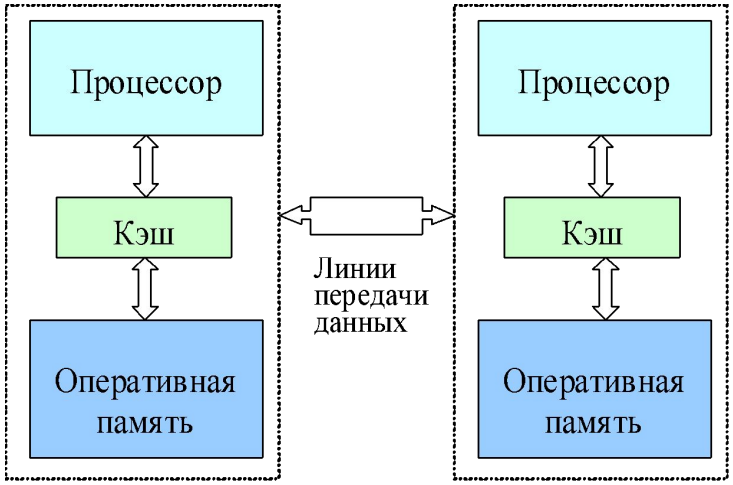
MPI

Message Passing Interface

Введение…

В вычислительных системах с распределенной памятью процессоры работают независимо друг от друга.

Для организации параллельных вычислений необходимо уметь:



- *распределять* вычислительную нагрузку,
- *организовать* информационное взаимодействие (*передачу данных*) между процессорами.

Решение всех перечисленных вопросов обеспечивает MPI - интерфейс передачи данных (message passing interface)

Введение…

- В рамках MPI для решения задачи разрабатывается одна программа, она запускается на выполнение одновременно на всех имеющихся процессорах
- Для организации различных вычислений на разных процессорах:
  - Есть возможность подставлять разные данные для программы на разных процессорах,
  - Имеются средства для идентификации процессора, на котором выполняется программа
- Такой способ организации параллельных вычислений обычно именуется как *модель "одна программа множество процессов"* (*single program multiple processes* or *SPMP*)

Введение…

- В MPI существует множество операций передачи данных:
  - Обеспечиваются разные способы пересылки данных,
  - Реализованы практически все основные коммуникационные операции.

Эти возможности являются наиболее сильной стороной MPI (об этом, в частности, свидетельствует и само название MPI)

Введение…

Что означает MPI?

- MPI - это стандарт, которому должны удовлетворять средства организации передачи сообщений.
- MPI – это программные средства, которые обеспечивают возможность передачи сообщений и при этом соответствуют всем требованиям стандарта MPI:
  - программные средства должны быть организованы в виде библиотек программных модулей (*библиотеки MPI*),
  - должны быть доступны для наиболее широко используемых алгоритмических языков C и Fortran.

Введение…

Достоинства MPI

- MPI позволяет существенно снизить остроту проблемы переносимости параллельных программ между разными компьютерными системами.
- MPI содействует повышению эффективности параллельных вычислений - практически для каждого типа вычислительных систем существуют реализации библиотек MPI.
- MPI уменьшает сложность разработки параллельных программ:
  - большая часть основных операций передачи данных предусматривается стандартом MPI,
  - имеется большое количество библиотек параллельных методов, созданных с использованием MPI.

MPI: основные понятия и определения…

В основу MPI положены четыре основные концепции:

- Тип операции передачи сообщения
- Тип данных, пересылаемых в сообщении
- Понятие коммуникатора (*группы процессов*)
- Понятие виртуальной топологии

Типы функций MPI

- функции инициализации и закрытия MPI-процессов;
- функции, реализующие коммуникационные операции типа точка-точка;
- функции, реализующие коллективные коммуникационные операции;
- функции для работы с группами процессов и коммуникаторами;
- функции для работы со структурами данных;
- функции формирования топологии процессов.

Способы выполнения функций MPI

1. ***Локальная функция –*** выполняется внутри вызывающего процесса. Ее завершение не требует коммуникаций.

2. ***Нелокальная функция*** – для ее завершения требуется выполнение MPI-процедуры другим процессом.

3. ***Глобальная функция*** – процедуру должны выполнять все процессы группы. Несоблюдение этого условия может приводить к зависанию задачи.

