Отчет по лабораторной работе №5

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

Запустим исходную программу без добавления параллелизма. Результат представлен на рисунке 1.



Рисунок 1 Результат работы программы

Добавили параллелизм в исходную программу. Для решения проблемы гонок в цикле добавим сокращение reducer для переменных x и S. Результат работы представлен на рисунке 2.



Рисунок 2 Результат работы программы

Запустим ***Amplifier XE*** и посмотрим, сколько задействовано потоков и сколько на них затрачивается времени. В моей версии IPS отсутствует ***Concurrency Analysis.*** На рисунке 3 можно это увидеть список доступных анализов.

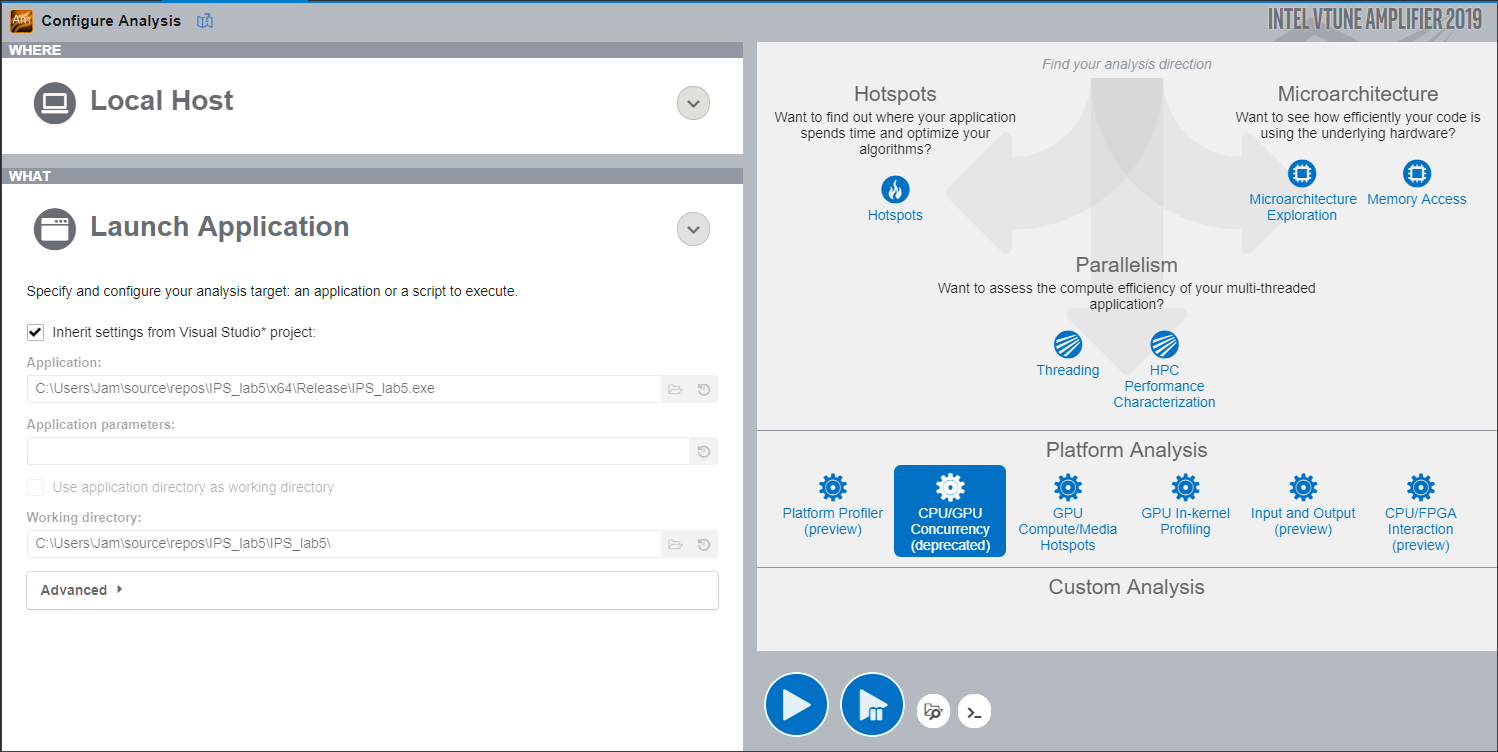


Рисунок 3 Список доступных анализов

Полистав документацию на ***Amplifier XE***, я не нашел упоминаний ***Concurrency Analysis,*** однако в 2017 версия документации он есть. Вероятно, он был убран, начиная с 2019 версии. Поэтому вместо него используется обычный ***Hotspot Analysis***. Результаты анализа на рисунках 4 и 5.

