**Вопросы по курсу «Технология программирования Message Passing Interface»**

1. **В чем состоят основы технологии MPI?**

MPI является наиболее распространённым стандартом интерфейса обмена данными в [параллельном программировании](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), существуют его реализации для большого числа компьютерных платформ. Используется при разработке программ для [кластеров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%B2)) и [суперкомпьютеров](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D0%B5%D1%80%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80). Основным средством коммуникации между [процессами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) в MPI является передача сообщений друг другу.

1. **В чем состоят основные преимущества и недостатки технологии MPI?**

В первую очередь MPI ориентирован на системы с [распределенной памятью](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2%D0%BD%D0%BE-%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0), то есть когда затраты на передачу данных велики, в то время как [OpenMP](https://ru.wikipedia.org/wiki/OpenMP" \o "OpenMP) ориентирован на системы с общей памятью (многоядерные с общим кэшем). Обе технологии могут использоваться совместно, чтобы оптимально использовать в кластере многоядерные системы.

1. **Что понимается под параллельной программой в рамках технологии MPI?**

MPI-программа - это множество параллельных взаимодействующих процессов, которые работают каждый в своей выделенной области памяти. По сути это N независимых программ, которые общаются между собой в ходе работы.

1. **Как происходит инициализация и завершение MPI программ?**

MPI\_Init(&argc, &argv);

// printf("Hello, World!\n");

MPI\_Finalize();

1. **Как происходит передача и прием сообщений MPI программе? Виды точечных взаимодействий.**

Стандартный режим отправки сообщений обеспечивает: − блокировку процесс-отправителя на время выполнения функции; − возможность повторного использования буфера после завершения функции; − поддерживает следующие состояния сообщения: сообщение находится в процессе отправления, сообщение находится в процессе передачи, сообщение хранится в процессе-получателе или сообщение принято процессом-получателем при помощи функции MPI\_Recv.

MPI\_Send Стандартная пересылка (Standard)

функция возвращает управление тогда, когда исходный буфер можно освобождать (т.е. данные или скопированы в промежуточный или отправлены)

MPI\_Bsend Буферизованная пересылка (Buffered)

функция возвращает управление тогда, когда данные скопированы в буфер, выделяемый пользователем

MPI\_Ssend Синхронная пересылка (Synchronous)

функция возвращает управление тогда, когда процесс-приемник преступил к выполнению соответствующей операции приема

MPI\_Rsend Режим передачи по готовности (Ready)

поведение функции не определено, если соответствующая операция приема не начала выполнения (для увеличения производительности)

1. **Как происходит передача данных от одного процесса всем?**

MPI\_BCAST(buffer, count, datatype, root, comm)

int MPI\_Scatter ( sendbuf, sendcnt, sendtype, recvbuf, recvcnt, recvtype, root, comm ) - обобщенная передача данных от одного процесса всем процессам.

1. **Как происходит передача данных от всем процессов одному?**

// для получения всех собираемых данных на каждом из процессов коммуникатора int MPI\_Allgather ( sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm );

//для получения результатов редукции данных на каждом из // процессов коммуникатора int MPI\_Allreduce ( sendbuf, recvbuf, count, datatype, op, comm )

1. **Какие используются в MPI для синхронизации вычислений?**

Синхронизация *процессов*, т.е. одновременное достижение *процессами* тех или иных точек *процесса* вычислений, обеспечивается при помощи функции *MPI*:

int MPI\_Barrier(MPI\_Comm comm)

1. **Как организуется неблокирующий обмен данными между процессами?**

Предназначены для перекрытия обменов и вычислений. Операция расщепляется на две: инициация и завершение.

int MPI\_Isend( buf, count, datatype, dest, tag, comm, request)

MPI\_Request \*request; /\* out \*/

MPI\_Ibsend(…), MPI\_Issend(…), MPI\_Irsend(…)

int MPI\_Irecv( buf, count, datatype, source, tag, comm, request )

MPI\_Request \*request; /\* out \*/

Завершение:

int MPI\_Wait (MPI\_Request \* request, MPI\_Status \* status)

int MPI\_Test(MPI\_Request \*request, int \*flag, MPI\_Status \*status)

int MPI\_Waitall(int count, MPI\_Request array\_of\_requests[], MPI\_Status array\_of\_statuses[] ) - завершились все операции

int MPI\_Waitany(int count, MPI\_Request array\_of\_requests[], int\* index, MPI\_Status \*status ) - завершилась по крайней мере одна операция

1. **Как организуется одновременное выполнение прием и передачи данных?**

int MPI\_Sendrecv(void \*sbuf,int scount,MPI\_Datatype stype, int dest, int stag, void \*rbuf,int rcount,MPI\_Datatype rtype, int source,int rtag, MPI\_Comm comm, MPI\_Status \*status),

где

· sbuf, scount, stype, dest, stag — параметры передаваемого сообщения;

· rbuf, rcount, rtype, source, rtag — параметры принимаемого сообщения;

· comm — коммуникатор, в рамках которого выполняется передача данных;

1. **Прием и передача по шаблону.**

В качестве параметров source и tag в функции MPI\_Recv могут быть использованы константы MPI\_ANY\_SOURCE и MPI\_ANY\_TAG соответственно. Допускается прием от процесса с произвольным номером и/или сообщения с произвольным тэгом.

