Суперкомпьютерное моделирование и технологии.

сентябрь – декабрь 2016 г.

Лекция
21 сентября 2016 г.

Материалы лекции представлены на сайте

Задание 2. Исследование эффективности решения задачи Дирихле для уравнения Лапласа (один из вариантов)

- Задача Дирихле для уравнения Лапласа (1).

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} + \dfrac{\partial^2 u}{\partial z^2} = 0, (x, y, z) \in D, \\ u(x, y, z) = g(x, y, z), (x, y, z) \in D^0, \end{cases} \quad (1)$$

где $u(x, y, z)$ - функция, удовлетворяющая в области D уравнению Лапласа и принимающая на границе D^0 области D значения $g(x, y, z)$.

М.В.Абакумов, А.В.Гулин Лекции по численным методам математической физики. Учебное пособие. – М.-:ИНФРА-М, 2013.-158

Постановка задачи. Разностная схема.

- Численный подход к решению задачи (1) основан на замене производных соответствующими конечными разностями (2).

$$\frac{u_{i+1,j,k}-2u_{i,j,k}+u_{i-1,j,k}}{h^2}+\frac{u_{i,j+1,k}-2u_{i,j,k}+u_{i,j-1,k}}{h^2}+\frac{u_{i,j,k+1}-2u_{i,j,k}+u_{i,j,k-1}}{h^2}=0, \quad (2)$$

где $h \geq 0$ *- шаг сетки,* $u_{i,j,k}$ *- значение функции u(x, y, z) в точке* $x = x_i = ih, i = \overline{0, M+1}, y = y_j = jh, j = \overline{0, N+1}, z = z_k = kh, h = \overline{0, L+1},$ *где* M, N, L *- количество внутренних узлов по каждой координате в области* D.

Метод Якоби.

- *итерационный метод Якоби (3).*

$$u^{n}_{i,j,k} = (u^{n-1}_{i-1,j,k} + u^{n-1}_{i+1,j,k} + u^{n-1}_{i,j+1,k} + u^{n-1}_{i,j-1,k} + u^{n-1}_{i,j,k+1} + u^{n-1}_{i,j,k-1})/6$$

$$u^{n}_{i,j,k} = g_{i,j,k}, (x,y,z) \in D^{0}, n = 1,2,.. \qquad (3)$$

где n - номер итерации.

Уравнение Лапласа (2D)

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \qquad x, y \in [0,1] \qquad (1)$$

Краевые условия:

$$u(x,0) = \sin(\pi x) \qquad 0 \le x \le 1$$

$$u(x,1) = \sin(\pi x)e^{-x} \qquad 0 \le x \le 1 \qquad (2)$$

$$u(0, y) = u(1, y) = 0 \qquad 0 \le y \le 1$$

Аналитическое решение:

$$u(x, y) = \sin(\pi x)e^{-xy} \qquad x, y \in [0,1] \qquad (3)$$

Дискретизация уравнения Лапласа

$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} \qquad i = 1,2,\ldots,m; \;\; j = 1,2,\ldots,m \qquad (4)$$