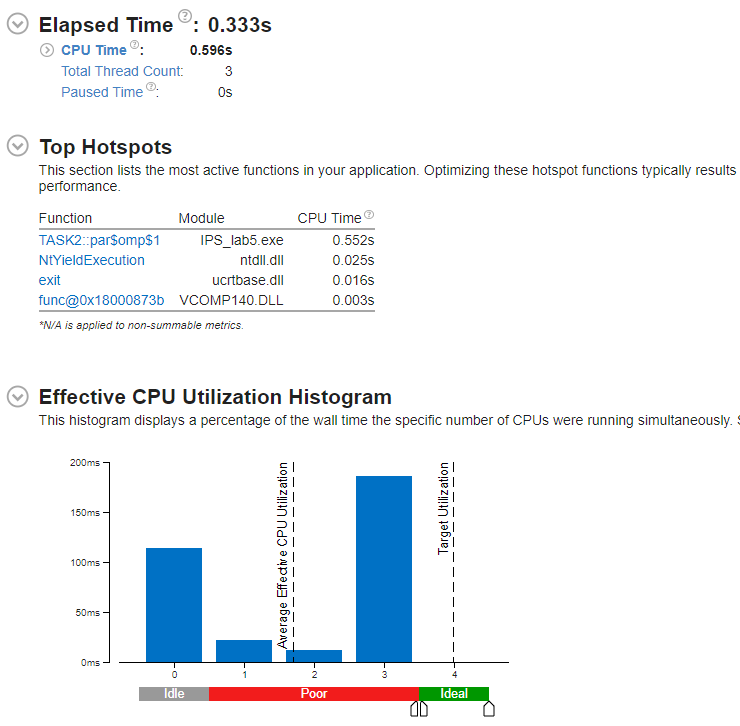


Рисунок 4 Результат работы Vtune Amplifier XE

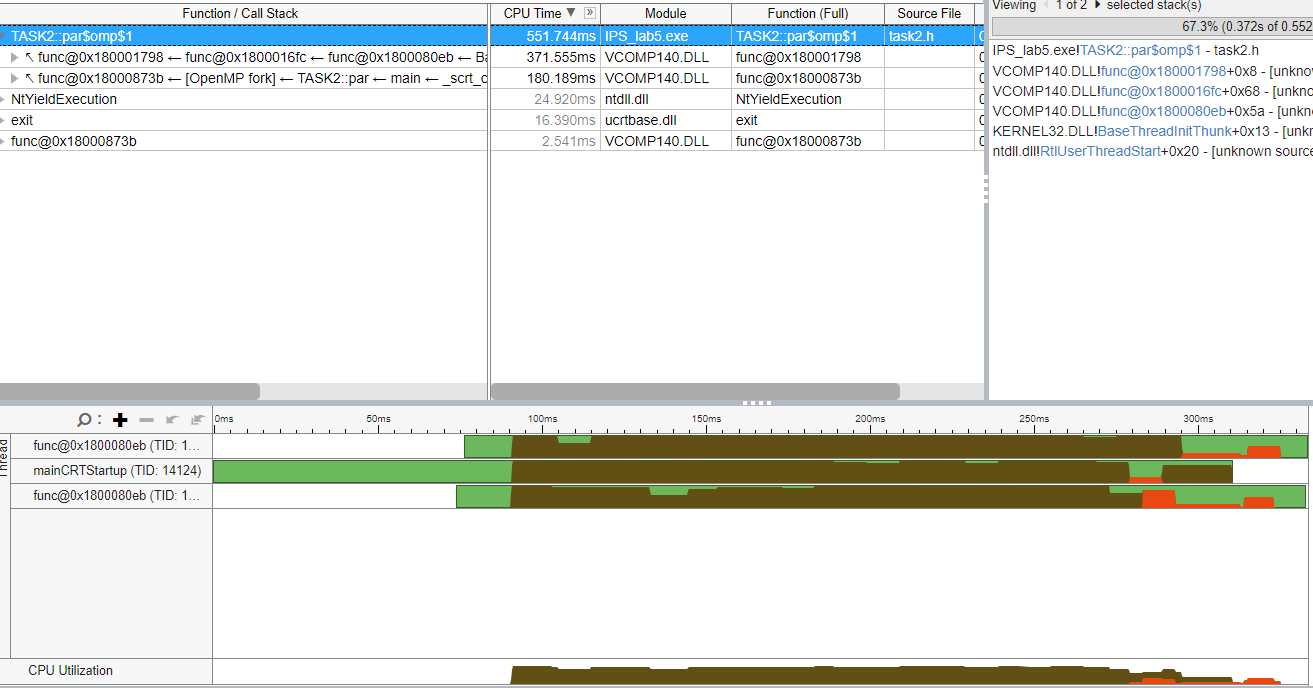


Рисунок 5 Результат работы Vtune Amplifier XE (вкладка Bottom-up)

Глядя на график мы видим, что большую часть времени ( ~210ms ) вычислители работают, остальное время (120ms) затрачивается на запуск и тд. Даже при большом времени вне вычислений мы все равно получаем выигрыш по отношению к последовательному варианту программы почти на 200ms.

## Задание 3

В функции ***par()***в цикле по ***i*** от ***0*** до ***num***после выражения ***S = S + 4.0 / (1.0 + x\*x);*** добавьте следующие 2 строки кода ***#pragma omp atomic***, ***inc++;***.

Результат работы программы представлен на рисунке 6.



Рисунок 6 Результат работы программы

Время работы программы существенно увеличилось. Запустим ***Amplifier XE***. Результат работы на рисунках 7 и 8.

