**Lobachevsky State University of Nizhni Novgorod**

***Computing Mathematics and Cybernetics faculty***

***Параллельное программирование для многопроцессорных распределенных систем памяти***

# Лекция 1
# Основы MPI

Сысоев А.В. , Гергель В.П.

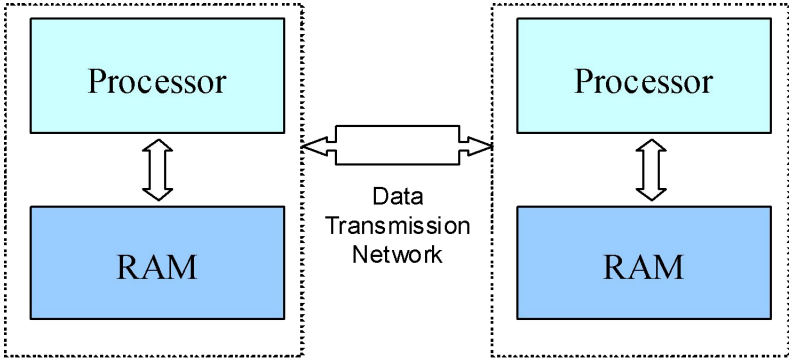
# Содержание

- Введение
- MPI: Основные понятия и определения
- Основы MPI
  - Инициализация и завершение программы MPI
  - Определение числа и ранга процессов
  - Операции отправки / получения сообщений
  - Оценка времени выполнения программы MPI
- Первая программа на MPI
- Введение в коллективную передачу данных

# ВВЕДЕНИЕ

# Введение…

- В вычислительных системах с распределенной памятью выслительные узлы работают независимо




- Необходимо иметь возможность:
  - распределять вычислительную нагрузку
  - организовывать информационное взаимодействие (передачу данных) между вычислительными узлами (далее узлы)

*Решение всех указанных вопросов обеспечивает MPI (message passing interface, интерфейс передачи сообщений)*

# Введение…

- При использовании MPI для решения задачи разрабатывается одна программа, которая запускается на выполнение одновременно на всех указанных узлах

- Эта модель реализации параллельных вычислений называется “одна программа множество процессов” или SPMP

# Введение…

- В MPI существует множество операций передачи данных:
  - Поддерживаются различные способы пересылки данных,
  - Реализованы практически все коммуникационные операции.

*Эти возможности являются основными преимуществами MPI (в частности, само название MPI свидетельствует об этом)*

# Введение…
# Что такое MPI?

- MPI – это стандарт организации передачи сообщений
- MPI – это программное обеспечение, которое должно обеспечивать возможность передачи сообщений и соответствовать всем требованиям стандарта MPI:
- Реализации MPI должны быть оформлены в виде библиотек программных модулей (библиотеки MPI)
- Реализации должны поддерживать широко используемые алгоритмические языки C и Fortran

# Введение…
# Преимущества MPI

- MPI позволяет в значительной степени уменьшить проблемы переносимости параллельных программ между различными компьютерными системами
- MPI способствует увеличению эффективности параллельных вычислений, так как реализации библиотеки MPI имеются практически для каждого типа вычислительной системы
- MPI снижает сложность разработки параллельных программ:
  - Большая часть основных операций передачи данных обеспечивается стандартом MPI
  - С использованием MPI разработано огромное число параллельных математических библиотек

# Введение

**История MPI (разработка стандарта MPI ведется международным консорциумом MPI Forum)**

- **1992 год** Начало исследований по интерфейсной библиотеке сообщений (Oak Ridge National Laboratory, Rice University)
- **Ноябрь 1992** Публикация рабочего варианта стандарта MPI-1
- **Ноябрь 1993** Обсуждение стандарта во время конференции Supercomputing'93
- **5 мая 1994** Окончательный вариант стандарта MPI-1.0
- **12 июня 1995** Новая версия стандарта – MPI-1.1
- **18 июля 1997** Опубликован стандарт MPI-2
- **21 сентября 2012** Опубликован стандарт MPI-3
- **4 июня 2015** Новая версия стандарта – MPI-3.1