4. ***Блокирующая функция*** – возврат управления из процедуры гарантирует возможность повторного использования параметров, участвующих в вызове. Никаких изменений в состоянии процесса, вызвавшего блокирующий запрос, до выхода из процедуры не может происходить.

5. ***Неблокирующая функция*** – возврат из процедуры происходит немедленно, без ожидания окончания операции и до того, как будет разрешено повторное использование параметров, участвующих в запросе. Завершение неблокирующих операций осуществляется специальными функциями.

MPI: основные понятия и определения…

Операции передачи данных

- Основу MPI составляют операции передачи сообщений.
- Среди предусмотренных в составе MPI функций различаются:
  - парные (*point-to-point* – точка-точка) операции между двумя процессами,
  - коллективные (*collective*) коммуникационные действия для одновременного взаимодействия нескольких процессов.

MPI: основные понятия и определения…

Понятие коммуникаторов…

- *Коммуникатор* в MPI - специально создаваемый служебный объект, объединяющий в своем составе группу процессов и ряд дополнительных параметров (*контекст)*:
  - парные операции передачи данных выполняются для процессов, принадлежащих одному и тому же коммуникатору,
  - Коллективные операции применяются одновременно для всех процессов коммуникатора.
- Указание используемого коммуникатора является обязательным для операций передачи данных в MPI.

MPI: основные понятия и определения…

Понятие коммуникаторов

- В ходе вычислений могут создаваться новые и удаляться существующие коммуникаторы.
- Один и тот же процесс может принадлежать разным коммуникаторам.
- Все имеющиеся в параллельной программе процессы входят в состав создаваемого по умолчанию коммуникатора с идентификатором MPI_COMM_WORLD.
- При необходимости передачи данных между процессами из разных групп необходимо создавать глобальный коммуникатор (*intercommunicator*).

MPI: основные понятия и определения…

Типы данных

- При выполнении операций передачи сообщений для указания передаваемых или получаемых данных в функциях MPI необходимо указывать тип пересылаемых данных.
- MPI содержит большой набор *базовых типов данных*, во многом совпадающих с типами данных в алгоритмических языках C и Fortran.
- В MPI имеются возможности для создания новых *производных типов данных* для более точного и краткого описания содержимого пересылаемых сообщений.

Основы MPI…

- Передача сообщений…

Базовые типы данных MPI для алгоритмического языка C

MPI_Datatype	C Datatype
MPI_BYTE	
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**

MPI: основные понятия и определения

Виртуальные топологии

- Логическая топология линий связи между процессами имеет структуру полного графа (независимо от наличия реальных физических каналов связи между процессорами).
- В MPI имеется возможность представления множества процессов в виде *решетки* произвольной размерности При этом, граничные процессы решеток могут быть объявлены соседними и, тем самым, на основе решеток могут быть определены структуры типа *тор*.
- В MPI имеются средства и для формирования логических (виртуальных) топологий любого требуемого типа.

Введение в разработку параллельных программ с использованием MPI…

Основы MPI…

- Инициализация и завершение MPI программ
  - *Первой вызываемой функцией* MPI должна быть функция:

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

(служит для инициализации среды выполнения MPI программы; параметрами функции являются количество аргументов в командной строке и текст самой командной строки.)

  - *Последней вызываемой функцией* MPI обязательно должна являться функция:

```
int MPI_Finalize (void)
```

Введение в разработку параллельных программ с использованием MPI…

Основы MPI…

- Инициализация и завершение MPI программ
  - структура параллельной программы, разработанная с использованием MPI, должна иметь следующий вид:

```
#include "mpi.h"
int main ( int argc, char *argv[] ) {
  <программный код без использования MPI функций>
  MPI_Init ( &agrc, &argv );
    <программный код с использованием MPI функций
>
  MPI_Finalize();
  <программный код без использования MPI функций >
  return 0;
}
```

Введение в разработку параллельных программ с использованием MPI…

Основы MPI…

- Определение количества и ранга процессов…
  - Определение *количества процессов* в выполняемой параллельной программе осуществляется при помощи функции:

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

  - Для определения *ранга процесса* используется функция:

```
int MPI_Comm_rank ( MPI_Comm comm, int *rank )
```