где *n* и *n+1* текущий и следующий шаг,

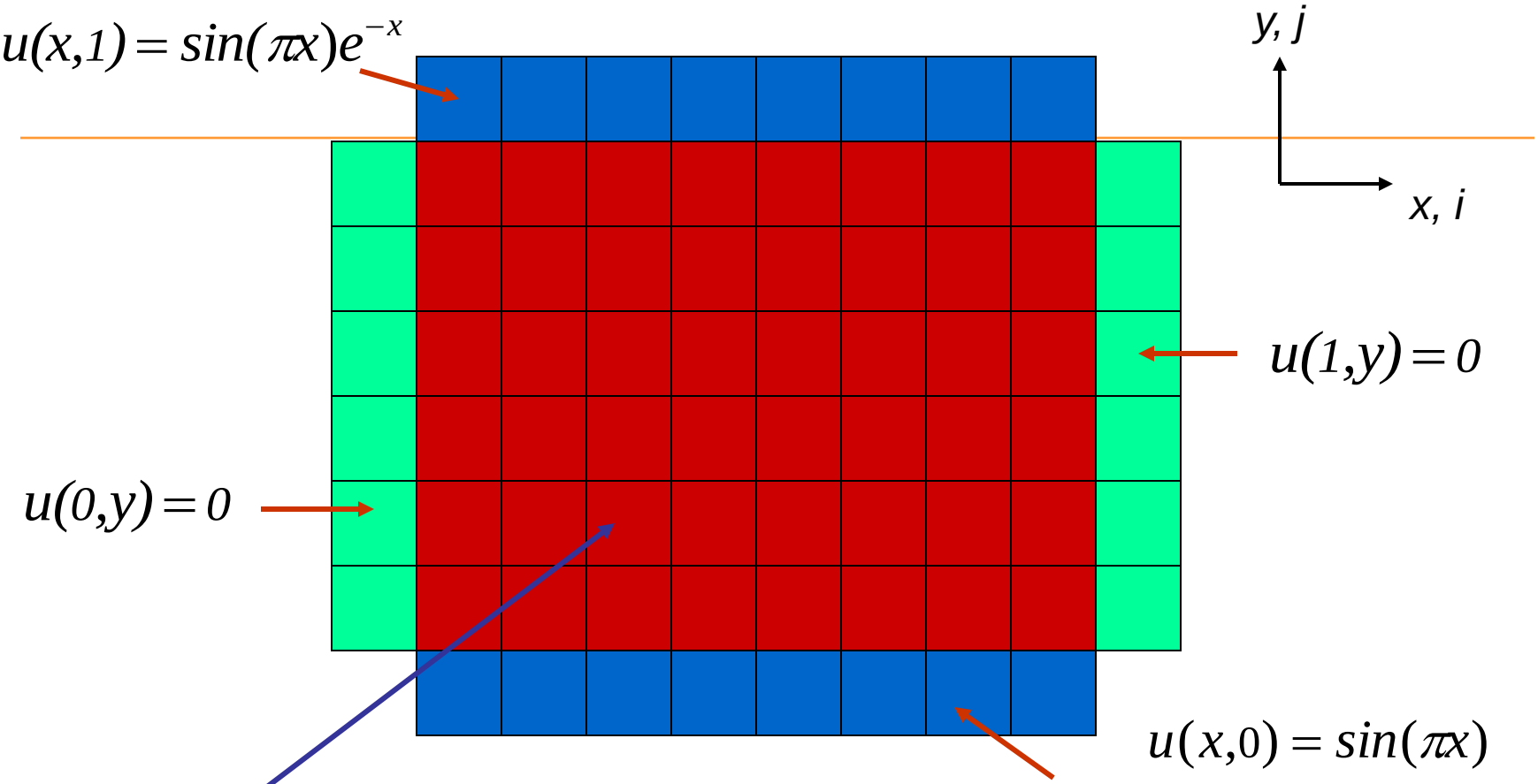
$$u_{i,j}^{n} = u^{n}(x_i, y_j) \qquad i = 0,1,2,\ldots,m+1; \;\; j = 0,1,2,\ldots,m+1 \qquad (5)$$

