Отчёт по лабораторной работе №5

Жуков Вадим, ИВТ-12М

Технические характеристики

OС: Windows 10

VS: Visual Studio 2017 v15.9.18

IPS: v2019

Железо: 2 физических ядра, 4 логических, 3.4ГГц

# Задание 1

Разберите последовательную программу по вычислению определенного интеграла [task\_lecture\_7.cpp](https://canvas.instructure.com/courses/1844563/files/87817254/download?wrap=1). Введите в нее параллелизм с помощью OpenMP. Установите количество рабочих процессов равным 3, для этого используйте оператор ***num\_threads(num\_of\_threads)***. Не забудьте настроить в свойствах проекта поддержку стандарта OpenMP: ***Свойства проекта*** → вкладка ***C\C++*** → ***Язык***→  ***Поддержка OpenMP***.

Результат работы изначальной версии программы приведён на рисунке 1.

C:\Users\zhuko\OneDrive\Рабочий стол\2.jpg

Рисунок 1. Результат 1-го варианта программы

Также я установил количество рабочих процессов равное 3 так, как указано в задании, и измерил необходимые показатели. Они представлены в результатах работы задания №2.

# Задание 2

После введения параллелизма запустите программу. На консоли Вы увидите подсчитанное значение и время выполнения программы. Сделайте скрин консоли, сохраните его, назвав соответствующим образом. Запустите ***Concurrency Analysis*** инструмента ***Amplifier XE*** из панели инструментов ***Visual Studio***. Во вкладке ***Summary***отчета Вы должны увидеть цикл функции ***par()***, использующий наибольшее время CPU. Нажав на него, Вы перейдете во вкладку ***Bottom-up***. Оцените загруженность вычислителей, представленную на графике ниже. Сделайте скрин вкладки ***Bottom-up***, сохраните его, назвав соответствующим образом. Текущую версию программы и скрины добавьте в коммит и загрузите в ***GitHub***.

Добавил в исходную программу параллелизм, для решения проблемы гонок данных добавил reducer для x и S. Результат работы приведён на рисунке 2.

C:\Users\zhuko\OneDrive\Рабочий стол\1.jpg

Рисунок 2. Результат работы 1-й программы с OpenMP

В используемой версии IPS, к сожалению, отсутствует Concurrency Analysis, поэтому вместо него я решил использовать Hotspot Analysis. Результаты приведены на рисунке 3, ***Summary***, и рисунке 4, скрин вкладки ***Bottom-Up***. Также я добавил reduction, чтобы избежать гонки данных. В сравнении со следующими заданиями, результаты, представленные в этом задании, показывают, что программа работает очень эффективно.

