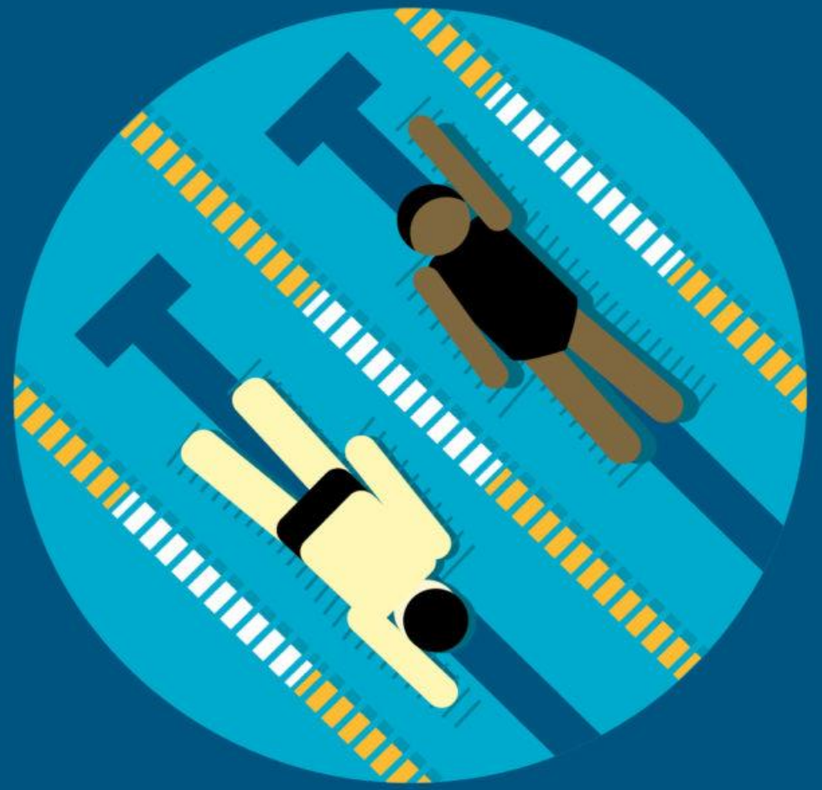
Асинхронные операции

особенности использова ния асинхронных операций для приема-передачи сообщений между различными MPI процессами.



Необходимость использования асинхронных операций

Рассмотрим асинхронные операции на примере MPI-процессов, где каждый процесс передаёт свой идентификатор следующему по рангу и получает сообщение от предыдущего. Эта задача иллюстрирует необходимость асинхронных операций для избежания блокировок при обмене сообщениями в кольцевой топологии.

```
from mpi4py import MPI
from numpy import array, int32


comm = MPI.COMM_WORLD
numprocs = comm.Get_size()
rank = comm.Get_rank()
a = array(rank, dtype=int32)
b = array(100, dtype=int32)
```

Здесь массив a хранит ранг процесса, а b изначально равен 100, чтобы при сбоях (если сообщение не получено) значение оставалось неизменным, что упрощает отладку.

```
if rank == 0:
    comm.Recv([b, 1, MPI.INT], source=1, tag=0, status=None)
    comm.Send([a, 1, MPI.INT], dest=numprocs-1, tag=0)
elif rank == numprocs - 1:
    comm.Recv([b, 1, MPI.INT], source=0, tag=0, status=None)
    comm.Send([a, 1, MPI.INT], dest=numprocs-2, tag=0)
else:
    comm.Recv([b, 1, MPI.INT], source=rank + 1, tag=0, status=None)
    comm.Send([a, 1, MPI.INT], dest=rank - 1, tag=0)
print('I, process with rank {}, got number {}'.format(rank, b))
```

В этом коде каждый процесс сначала получает сообщение от соседа и затем отправляет свой ранг другому. Если запустить скрипт с `mpiexec -n 4 python script.py`, можно ожидать, что каждый процесс выведет число на 1 больше его ранга. Однако, так как `Send` и `Recv` являются блокирующими, возможна ситуация, когда программа зависает, так как все процессы ожидают данные, и ни один не инициирует отправку.

Для избежания блокировки можно изменить топологию или порядок команд, например, в линейке:

```
if rank == 0:
    comm.Recv([b, 1, MPI.INT], source=1, tag=0, status=None)
elif rank == numprocs - 1:
    comm.Send([a, 1, MPI.INT], dest=numprocs-2, tag=0)
else:
    comm.Recv([b, 1, MPI.INT], source=rank + 1, tag=0, status=None)
    comm.Send([a, 1, MPI.INT], dest=rank - 1, tag=0)
print('I, process with rank {}, got number {}'.format(rank, b))
```

Запуск `mpiexec -n 4 python script.py` выведет:

```
I, process with rank 3, got number 100
I, process with rank 2, got number 3
I, process with rank 1, got number 2
I, process with rank 0, got number 1
```

Тут видно, что ранг 3 ничего не получает, так как крайние процессы в линейке не имеют замкнутой связи.

