Пухов Максим Станиславович (ИС-242)

Отчёт
Практическая работа 1

Задание

Требуется на базе операций MPI_Send, MPI_Recv, MPI_Sendrecv, MPI_Isend, MPI_Irecv реализовать следующие схемы обменов.

Описание работы программы

В данной работе было разработано и протестировано несколько MPI-программ, каждая из которых реализует различные схемы обмена сообщениями между процессами. Все программы используют возможности параллельной обработки данных и выполняются на кластере с различной конфигурацией узлов и процессов. Основные операции передачи данных включают в себя такие функции MPI, как `MPI_Send`, `MPI_Recv`, `MPI_Isend`, `MPI_Irecv`, и другие. Рассмотрим особенности каждой программы и детали их реализации.

1. *кольцевой обмен**

Программа "Ring" реализует кольцевую схему обмена данными. В этой схеме каждый процесс передает сообщение своему соседнему процессу и принимает сообщение от предыдущего. Цель — передать сообщение через все процессы, организованные в кольцо, где последний процесс связывается с первым.

Описание работы программы:

- В начале программа инициализирует MPI и определяет количество процессов (`MPI_Comm_size`) и текущий ранг процесса (`MPI_Comm_rank`).

- Основная задача каждого процесса заключается в том, чтобы отправить сообщение следующему процессу по кольцу и получить сообщение от предыдущего.

- Для реализации этой схемы используется функция `MPI_Sendrecv`. Она одновременно отправляет сообщение следующему процессу и получает сообщение от предыдущего. Например:

```

MPI_Sendrecv(&send_data, 1, MPI_INT, (rank + 1) % size, 0, &recv_data, 1, MPI_INT, (rank - 1 + size) % size, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```

Здесь `(rank + 1) % size` и `(rank - 1 + size) % size` обеспечивают кольцевую связь между процессами.

- Процесс обмена выполняется `p - 1` раз, где `p` — количество процессов. Это гарантирует, что все сообщения пройдут через каждый процесс.

- Время выполнения обмена измеряется с помощью функции `MPI_Wtime` для дальнейшего анализа производительности программы.

Пример работы программы:

```

Процесс 0 передает сообщение процессу 1

Процесс 1 передает сообщение процессу 2

...

Процесс N передает сообщение процессу 0

```

#### 2. *трансляционная передача**

Программа "Broadcast" реализует схему передачи данных от одного процесса ко всем остальным. Корневой процесс (обычно процесс с рангом 0) передает свое сообщение всем остальным процессам. Это важно для передачи глобальных данных, таких как настройки программы или исходные данные для расчётов.

**Описание работы программы**:

- Корневой процесс передает данные всем остальным процессам с помощью функции `MPI_Bcast`.

```

MPI_Bcast(&data, count, MPI_CHAR, 0, MPI_COMM_WORLD);

```

Здесь `data` — это сообщение, которое передается всем процессам, `count` — количество передаваемых элементов, `MPI_CHAR` — тип данных, а 0 — ранг корневого процесса.
```

- Все процессы получают одно и то же сообщение, что гарантирует синхронное выполнение дальнейших шагов программы.

- Время передачи данных измеряется как на стороне отправителя (процесс 0), так и на стороне получателей (процессы 1, 2, ..., p-1) для оценки задержек и производительности обменов.

Пример работы программы:

```

Процесс 0 отправляет сообщение всем процессам

Процесс 1 получает сообщение

Процесс 2 получает сообщение

...

Процесс N получает сообщение

```

3. *коллекторный приём**

Программа "Gather" выполняет обратную задачу по сравнению с "Broadcast". В этой схеме все процессы отправляют свои данные одному процессу, который собирает эти данные в свой буфер. Эта схема полезна для сбора результатов вычислений или сбора данных от различных процессов для их дальнейшей обработки.

Описание работы программы:

- Каждый процесс отправляет данные в буфер корневого процесса с помощью функции `MPI_Gather`.

  ```

  MPI_Gather(&send_data, count, MPI_CHAR, recv_buf, count, MPI_CHAR, 0, MPI_COMM_WORLD);

  ```

  Здесь `send_data` — данные, которые процесс отправляет, `recv_buf` — буфер корневого процесса для приема данных от всех процессов.

- Корневой процесс собирает данные в свой буфер, который имеет размер, соответствующий количеству процессов.

- Время выполнения замеров осуществляется как для отправителей, так и для корневого процесса.

Пример работы программы:

```

Процесс 0 принимает данные от всех процессов

Процесс 1 отправляет свои данные

Процесс 2 отправляет свои данные

...

Процесс N отправляет свои данные

```

4. **All-to-all (все ко всем)**

Программа "All-to-all" реализует схему, при которой каждый процесс передает сообщение всем остальным процессам и одновременно получает сообщения от них. Для реализации этой схемы используется неблокирующий обмен с помощью функций `MPI_Isend` и `MPI_Irecv`. Это наиболее сложная схема обмена данными, так как каждый процесс участвует во множественных операциях одновременно.

Описание работы программы:

- Программа инициализирует несколько неблокирующих отправок и приемов, используя `MPI_Isend` и `MPI_Irecv`.

  ```

  MPI_Isend(&send_data, count, MPI_CHAR, dest, 0, MPI_COMM_WORLD, &request_send);

  MPI_Irecv(&recv_data, count, MPI_CHAR, source, 0, MPI_COMM_WORLD, &request_recv);

  ```

Здесь каждый процесс отправляет сообщение всем другим процессам и одновременно получает сообщения от всех.

- После завершения всех операций программа синхронизирует их с помощью `MPI_Waitall`, что гарантирует, что все передачи и приемы завершились до продолжения программы.

```

MPI_Waitall(size, requests, MPI_STATUSES_IGNORE);

```

- Время выполнения обменов замеряется на каждом процессе для оценки задержек при параллельной передаче данных.

Пример работы программы:

```

Процесс 0 отправляет сообщения процессам 1, 2, ..., N

Процесс 1 отправляет сообщения процессам 0, 2, ..., N

...

Процесс N отправляет сообщения процессам 0, 1, ..., N-1

```

Итоги

Каждая из описанных программ демонстрирует различные способы обмена сообщениями между процессами в распределенных системах. Эти схемы важны для понимания и оптимизации работы параллельных вычислений. В ходе экспериментов было проведено измерение времени выполнения обменов при различных размерах сообщений и конфигурациях узлов, что позволит оценить эффективность каждой схемы и ее применение в реальных вычислительных задачах.