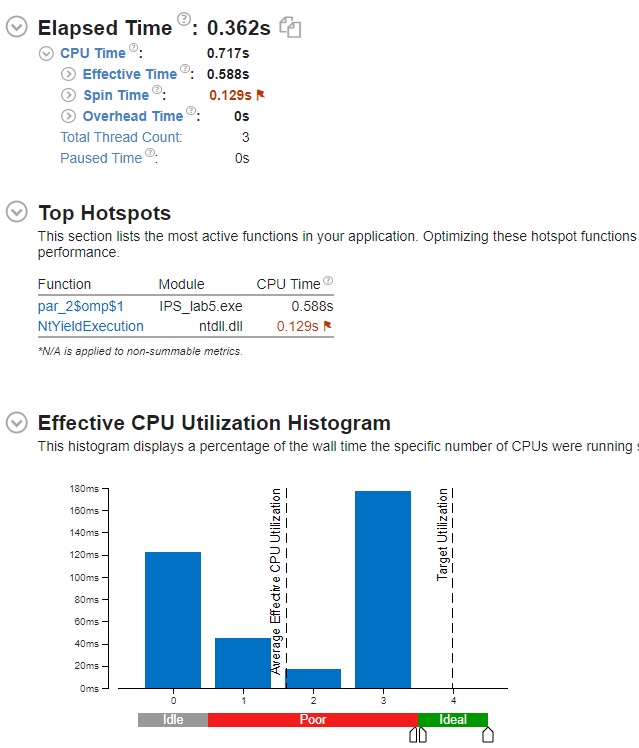


Рисунок 3. Summary

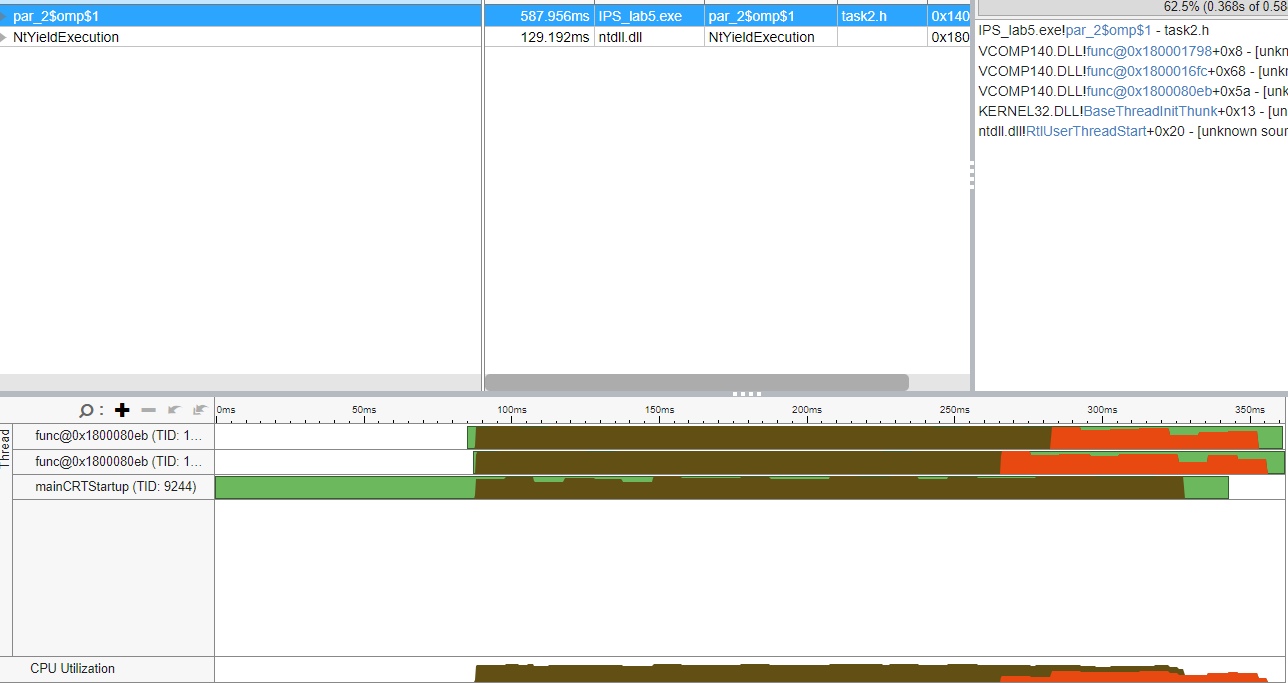


Рисунок 4. Вкладка Bottom-Up

# 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++;***. Пересоберите решение. Запустите программу, сделайте скрин консоли, сохраните его. Далее запустите ***Concurrency Analysis***. Перейдя во вкладку ***Summary***отчета, Вы увидите, что теперь наибольшее время затрачивается на выполнение новых двух добавленных строк кода. Чем Вы объясните такие изменения?

Далее, нажав по соответствующей строке отчета ***Summary***, перейдите во вкладку ***Bottom-up***. Проанализируйте загруженность вычислителей в данном случае. Сохраните скрин вкладки ***Bottom-up***. Текущую версию программы и скрины добавьте в коммит и загрузите в ***GitHub***.

Результат работы сильно ухудшился, потому что увеличилось время выполнения. Это видно на рисунке 5.

C:\Users\zhuko\OneDrive\Рабочий стол\5.jpg

Рисунок 5. Результат работы после добавления atomic

Результаты работы Hotspot Analysis представлены на рисунке 6, ***Summary***, и рисунке 7, вкладка ***Bottom-Up***. По изменениям в коде и результатам анализа загруженности видно, что такое увеличение по времени вызвало введение ***#pragma omp atomic***. Вычислители ожидают завершения атомарного оператора и в это время простаивают, поэтому время выполнения программы увеличилось.

