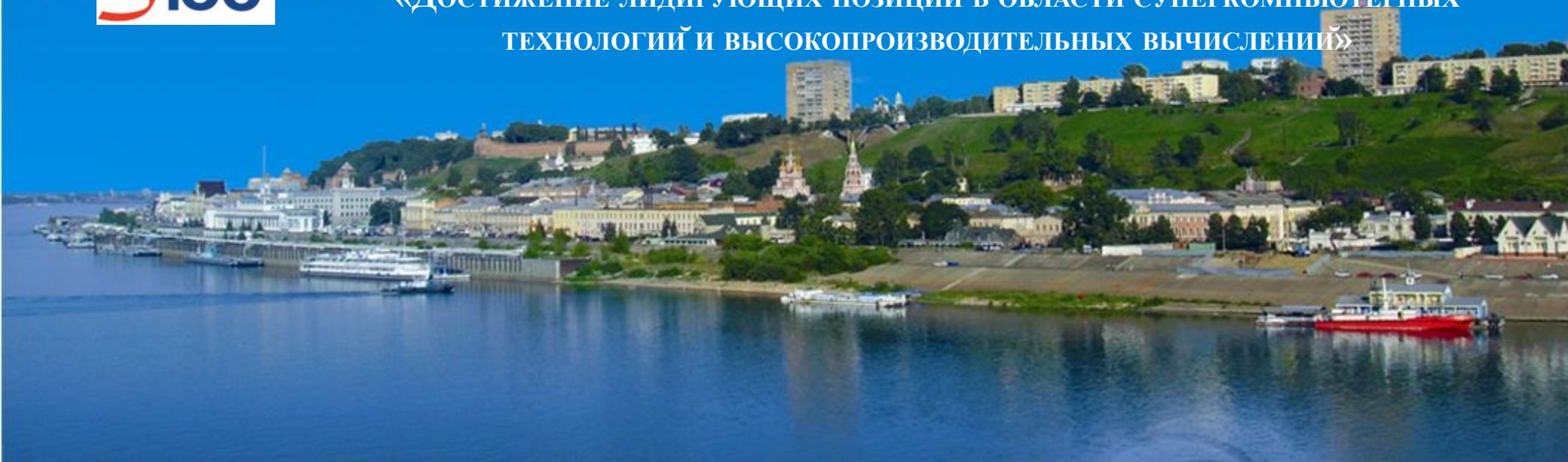



Нижегородский государственный университет имени Н.И. Лобачевского

Институт Информационных технологий, математики и механики

Проект повышения конкурентоспособности ведущих российских университетов среди ведущих мировых научно-образовательных центров

Стратегическая инициатива «Достижение лидирующих позиций в области суперкомпьютерных технологий и высокопроизводительных вычислений»



Нижегородский государственный университет
имени Н.И. Лобачевского
Институт Информационных технологий, математики
и механики

Параллельное программирование для многопроцессорных систем с распределенной памятью

Лекция 02
Коллективные и парные взаимодействия

Гергель В.П., Сысоев А.В.
Кафедра МОСТ

Содержание

- Коллективное взаимодействие
  - Широковещательный обмен
  - Операции редукции
  - Пример: вычисление числа π
  - Рассылка и сбор данных
  - Пример: вычисление скалярного произведения
  - Взаимодействие «Каждый к каждому»
  - Синхронизация вычислений
- Взаимодействие между двумя процессами
  - Режимы взаимодействия
  - Неблокирующее взаимодействие
  - Совмещенные прием и передача сообщений

КОЛЛЕКТИВНОЕ ВЗАИМОДЕЙСТВИЕ

- Широковещательный обмен
- Операции редукции
- Пример: вычисление числа π
- Рассылка и сбор данных
- Пример: вычисление скалярного произведения
- Взаимодействие «Каждый к каждому»
- Синхронизация вычислений

Коллективное взаимодействие… Широковещательный обмен

- Для эффективного широковещательного обмена данными следует использовать следующую функцию MPI:

```
int MPI_Bcast(void *buf, int count, MPI_Datatype type,
  int root, MPI_Comm comm);
```

