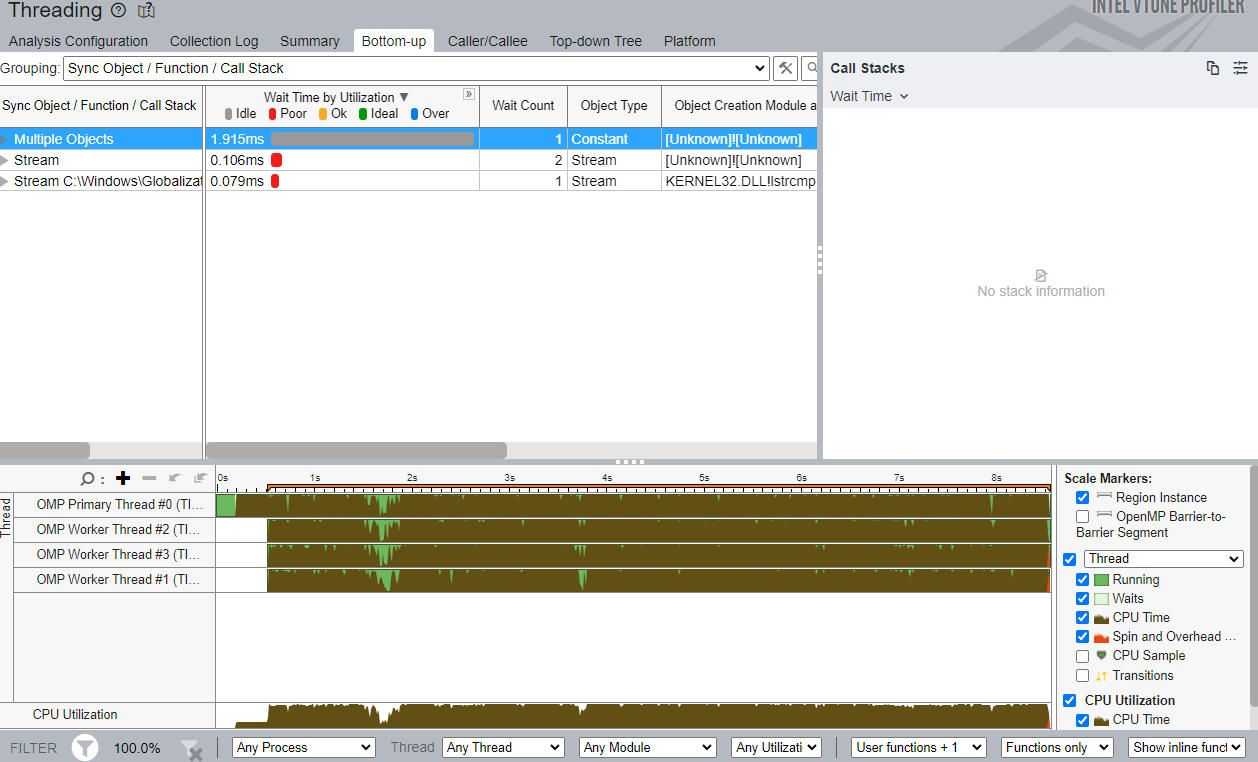


Рисунок 2.9 – Детальная загрузка каждого ядра с использованием «schedule»

## Ускорение и эффективность

В конце рассчитаем ускорение и эффективность и получим результат, приведенный на рисунке 2.10.

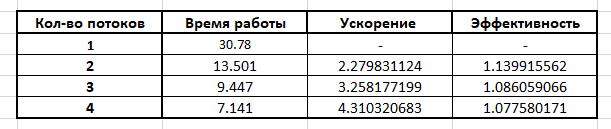


Рисунок 2.10 – Расчет ускорения и эффективности

# Вывод

В ходе лабораторной работы мы на примере задачи вычисления суммы ряда научились разрабатывать простейшие параллельные программы средствами «OpenMP», а также использовать инструменты для проверки корректности программ Intel Inspector и для профилирования производительности программ «Intel VTune Profiler».

# Приложение

## Код программы

#include <iostream>

#include <omp.h>

using namespace std;

int main() {

long long N = 15000000000;

double sum = 0;

double start\_time = clock();

#pragma omp parallel reduction(+:sum) num\_threads(4)

{

#pragma omp for schedule(guided, 1000)

for (long long n = 1; n <= N; n += 2) {

sum += -1.0 / (n - log(n));

sum += 1.0 / (n + 1. - log(n + 1.));

}

}

double end\_time = clock();

cout << "SUM = " << sum << "\n";

cout << "Time = " << (end\_time - start\_time) / CLK\_TCK << "\n\n\n";

return 0;

}