# MPI: ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

# MPI: основные понятия и определения ... Концепция параллельной программы

- В рамках MPI под параллельной программой понимается несколько одновременно выполняемых процессов:
  - Процессы могут выполняться на разных процессорах, несколько процессов могут быть расположены на одном процессоре
  - Каждый параллельный процесс создается как копия одного и того же программного кода (модель SPMP)
- Исходный код разрабатывается на алгоритмических языках C или Fortran с использованием реализации библиотеки MPI
- Количество процессов определяется в момент запуска параллельной программы
- Все программные процессы последовательно нумеруются. Номер процесса называется рангом

# MPI: основные понятия и определения ...

- В основе MPI лежат четыре основные концепции:
  - Тип операций передачи данных
  - Тип данных, пересылаемых в сообщении
  - Концепция коммуникатора
  - Концепция виртуальной топологии

# MPI: основные понятия и определения ... Операции передачи данных

- Операции передачи данных составляют основу MPI
- Среди функций приема/передачи сообщений в MPI выделяют:
  - парные (*point-to-point*) операции, в которых участвуют два процесса,
  - коллективные (*collective*) операции, в которых одновременно участвуют несколько процессов.

# MPI: основные понятия и определения ... Коммуникаторы

- *Коммуникатор* в MPI – служебный объект, который объединяет в себе группу процессов и ряд дополнительных параметров (контекст):
  - Парные операции передачи данных выполняются для процессов, принадлежащих одному и тому же коммуникатору,
  - *Коллективные операции* применяются одновременно ко всем процессам коммуникатора
- Указание коммуникатора обязательно для любой операций передачи данных в MPI

# MPI: основные понятия и определения ... Коммуникаторы

- Во время вычислений могут быть созданы новые коммуникаторы, а уже существующие могут быть удалены
- Один и тот же процесс может принадлежать разным коммуникаторам
- Все процессы, доступные в параллельной программе, принадлежат коммуникатору с идентификатором MPI_COMM_WORLD, который создается по умолчанию
- Если необходимо передать данные между процессами, принадлежащими разным группам, необходимо создать *intercommunicator*.

# MPI: основные понятия и определения … Типы данных

- В любой операции передачи данных MPI необходимо указывать тип передаваемых данных
- MPI содержит широкий набор базовых типов данных, в основном совпадающих с типами данных алгоритмических языков C и Fortran
- MPI позволяет создавать производные типы данных для более точного описания содержимого пересылаемых сообщений

# MPI: основные понятия и определения
# Виртуальные топологии

- Логическая топология линий связи между процессами представляет собой полный граф (независимо от наличия реальных физических каналов связи между процессорами)
- MPI предоставляет возможность представить набор процессов как *решетку* произвольной размерности.
- MPI обеспечивает возможность формирования *логических (виртуальных) топологий* любого требуемого типа

# ОСНОВЫ MPI

- Инициализация и завершение MPI программы
- Определение числа и ранга процессов
- Операции отправки / получения сообщений
- Оценка времени выполнения MPI программы

# Основы MPI…
# Инициализация и завершение MPI программы

- **Первая вызываемая функция MPI** должна быть

```
int MPI_Init(int *agrc, char ***argv);
```

(служит для инициализации среды выполнения MPI-программы, параметрами функции являются количество аргументов командной строки и сами аргументы)

- **Последней вызываемой функцией MPI** должна быть

```
int MPI_Finalize(void);
```

# Основы MPI…
# Инициализация и завершение MPI программы

- Структура параллельной программы на основе MPI должна выглядеть следующим образом:

```
#include "mpi.h"
int main(int argc, char *argv[])
{
  <program code without the use of MPI functions>
  MPI_Init(&agrc, &argv);
  <program code with the use of MPI functions>
  MPI_Finalize();
  <program code without the use of MPI functions>
  return 0;
}
```

# Основы MPI…
# Определение числа и ранга процессов

- **Количество процессов** в выполняемой параллельной программе может быть получено с помощью следующей функции:

```
int MPI_Comm_size(MPI_Comm comm, int *size);
```

- Для определения **ранга процесса** используется функция

```
int MPI_Comm_rank(MPI_Commcomm, int *rank);
```

# Основы MPI…
# Определение числа и ранга процессов

- Как правило, функции **MPI_Comm_size()** и **MPI_Comm_rank()** вызываются сразу после **MPI_Init()**

```
#include "mpi.h"
int main(int argc, char *argv[])
{
  int ProcNum, ProcRank;
  <program code without the use of MPI functions>
  MPI_Init(&agrc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
  <program code with the use of MPI functions>
  MPI_Finalize();
  <program code without the use of MPI functions>
  return 0;
}
```