- **buf** – адрес начала буфера, содержащего передаваемые данные
- **count** – число элементов в буфере
- **type** – тип данных элементов буфера
- **root** – номер корневого процесса
- **comm** – коммуникатор, внутри которого будут передаваться данные

- Функция **MPI_Bcast()** передает данные из буфера **buf**, который содержит **count** элементов типа **type**, от корневого процесса **root** процессам коммуникатора **comm**

Коллективное взаимодействие… Широковещательный обмен

- Функция **`MPI_Bcast()`** коллективная, ее вызов должен быть выполнен всеми процессами коммуникатора **`comm`**
- Буфер, указанный в **`MPI_Bcast()`** имеет разное назначение в различных процессах:
  - Для корневого (**`root`**) процесса, который передает данные, буфер должен содержать передаваемое сообщение
  - Для остальных – принимаемые данные

Коллективное взаимодействие… Редукция данных

- Чтобы “редуцировать” данные со всех процессов в один выбранный, используется следующая функция MPI:

```
int MPI_Reduce(void *sendbuf, void *recvbuf, int count,
  MPI_Datatype type, MPI_Op op, int root, MPI_Comm comm);
```

- **sendbuf** – буфер с передаваемыми данными
- **recvbuf** – буфер, содержащий сообщение-результат (только для корневого процесса)
- **count** – число элементов в сообщении
- **type** – тип данных элементов в сообщении
- **op** – операция, которая должна быть выполнена над данными
- **root** – номер процесса, который должен получить результат
- **comm** – коммуникатор, внутри которого будет выполнена операция

Коллективное взаимодействие… Редукция данных

- Базовые типы операций MPI для редукции данных

Операция	Описание
`MPI_MAX`	Вычисление максимального значения
`MPI_MIN`	Вычисление минимального значения
`MPI_SUM`	Вычисление суммы значений
`MPI_PROD`	Вычисление произведения значений
`MPI_LAND`	Вычисление операции логическое «И» над данными в сообщении
`MPI_BAND`	Вычисление операции побитовое «И» над данными в сообщении
`MPI_LOR`	Вычисление операции логическое «ИЛИ» над данными в сообщении
`MPI_BOR`	Вычисление операции побитовое «ИЛИ» над данными в сообщении
`MPI_LXOR`	Вычисление операции логическое «ИСКЛЮЧАЮЩЕЕ ИЛИ» над данными в сообщении
`MPI_BXOR`	Вычисление операции побитовое «ИСКЛЮЧАЮЩЕЕ ИЛИ» над данными в сообщении
`MPI_MAXLOC`	Вычисление максимальных значений и их индексов
`MPI_MINLOC`	Вычисление минимальных значений и их индексов

Коллективное взаимодействие… Редукция данных

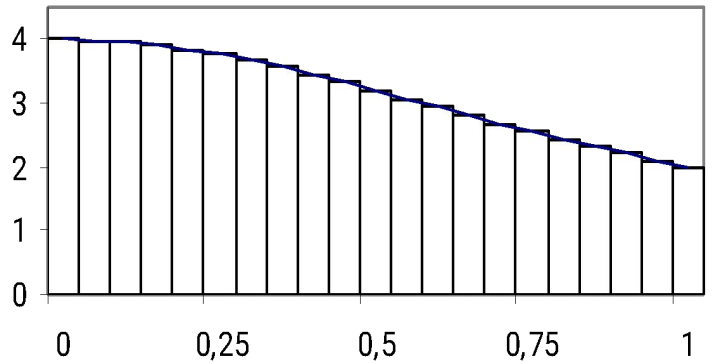
- Функция **`MPI_Reduce()`** коллективная, ее вызов должен быть выполнен всеми процессами коммуникатора **`comm`**.
- Все вызовы должны содержать одинаковые значения параметров **`count`**, **`type`**, **`op`**, **`root`**, **`comm`**.
- Передача данных должна быть выполнена всеми процессами.
- Результат операции будет получен только корневым (**`root`**) процессом.
- Операция **`op`** редукции выполняется над отдельными элементами передаваемого сообщения.

