Лабораторная работа 4. OpenMP-2

1. Прямоугольники Написать программу, которая приближенно вычисляет определенный интеграл по формуле прямоугольников. Подъинтегральную функцию задавать в виде отдельной функции, чтобы программа могла считать разные интегралы. Проверить на различных функциях (не менее 10 тестов). Для распараллеливания использовать OpenMP, использовать весь ресурс параллелизма (параллелить, где можно) и оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

2. МПИ Составить программу, решающую систему линейных алгебраических уравнений произвольной размерности методом простых итераций. Задаются матрица коэффициентов и столбец свободных членов. Проверить на примерах (не менее 5 тестов). Для распараллеливания использовать OpenMP, использовать весь ресурс параллелизма (параллелить, где можно) и оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

3. Произведение матриц. Составить программу, находящую произведение матриц A*B. Проверить на различного размера матрицах (не менее 5 тестов), сверх 5 добавить тесты для случаев, когда матрицы невозможно перемножить. Для распараллеливания использовать OpenMP, использовать весь ресурс параллелизма (параллелить, где можно) и оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

4. Ньютон. Напишите программу, которая выполняет интерполяцию произвольной функции, заданной таблично, по полиному Ньютона. Для распараллеливания использовать OpenMP, использовать весь ресурс параллелизма (параллелить, где можно) и оптимизации компилятора. Проверить на различных функциях (не менее 5 тестов). Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

5. Метод Ньютона. Напишите программу, решающую уравнение Ax^2-Bx=C методом Ньютона. A,B,C – произвольные действительные числа. Выбрать начальное приближение x = -4…4 с шагом 1 так, что в каждом потоке OpenMP проводится решение со своим начальным приближением. Проверить на различных уравнениях (не менее 5 тестов). Использовать оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

6. Сумма ряда. Написать программу, находящую сумму ряда. Протестировать на нескольких (не менее 5) рядах с 1000000 членов, используя 1, 2, 4, 8 потоков OpenMP. Использовать parallel for. Сравнить быстродействие в каждом случае при использовании статической и динамической планировки распределения итераций цикла. Использовать оптимизации компилятора. Построить график зависимости ускорения многопоточной программы по сравнению с 1 потоком и сравнить с линейным ускорением.

7. Accumulate. Написать свою версию стандартной функции std::accumulate, которая принимает два итератора, и суммирует все элементы контейнера, используя потоки OpenMP. Число потоков должно определяться в зависимости от размера контейнера (установите для вашего оборудования оптимальное число самостоятельно). Использовать оптимизации компилятора. Построить график зависимости ускорения многопоточной программы по сравнению с 1 потоком и сравнить с линейным ускорением. Протестировать на нескольких (не менее 5) примерах, в т.ч., на элементах недопустимого типа.

8. MinMax. Вам даётся файл с 3 миллионами строк и 10 столбцами. Написать программу, которая находит минимум и максимум в каждой строке. Протестировать на нескольких (не менее 5) примерах файлов с небольшим количеством чисел. Для распараллеливания использовать OpenMP, использовать весь ресурс параллелизма (параллелить, где можно) и оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

9. ExpoFinder. Вам даётся файл с 100 тысячами строк. Каждая из строк была рассчитана по формуле i * e^x (0 < i < 100000), однако во время расчётов возникли неполадки, и все значения содержат погрешности. Написать программу, которая находит показатель экспоненты по файлу. Протестировать нахождение x на нескольких (не менее 3) примерах небольших файлов, рассчитанных по точной формуле. Для распараллеливания использовать OpenMP, использовать весь ресурс параллелизма (параллелить, где

можно) и оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

10. Дисперсия. Вам даётся файл с 3 миллионами строк и 10 столбцами. Написать программу, которая находит математическое ожидание и дисперсию величины в каждой строке. Протестировать на файле с не менее 4 строками, для которых известны значения математического ожидания и дисперсии в строках. Для распараллеливания использовать OpenMP, использовать весь ресурс параллелизма (параллелить, где можно) и оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

11. Редукция. Вам даётся файл с 100 тысячами строк. Необходимо найти среднее и максимальное значение в файле. Протестировать нахождение необходимых величин на нескольких (не менее 3) примерах небольших файлов. Для распараллеливания использовать OpenMP, использовать весь ресурс параллелизма (параллелить, где можно) и оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

12. Двойная обработка. Вам даётся файл с 100 тысячами строк. Необходимо удвоить каждый четный элемент (нулевой, второй...) строки на 0, и с полученной матрице в каждом столбце найти наибольший элемент. Протестировать нахождение необходимых величин на нескольких (не менее 3) примерах небольших файлов. Для распараллеливания использовать OpenMP, использовать весь ресурс параллелизма (параллелить, где можно) и оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

13. find_all. Написать свою версию алгоритма find_all, который принимает два итератора и значение, и возвращает контейнер с итераторами, указывающими на элементы контейнера с этим значением, используя потоки OpenMP. Число потоков должно определяться в зависимости от размера контейнера (установите для вашего оборудования оптимальное число самостоятельно). Использовать оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением. Протестировать на нескольких (не менее 5) примерах, в т.ч., на элементах недопустимого типа.