# Основы MPI…
# Определение числа и ранга процессов

- Коммуникатор **`MPI_COMM_WORLD`** создается по умолчанию и представляет все процессы параллельной программой, созданные при ее запуске

- Ранг, полученный с помощью функции **`MPI_Comm_rank()`**, представляет собой ранг процесса, который вызвал эту функцию, то есть переменная **`ProcRank`** будет принимать разные значения в разных процессах

# Основы MPI…
# Передача сообщений

- Для передачи данных процесс-отправитель должен выполнить функцию

```
int MPI_Send(void *buf, int count, MPI_Datatype type,
  int dest, int tag, MPI_Comm comm);
```

- **buf** – адрес буфера памяти, который содержит данные сообщения, подлежащие передаче
- **count** – количество элементов данных в сообщении
- **type** – тип элементов данных в сообщении
- **dest** – ранг процесса, который должен получить сообщение
- **tag** – значение тега, который используется для идентификации сообщения
- **comm** – коммуникатор, внутри которого передаются данные

# Основы MPI…
# Передача сообщений

## Типы данных MPI для алгоритмического языка C

| MPI_Datatype | C Datatype |
|---|---|
| MPI_CHAR | signed char |
| MPI_DOUBLE | double |
| MPI_FLOAT | float |
| MPI_INT | int |
| MPI_LONG | long |
| MPI_LONG_DOUBLE | long double |
| MPI_PACKED | |
| MPI_SHORT | short |
| MPI_UNSIGNED_CHAR | unsigned char |
| MPI_UNSIGNED | unsigned int |
| MPI_UNSIGNED_LONG | unsigned long |
| MPI_UNSIGNED_SHORT | unsigned short |

# Основы MPI…
# Передача сообщений

- Отправляемое сообщение определяется путем указания на блок памяти (**buffer**), который содержит сообщение.

  Триада, которая используется для указания буфера (**buf**, **count**, **type**), включается в параметры практически всех функций передачи данных
- Процессы, между которыми передаются данные, должны принадлежать коммуникатору, указанному в функции **MPI_Send ()**
- Параметр **tag** может использоваться, когда необходимо различать передаваемые сообщения
  В противном случае в качестве значения параметра используется произвольное целое число

# Основы MPI…
# Получение сообщений

- Для приема данных процесс-получатель должен выполнить функцию

```
int MPI_Recv(void *buf, int count, MPI_Datatype type,
  int source, int tag, MPI_Comm comm, MPI_Status *status);
```

- **buf** – адрес буфера памяти, который будет содержать принимаемые данные
- **count** – количество элементов данных в сообщении
- **type** – тип элементов данных в сообщении
- **source** – ранг процесса, от которого должно быть принято сообщение
- **tag** – тег сообщения
- **comm** – коммуникатор, внутри которого передаются данные
- **status** – указатель на структуру данных с информацией о результате выполнения операции передачи данных

# Основы MPI…
# Получение сообщений

- Буфер памяти должен быть достаточным для приема данных, а типы элементов отправленного и полученных сообщений должны совпадать.
  В случае нехватки памяти часть сообщения будет потеряна, а в коде завершения функции будет зарегистрирована ошибка переполнения

- Для получения сообщения от источника с любым рангом для параметра **`source`** может быть использовано значение **`MPI_ANY_SOURCE`**

- Для получения сообщения с любым тегом для параметра **`tag`** может быть указано значение **`MPI_ANY_TAG`**

# Основы MPI…
# Получение сообщений

- Параметр **status** позволяет определить ряд характеристик полученного сообщения

  - **status.MPI_SOURCE** – ранг процесса, который отправил полученное сообщение
  - **status.MPI_TAG** – тег полученного сообщения

- Функция

```
int MPI_Get_count(MPI_Status *status, MPI_Datatype type,
  int *count);
```

возвращает в переменной **count** количество элементов типа **type** в полученном сообщении

# Основы MPI…
# Получение сообщений

- Функция **`MPI_Recv()`** является блокирующей для процесса-получателя
- Выполнение процесса приостанавливается до тех пор, пока функция не завершит свою работу
- Если по какой-либо причине ожидаемое сообщение отсутствует, выполнение процесса будет блокировано