Коллективное взаимодействие…
Пример: вычисление числа π

- Значение π может быть вычислено через интеграл

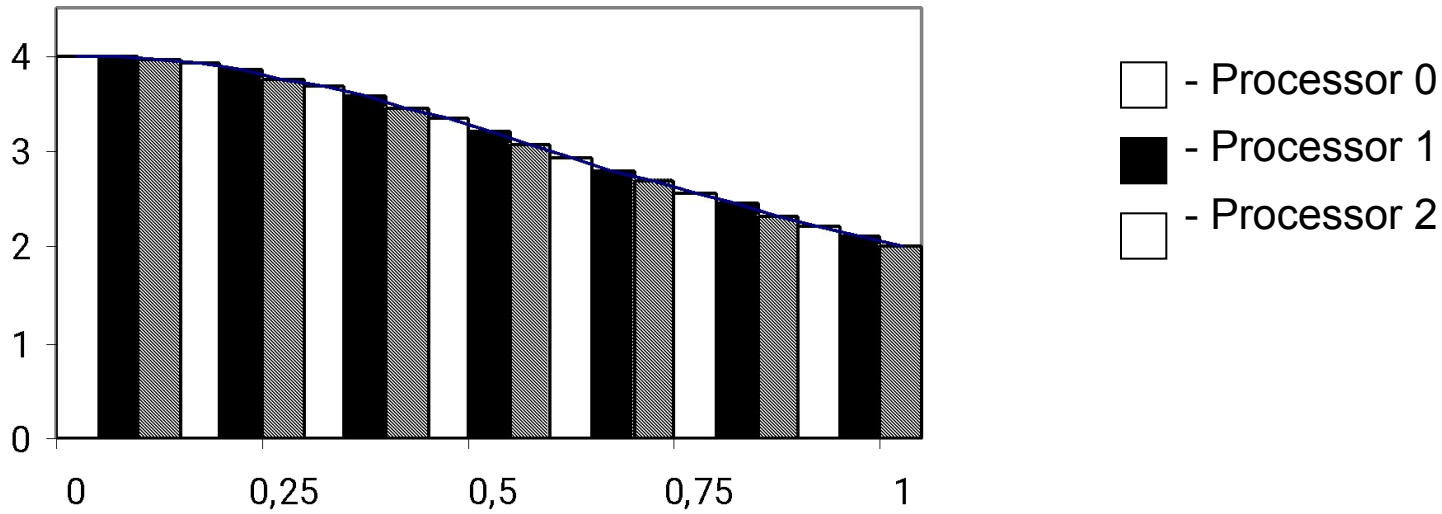
$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx$$

- Для вычисления определенного интеграла можно воспользоваться методом прямоугольников



Коллективное взаимодействие…
Пример: вычисление числа π

- Для распределения вычислений между процессами может использоваться циклическая схема
- Частичные суммы, посчитанные разными процессами, должны быть сложены



Коллективное взаимодействие…
Пример: вычисление числа π

```
#include "mpi.h"
#include <math.h>
double f(double a){
  return (4.0 / (1.0 + a*a));
}
void main(int argc, char *argv[]){
  int ProcRank, ProcNum, done = 0, n = 0, i;
  double PI25DT = 3.14159265358979323846264643;
  double mypi, pi, h, sum, x, t1, t2;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
  while (!done){ // главный цикл
    if (ProcRank == 0){
      printf("Enter the number of intervals: ");
      scanf("%d", &n);
      t1 = MPI_Wtime();
    }
```

