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**.**

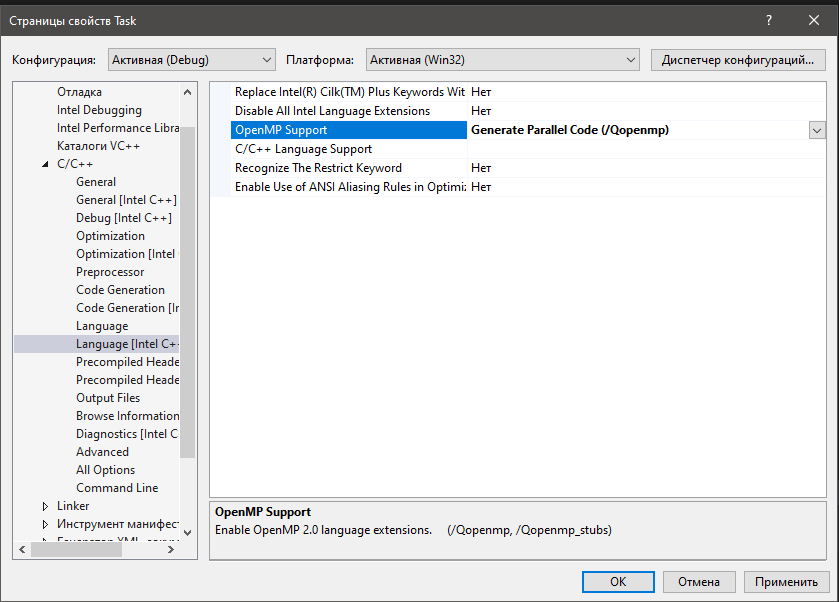


Рис.1 – Настройка поддержки OpenMP

1. **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**.**

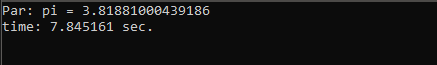


Рис.2 – Результат выполнения программы после добавления num\_threads

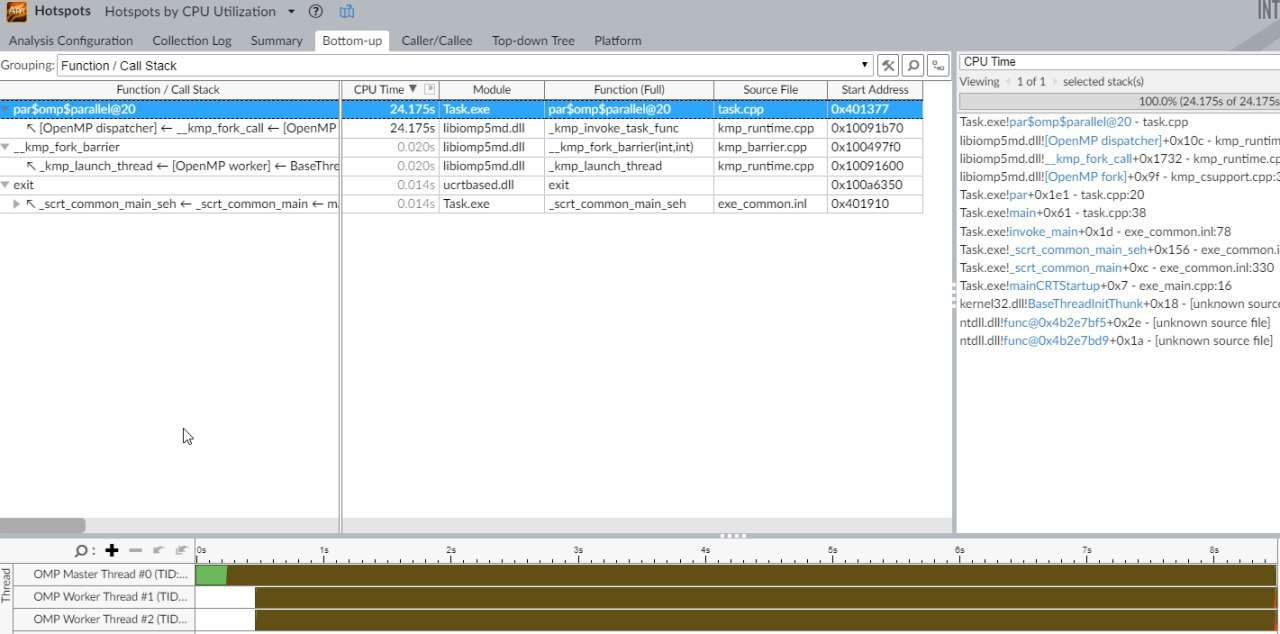


Рис.3 – Оценка работы программы

1. **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?**

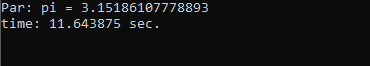


Рис.4 – Результат выполнения программы после добавления #pragma omp atomic и inc++