# Основы MPI…
# Оценка времени выполнения MPI программы

- Для оценки ускорения параллельной программы необходимо уметь получать время выполнения
- Получение текущего момента выполнения программы обеспечивается с помощью функции

```
double MPI_Wtime(void);
```

- Точность измерения времени может зависеть от среды выполнения параллельной программы.
  Для определения текущего значения точности измерения времени может использоваться функция

```
double MPI_Wtick(void);
```

# ПЕРВАЯ ПРОГРАММА НА MPI

# Первая параллельная MPI программа…

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if (ProcRank == 0)
{
  printf("Hello from %d!\n", ProcRank);
  for (int i = 1; i < ProcNum; i++)
  {
    MPI_Recv(&ProcRank, 1, MPI_INT, MPI_ANY_SOURCE,
      MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    printf("Hello from %d!\n", ProcRank);
  }
}
else
  MPI_Send(&ProcRank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
MPI_Finalize();
```

# Первая параллельная MPI программа…

- Каждый процесс узнает свой ранг, после чего все операции в программе разделяются (разные процессы выполняют разный код)
- Все процессы, за исключением процесса с рангом 0, отправляют значение его ранга нулевому процессу
- Нулевой процесс печатает значение своего ранга, получает сообщения от других процессов и последовательно печатает их ранги
- Возможный вариант результата выполнения программы

```
Hello from process 0
Hello from process 2
Hello from process 1
Hello from process 3
```

# Первая параллельная MPI программа…

- Следует отметить, что порядок приема сообщений не определен и зависит от условий выполнения параллельной программы.
  Более того, порядок может меняться от запуска к запуску.
- Если это не приводит к потерям эффективности, можно обеспечить однозначность порядка приема данных

```
MPI_Recv(&ProcRank, 1, MPI_INT, i, MPI_ANY_TAG,
  MPI_COMM_WORLD, &status);
```

Настройка ранга процесса отправки регулирует порядок приема сообщений

# Первая параллельная MPI программа

- Все функции MPI возвращают код завершения
- Если функция успешно завершена, код возврата - MPI_SUCCESS
- Другие значения кода завершения свидетельствуют о том, что в ходе выполнения функции были обнаружены ошибки
- Чтобы узнать тип обнаруженной ошибки, используются именованные константы, например

- **MPI_ERR_BUFFER** – неправильный указатель на буффер
- **MPI_ERR_COMM** – неправильный коммуникатор
- **MPI_ERR_RANK** – неправильный ранг процеса

# ВВЕДЕНИЕ В КОЛЛЕКТИВНУЮ ПЕРЕДАЧУ ДАННЫХ

# Введение в коллективную передачу данных… Проблема суммирования

- Рассмотрим задачу суммирования элементов вектора *x*

$$S = \sum_{i=1}^{n} x_i$$

- Для разработки параллельной реализации необходимо
  - разделить данные на «равные» блоки
  - распределить блоки по процессам
  - провести в каждом процессе суммирование полученных данных
  - собрать значения частичных сумм на одном из процессов
  - Сложить значения частичных сумм, чтобы получить общий результат

# Введение в коллективную передачу данных… Передача данных

- Предположим, что мы имеем p процессов и n % p = 0
- В этом случае размер каждого блока будет n / p
- Предположим, что только процесс с рангом 0 знает размер n и вектор x
- Перед распределением вектора x между процессами мы должны передать размер n каждому процессу, кроме нулевого
- Мы можем использовать следующий код