Коллективное взаимодействие…
Пример: вычисление числа π

```
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (n > 0){ // вычисление локальных сумм
      h = 1.0 / (double) n;
      sum = 0.0;
      for (i = ProcRank + 1; i <= n; i += ProcNum){
        x = h * ((double)i - 0.5);
        sum += f(x);
      }
      mypi = h * sum;
      MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
        MPI_COMM_WORLD);
      if (ProcRank == 0){ // вывод результата
        t2 = MPI_Wtime();
        printf("pi = %.15f, Error = %.15f\n", pi, fabs(pi - PI25DT));
        printf("time = %f\n", t2 - t1);
      }
    }
    else done = 1;
  }
  MPI_Finalize();
}
```

Коллективное взаимодействие… Рассылка и сбор данных

- Чтобы распределить (“scatter”) данные от выбранного процесса остальным, используется следующая функция MPI:

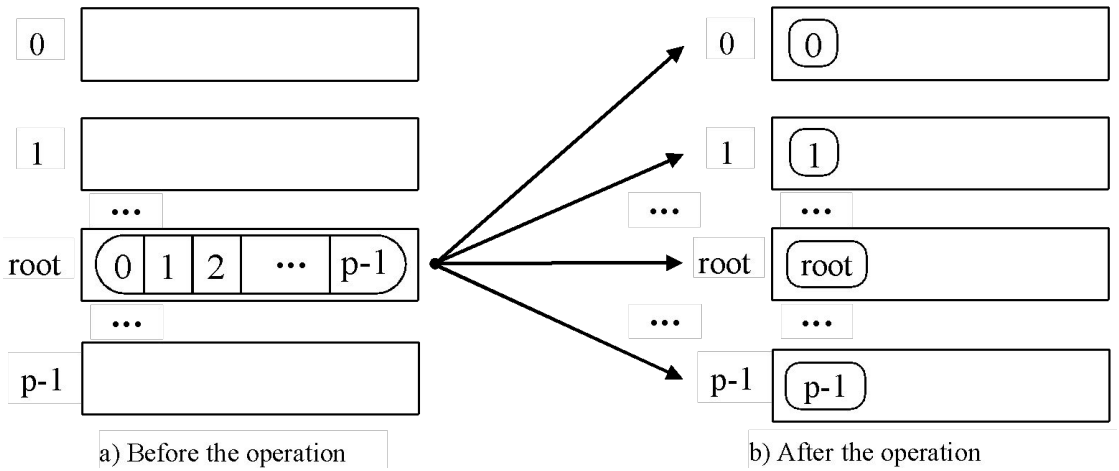
```
int MPI_Scatter(void *sbuf, int scount, MPI_Datatype stype,
                void *rbuf, int rcount, MPI_Datatype rtype,
                int root, MPI_Comm comm);
```

- **sbuf, scount, stype** – параметры передаваемого сообщения (scount определяет числе элементов, передаваемых каждому процессу)
- **rbuf, rcount, rtype** – параметры получаемого сообщения
- **root** – номер процесса, который выполняет распределение
- **comm** – коммуникатор, внутри которого должна быть выполнена операция

- Когда размер сообщения различается для каждого процесса, для рассылки используется функция **MPI_Scatterv()**

Коллективное взаимодействие… Рассылка и сбор данных

- Функция **`MPI_Scatter()`** должна быть вызвана всеми процессами коммуникатора **`comm`**.
- Корневой (**`root`**) процесс передает сообщения одинаковой длины (**`scount`**) из буфера **`sbuf`** всем процессам
- Каждый процесс (включая корневой) получает сообщение длины **`rcount=scount`** в буфер **`rbuf`**