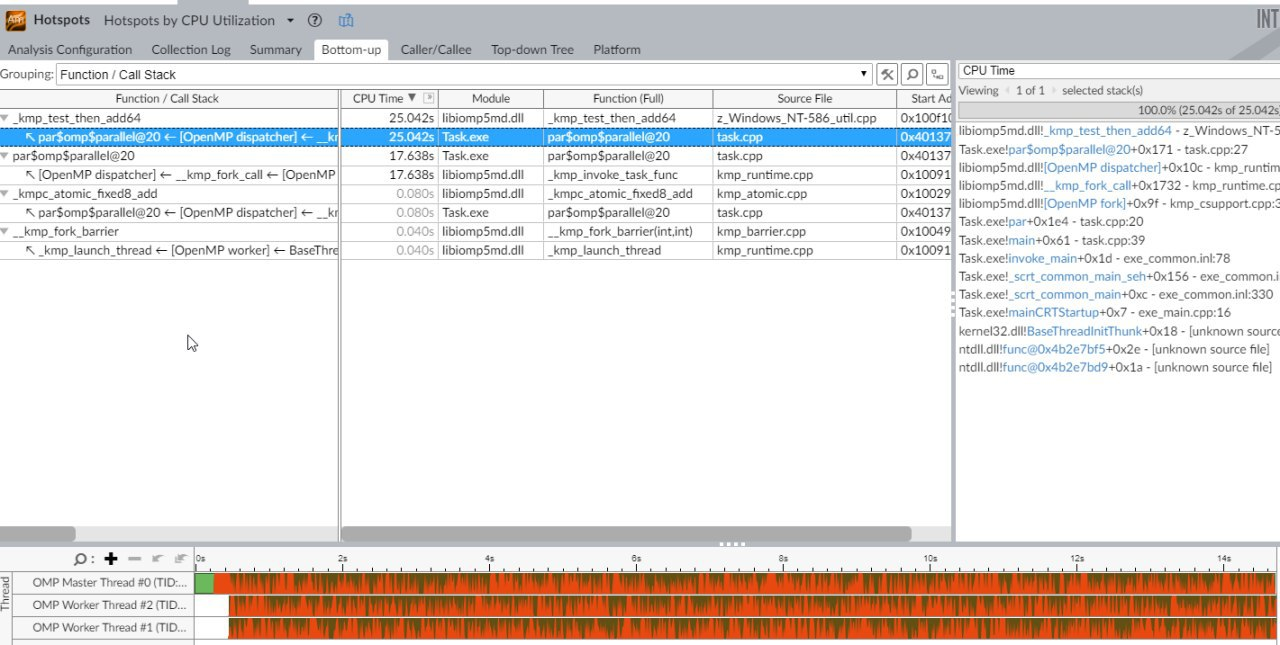


Рис.5 – Оценка работы программы после добавления строк

Объяснение результата: После введения в программу двух строк кода #pragma omp atomic и inc++ время работы программы действительно изменилось, это показано на рисунках 4 и 5. Время работы увеличилось потому, что используется распределение данных между потоками и затрачивается время на работу планировщика потоков OMP.

**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?**



Рис.6 – Результат выполнения программы после замены #pragma omp atomic на #pragma omp critical