```
MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
for (i = 1; i < ProcNum; i++)
  MPI_Send(&n, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
```

- Эта версия очень неэффективна! Повторение передачи данных приводит к суммированию латентностей

# Введение в коллективную передачу данных… Передача данных

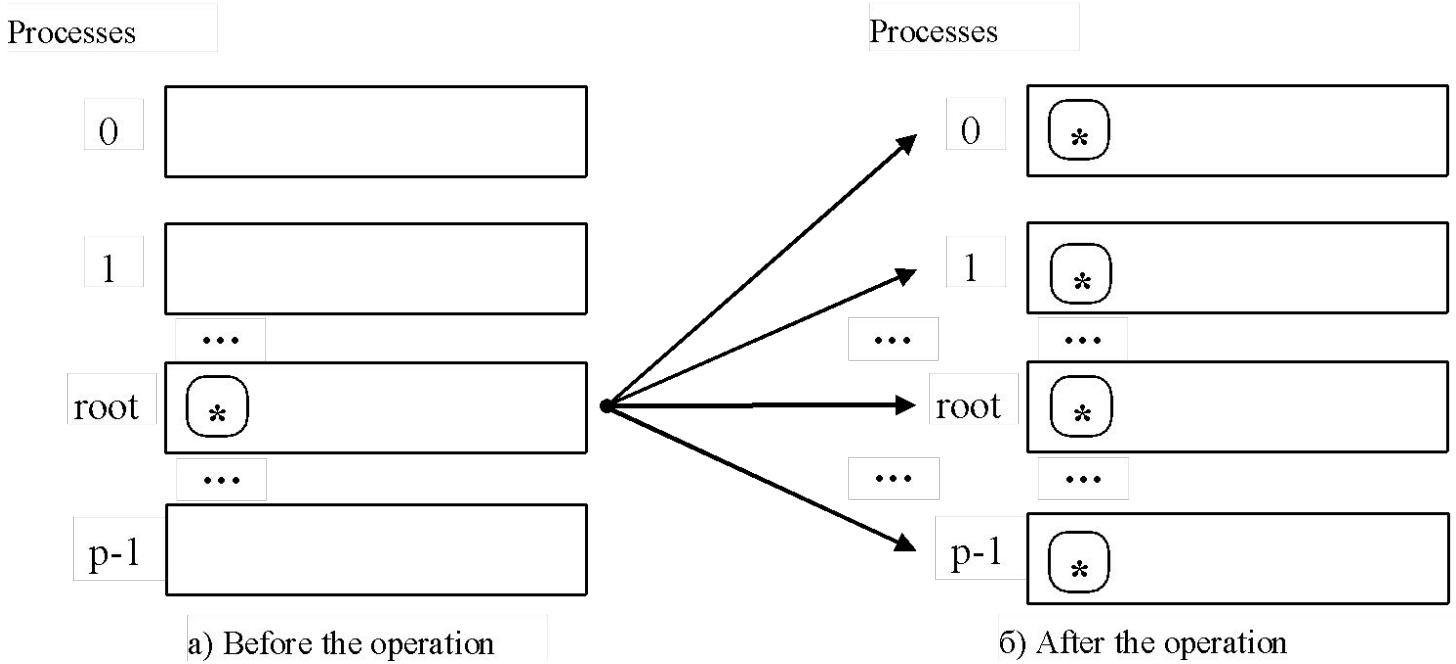
- Для обеспечения эффективной передачи можно использовать следующую функцию MPI

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

- **buf** – адрес буфера памяти, который содержит данные сообщения, подлежащие передаче
- **count** – количество элементов данных в сообщении
- **type** – тип элементов данных в сообщении
- **root** – ранг процесса, который осуществляет передачу данных
- **comm** – коммуникатор, внутри которого передаются данные

# Введение в коллективную передачу данных… Передача данных

- Функция **`MPI_Bcast()`** выполняет передачу данных из буффера **`buf`**, который содержит элементы типа **`count`**, от процесса с рангом **`root`** процессам внутри коммуникатора **`comm`**




а) Before the operation

б) After the operation

# Введение в коллективную передачу данных… Передача данных

- Функция **`MPI_Bcast()`** – это коллективная операция, ее вызов должен выполняться всеми процессами коммуникатора **`comm`**
- Буфер памяти, указанный в функции **`MPI_Bcast()`**, имеет разные назначения в разных процессах:
  - Для **`root`** процесса, из которого выполняется передача данных, этот буфер должен содержать переданное сообщение
  - Для остальных процессов буфер предназначен для приема данных
- Таким образом, мы можем изменить код на

```
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

# Введение в коллективную передачу данных… Распределение данных и вычислений

- Чтобы распределить вектор x среди процессов, мы можем использовать следующий код

