Отчет по лабораторной работе №4

Сайфуллин Джамиль ИВТ-11М

Технические характеристики

OS: Windows 10

Vs: Visual Studio 2017 v15.9.18

Ips: v2019

Железо: 2 ядра физических, 4 логических 3.4ГГц

# 

# Задание 1-2

Разберите программу, представленную в файле task\_for\_lecture5.cpp. В программе создается 2 потока, каждый из которых вычисляет средние значения матрицы, один по строкам исходной матрицы ***matrix***, а другой - по столбцам.

Запустим программу. Результат программы представлен на рисунке 1. Как можно заметить, программа работает корректно.

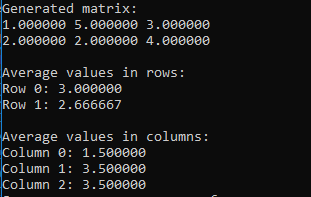


Рисунок 1 Результат работы программы task\_for\_lecture5

Проанализировав программу, я пришел к выводу, что улучшить ее производительность вряд ли получится. Она итак весьма хорошо оптимизирована, за исключением утечек памяти, но об этом позже. Дабы убедиться в этом, я решил попробовать изменить программу 3 способами. 1 вариант изменения – смена динамического выделения памяти для массивов на векторы. Такой вариант удобнее использовать, но вероятно не даст увеличение производительности, т.к. простые массивы быстрее. 2 вариант – убрать std потоки и заменить их на cilk\_for. 3 вариант совместить cilk\_for и std::threads, однако не уверен как будет работать создание потока внутри другого потока.

Результат представлен на рисунке 2.

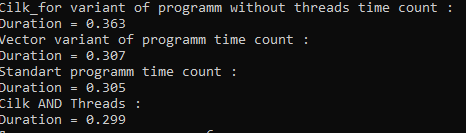


Рисунок 2 Результат работы 4-х вариантов программы для матрицы 5000х5000

Как видно из рисунка 2, стандартный вариант программы работает примерно с такой же скоростью, что и векторный. Cilk\_for без использование std::threads отстает. Наибыстрейшим, к моему удивлению, становится вариант совместного использования cilk\_for и std::threads. Такой вариант и будет улучшением базовой программы. Остается только избавиться от утечек данных, но об этом позже.

# Задание 3-4

Определим, есть ли в нашей программе проблемы с гонками данных, утечками памяти и дедлоками. Запусти Intel Parallel Inspector. Результат представлен на рисунках 3 и 4. Как можно заметить, у нас имеются утечки памяти. Это связано с тем, что мы выделяем динамически память, а потом ее не освобождаем.

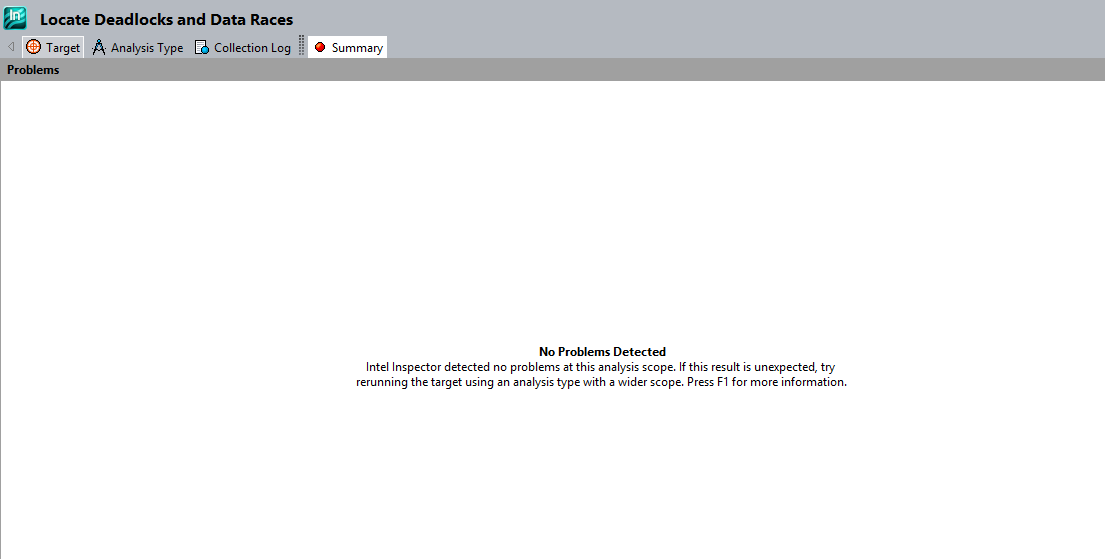


Рисунок 3 результат работы Intel Parallel Inspector по поиску гонок данных и дедлоков