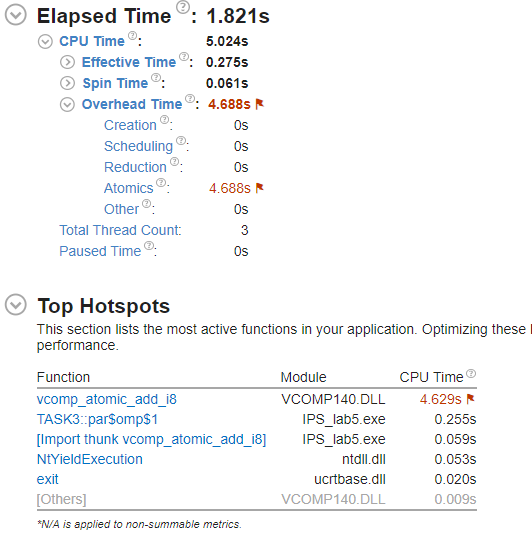


Рисунок 7 Результат работы Vtune Amplifier XE

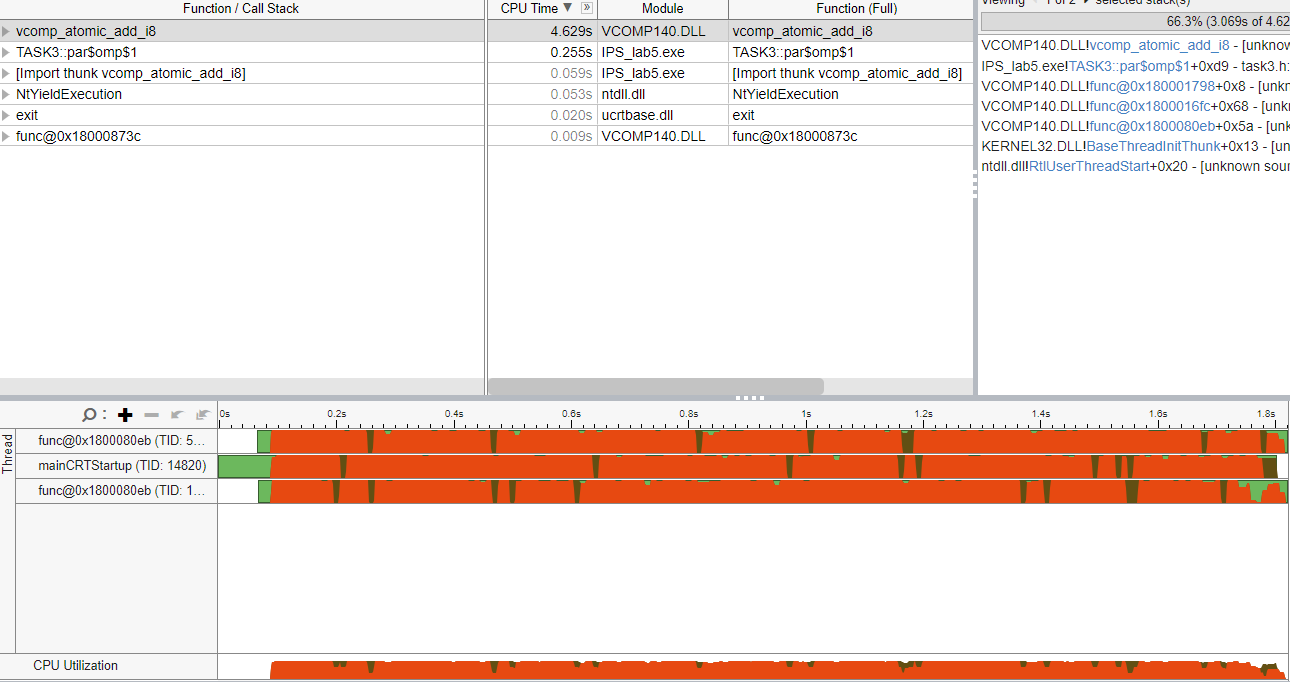


Рисунок 8 Результат работы Vtune Amplifier XE (вкладка Bottom-up)

## Как можно видеть, много времени ушло на атомарные операции. Ожидание потоком завершение атомарной операции и вызвало такое увеличение по времени. Поэтому большую часть времени вычислители проставивают.

## Задание 4

 Замените строку ***#pragma omp atomic*** строкой ***#pragma omp critical***. Результат работы такой программы представлен на рисунке 9.



Рисунок 9 Результат работы программы

Программа стала еще медленнее. Запустим Amplifier XE и посмотрим, что заняло столько времени. Результат представлен на рисунках 10 и 11.