$$= u^{n}(i\Delta x,\, j\Delta y)$$

Для простоты

$$\Delta x = \Delta y = \frac{1}{m+1}$$

Вычислительная область



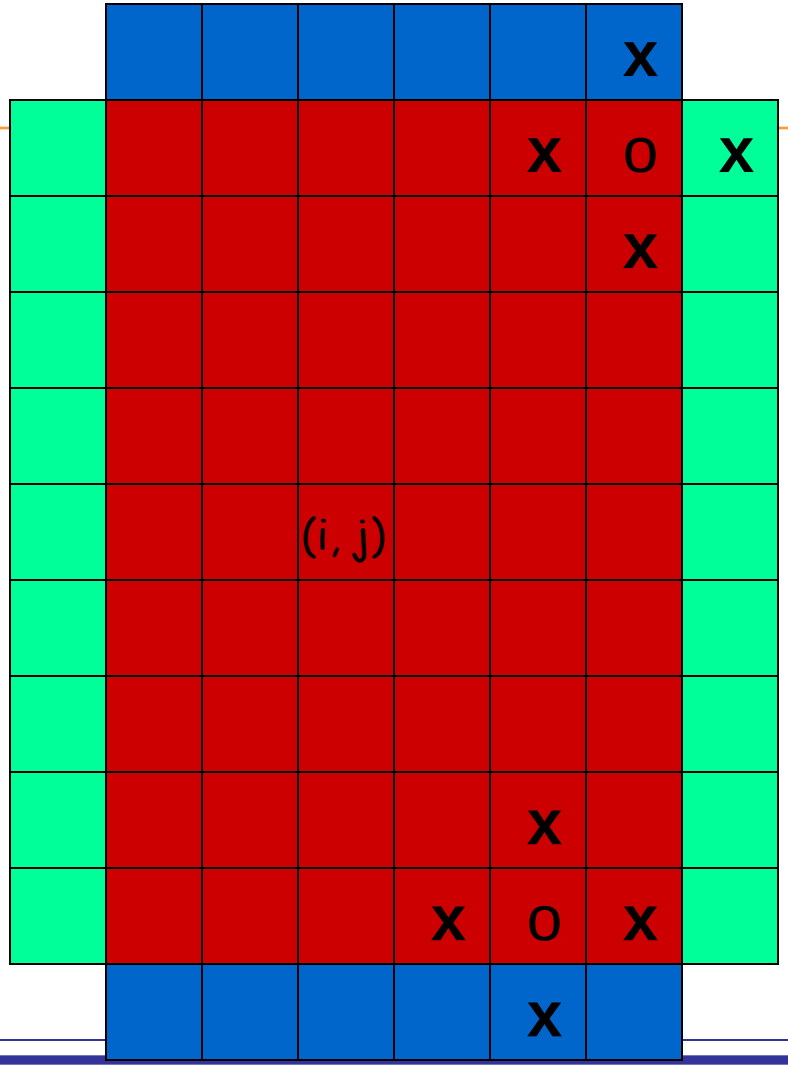
$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} \qquad i = 1,2,\ldots,m; \quad j = 1,2,\ldots,m$$

5-точечный шаблон

$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4}$$

внутренняя область, на которой ищется решение уравнения

граничная область.