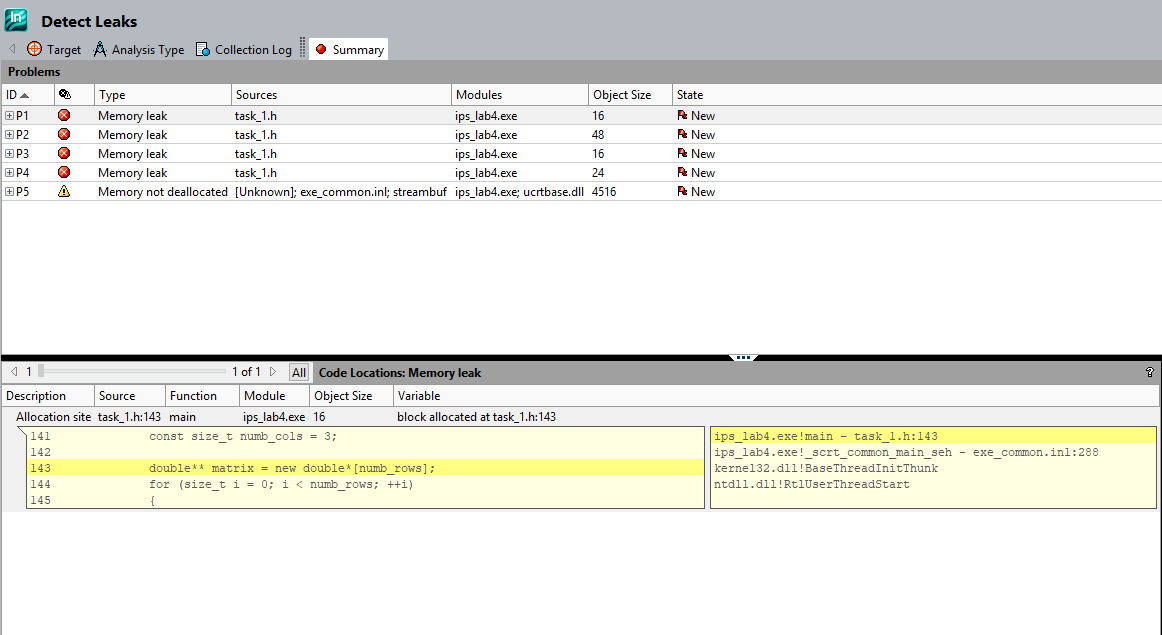


Рисунок 4 результат работы Intel Parallel Inspector по поиску утечек данных

Изменим программу, добавив в нее освобождение выделенной памяти, и запустим Intel Parallel Inspector еще раз. Результат на рисунке 5. Как видим, проблемы с утечкой памяти устранились.

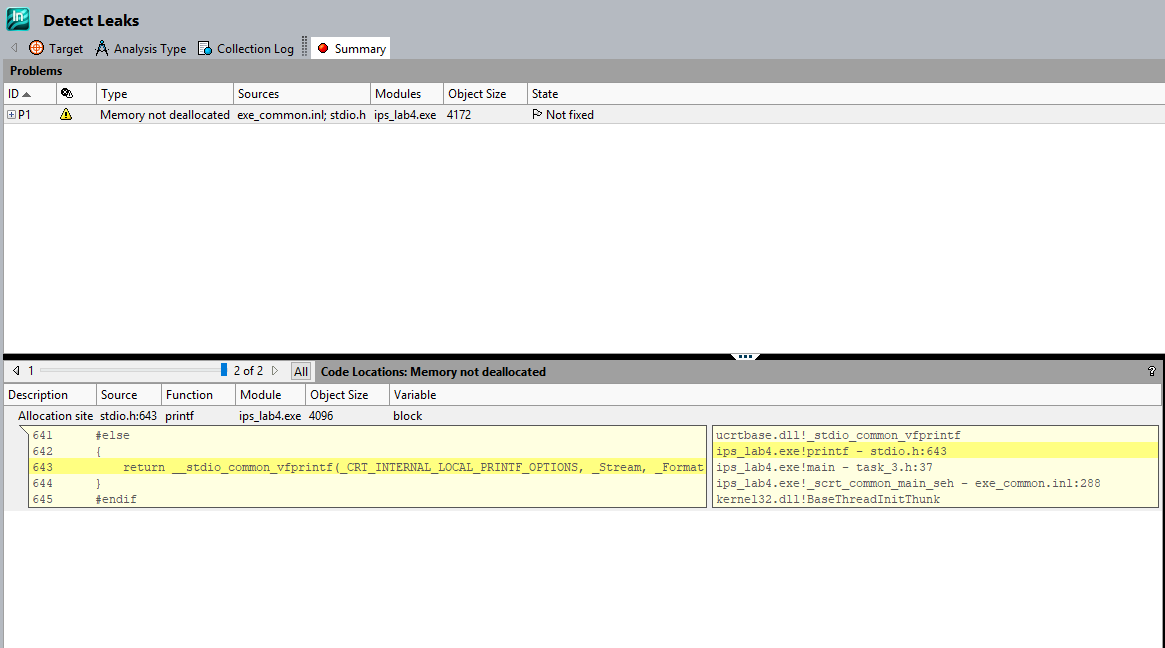


Рисунок 5 результат работы Intel Parallel Inspector по поиску утечек данных

В результат мы получили более производительную программу и избавились от проблем с утечками памяти.