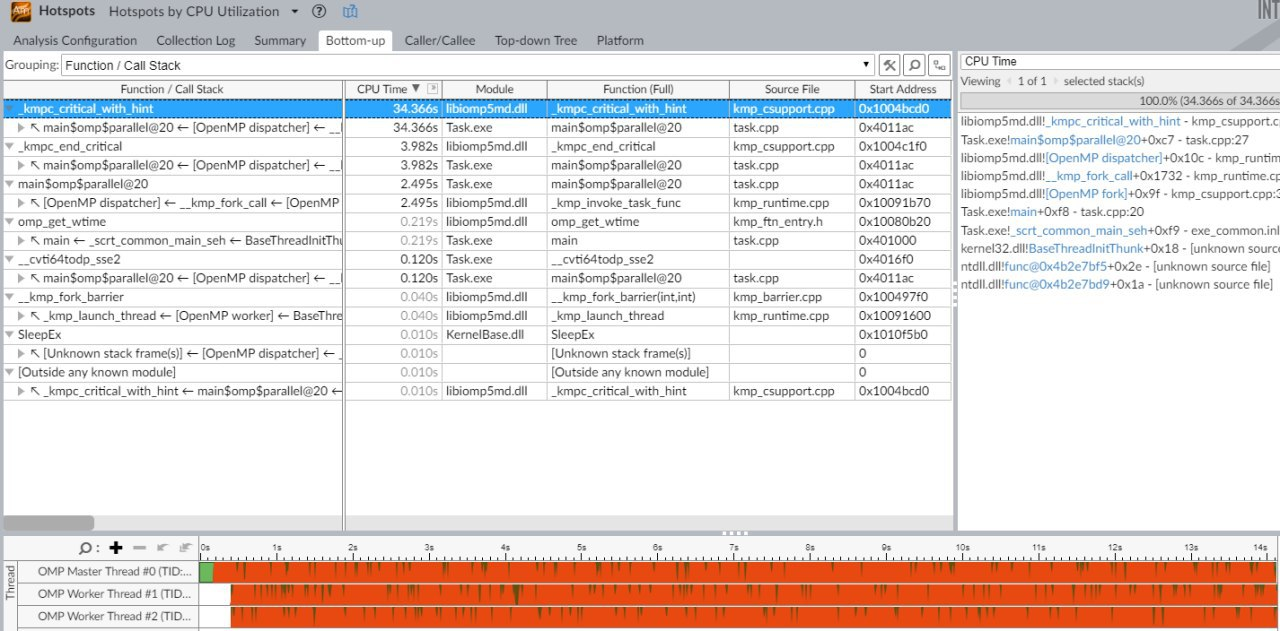


Рис.7 – Оценка работы программы после замены строк

Объяснение результата: После замены строк время работы программы сново изменилось, это показано на рисунках 6 и 7. #pragma omp atomic и #pragma omp critical похожи, однако #pragma omp critical имеет больше накладных расходов, так как не использует аппаратное приращение. Поэтому время работы увеличилось.

**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.**

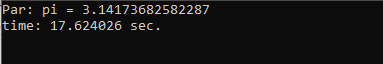


Рис.8 – Результат выполнения программы после замены #pragma omp critical на lock

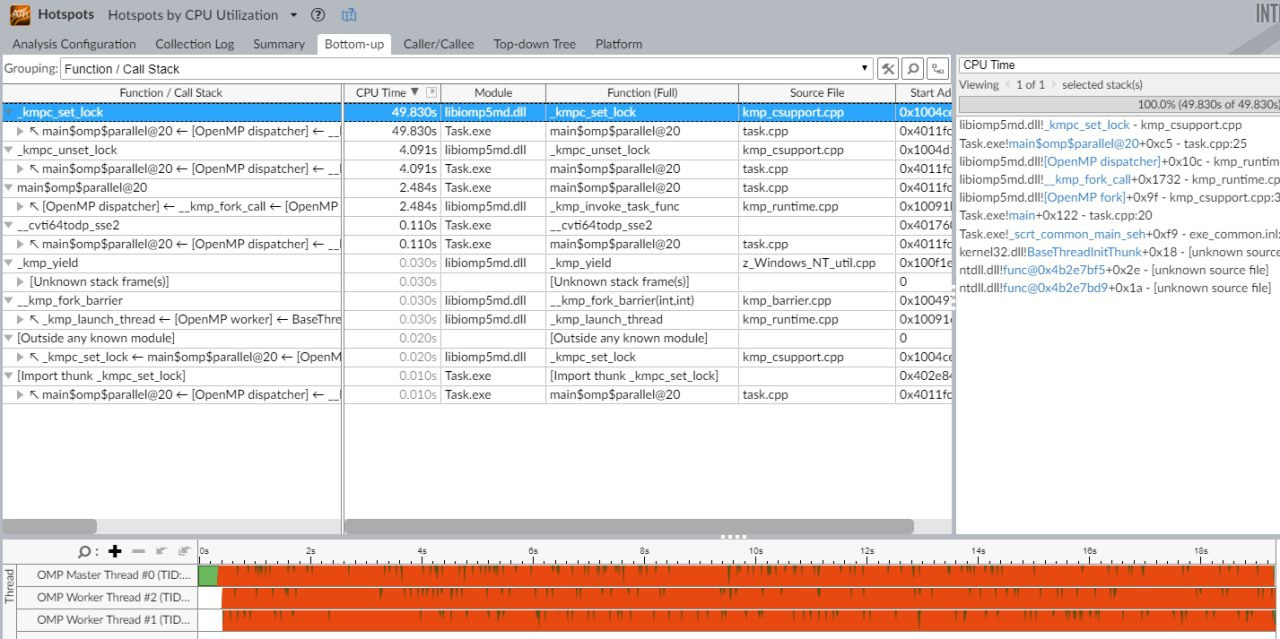


Рис.9 – Оценка работы программы после замены #pragma omp critical на lock

Объяснение результата: После замены critical на конструкции lock время работы программы снова увеличилось (скриншоты 8 и 9). Причиной этому изменению служит тот факт, что механизм замков тратит больше времени на распределение, чем #pragma omp critical.

Вывод: В рамках данной лабораторной работы мы познакомились с OpenMP. Мы попробовали различные способы защиты переменной. Наилучший результат был достигнут с помощью atomic. Однако при использовании atomic необходимо помнить, что он используется только на ограниченном числе операций. В остальных, при использовании более сложных конструкций следует обратиться к critical или конструкции lock.