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**.**

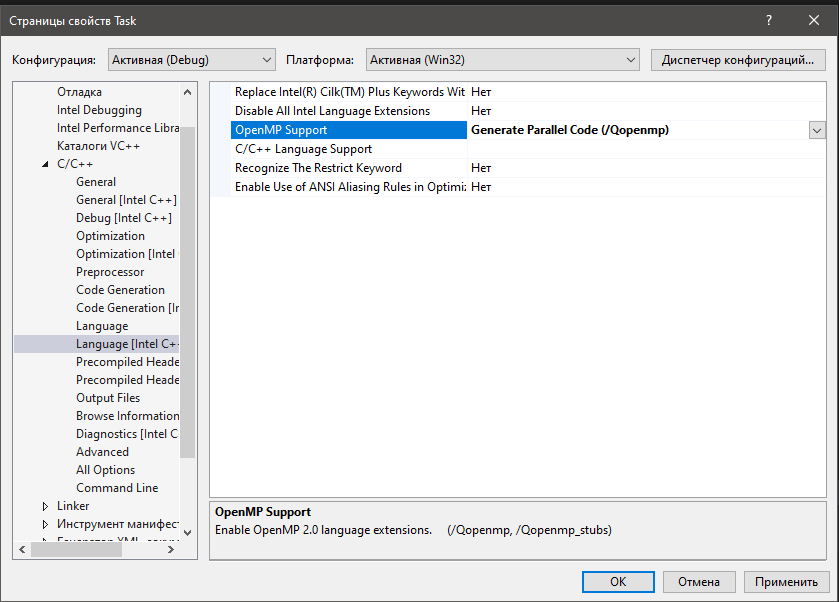


Рис.1 – Настройка поддержки OpenMP

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

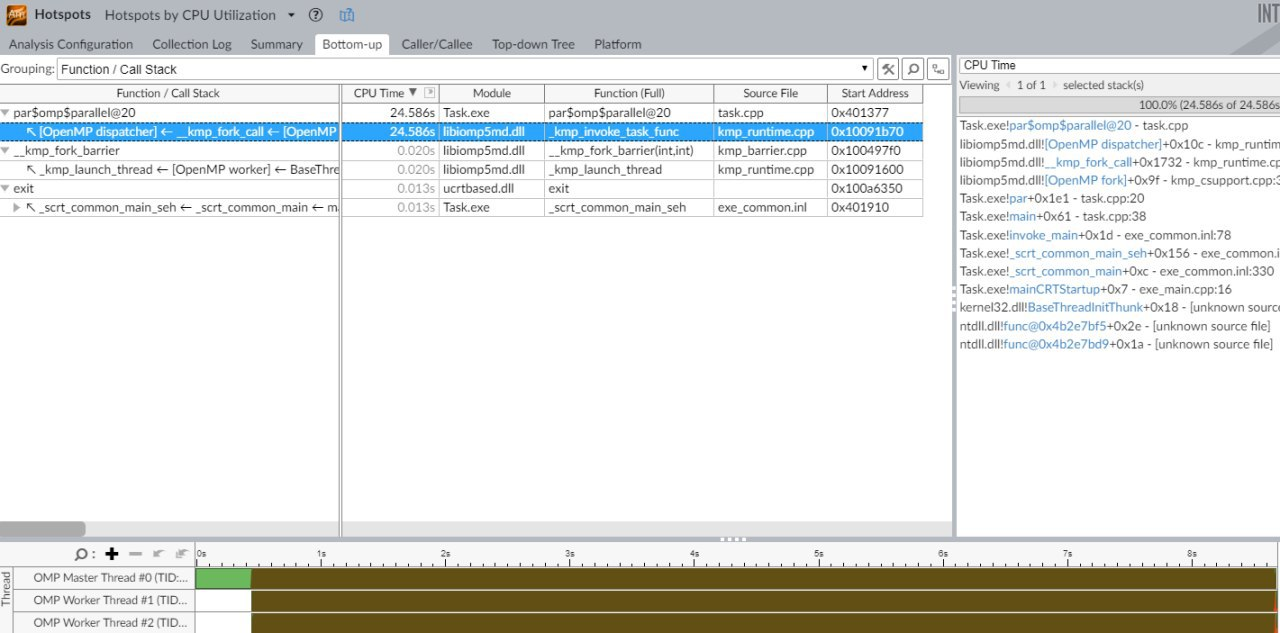


Рис.3 – Результат анализа Vtune Amplifier

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?**

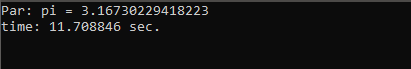


Рис.4 – Результат выполнения программы после добавления #pragma omp atomic и inc++