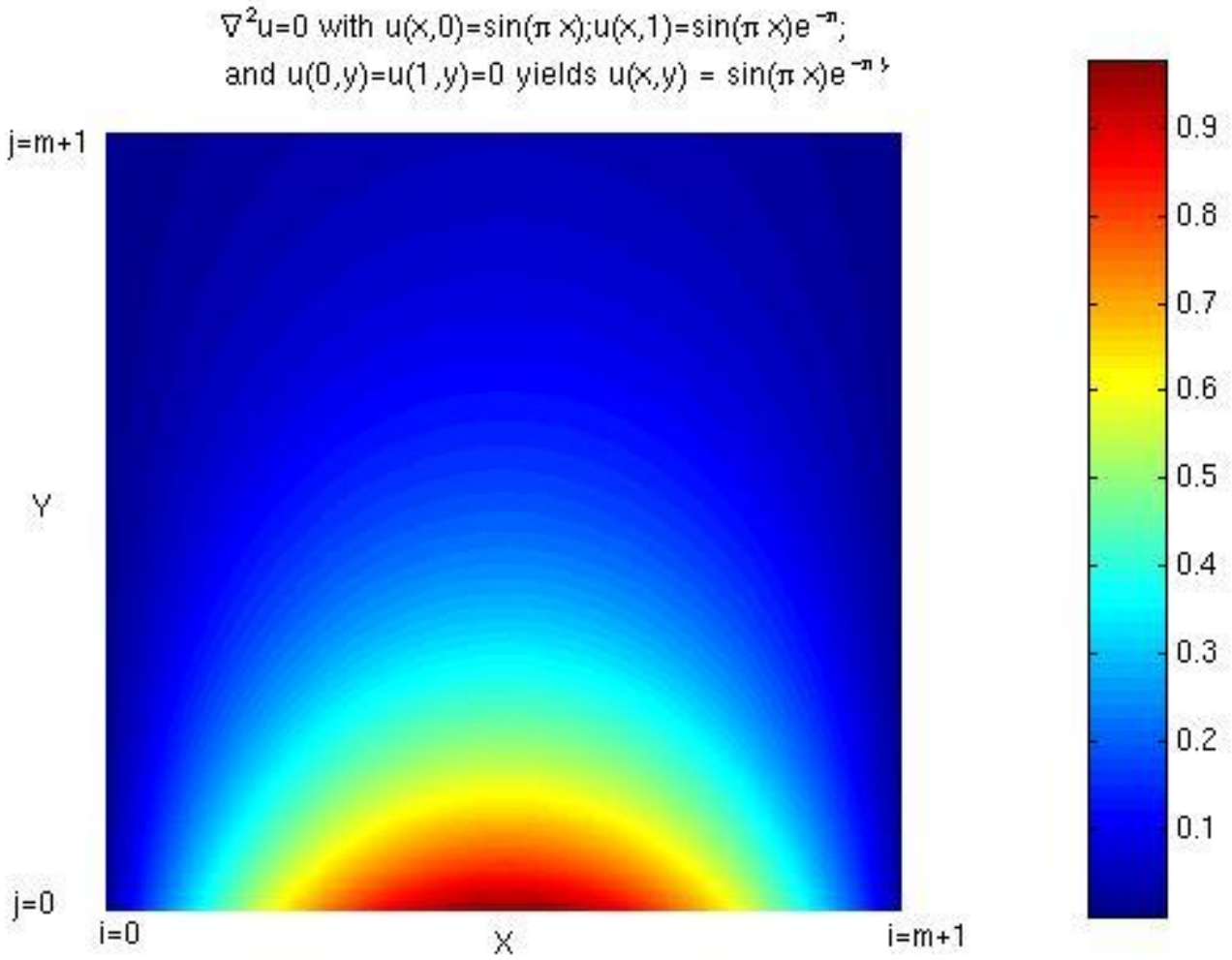


Метод Якоби

1. Задать начальные значения *и во всех внутренних точках (i,j) в момент времени n=0*.
2. Используя 5-точечный шаблон, вычислить значения во внутренних точках $u_{i,j}^{n+1}$ *(i,j)*.
3. Завершить процесс, если заданная точность достигнута.
4. Иначе: $u_{i,j}^{n} = u_{i,j}^{n+1}$ для всех внутренних точек.
5. Перейти на шаг 2.

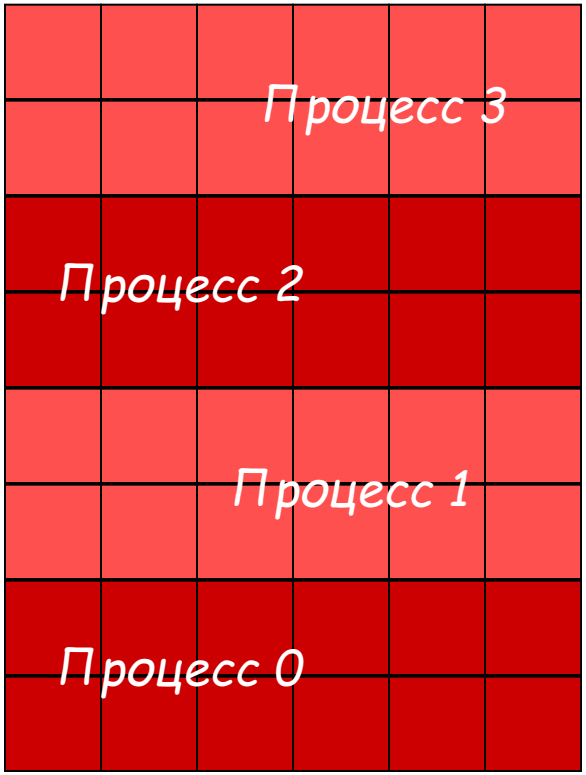
Это очень простая схема. Медленно сходится, поэтому не используется для решения реальных задач.