Введение в разработку параллельных программ с использованием MPI…

Основы MPI…

- Определение количества и ранга процессов…
  - Как правило, вызов функций *MPI_Comm_size* и *MPI_Comm_rank* выполняется сразу после *MPI_Init*:

```
#include "mpi.h"
int main ( int argc, char *argv[] ) {
  int ProcNum, ProcRank;
  <программный код без использования MPI функций>
  MPI_Init ( &agrc, &argv );
  MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank);
    <программный код с использованием MPI функций >
  MPI_Finalize();
  <программный код без использования MPI функций >
  return 0;
}
```

Введение в разработку параллельных программ с использованием MPI…

Основы MPI…

- Определение количества и ранга процессов…
  - Коммуникатор *MPI_COMM_WORLD* создается по умолчанию и представляет все процессы выполняемой параллельной программы;
  - Ранг, получаемый при помощи функции *MPI_Comm_rank*, является рангом процесса, выполнившего вызов этой функции, и, тем самым, переменная *ProcRank* будет принимать различные значения в разных процессах.
  - *MPI_Comm_rank* = 0 .. *MPI_Comm_size*

Введение в разработку параллельных программ с использованием MPI…

Основы MPI…

- Передача сообщений…
  - Для *передачи сообщения* процесс-отправитель должен выполнить функцию:

```
int MPI_Send(void *buf, int count, MPI_Datatype type,
  int dest, int tag, MPI_Comm comm),
где
  - buf   - адрес буфера памяти, в котором располагаются
данные
            отправляемого сообщения,
  - count - количество элементов данных в сообщении,
  - type  - тип элементов данных пересылаемого сообщения,
  - dest  - ранг процесса, которому отправляется сообщение,
  - tag   - значение-тег, используемое для идентификации
            сообщений,
  - comm  - коммуникатор, в рамках которого выполняется
    передача данных.
```

Основы MPI…

- Передача сообщений…

Базовые типы данных MPI для алгоритмического языка C

MPI_Datatype	C Datatype
MPI_BYTE	
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**

Введение в разработку параллельных программ с использованием MPI…

Основы MPI…

- Передача сообщений
  - Отправляемое сообщение определяется через указание блока памяти (*буфера*), в котором это сообщение располагается. Используемая для указания буфера триада (*buf, count, type*) входит в состав параметров практически всех функций передачи данных,
  - Процессы, между которыми выполняется передача данных, обязательно должны принадлежать коммуникатору, указываемому в функции *MPI_Send*,
  - Параметр *tag* используется только при необходимости различения передаваемых сообщений, в противном случае в качестве значения параметра может быть использовано произвольное целое число.

Основы MPI…

- Прием сообщений…
  - Для *приема сообщения* 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),
где
  - buf, count, type – буфер памяти для приема сообщения
  - source - ранг процесса, от которого должен быть выполнен прием
             сообщения,
  - tag    - тег сообщения, которое должно быть принято для
             процесса,
  - comm   - коммуникатор, в рамках которого выполняется передача
             данных,
  - status – указатель на структуру данных с информацией о
             результате выполнения операции приема данных.
```

Основы MPI…

- Прием сообщений…
  - Буфер памяти должен быть достаточным для приема сообщения, а тип элементов передаваемого и принимаемого сообщения должны совпадать; при нехватке памяти часть сообщения будет потеряна и в коде завершения функции будет зафиксирована ошибка переполнения,
  - При необходимости приема сообщения от любого процесса-отправителя для параметра *source* может быть указано значение *MPI_ANY_SOURCE*,
  - При необходимости приема сообщения с любым тегом для параметра *tag* может быть указано значение *MPI_ANY_TAG*,

Введение в разработку параллельных программ с использованием MPI…

Для контроля правильности выполнения все функции MPI возвращают в качестве своего значения *код завершения*. При успешном выполнении функции возвращаемый код равен *MPI_SUCCESS*. Другие значения кода завершения свидетельствуют об обнаружении тех или иных ошибочных ситуаций в ходе выполнения функций:

- MPI_ERR_BUFFER – неправильный указатель на буфер,
- MPI_ERR_COMM – неправильный коммуникатор,
- MPI_ERR_RANK – неправильный ранг процесса
и др.

Операции передачи данных между двумя процессами

Одновременное выполнение передачи и приема

- Функция, позволяющая эффективно одновременно выполнить передачу и прием 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,
- status - структура данных с информацией о результате выполнения
операции.
```

```
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).
```

