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?

Здесь директива #pragma omp atomic используется , чтобы при выполнении сложения inc++; не происходило гонок данных . Директива #pragma omp atomic показывает , что inc++ является атомарной операцией , а значит может быть напрямую заменена командой процессора . Однако , хоть данное решение и дает корректный результат , оно является крайне неэффективным . Скорость работы данного кода будет ниже, чем скорость последовательного варианта. Во время работы алгоритма постоянно будут возникать блокировки, в результате чего практически вся работа ядер сведется к ожиданию. На практике использование этой директивы рационально при относительно редком обращении к общим переменным.

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?

В данном случае , #pragma omp critical – это критическая секция , в которой находится команда inc++ . Наличие критической секции в параллельном блоке гарантирует, что она в каждый конкретный момент времени будет выполняться только одним потоком. Т.е. когда один поток находится в критической секции, все остальные потоки, которые готовы в нее войти, находятся в приостановленном состоянии. Из за постоянных блокировок (благодаря этой критической секции) при работе цикла , программа будет медленно работать . Директива atomic работает быстрее, чем критические секции, поскольку некоторые атомарные операции могут быть напрямую заменены командами процессора.

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.pngПросмотр в новом окне. 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.

void omp_init_lock(omp_lock_t *lock) - инициализирует блокировку и связывает ее с параметром lock.

void omp_set_lock(omp_lock_t *lock) - Блокирует выполнение потока до тех пор пока блокировка на переменную lock не станет доступной.

void omp_unset_lock(omp_lock_t *lock) - Снимает блокировку с переменной lock

void omp_destroy_lock(omp_lock_t *lock) - деинициализирует переменную, связанную с параметром lock.

Данные вызовы в openmp аналогичны мьютексам

Между мьютексом и критической секцией есть терминологические различия, так процедура, аналогичная захвату мьютекса, называется входом в критическую секцию , снятию блокировки мьютекса — выходом из критической секции.

Процедура входа и выхода из критических секций обычно занимает меньшее время, нежели аналогичные операции мьютекса, что связано с отсутствием необходимости обращаться к ядру ОС.

Разница между мьютексом и критической секцией в том, что мьютекс является объектом ядра и может быть использован несколькими процессами одновременно, критическая секция же принадлежит процессу и служит для синхронизации только его потоков.

6. Сделайте выводы по проделанной Вами работе

Опытным путем я убедился , что с точки зрения убывания быстродействия, средства защиты общих данных от одновременной записи располагаются так: atomic (если можно использовать атомарные операции), critical, omp_set_lock . Также нужно не злоупотреблять с блокировкой потоков , делать так , чтобы в программе было немного блокировок (только там , где они действительно необходимы) , особенно осторожным нужно быть при работе с циклами .