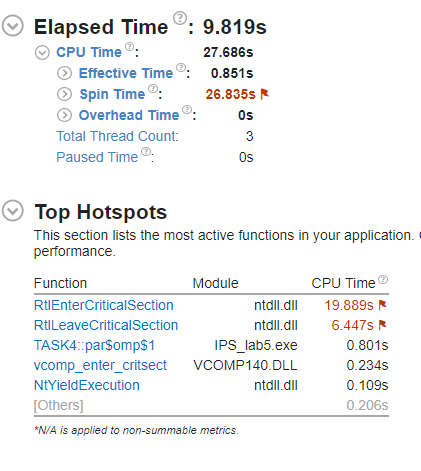


Рисунок 10 Результат работы Vtune Amplifier XE

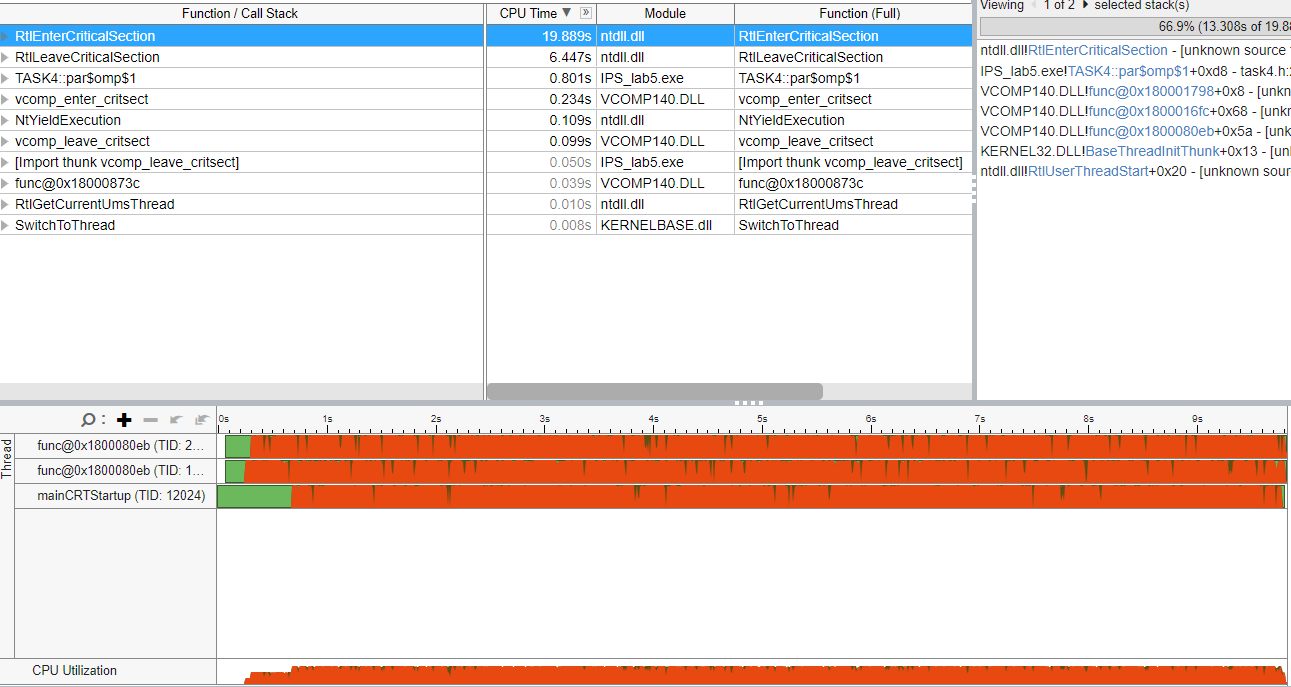


Рисунок 11 Результат работы Vtune Amplifier XE (вкладка Bottom-up)

Такое увеличение времени работы программы связано с тем, что операции инкрементации переменной Inc может выполняться одновременно лишь в одном потоке. Это вызвано строкой #pragma omp crtitical. Поэтому большую часть времени вычислители простаивают.

## Задание 5

Заменим #pragma omp critical на конструкцию с использованием замков. Результат представлен на рисунке 12.



Рисунок 12 Результат работы программы

Как можно заметить, программа стала работать еще медленнее. Запустим **Amplifier XE** и посмотрим, что замедляет программу. Результат представлен на рисунках 13 и 14.