Основы MPI…

- Первая параллельная программа с использованием MPI (замечания)…
  - Можно рекомендовать при увеличении объема разрабатываемых программ выносить программный код разных процессов в отдельные программные модули (функции). Общая схема MPI программы в этом случае будет иметь вид:

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoProcess0();
else if ( ProcRank == 1 ) DoProcess1();
else if ( ProcRank == 2 ) DoProcess2();
```

Введение в разработку параллельных программ с использованием MPI…

Определение времени выполнение MPI программы

- Необходимо определять время выполнения вычислений для оценки достигаемого ускорения за счет использования параллелизма,
- Получение времени текущего момента выполнения программы обеспечивается при помощи функции:

```
double MPI_Wtime(void)
```

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

```
double MPI_Wtick(void)
```

(время в секундах между двумя последовательными показателями времени аппаратного таймера используемой системы)

Основы MPI…
Первая параллельная программа с использованием MPI…

```
#include "mpi.h "
int main(int argc, char* argv[]) {
  int ProcNum, ProcRank, RecvRank;
  MPI_Status Status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
  if ( ProcRank == 0 ) { // Действия для процесса 0
     printf ("\n Hello from process %3d", ProcRank);
     for ( int i=1; i < ProcNum; i++ ) {
       MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
         MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
       printf("\n Hello from process %3d", RecvRank);
    }
  }
  else // Действия для всех процессов, кроме процесса 0
    MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
  // Завершение работы
  MPI_Finalize();
  return 0;
 }
```

Основы MPI…

- Первая параллельная программа с использованием MPI…
  - Каждый процесс определяет свой ранг, после чего действия в программе разделяются (разные процессы выполняют различные действия),
  - Все процессы, кроме процесса с рангом 0, передают значение своего ранга нулевому процессу,
  - Процесс с рангом 0 сначала печатает значение своего ранга, а далее последовательно принимает сообщения с рангами процессов и также печатает их значения,
  - Возможный вариант результатов печати процесса 0:

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

Сумма вектора

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
int main(int argc, char* argv[]){
double x[100], TotalSum, ProcSum = 0.0;
int ProcRank, ProcNum, N=100;
MPI_Status Status;
// инициализация
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&ProcNum);
MPI_Comm_rank(MPI_COMM_WORLD,&ProcRank);
// подготовка данных
if ( ProcRank == 0 ) DataInitialization(x,N);
```

Сумма вектора

```
// рассылка данных на все процессы
    if ( ProcRank == 0 )
for ( int i=1; i < ProcNum; i++ )
MPI_Send(&X, N, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
    else
// прием вектора другими процессами
MPI_Recv(&X, N, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
&Status);

// вычисление частичной суммы на каждом из процессов
// на каждом процессе суммируются элементы вектора x
// от i1 до i2
int k = N / ProcNum;
int i1 = k * ProcRank;
int i2 = k * ( ProcRank + 1 );
if ( ProcRank == ProcNum-1 ) i2 = N;
for ( int i = i1; i < i2; i++ )
ProcSum = ProcSum + x[i];
```

Сумма вектора

```
// сборка частичных сумм на процессе с рангом 0
if ( ProcRank == 0 ) {
TotalSum = ProcSum;
for ( int i=1; i < ProcNum; i++ ) {
MPI_Recv(&ProcSum, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 0,
MPI_COMM_WORLD,&Status);
TotalSum = TotalSum + ProcSum;
}

else // все процессы отсылают свои частичные суммы
MPI_Send(&ProcSum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
}
// вывод результата
if ( ProcRank == 0 )
printf("\nTotal Sum = %10.2f",TotalSum);
Finalize();
}
```

Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от одного процесса всем процессам программы…
  - Необходимо передать значения вектора *x* всем процессам параллельной программы,
  - Можно воспользоваться рассмотренными ранее функциями парных операций передачи данных:

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

Повторение операций передачи приводит к суммированию затрат (латентностей) на подготовку передаваемых сообщений,

Данная операция может быть выполнена за меньшее число операций передачи данных.

Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

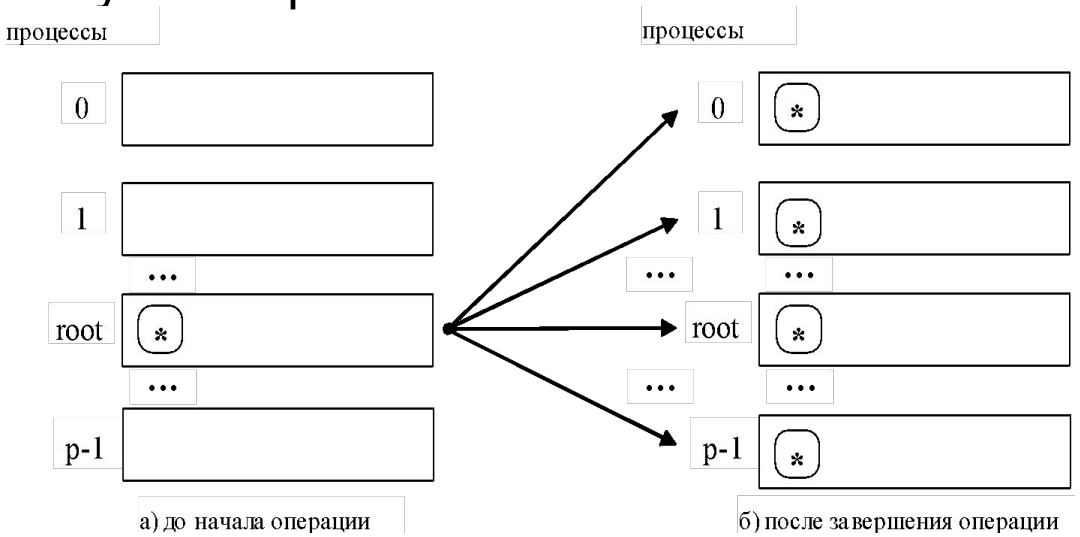
- Передача данных от одного процесса всем процессам программы…
  - *Широковещательная рассылка* данных может быть обеспечена при помощи функции MPI:

```
int MPI_Bcast(void *buf,int count,MPI_Datatype type,
    int root,MPI_Comm comm),
где
- buf, count, type – буфер памяти с отправляемым
          сообщением (для процесса с рангом 0), и для
          приема сообщений для всех остальных
процессов,
- root - ранг процесса, выполняющего рассылку
данных,
- comm - коммуникатор, в рамках которого выполняется
          передача данных.
```

Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от одного процесса всем процессам программы…
  - Функция *MPI_Bcast* осуществляет рассылку данных из буфера *buf*, содержащего *count* элементов типа *type* с процесса, имеющего номер *root*, всем процессам, входящим в коммуникатор *comm*



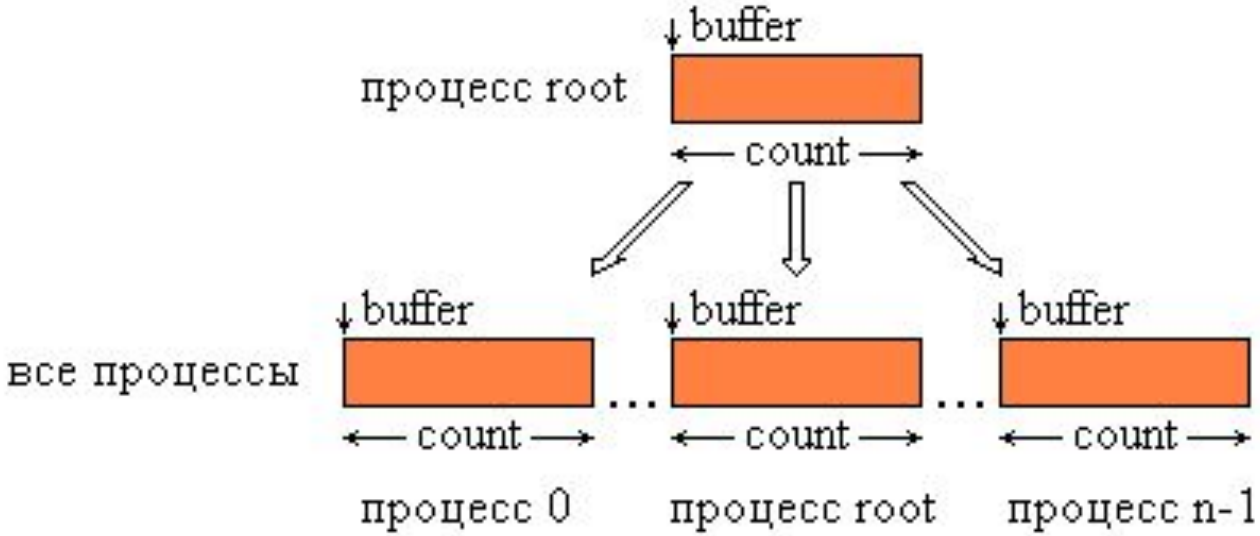
Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от одного процесса всем процессам программы
  - Функция *MPI_Bcast* определяет коллективную операцию, вызов функции *MPI_Bcast* должен быть осуществлен всеми процессами указываемого коммуникатора,
  - Указываемый в функции *MPI_Bcast* буфер памяти имеет различное назначение в разных процессах:
    - Для процесса с рангом *root*, с которого осуществляется рассылка данных, в этом буфере должно находиться рассылаемое сообщение.
    - Для всех остальных процессов указываемый буфер предназначен для приема передаваемых данных.