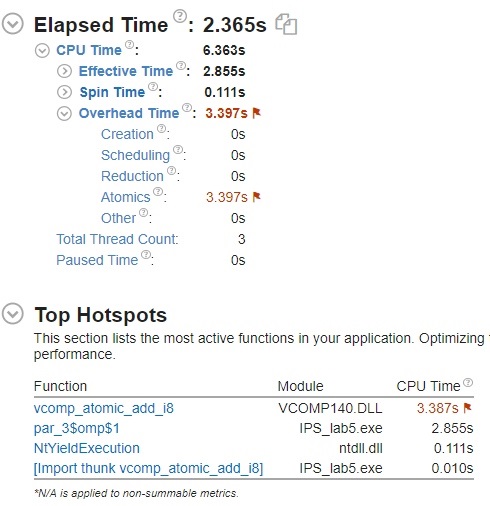


Рисунок 6. Summary

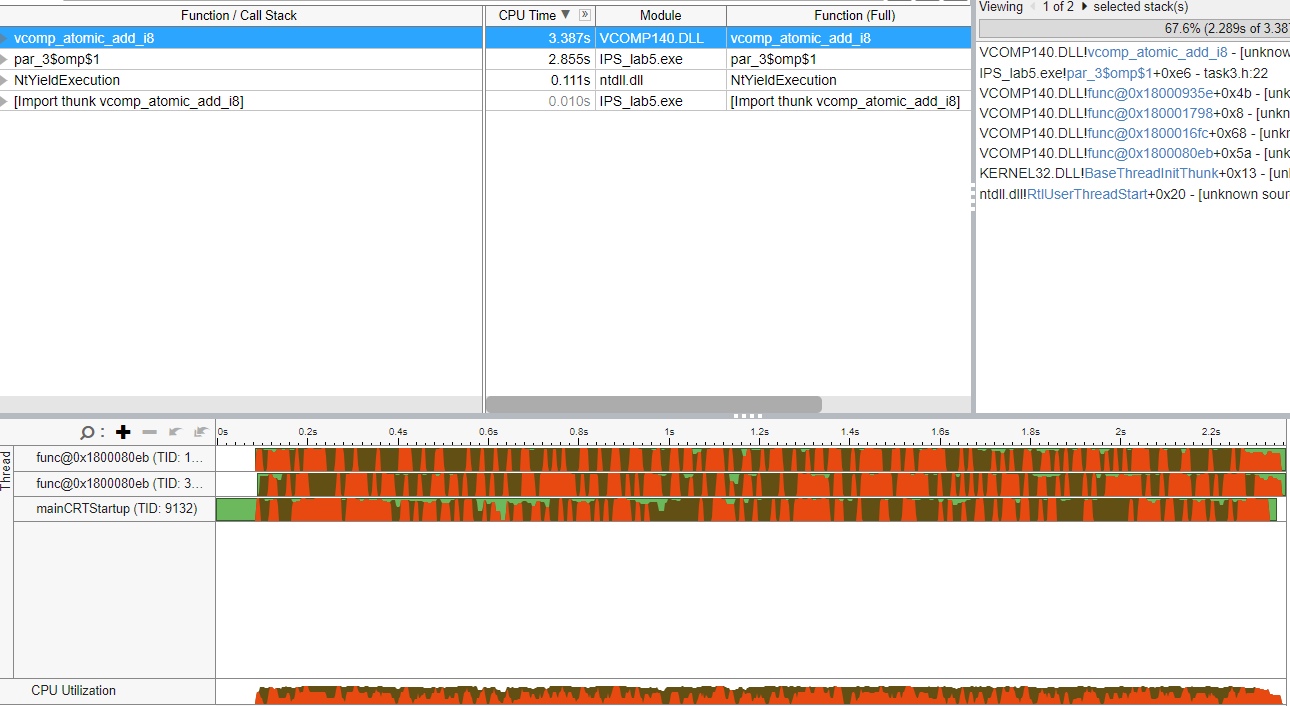


Рисунок 7. Bottom-Up

# Задание 4

Замените строку ***#pragma omp atomic*** строкой ***#pragma omp critical***. Пересоберите решение проекта, запустите программу. Сделайте скрин консоли, где отображено вычисленное значение и время выполнения программы.

Запустите ***Concurrency Analysis***. Перейдя во вкладку ***Summary***отчета Вы увидите изменения по сравнению с предыдущей версией программы. Чем Вы объясните такие изменения?

Далее, нажав по соответствующей строке отчета ***Summary***, перейдите во вкладку ***Bottom-up***. Проанализируйте загруженность вычислителей. сохраните скрин вкладки ***Bottom-up***. Текущую версию программы и скрины добавьте в коммит и загрузите в ***GitHub***.

Программа стала ещё медленнее, это видно на рисунке 8.



Рисунок 8. Результат работы после введения critical

Результаты работы Hotspot Analysis представлены на рисунке 9, ***Summary***, и рисунке 10, вкладка ***Bottom Up***. Вызов ***#pragma omp critical*** делает так, что инкрементация переменной Inc может выполняться лишь в одном потоке. Это блокирует все остальные операции, и вычислители простаивают.