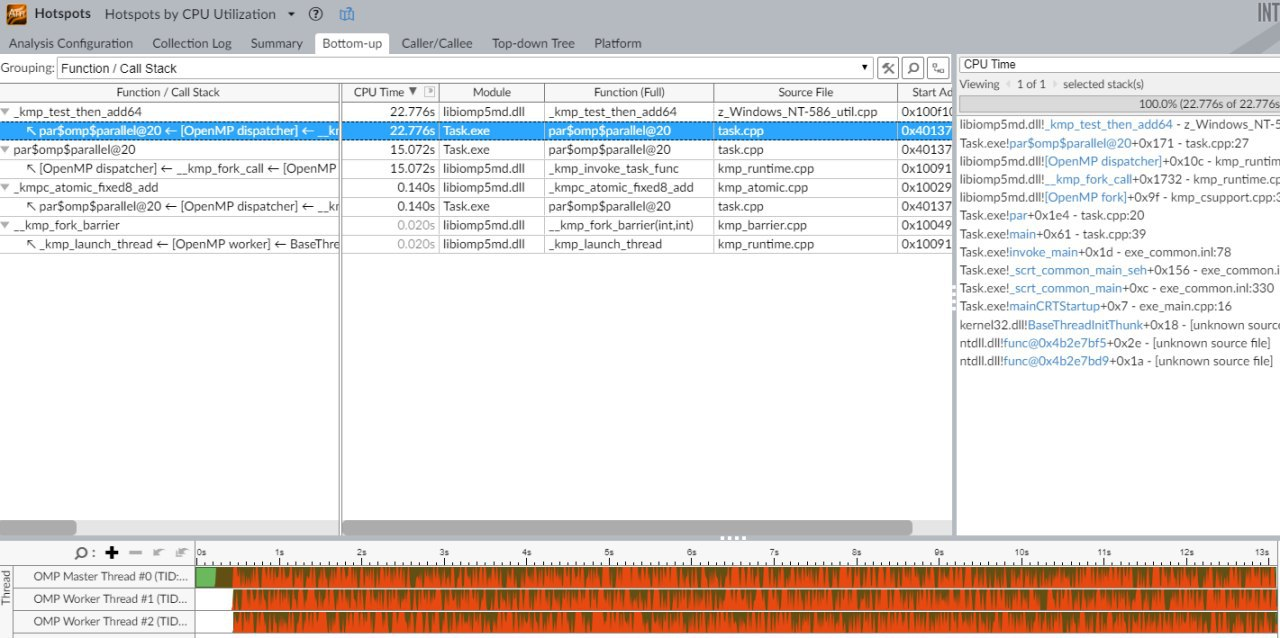


Рис.5 – Результат анализа Vtune Amplifier после внесения изменений

После добавления в программу 2-х строк кода #pragma omp atomic и inc++ время работы программы увеличилось, как показано на скриншотах 4 и 5. Время работы программы изменилось потому, что теперь мы используем распределение данных между потоками, т. о. 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