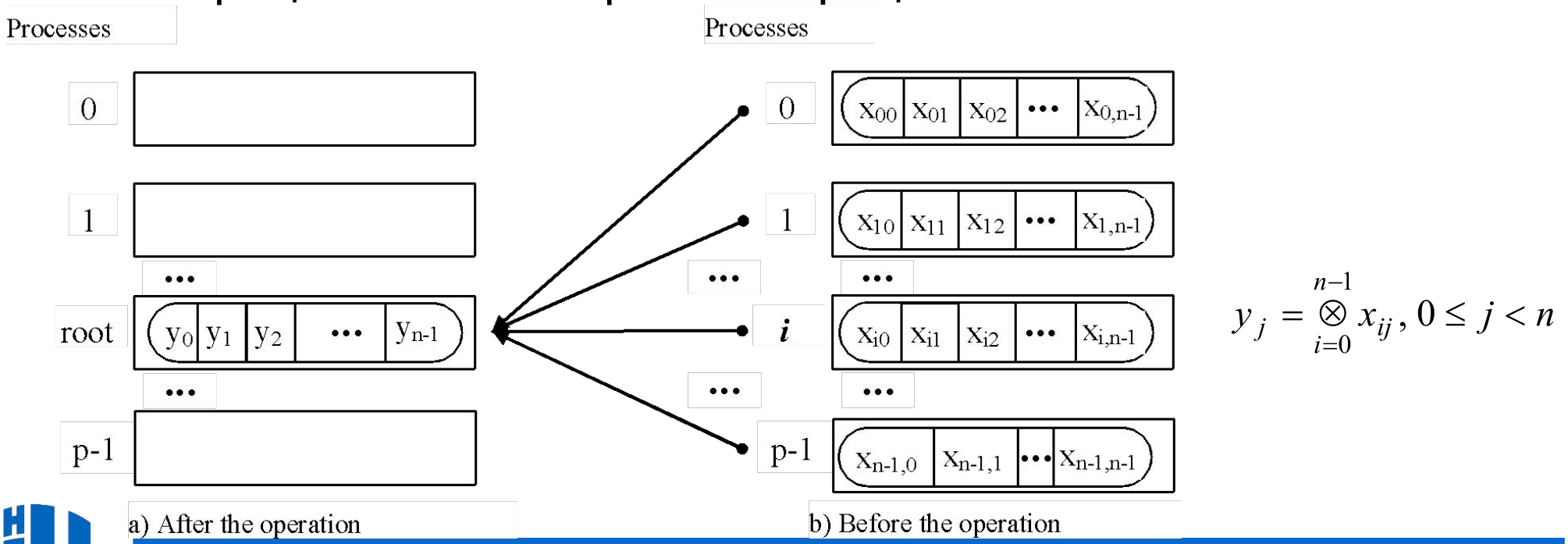
```
MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
for (i = 1; i < ProcNum; i++)
  MPI_Send(&x[n/ProcNum*i], n/ProcNum, MPI_DOUBLE, i, 0,
    MPI_COMM_WORLD);
```

- Этот код может быть реализован более эффективно (см. Лекцию 02)
- Теперь мы можем выполнить суммирование в каждом процессе, используя следующий простой код

```
sum = 0.0;
for (i = 0; i < n/ProcNum; i++)
  sum += x[i];
```

# Введение в коллективную передачу данных… Редукция данных

- Последний этап – собрать значения вычисленных частичных сумм и просуммировать их для получения общего результата
- Такая процедура сбора и суммирования данных является примером широко используемой операции редукции данных от всех процессов на выбранном процессе




$$y_j = \bigotimes_{i=0}^{n-1} x_{ij}, \; 0 \le j < n$$

# Введение в коллективную передачу данных… Редукция данных

- Для «редукции» данных со всех процессов на выбранном может использоваться 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 памяти с переданным сообщением
- **recvbuf** – буфер памяти с полученным сообщением (только для root процесса)
- **count** – количество элементов в сообщении
- **type** – тип элементов в сообщении
- **op** – операция, которая должна выполняться над данными
- **root** – ранг процесса, на котором должен быть получен результат
- **comm** – коммуникатор, внутри которого выполняется операция