Программа

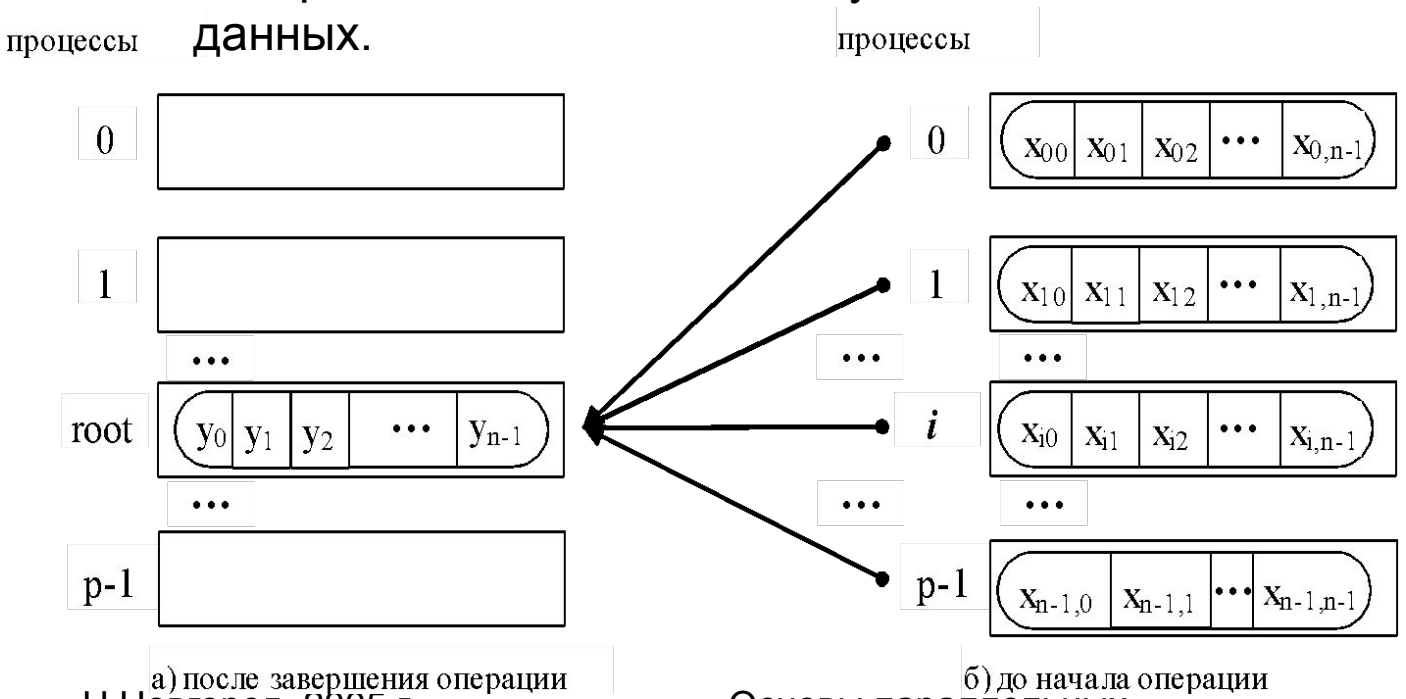
Графическая интерпретация операции Bcast



Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от всех процессов одному процессу…
  - Процедура сбора и последующего суммирования данных является примером часто выполняемой коллективной *операции передачи данных от всех процессов одному процессу*, в которой над собираемыми значениями осуществляется та или иная обработка данных.