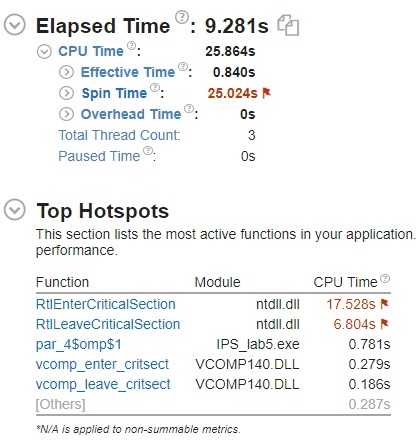


Рисунок 9. Summary

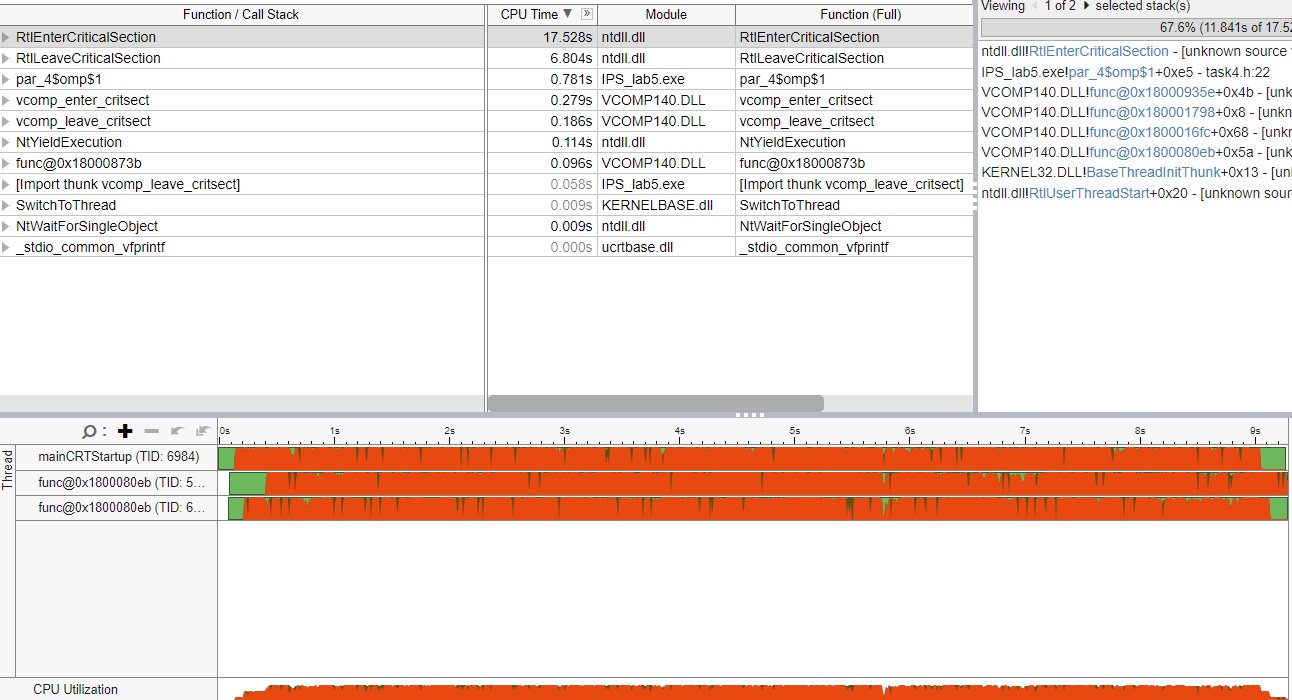


Рисунок 10. Bottom-Up

# Задание 5

Замените строку ***#pragma omp critical***. Введите в программу изменения: перед инкрементом переменной ***inc***необходимо поставить вызов ***omp\_set\_lock (&writelock)***, после него вызов ***omp\_unset\_lock (&writelock)***. Пример правильного использования этих двух функций показан на изображении [init\_lock\_openmp.png](https://canvas.instructure.com/courses/1844563/files/87817214/download?wrap=1). После введенных изменений пересоберите решение, запустите программу. Сделайте скрин консоли. Запустите ***Concurrency Analysis***. Во вкладке ***Summary***отчета Вы должны увидеть, что в данном случае наибольшее время затрачивается на вызов функций ***omp\_set\_lock (&writelock)***и ***omp\_unset\_lock (&writelock)***. Нажав по соответствующей строке отчета ***Summary***, Вы перейдете во вкладку ***Bottom-up***. Проанализируйте загруженность вычислителей. Сделайте скрин вкладки ***Bottom-up***, сохраните его.