Коллективное взаимодействие… Рассылка и сбор данных

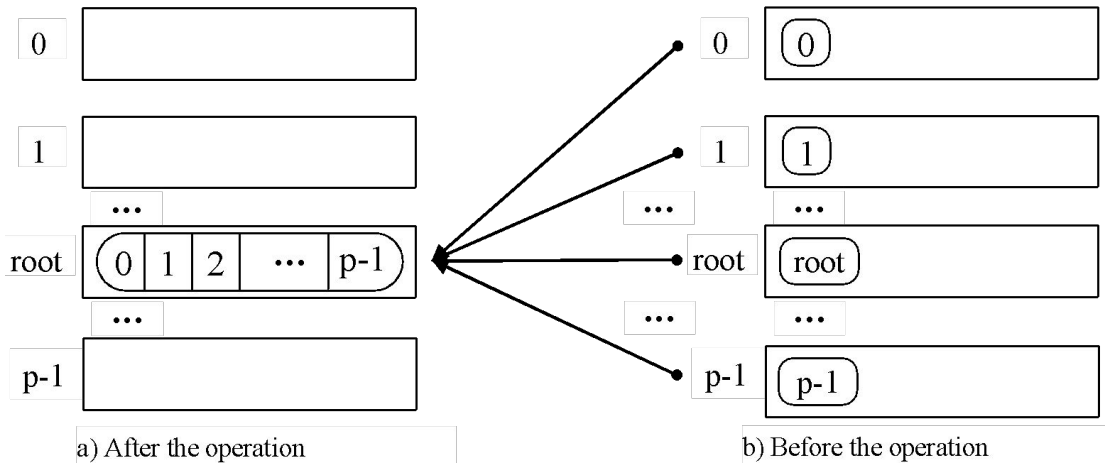
- Операция сбора данных от всех процессов в один противоположна рассылке. Для ее выполнения используется следующая функция MPI:

```
int MPI_Gather(void *sbuf, int scount, MPI_Datatype stype,
               void *rbuf, int rcount, MPI_Datatype rtype,
               int root, MPI_Comm comm);
```

- **sbuf, scount, stype** – параметры передаваемого сообщения
- **rbuf, rcount, rtype** – параметры получаемого сообщения
- **root** – номер процесса, который должен получить результат
- **comm** – коммуникатор, внутри которого операция должна быть выполнена

Коллективное взаимодействие… Рассылка и сбор данных

- Функция **`MPI_Gather()`** должна быть вызвана всеми процессами коммуникатора **`comm`**.
- Корневой (**`root`**) процесс получает от всех процессов сообщения одинаковой длины (**`rcount`**) в буфер **`rbuf`**
- Каждый процесс (включая корневой) передает данные длины **`scount=rcount`** из буфера **`sbuf`**



Коллективное взаимодействие… Пример: вычисление скалярного произведения

- Рассмотрим следующую задачу:

$$S = \sum_{i=1}^{n} a_i \cdot b_i$$

- Для разработки параллельной реализации необходимо:
  - разбить векторы a и b на “одинаковые” блоки
  - передать эти блоки процессам
  - вычислить частичные скалярные произведения в каждом процессе
  - редуцировать значения вычисленных частичных сумм в одном процессе и получить итоговый результат

Коллективное взаимодействие…
Пример: вычисление скалярного произведения

```
#include "mpi.h"
#include "stdio.h"

void main(int argc, char *argv[]) {
  double *a, *b;
  int i, n, ProcNum, ProcRank;
  double sum, sum_all;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
  if (ProcRank == 0){
    scanf("%d", &n);
    a = new double[n];
    b = new double[n];
    // инициализация векторов a и b
  }
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
  n = n / ProcNum;
```

Коллективное взаимодействие… Пример: вычисление скалярного произведения

```
  if (ProcRank != 0) {
    a = new double[n];
    b = new double[n];
  }
  MPI_Scatter(a, n, MPI_DOUBLE, a, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  MPI_Scatter(b, n, MPI_DOUBLE, b, n, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  sum = sum_all = 0;
  for (i = 0; i < n; i++)
    sum += a[i] * b[i];
  MPI_Reduce(&sum, &sum_all, 1, MPI_DOUBLE, MPI_SUM, 0,
    MPI_COMM_WORLD);
  if (ProcRank == 0)
    printf("\nInner product = %10.2f", sum_all);

  MPI_Finalize();
  delete [] a;
  delete [] b;
}
```

Коллективное взаимодействие… Взаимодействие «Каждый к каждому»

