Отчет к заданию по занятию №7

Зуева Ирина, ИВТ-21М

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***.

В первом случае загруженность вычислителей не высокая, т е время выполнения метода par() составляет 612 мс.

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***.

Для ряда операций более эффективно использовать директиву atomic, чем критическую секцию. Она ведет себя также, но работает чуть быстрее. Применять ее можно для операций префиксного/постфиксного инкремента/декремента. Директива гарантирует корректную работу с общей переменной, стоящей в левой части оператора присваивания.

Результат работы: программа выполняется дольше на 0.05 сек.

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***.

Выше уже было описано, что директива critical работает аналогично atomic, но медленнее. Директива critical применяется в тех случаях, когда параллельное выполнение кода несколькими нитями может приводить к неоднозначности результата. Приведем пример программы, иллюстрирующей действие директивы critical.

Результат: программа выполнялась на 0.1 секунду дольше, чем с atomic.

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. 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.

Один из вариантов синхронизации в OpenMP реализуется через механизм замков (locks). В качестве замков используются общие целочисленные переменные (размер должен быть достаточным для хранения адреса).
Замок может находиться в одном из трёх состояний: неинициализированный, разблокированный или заблокированный. Разблокированный замок может быть захвачен некоторой нитью. При этом он переходит в заблокированное состояние. Нить, захватившая замок, и только она может его освободить, после чего замок возвращается в разблокированное состояние.

Проанализировав CPU time можно сделать вывод, что на установку замка и его снятие тратится большое количество ресурсов.
Время выполнения программы 4 секунды.

Вывод: с помощью инструмента ***Concurrency Analysis*** инструмента ***Amplifier XE*** из панели инструментов ***Visual Studio*** можно наглядно оценить эффективность применения тех или иных инструментов параллельного программирования. Для оптимальной работы данного примера было достаточно ввести распараллеливание по 3 потокам.