Решение (линии уровня)

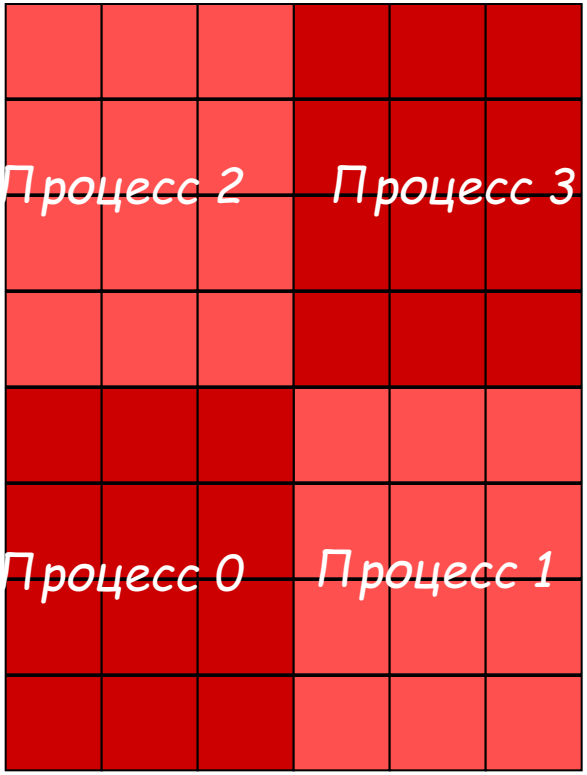


Параллельная реализация

1D Domain Decomposition



2D Domain Decomposition



Распределение данных – 1D

5-точечная схема требует значений от соседних потоков