14. max. Написать свою версию алгоритма max, который ищет в контейнере, содержащем числа, максимальный элемент, используя потоки OpenMP. Число потоков должно определяться в зависимости от размера контейнера (установите для вашего оборудования оптимальное число самостоятельно). Использовать оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением. Протестировать на нескольких (не менее 5) примерах, в т.ч., на элементах недопустимого типа.

15. Книголюб. Посчитать число вхождений каждой буквы русского алфавита в «Братьях Карамазовых». Составить рейтинг популярности букв. Игнорировать регистр букв. Найти оптимальное число потоков для этой обработки на вашем компьютере. Книгу взять с сайта https://royallib.com/author/dostoevskiy_fedor.html Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

16. Корпус Русского языка. Посчитать число вхождений каждого слова на русском языке в «Преступлении и наказании». Составить рейтинг самых популярных слов (30 слов). Игнорировать регистр букв. Для простоты не делать усечение окончаний. Найти оптимальное число потоков для этой обработки на вашем компьютере. Книгу взять с сайта https://royallib.com/author/dostoevskiy_fedor.html Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

17. Уравнение-1. Решить уравнение с использованием явной разностной схемы, осуществив распараллеливание (OpenMP) по всем возможным независимым переменным:
Решить уравнение с использованием явной разностной схемы, осуществив распараллеливание (OpenMP) по всем возможным независимым переменным:

$$\frac{\partial c}{\partial t} + v\frac{\partial c}{\partial x} = 4x$$

$x \in 0..1$, $t \in 0..1$. v > 0 задает пользователь.
Граничное условие c(t,0) = 1, начальное условие c(0,x) = sin(x) + 1. Построить график зависимости

ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

18. Уравнение-2. Решить уравнение с использованием явной разностной схемы, осуществив распараллеливание (OpenMP) по всем возможным независимым переменным:

$$\frac{\partial c}{\partial t} + 4\frac{\partial c}{\partial x} = t + x$$

$x \in 0..1$, $t \in 0..1$

Граничное условие c(t,0) = 0.6, начальное условие c(0,x) = 0.6 cos(x). Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

19. Давайте перемножим это. Напишите программу с использованием OpenMP, позволяющую умножать пары матриц. Первая матрица в паре содержит 1000 строк, вторая – 250 столбцов. Каждый из 20 потоков рассчитывает перемножает 5 строк первой матрицы в паре и 5 столбцов второй матрицы в паре, затем получает новое задание. Всего перемножить 20 пар матриц. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

20. А давайте еще перемножим. Написать программу, находящую произведение двух больших матриц. Протестировать на нескольких (не менее 5) матрицах, имеющих 100тыс.-10 млн элементов (выбрать самостоятельно, чтобы оборудование было загружено несколько секунд), используя 1, 2, 4, 8 потоков OpenMP. Использовать parallel for. Сравнить быстродействие в каждом случае при использовании статической, динамической, управлеяемой (guided) планировки распределения итераций цикла. Использовать оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

21. А давайте посуммируем. Написать программу, находящую сумму двух больших матриц. Протестировать на нескольких (не менее 5) матрицах, имеющих 10-100 млн элементов (выбрать самостоятельно, чтобы оборудование было загружено несколько секунд), используя 1, 2, 4, 8 потоков OpenMP. Использовать parallel for. Сравнить быстродействие в каждом случае при использовании статической, динамической, управлеяемой (guided) планировки распределения итераций цикла. Использовать оптимизации компилятора. Построить график зависимости ускорения многопоточной программыпо сравнению с 1 потоком и сравнить с линейным ускорением.

22. Майнинг. Написать программу, осуществляющую подбор шестнадцатеричного хэша длиной N. N выбрать самостоятельно, чтобы оборудование было загружено примерно 5-20 секунд. Построить график зависимости ускорения подбора в потоках по сравнению с 1 потоком и сравнить с линейным ускорением.