- Чтобы получить все собранные данные в каждом процессе коммуникатора, необходимо использовать функцию сбора и распределения **MPI_Allgather()**

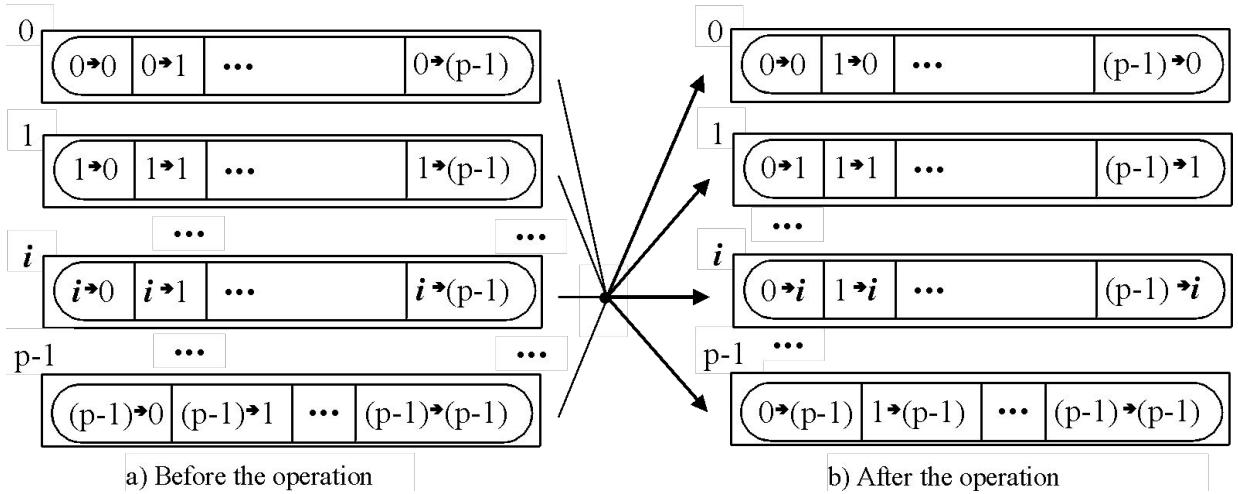
```
int MPI_Allgather(
  void *sbuf, int scount, MPI_Datatype stype,
  void *rbuf, int rcount, MPI_Datatype rtype,
  MPI_Comm comm);
```

- Сбор данных в общем случае, когда сообщения могут иметь разный размер, выполняется функциями **MPI_Gatherv()** и **MPI_Allgatherv()**

Коллективное взаимодействие… Взаимодействие «Каждый к каждому»

- Обмен данными между процессами выполняется функцией

```
int MPI_Alltoall(void *sbuf, int scount, MPI_Datatype stype,
  void *rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm);
```



- В случае, если сообщения могут иметь разную длину, обмен осуществляется функцией **`MPI_Alltoallv()`**

Коллективное взаимодействие… Взаимодействие «Каждый к каждому»

- Функция **`MPI_Reduce()`** позволяет получить данные только в одном процессе
- Для получения результата редукции в каждом процессе коммуникатора необходимо использовать функцию **`MPI_Allreduce()`**

```
int MPI_Allreduce(void *sendbuf, void *recvbuf, int count,
  MPI_Datatype type, MPI_Op Op, MPI_Comm comm);
```

Коллективное взаимодействие… Синхронизация вычислений

- Синхронизация, т.е. одновременное достижение разными процессами параллельной программы одной точки, выполняется функцией

```
int MPI_Barrier(MPI_Commcomm);
```

- Функция **`MPI_Barrier()`** коллективная
- Она должна быть вызвана каждым процессом коммуникатора **`comm`**
- При вызове функции **`MPI_Barrier()`** выполнение процесса приостанавливается. Выполнение продолжится только после вызова **`MPI_Barrier()`** всеми процессами коммуникатора