а) после завершения операции

б) до начала операции

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

Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

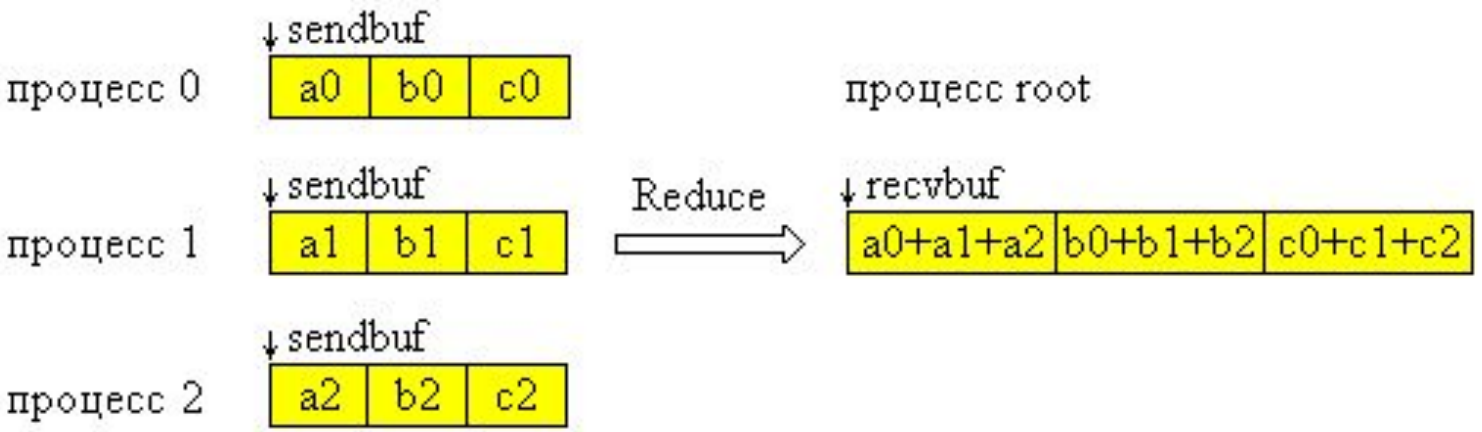
- Передача данных от всех процессов одному процессу…

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

где

- **sendbuf** – буфер памяти с отправляемым сообщением,
- **recvbuf** – буфер памяти для результирующего сообщения (только для процесса с рангом root),
- **count** – количество элементов в сообщениях,
- **type** – тип элементов сообщений,
- **op** – операция, которая должна быть выполнена над данными,
- **root** – ранг процесса, на котором должен быть получен результат,
- **comm** – коммуникатор, в рамках которого выполняется операция.

Графическая интерпретация операции Reduce



Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

- Типы операций MPI для функций редукции данных…

Операция	Описание
MPI_MAX	Определение максимального значения
MPI_MIN	Определение минимального значения
MPI_SUM	Определение суммы значений
MPI_PROD	Определение произведения значений
MPI_LAND	Выполнение логической операции "И" над значениями сообщений
MPI_BAND	Выполнение битовой операции "И" над значениями сообщений
MPI_LOR	Выполнение логической операции "ИЛИ" над значениями сообщений
MPI_BOR	Выполнение битовой операции "ИЛИ" над значениями сообщений
MPI_LXOR	Выполнение логической операции исключающего "ИЛИ" над значениями сообщений
MPI_BXOR	Выполнение битовой операции исключающего "ИЛИ" над значениями сообщений
MPI_MAXLOC	Определение максимальных значений и их индексов
MPI_MINLOC	Определение минимальных значений и их индексов

Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

- Типы операций MPI для функций редукции данных…
  - **MPI_MAX** и **MPI_MIN** ищут поэлементные максимум и минимум;
  - **MPI_SUM** вычисляет поэлементную сумму векторов;
  - **MPI_PROD** вычисляет поэлементное произведение векторов;
  - **MPI_LAND**, **MPI_BAND**, **MPI_LOR**, **MPI_BOR**, **MPI_LXOR**, **MPI_BXOR** - логические и двоичные операции И, ИЛИ, исключающее ИЛИ;
  - **MPI_MAXLOC**, **MPI_MINLOC** - поиск индексированного минимума/максимума