Чтобы решить эту проблему в кольце, попробуем поменять порядок `Send` и `Recv` для последнего процесса:

```
if rank == 0:
    comm.Recv([b, 1, MPI.INT], source=1, tag=0, status=None)
    comm.Send([a, 1, MPI.INT], dest=numprocs-1, tag=0)
elif rank == numprocs - 1:
    comm.Send([a, 1, MPI.INT], dest=numprocs-2, tag=0)
    comm.Recv([b, 1, MPI.INT], source=0, tag=0, status=None)
else:
    comm.Recv([b, 1, MPI.INT], source=rank + 1, tag=0, status=None)
    comm.Send([a, 1, MPI.INT], dest=rank - 1, tag=0)
print('I, process with rank {}, got number {}'.format(rank, b))
```

Запуск этого варианта приведет к последовательному обмену, начиная с предпоследнего и двигаясь по кругу до последнего.

Если же отправляемое сообщение небольшое, и оно может быть буферизовано, например, так:

```
if rank == 0:
    comm.Send([a, 1, MPI.INT], dest=numprocs-1, tag=0)
    comm.Recv([b, 1, MPI.INT], source=1, tag=0, status=None)
elif rank == numprocs - 1:
    comm.Send([a, 1, MPI.INT], dest=numprocs-2, tag=0)
    comm.Recv([b, 1, MPI.INT], source=0, tag=0, status=None)
else:
    comm.Send([a, 1, MPI.INT], dest=rank - 1, tag=0)
    comm.Recv([b, 1, MPI.INT], source=rank + 1, tag=0, status=None)
print('I, process with rank {}, got number {}'.format(rank, b))
```

При малом объеме данных `Send` может сразу завершиться, и программа выполнится без блокировок, что подтверждает важность асинхронных операций.

Асинхронные операции

Вспомним аргументы `Send`:

```
comm.Send([a, 1, MPI.INT], dest=numprocs-1, tag=0)
```

Первый аргумент указывает на начало области памяти массива `a`, берется один элемент типа `int`, отправляется процессу с ID `dest`, сообщение помечено `tag=0`. Асинхронная версия:

```
requests = comm.Isend([a, 1, MPI.INT], dest=numprocs-1, tag=0)
```

Аргументы те же, но добавляется `requests` типа `MPI_Request`, который хранит информацию о завершении передачи сообщения. Выход происходит сразу. Аналогично для `Recv`:

```
comm.Recv([b, 1, MPI.INT], source=1, tag=0, status=None)
```

Асинхронный аналог:

```
requests = comm.Irecv([b, 1, MPI.INT], source=1, tag=0)
```

Теперь заменим `Send` и `Recv` асинхронными версиями:

```
requests = [MPI.Request() for i in range(2)]
if rank == 0:
    requests[0] = comm.Isend([a, 1, MPI.INT], dest=numprocs-1, tag=0)
    requests[1] = comm.Irecv([b, 1, MPI.INT], source=1, tag=0)
elif rank == numprocs - 1:
    requests[0] = comm.Isend([a, 1, MPI.INT], dest=numprocs-2, tag=0)
    requests[1] = comm.Irecv([b, 1, MPI.INT], source=0, tag=0)
else:
    requests[0] = comm.Isend([a, 1, MPI.INT], dest=rank-1, tag=0)
    requests[1] = comm.Irecv([b, 1, MPI.INT], source=rank+1, tag=0)
print('I, process with rank {}, got number {}'.format(rank, b))
```

После запуска на четырех процессах:

```
I, process with rank 2, got number 100
I, process with rank 0, got number 100
I, process with rank 1, got number 100
I, process with rank 3, got number 100
```

Строки меняются местами, вывод неверный, так как процессы выполняют `print` до завершения обмена сообщениями. Чтобы избежать этого, добавим блокировку перед `print`:

```
MPI.Request.Waitall(requests, statuses=None)
print('I, process with rank {}, got number {}'.format(rank, b))
```

Теперь результат корректен:

```
I, process with rank 2, got number 3
I, process with rank 0, got number 1
I, process with rank 1, got number 2
I, process with rank 3, got number 0
```

Если прибавить к `a` значение после отправки:

```
a += 10
MPI.Request.Waitall(requests, statuses=None)
print('I, process with rank {}, got number {}'.format(rank, b))
```

Результат:

```
I, process with rank 1, got number 12
I, process with rank 0, got number 11
I, process with rank 3, got number 0
I, process with rank 2, got number 3
```

Чтобы вычисление произошло после отправки, можно написать:

```
MPI.Request.Wait(requests[0], statuses=None)
a += 10
MPI.Request.Waitall(requests, statuses=None)
print('I, process with rank {}, got number {}'.format(rank, b))
```

Можно комбинировать асинхронные и блокирующие операции, например, в линейной топологии:

```
requests = [MPI.Request() for i in range(2)]
if rank == 0:
    comm.Recv([b, 1, MPI.INT], source=1, tag=0, status=None)
elif rank == numprocs - 1:
    comm.Send([a, 1, MPI.INT], dest=numprocs-2, tag=0)
else:
    requests[0] = comm.Isend([a, 1, MPI.INT], dest=rank-1, tag=0)
    requests[1] = comm.Irecv([b, 1, MPI.INT], source=rank+1, tag=0)
    MPI.Request.Waitall(requests, statuses=None)
print('I, process with rank {}, got number {}'.format(rank, b))
```

На нулевом процессе нет смысла использовать `Irecv`, так как он не отправляет сообщения, аналогично, на последнем процессе можно использовать обычный `Send`.

Ранее для обмена с соседями применялись Sendrecv и Sendrecv_replace, что позволило избежать блокировок. Эти проблемы также можно решить с помощью асинхронных операций, которые предпочтительно использовать, если такие команды доступны.

Спасибо за внимание!