ВЗАИМОДЕЙСТВИЕ МЕЖДУ ДВУМЯ ПРОЦЕССАМИ

Режимы взаимодействия

Неблокирующее взаимодействие

Совмещенные прием и передача сообщений

Взаимодействие между двумя процессами… Режимы взаимодействия

Стандартный (**Standard**) режим:

- Предоставляется функцией **`MPI_Send()`**
- Отправляющий процесс блокируется на время выполнения функции
- Буфер может использоваться повторно после завершения выполнения функции
- Состояние передаваемого сообщения может быть разным в момент завершения, т.е. сообщение может располагаться в отправляющем процессе, передаваться, храниться в принимающем процессе или приниматься функцией **`MPI_Recv()`**

Взаимодействие между двумя процессами… Режимы взаимодействия

Синхронный (**Synchronous**) режим

- Функция передачи завершается лишь после подтверждения от принимающего процесса факта начала приема сообщения

`MPI_Ssend` – функция отправки сообщения в синхронном режиме

Режим «по готовности» (**Ready**)

- Может использоваться только если процесс приема уже инициирован. Буфер сообщения может повторно использоваться после завершения функции передачи сообщения.

`MPI_Rsend` – функция отправки сообщения в режиме по готовности

Взаимодействие между двумя процессами… Режимы взаимодействия

Буферизированный (**Buffered**) режим

- Предполагает использование дополнительного буфера для копирования передаваемых сообщений. Функция отправки сообщения завершается сразу после копирования сообщения в буфер.

MPI_Bsend – функция отправки сообщения в буферизированном режиме

- Для использования этого режима нужно создать MPI буфер для буферизации сообщений и передать его MPI

```
int MPI_Buffer_attach(void *buf, int size)
```
- **buf** – буфер для буферизированных сообщений
- **size** – размер буфера

- После завершения операций буфер должен быть отсоединен функцией

```
int MPI_Buffer_detach(void *buf, int *size);
```

Взаимодействие между двумя процессами… Режимы взаимодействия

- Режим **по готовности** формально самый быстрый, но используется довольно редко, так как обычно достаточно сложно обеспечить готовность принимать сообщения.
- **Стандартный** и **буферизированный** режимы могут быть выполнены достаточно быстро, но могут привести к ощутимому расходу ресурсов (памяти). В целом, они могут быть рекомендованы для передачи коротких сообщений.
- **Синхронный** режим самый медленный из всех, так как требует подтверждения приема. В то же время он является самым надежным. Может быть рекомендован для передачи длинных сообщений.

Взаимодействие между двумя процессами… Неблокирующие взаимодействие

- **Блокирующие функции** блокируют выполнение до тех пор, пока вызываемая функция не завершится
- **Неблокирующие функции** предоставляют возможность выполнять функции обмена данными без блокирования, чтобы осуществлять коммуникацию и вычисления параллельно
- Неблокирующий метод
  - Довольно сложный
  - Может вызывать падение эффективности параллельных вычислений из-за довольно медленных операций взаимодействия

Взаимодействие между двумя процессами… Неблокирующие взаимодействие

- Имена неблокирующих функций отличаются от соответствующих блокирующих префиксом **I** (Immediate, немедленный)

```
int MPI_Isend(void *buf, int count, MPI_Datatype type,
  int dest, int tag, MPI_Comm comm, MPI_Request *request);
int MPI_Issend(void *buf, int count, MPI_Datatype type,
  int dest, int tag, MPI_Comm comm, MPI_Request *request);
int MPI_Ibsend(void *buf, int count, MPI_Datatype type,
  int dest, int tag, MPI_Comm comm, MPI_Request *request);
int MPI_Irsend(void *buf, int count, MPI_Datatype type,
  int dest, int tag, MPI_Comm comm, MPI_Request *request);
int MPI_Irecv(void *buf, int count, MPI_Datatype type,
  int src, int tag, MPI_Comm comm, MPI_Request *request);
```

Взаимодействие между двумя процессами… Неблокирующие взаимодействие

- Состояние выполнения неблокирующей функции может быть проверено вызовом следующей функции

```
int MPI_Test( MPI_Request *request, int *flag,
  MPI_status *status);
```