1. **Коллективные взаимодействия процессов**

Broadcast (широкое распространение "один-всем")

Gather (сборка "каждый-одному") и Scatter (размещение "один-каждому")

Gatherv (гибкая сборка "переменный каждый-одному")

Scatterv (гибкое размещение "переменный один-каждому")

Allgather (всеобщая сборка "все-каждому")

Alltoall (всеобщий обмен "каждый-каждому")

13. Интер- и интра-коммуникаторы

Коммуникатор – служебный объект, объединяющий в своем составе группу процессов и ряд дополнительных параметров (контекст), используемых при выполнении операций передачи данных.

Интра-коммуникаторы объединяют процессы из одной группы. Интер-коммуникатор позволяет передавать данные между процессами из разных интра-коммуникаторов. Интер-коммуникаторы не могут использоваться в коллективных взаимодействиях

14. **Основные функции работы с коммуникаторами**

**Функция дублирования коммуникатора MPI\_Comm\_dup**

С:

MPI\_Comm\_dup(MPI\_Comm comm, MPI\_Comm \*newcomm)

|  |  |  |
| --- | --- | --- |
| IN | comm | - коммуникатор; |
| OUT | newcomm | - копия коммуникатора. |

**Функция создания коммуникатора MPI\_Comm\_create**

С:

MPI\_Comm\_create(MPI\_Comm comm, MPI\_Group group,

MPI\_Comm \*newcomm)

|  |  |  |  |
| --- | --- | --- | --- |
| IN | comm | - | родительский коммуникатор; |
| IN | group | - | группа, для которой создается коммуникатор; |
| OUT | newcomm | - | новый коммуникатор. |

**Функция расщепления коммуникатора MPI\_Comm\_split**

С:

MPI\_Comm\_split(MPI\_Comm comm, int color, int key,

MPI\_Comm \*newcomm)

**Функция уничтожения коммуникатораа MPI\_Comm\_free**

С:

MPI\_Comm\_free(MPI\_Comm \*comm)

16. Информационные функции для работы с группами

Определение размера группы:

int MPI\_Group\_size(MPI\_Group group, int \*size) group – группа; size – указатель на область памяти для записи информации о количестве процессов в группе;

Определение номера процесса, выполняющего вызов функции, в группе:

int MPI\_Group\_rank(MPI\_Group group, int \*rank) group – группа; rank – указатель на область памяти для сохранения номера процесса;

Установление соответствия между номерами процессов в различных группах:

int MPI\_Group\_translate\_ranks ( MPI\_Group group1, int n, int \*ranks1, MPI\_Group group2, int \*ranks2) group1 – первая группа; n – число элементов массивов ranks1 и ranks2; ranks1 – массив номеров процессов в первой группе; group2 – вторая группа; ranks2 – массив для сохранения номеров процессов во второй группе;

Сравнение двух групп процессов:

int MPI\_Group\_compare(MPI\_Group group1,MPI\_Group group2, int \*result) group1 – первая группа; group2 – вторая группа; result – указатель на область памяти для сохранения результата;

Объединение двух групп:

int MPI\_Group\_union(MPI\_Group gr1, MPI\_Group g2, MPI\_Group\* gr3) gr1 – первая группа; gr2 – вторая группа; gr3 – указатель на область для сохранения результата операции;

Пересечение двух групп:

int MPI\_Group\_intersection(MPI\_Group gr1, MPI\_Group g2, MPI\_Group\* gr3) gr1 – первая группа; gr2 – вторая группа; gr3 – указатель на область для сохранения результата операции;

Разность двух групп:

int MPI\_Group\_difference(MPI\_Group gr1, MPI\_Group g2, MPI\_Group\* gr3) gr1 – первая группа; gr2 – вторая группа; gr3 – указатель на область для сохранения результата операции;

Переупорядочивание (с возможным удалением) процессов в существующей группе:

int MPI\_Group\_incl(MPI\_Group\* group, int n, int\* ranks, MPI\_Group\* newgroup) group – исходная группа; n – число элементов в массиве ranks; ranks – массив номеров процессов, из которых будет создана новая группа; newgroup – указатель на область для сохранения результата операции;

Удаление процессов из группы:

int MPI\_Group\_excl(MPI\_Group\* group, int n, int\* ranks, MPI\_Group\* newgroup) group – исходная группа; n – число элементов в массиве ranks; ranks – массив номеров удаляемых процессов; newgroup – указатель на область для сохранения результата операции;