Необходим обмен данными на границах области

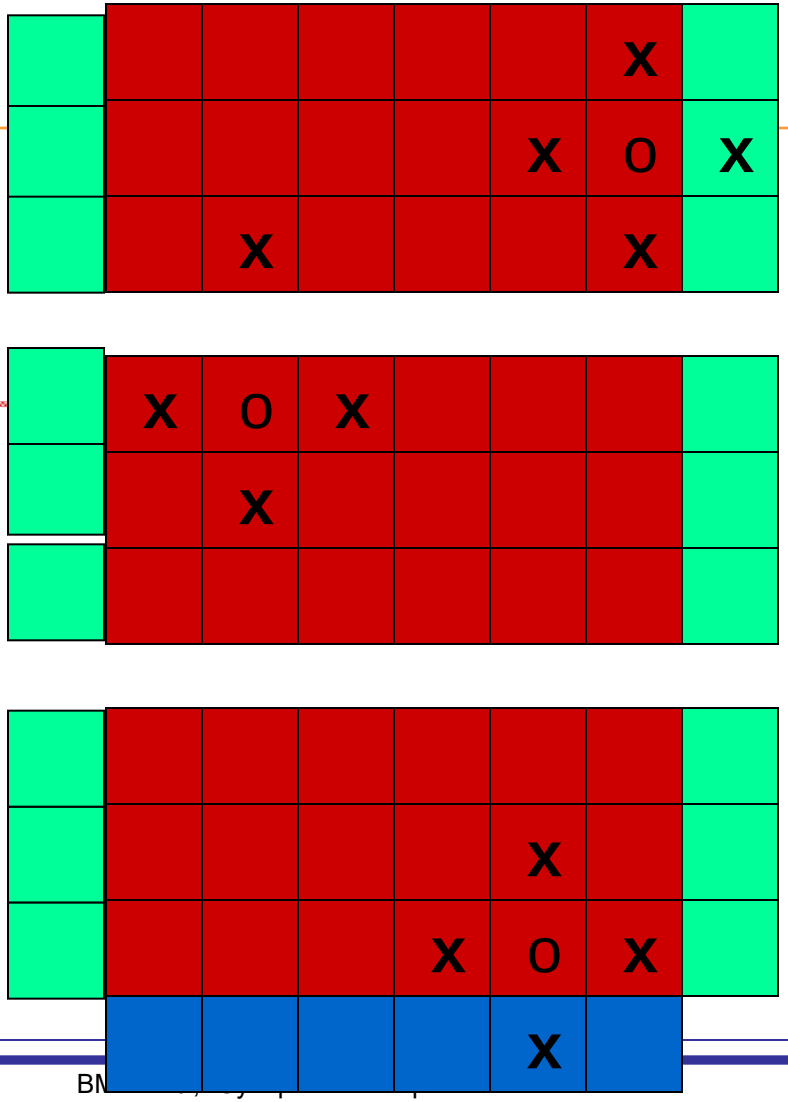
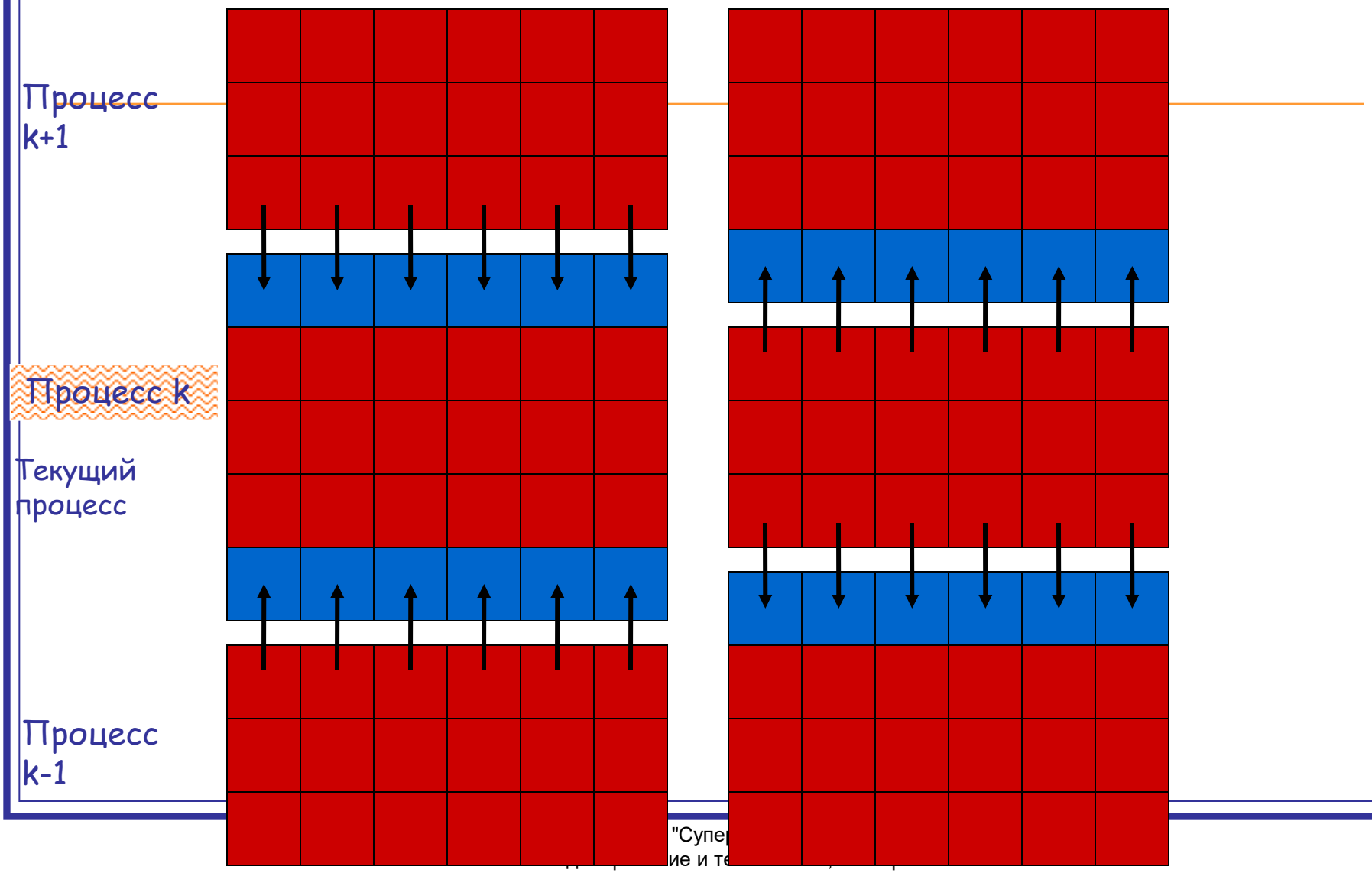


Схема передачи данных



Обзор технологий параллельного программирования.

Основные возможности MPI.