- **request** - дескриптор операции, который определяется при вызове неблокирующей функции
- **flag** - результат проверки (=true, если операция завершилась)
- **status** - результат выполнения функции (только для завершенных операций)

- Функция MPI_Test неблокирующая

Взаимодействие между двумя процессами… Неблокирующие взаимодействие

- Возможна следующая схема совмещения вычислений и неблокирующего взаимодействия

```
MPI_Irecv(buf, count, type, dest, tag, comm, &request);
...
do {
   ...
  MPI_Test(&request, &flag, &status);
} while (!flag );
```

- Блокирующее ожидание завершения неблокирующей операции

```
int MPI_Wait(MPI_Request *request, MPI_status *status);
```

Взаимодействие между двумя процессами… Неблокирующие взаимодействие

- Дополнительные функции проверки и ожидания для неблокирующих операций обмена

MPI_Testall – проверяет завершение всех перечисленных операций
MPI_Waitall – ожидает завершения всех перечисленных операций
MPI_Testany – проверяет завершение хотя бы одной из перечисленных операций
MPI_Waitany – ожидает завершения любой из перечисленных операций
MPI_Testsome – проверяет завершение всех разрешенных операций в списке
MPI_Waitsome – ожидает завершения хотя бы одной из перечисленных операций и оценивает состояния всех операций

Взаимодействие между двумя процессами… Совмещенные прием и передача сообщении

- Эффективное одновременное выполнение приема и передачи данных выполняется функцией

```
int MPI_Sendrecv(void *sbuf, int scount, MPI_Datatype stype,
  int dest, int stag, void *rbuf, int rcount, MPI_Datatype
rtype,
  int src, int rtag, MPI_Comm comm, MPI_Status *status),
```

- **sbuf**, **scount**, **stype**, **dest**, **stag** – параметры передаваемого сообщения
- **rbuf**, **rcount**, **rtype**, **src**, **rtag** – параметры получаемого сообщения
- **comm** – коммуникатор, внутри которого происходит обмен данными
- **status** – результат выполнения операции

- В случае, если сообщения одинакового типа, MPI может использовать один буфер

```
int MPI_Sendrecv_replace(void *buf, int count,
  MPI_Datatype type, int dest, int stag, int source,
  int rtag, MPI_Comm comm, MPI_Status *status);
```

Резюме

- Рассмотрены операции коллективного взаимодействия
- Разобрано взаимодействие между двумя процессами
- Детально описаны режимы выполнения операций (стандартный, синхронный, буферизированный и по готовности)
- Рассмотрено неблокирующее взаимодействие между процессами для каждой операции
- Представлены примеры параллельных программ с использованием MPI

Упражнения

- Разработать простую программу для демонстрации каждой коллективной операции, доступной в MPI.
- Разработать реализацию коллективных операций с помощью парного взаимодействия. Провести вычислительный эксперимент и сравнить время выполнения разработанной программы с функциями коллективных операций MPI.
- Разработать программу, выполнить эксперименты и сравнить результаты работы разных алгоритмов сбора, обработки и распространения данных (функция **`MPI_Allreduce()`**).

Ссылки

1. Интернет-ресурс, описывающий стандарт MPI: http://www.mpiforum.org
2. Одна из самых широко используемых реализаций стандарта, библиотека MPICH, размещена по адресу http://www.mpich.org
3. Quinn, M.J. (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.
4. Pacheco, P. (1996). Parallel Programming with MPI. - Morgan Kaufmann.
5. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J. (1996). MPI: The Complete Reference. – MIT Press, Boston, 1996.
6. Group, W., Lusk, E., Skjellum, A. (1999). Using MPI – 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation). – MIT Press.
7. Group, W., Lusk, E., Thakur, R. (1999). Using MPI-2: Advanced Features of the Message Passing Interface (Scientific and Engineering Computation). – MIT Press.