Лабораторная работа 3. Разработка простейших многопоточных программ

Цель работы: реализовать параллельные алгоритмы, учитывая особенности и правила многопоточности.

1. Разработать программу, реализующую параллельный алгоритм.

2. Выполнить анализ эффективности параллельной программы:

- построить графики зависимости ускорения и коэффициента эффективности от числа потоков,
- оценить ускорение по закону Амдала (для нерекурсивных алгоритмов),
- оценить масштабирование.

Варианты заданий:

Вариант 1. Генерация множества Мандельброта.

На *вход* программы поступают числа a, b, определяющие размер прямоугольной области для поиска точек, принадлежащих множеству Мандельброта.

В *результате* выполнения программы необходимо получить подмножество n точек, принадлежащих множеству Мандельброта. Для этого прямоугольная область равномерно разделяется между потоками, и каждый поток выполняет поиск точек множества Мандельброта в своей области. Для проверки принадлежности точки множеству Мандельброта рекомендуется использовать алгоритм Escape Time. Необходимо учитывать проблему дисбаланса загрузки процессорных ядер и убедиться в корректности определения времени выполнения параллельной программы.

Вариант 2. Алгоритм быстрой сортировки.

Входными данными для программы является неотсортированный числовой массив **a**, заполненный случайными значениями.

Результат выполнения программы – отсортированный массив **a**. Необходимо реализовать рекурсивный алгоритм быстрой сортировки. При каждом вызове функции быстрой сортировки необходимо порождать поток. Порождение потоков необходимо остановить при достижении количества

потоков, равного заданному числу, которое варьируется от 1 до 10 числа процессорных ядер на вычислительном узле. В качестве опорного элемента выбирать первый элемент последовательности.

Вариант 3. Алгоритм сортировки Шелла.

На *вход* алгоритма поступает неотсортированный массив a длины n, заполненный случайными числовыми значениями.

Результат программы –отсортированный массив a. Длины di промежутков следует выбирать следующим образом:

$$d1 = n / 2, d2 = n / 4, di = di – 1 / 2, dk = 1.$$

В ходе выполнения алгоритма сортировку каждого подмассива (для текущей длины d промежутка) необходимо выполнять в отдельном потоке. Число потоков не должно превышать количества процессорных ядер в системе.

Вариант 4. Алгоритм бинарного поиска.

Необходимо реализовать две версии алгоритма поиска: для неотсортированного массива и для отсортированного массива. Ключи могут встречаться несколько раз в массиве. На вход подаётся случайный числовой массив a (отсортированный или неотсортированный), в котором осуществляется поиск, и набор чисел для поиска, представленных массивом b.

Выходные данные: если элемент массива b встречается в массиве a, то необходимо вывести индекс этого элемента в массиве a. В случае неотсортированного массива элементы массива a равномерно распределяется

между потоками, каждый поток выполняет поиск элементов в своём подмассиве. В случае отсортированного массива целесообразно распределить между потоками числа для поиска, представленные массивом b. Число потоков не должно превышать количество процессорных ядер.

Вариант 5. Решение системы линейных алгебраических уравнений (СЛАУ) методом Гаусса. На вход программы поступает случайно сформированная СЛАУ Ax = b из n уравнений. Вид матрицы – произвольный. Выходные данные: вектор x решений СЛАУ. При распараллеливании элементы матрицы равномерно распределяются между потоками по строкам. Необходимо реализовать параллельное выполнение как прямого, так и обратного хода метода Гаусса.

Вариант 6. Алгоритм Прима построения минимального остовного дерева в графе.

Входные данные: случайный связный неориентированный граф G = (V, E). Для каждого ребра графа задана его стоимость. Выходные данные: набор вершин, образующих минимальное остовное дерево. На каждой итерации алгоритма Прима поиск ребра с наименьшей стоимостью выполняется параллельно. Для этого всё множество рёбер, инцидентных вершинам текущего остовного дерева, равномерно распределяется между потоками. Число потоков не должно превышать количество процессорных ядер.

Вариант 7. Алгоритм Флойда поиска всех минимальных путей в графе.

Входные данные: случайный неориентированный граф G = (V, E) из вершин. Для каждого ребра графа задана его стоимость.

Выходные данные: все минимальные пути в графе. На каждой итерации алгоритма множество всех вершин k, через которые может пройти кратчайший путь, равномерно распределяется между потоками. Число потоков не должно превышать количество процессорных ядер.

Вариант 8. Алгоритм поиска подстрок в строке.

Входные данные: строка s, строка d.

Выходные данные: номера символов, с которых начинается строка d, найденная в строке s. Элементы строки s равномерно распределяются между потоками, каждый поток выполняет поиск в своей подстроке. Число потоков не должно превышать количество процессорных ядер.

Вариант 9. Разработайте многопоточное приложение, выполняющее вычисление произведения матриц A (m×n) и B (n×k). Элементы c_{ij} матрицы произведения C = A×B вычисляются параллельно p однотипными потоками. Если некоторый поток уже вычисляет элемент c_{ij} матрицы C, то следующий приступающий к вычислению поток выбирает для расчета элемент c_{ij+1}, если j<k, и c_{i+1k}, если j=k. Выполнив вычисление элемента матрицы-произведения, поток проверяет, нет ли элемента, который еще не рассчитывается. Если такой элемент есть, то приступает к его расчету. В противном случае отправляет (пользовательское) сообщение о завершении своей работы и приостанавливает своё выполнение. Главный поток, получив сообщения о завершении вычислений от всех потоков, выводит результат на экран и запускает поток, записывающий результат в конец файла-протокола. В каждом потоке должна быть задержка в выполнении вычислений (чтобы дать возможность поработать всем потокам). Синхронизацию потоков между собой организуйте через критическую секцию или мьютекс.