Результаты стали ещё хуже, программа выполняется за 13 секунд, это видно на рисунок 11.

C:\Users\zhuko\OneDrive\Рабочий стол\11.jpg

Рисунок 11. Результат работы в задании 5

Результаты работы Hotspot Analysis представлены на рисунке 12, ***Summary***, и на рисунке 13, вкладка ***Bottom-Up***. Замок блокирует всю программу в пределах выполнения замка, поэтому происходит тоже самое, что и с ***#pragma omp critical***. Время получилось ещё больше, потому что lock и unlock также занимают время.

Самым эффективным оказался вариант программы во 2 задании.

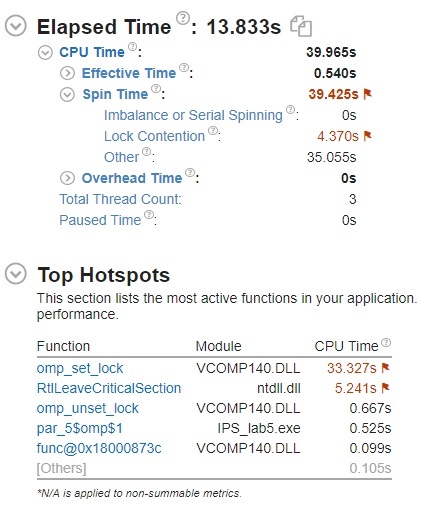


Рисунок 12. Summary

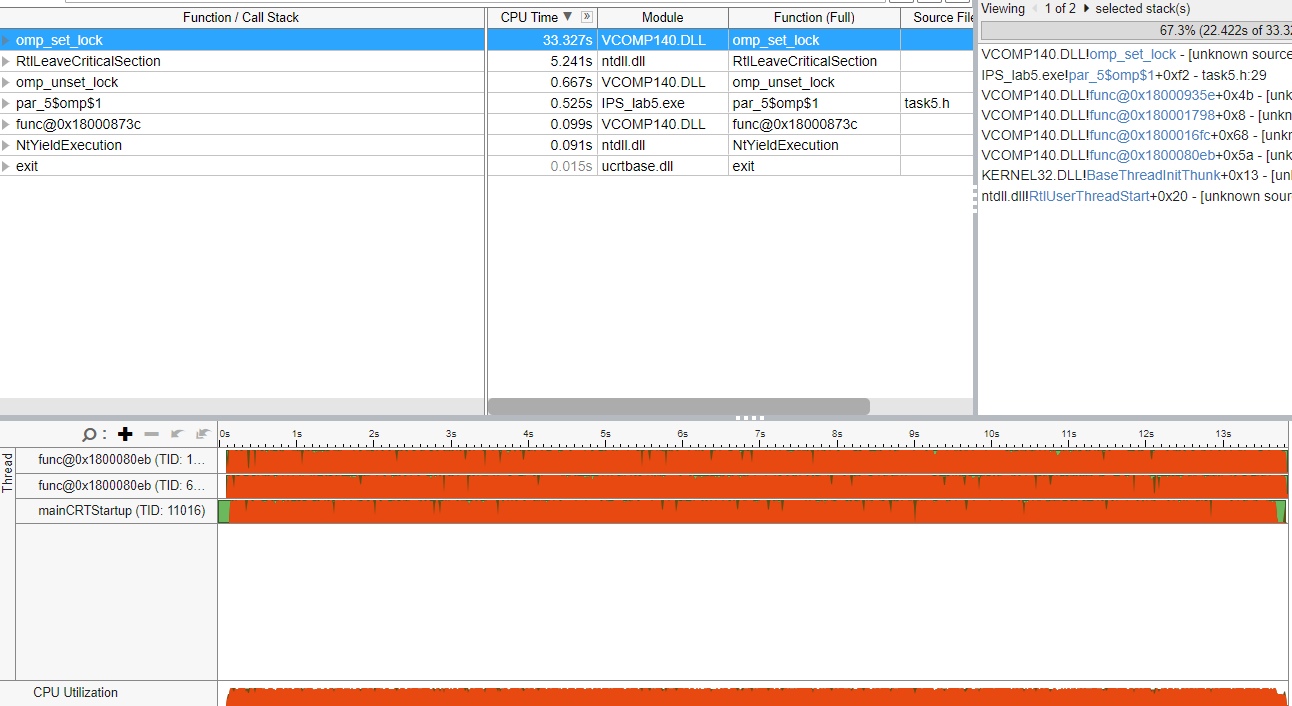


Рисунок 13. Bottom-Up