# Введение в коллективную передачу данных… Редукция данных

- Основные типы операций MPI для редукции данных

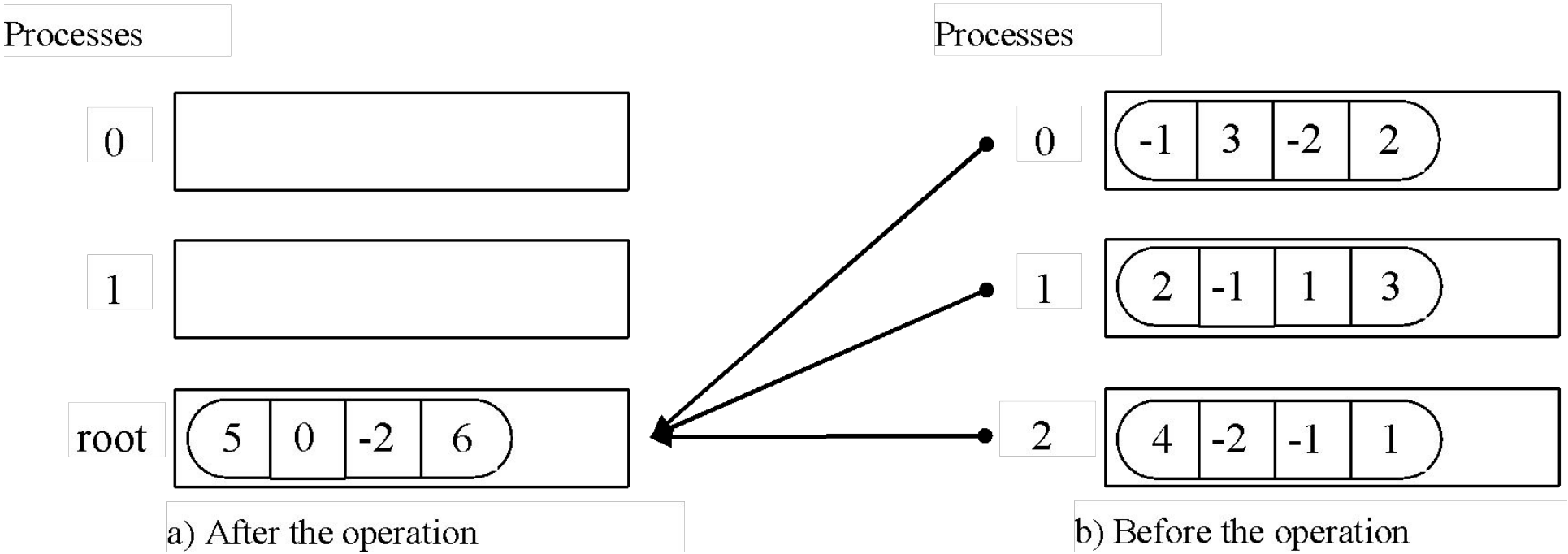
| Operation | Description |
|---|---|
| `MPI_MAX` | Вычисление максимального значения |
| `MPI_MIN` | Вычисление минимального значения |
| `MPI_SUM` | Вычисление суммы элементов |
| `MPI_PROD` | Вычисление произведения элементов |
| `MPI_LAND` | Выполнение логической операции “И” над значениями сообщения |
| `MPI_BAND` | Выполнение битовой операции “И” над значениями сообщения |
| `MPI_LOR` | Выполнение логической операции “ИЛИ” над значениями сообщения |
| `MPI_BOR` | Выполнение битовой операции “ИЛИ” над значениями сообщения |
| `MPI_LXOR` | Выполнение исключающей логической операции “ИЛИ” над значениями сообщения |
| `MPI_BXOR` | Выполнение исключающей битовой операции “ИЛИ” над значениями сообщения |
| `MPI_MAXLOC` | Вычисление максимальных значений и их индексов |
| `MPI_MINLOC` | Вычисление минимальных значений и их индексов |

# Введение в коллективную передачу данных… Редукция данных

- Функция **`MPI_Reduce()`** является коллективной операцией, ее вызов должен выполняться всеми процессами коммуникатора **`comm`**.
- Все вызовы должны содержать одинаковые значения параметров **`count`**, **`type`**, **`op`**, **`root`**, **`comm`**
- Передача данных должна осуществляться всеми процессами.
- Результат операции будет получен только **`root`** процессом.
- Выполнение операции **`op`** происходит над отдельными элементами передаваемых сообщений

# Введение в коллективную передачу данных… Сжатие данных

- Пример вычисления суммы значений



# Итоги

- Рассмотрен ряд концепций и определений, которые являются основными для стандарта MPI (параллельная программа, операции передачи сообщений, типы данных, коммуникаторы, виртуальные топологии)
- Дано краткое введение в разработку параллельных программ на основе MPI
- Представлены примеры параллельных программ на основе MPI

# Упражнения

- Разработать программу для нахождения минимального (максимального) значения среди элементов вектора
- Разработать программу вычисления скалярного произведения двух векторов

# Ссылки


1. Описание стандарта MPI: http://www.mpiforum.org
2. Одна из наиболее широко используемых реализаций MPI, библиотека MPICH 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.