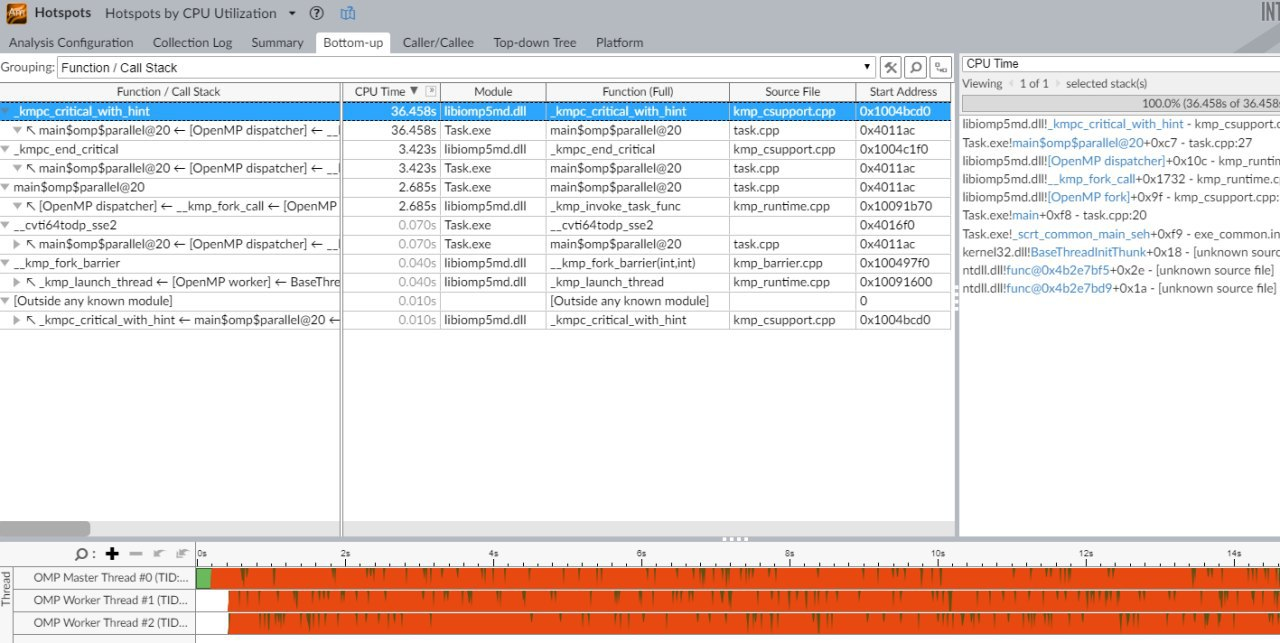


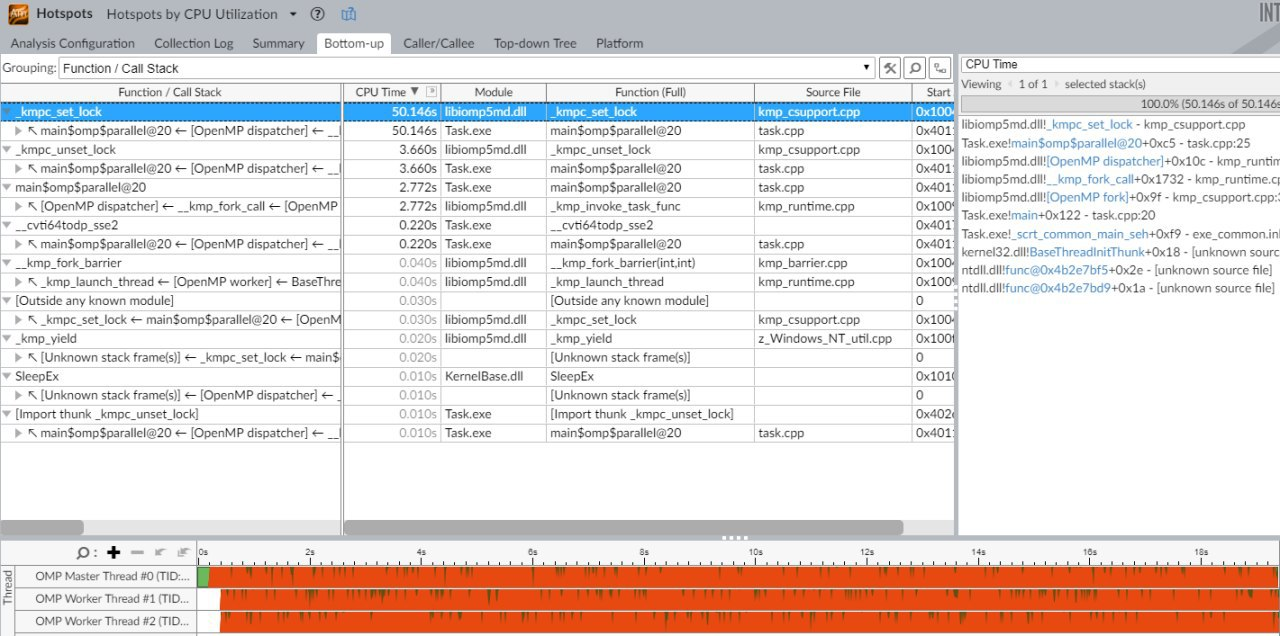
Рис.7 – Результат анализа Vtune Amplifier после замены

После внесенных изменений время работы программы увеличилось. Результат можно объяснить следующим образом: #pragma omp critical имеет больше накладных временных расходов по выполнению в отличии от #pragma omp atomic, так как не использует аппаратное приращение.

**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 – Результат анализа Vtune Amplifier после замены

После введения конструкции lock от OMP время работы программы снова увеличилось (рис. 8 и рис. 9), так как механизм lock тратит больше времени на распределение, critical.

Вывод: В рамках данной лабораторной работы мы познакомились с OpenMP. Мы поняли, что для защиты переменной, используемой параллельно, лучше всего применять atomic для лучшего быстродействия программы. Однако следует помнить, что atomic работает лишь на ограниченном множестве операций, поэтому в более сложных случаях необходимо применять critical или конструкцию lock.