Пример параллельной программы программы (C, OpenMP)

Сумма элементов массива

```
#include <stdio.h>
#define N 1024
int main()
{ double sum;
  double a[N];
  int   i, n =N;
  for (i=0; i<n; i++){
    a[i] = i*0.5;  }
 sum =0;
#pragma omp for reduction (+:sum)
 for (i=0; i<n; i++)
  sum = sum+a[i];
printf (“Sum=%f\n”, sum);
}
```

Пример параллельной программы программы (C, MPI)

```
#include <stdio.h>
#include <mpi.h>
#define N 1024
int main(int argc, char *argv[])
{ double sum, all_sum;
  double a[N];
  int  i, n =N;
  int size, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
```

```
n= n/ size;
  for (i=rank*n;  i<n; i++){
    a[i] = i*0.5;  }

 sum =0;
for (i=rank*n; i<n; i++)
  sum = sum+a[i];
MPI_Reduce(& sum,& all_sum, 1,
 MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
If ( !rank)
printf ("Sum =%f\n", all_sum);
MPI_Finalize();
return 0;
}
```

Гибрид: MPI+OpenMP

```
#include <stdio.h>
#include <mpi.h>
#define N 1024
int main(int argc, char *argv[])
{ double sum, all_sum;
  double a[N];
  int  i, n =N;
  int size, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
```

```
n= n/ size;
  for (i=rank*n;  i<n; i++){
    a[i] = i*0.5;  }

 sum =0;
#pragma omp for reduction (+:sum)
for (i=rank*n; i<n; i++)
  sum = sum+a[i];
MPI_Reduce(& sum,& all_sum, 1,
 MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
If ( !rank)
printf ("Sum =%f\n", all_sum);
MPI_Finalize();
return 0;
}
```

MPI

- MPI 1.1 Standard разрабатывался 92-94
- MPI 2.0 - 95-97
- MPI 2.1 - 2008
- MPI 3.1 – 2015
- Стандарты
  - **http://mpi-forum.org/docs/**
  - **http://www.mpi-forum.org/docs/docs.html**

  Описание функций
  - **http://www-unix.mcs.anl.gov/mpi/www/**

Реализации MPI

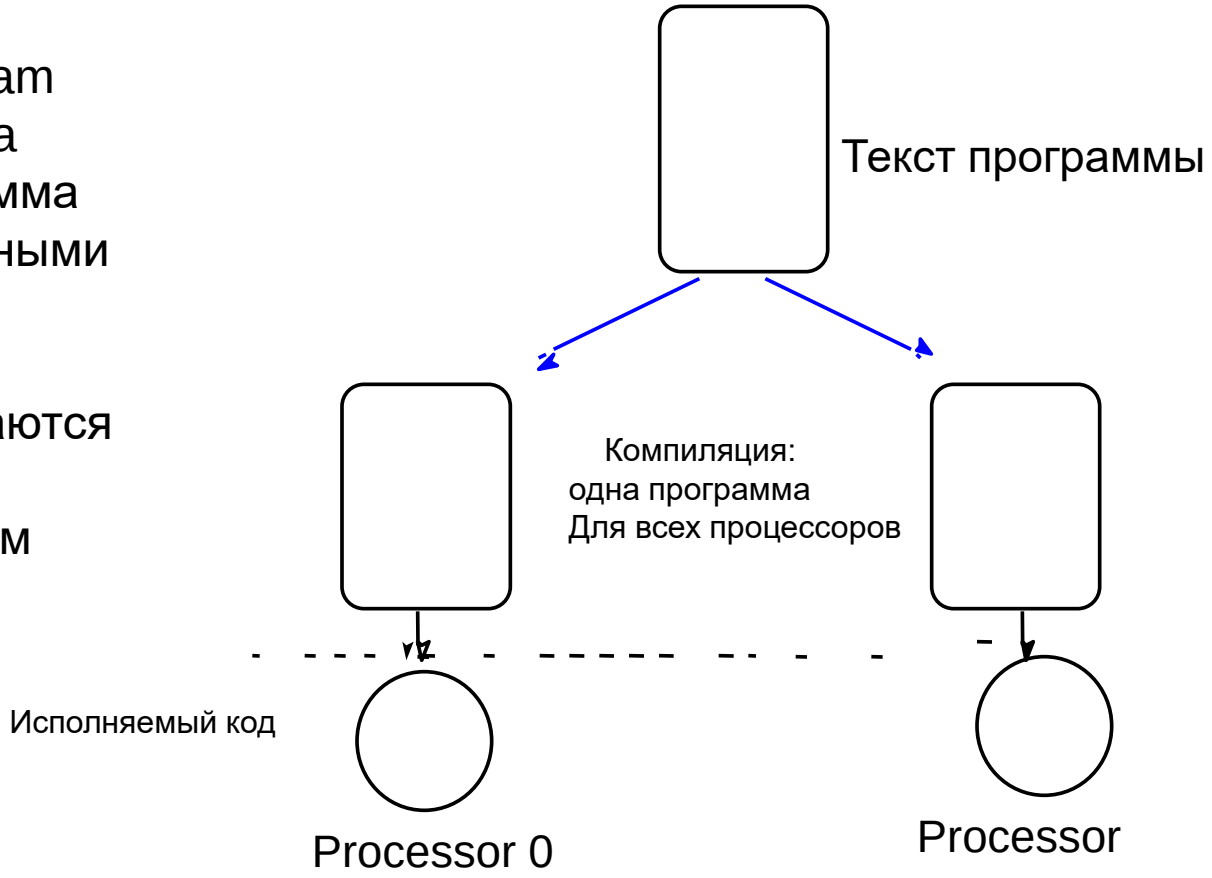
- MPICH
- LAM/MPI
- Mvapich
- OpenMPI
- Коммерческие реализации Intel,IBM и др.