Введение в разработку параллельных программ с использованием MPI…

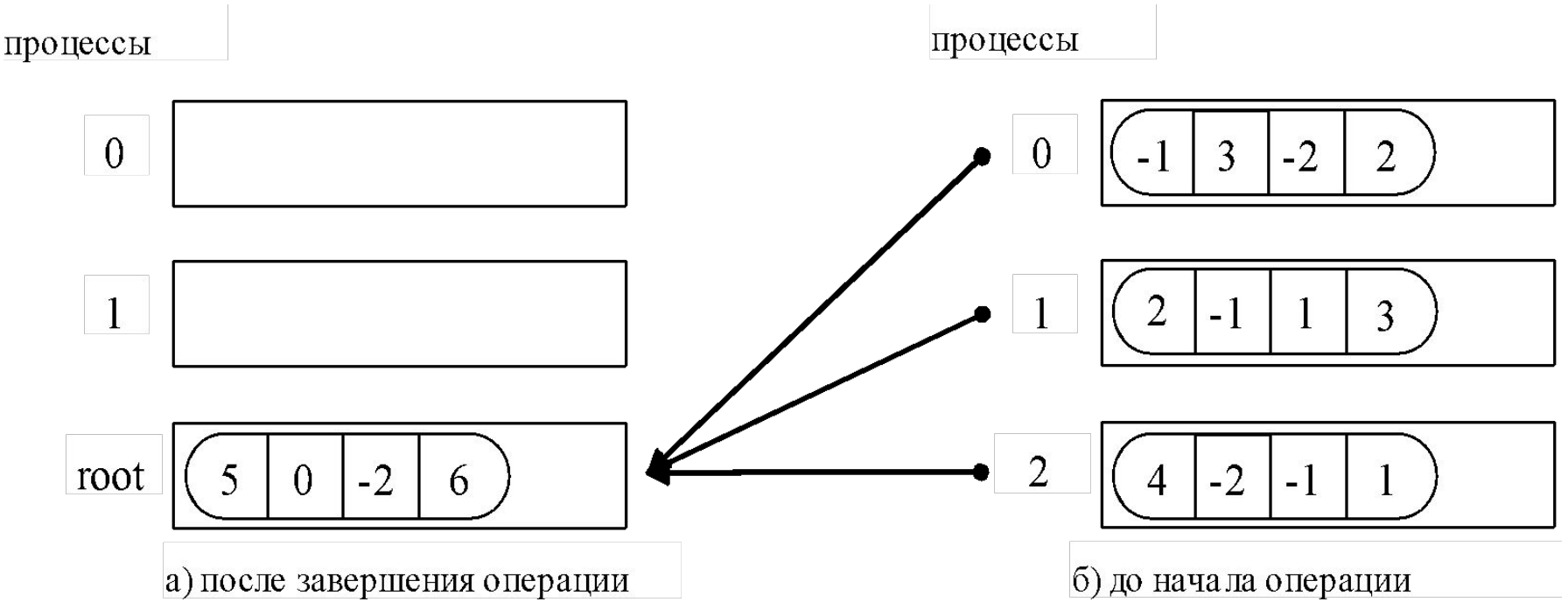
Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от всех процессов одному процессу…
  - Функция *MPI_Reduce* определяет коллективную операцию и, тем самым, вызов функции должен быть выполнен всеми процессами указываемого коммуникатора, все вызовы функции должны содержать одинаковые значения параметров *count*, *type*, *op*, *root*, *comm*,
  - Передача сообщений должна быть выполнена всеми процессами, результат операции будет получен только процессом с рангом *root*,
  - Выполнение операции редукции осуществляется над отдельными элементами передаваемых сообщений.

Введение в разработку параллельных программ с использованием MPI…

Начальное знакомство с коллективными операциями передачи данных…

- Передача данных от всех процессов одному процессу (пример для операции суммирования)



Сумма вектора
Использование коллективных функций передачи данных

```
// коллективная рассылка вектора

MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);



// Использование функции MPI_Reduce

// сборка частичных сумм на процессе с рангом 0
MPI_Reduce(&ProcSum,&TotalSum, 1, MPI_DOUBLE, MPI_SUM,
0, MPI_COMM_WORLD);
```

Начальное знакомство с коллективными операциями передачи данных

- Синхронизация 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);
```

  - Функция *MPI_Barrier* определяет коллективную операцию, при использовании должна вызываться всеми процессами коммуникатора.
  - Продолжение вычислений любого процесса произойдет только после выполнения функции *MPI_Barrier* всеми процессами коммуникатора.