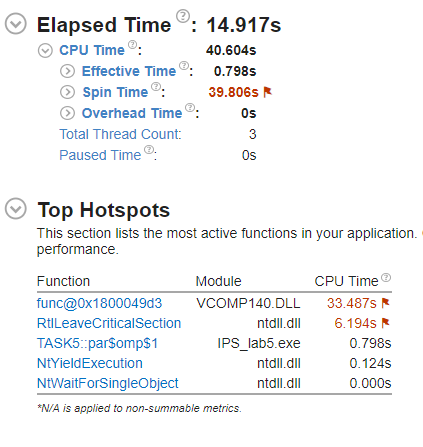


Рисунок 13 Результат работы Vtune Amplifier XE

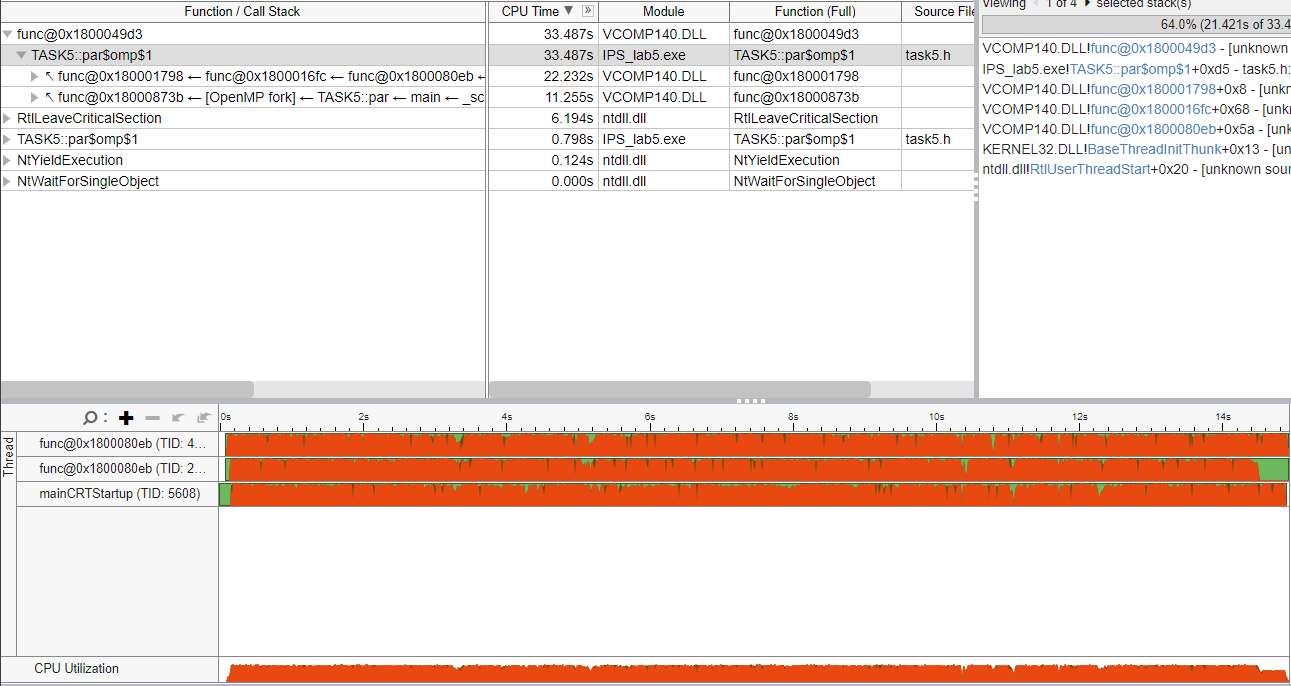


Рисунок 14 Результат работы Vtune Amplifier XE (вкладка Bottom-up)

Использование замков схоже с использованием #pragma omp critical. Замок блокирует выполнение кода в пределах ограничения замка для всех потоков, кроме того, что активировал замок. Данный вариант оказался медленнее чем #pragma omp critical вероятно потому, что необходимо дополнительное время для установки и снятия замка.

Для оптимизации программы можно добавить сокращение reducer как и для других переменных с гонками данных. Тогда скорость программы снова станет ~300ms. Результат на рисунке 15.



Рисунок 15 Результат работы конечной программы