Практическая работа №9: "Распределенная обработка данных с использованием MPI"

Цель работы:

1. Изучить продвинутые операции MPI (передача сообщений, коллективные операции, односторонняя передача).

2. Реализовать программы для сложной распределённой обработки данных.

3. Исследовать производительность и масштабируемость программ на кластере.

Задание 1: Распределённое вычисление среднего значения и стандартного отклонения

Описание задачи:

Напишите программу, которая выполняет следующие шаги:

1. Создайте массив случайных чисел на процессе с "rank = 0". Размер массива — N (например, N = 10^6).

2. Разделите массив между всеми процессами с помощью функции "MPI_Scatterv" (учитывая, что массив может не делиться нацело между процессами).

3. Каждый процесс вычисляет:

- Сумму элементов своей части массива.

- Сумму квадратов элементов своей части массива.

4. Соберите локальные суммы на процессе с "rank = 0" с помощью функции "MPI_Reduce".

5. На основе собранных данных вычислите:

- Среднее значение массива.

- Стандартное отклонение, формула:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} x_i\right)^2}$$

6. Выведите результаты на экран.

Требования:

- Используйте "MPI_Scatterv" для учёта остатка при разделении массива.

- Убедитесь, что программа работает корректно при любом количестве процессов.

Задание 2: Распределённое решение системы линейных уравнений методом Гаусса

Описание задачи:

Напишите программу для распределённого решения системы линейных уравнений методом Гаусса:

1. Процесс с "rank = 0" создаёт матрицу коэффициентов A размером NxN и вектор правых частей b.

2. Разделите строки матрицы между процессами с помощью функции "MPI_Scatter".

3. Реализуйте следующие шаги метода Гаусса:

- Прямой ход: каждый процесс выполняет вычитание строк для своей части матрицы.

- Обратный ход: соберите результаты на процессе с "rank = 0" и завершите вычисления.

4. Выведите решение системы уравнений на экран.

Требования:

- Размеры матрицы должны быть заданы параметрами программы.

- Убедитесь, что программа работает корректно при любом количестве процессов.

- Используйте "MPI_Bcast" для передачи текущей строки другим процессам во время прямого хода.

Задание 3: Параллельный анализ графов (поиск кратчайших путей)

Описание задачи:

Напишите программу для параллельного поиска кратчайших путей в графе с использованием алгоритма Флойда-Уоршелла:

1. Процесс с "rank = 0" создаёт матрицу смежности графа G размером NxN.

2. Разделите строки матрицы между процессами с помощью функции "MPI_Scatter".

3. Реализуйте алгоритм Флойда-Уоршелла:

- Каждый процесс обновляет свою часть матрицы для текущей итерации.

- Передайте обновлённые данные между процессами с помощью функции "MPI_Allgather".

4. После завершения всех итераций соберите матрицу на процессе с "rank = 0" и выведите её на экран.

Требования:

- Размеры графа должны быть заданы параметрами программы.
- Убедитесь, что программа работает корректно при любом количестве процессов.
- Используйте "MPI_Allgather" для обмена данными между процессами.

Дополнительные инструкции:

1. Компиляция и запуск

- Скомпилируйте программу с помощью команды:

mpic++ program.cpp -o program

- Запустите программу на нескольких процессах:

mpirun -np <number_of_processes> ./program

2. Исследование производительности

- Для каждого задания измерьте время выполнения программы с помощью функции "MPI_Wtime()":

```
  double start_time = MPI_Wtime(); // В начале программы
  double end_time = MPI_Wtime();   // В конце программы
  if (rank == 0) {
      std::cout << "Execution time: " << end_time - start_time << "
seconds." << std::endl;
  }
```

- Сравните время выполнения для разных значений "-np".

Контрольные вопросы:

1. Как изменяется время выполнения программы при увеличении количества процессов? Почему?
2. Какие факторы могут влиять на производительность программы?
3. Как можно оптимизировать передачу данных между процессами?
4. Какие ограничения возникают при работе с большими данными?

Заключение

Эта практическая работа помогает освоить продвинутые операции MPI и решать сложные задачи, такие как статистический анализ, решение систем уравнений и анализ графов. Вы можете использовать её как основу для дальнейшего изучения параллельного программирования.