Модель MPI

- Параллельная программа состоит из процессов, процессы могут быть многопоточными.
- MPI реализует передачу сообщений между процессами.
- Межпроцессное взаимодействие предполагает:
  - синхронизацию
  - перемещение данных из адресного пространства одного процесса в адресное пространство другого процесса.

Модель MPI-программ

- SPMD – Single Program Multiple Data
- Одна и та же программа выполняется различными процессорами
- Управляющими операторами выбираются различные части программы на каждом процессоре.

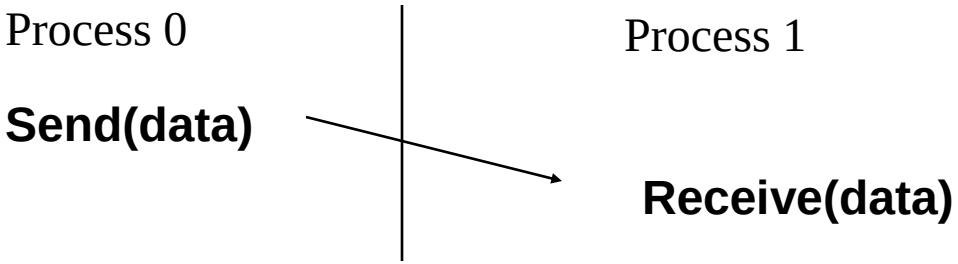


Модель выполнения MPI- программы

- Запуск: ***mpirun***
- При запуске указываем число требуемых процессоров **np** и название программы: пример: ***mpirun –np 3 prog***
- Каждый процесс MPI-программы получает два значения:
  - ***np –*** *число процессов*
  - ***rank*** из диапазона [0 … ***np-1***] – номер процесса
- Любые два процесса могут непосредственно обмениваться данными с помощью функций передачи сообщений

Основы передачи данных в MPI

- Технология передачи данных MPI предполагает кооперативный обмен.
- Данные посылаются одним процессом и принимаются другим.
- Передача и синхронизация совмещены.



Основные группы функций MPI

- Определение среды
- Передачи «точка-точка»
- Коллективные операции
- Производные типы данных
- Группы процессов
- Виртуальные топологии
- Односторонние передачи
- Параллельный ввод-вывод

Основные понятия

- Процессы объединяются в ***группы***.
- Каждое сообщение посылается в рамках некоторого контекста и должно быть получено в том же контексте.
- Группа и контекст вместе определяют коммуникатор.
- Процесс идентифицируется своим номером в группе, ассоциированной с коммуникатором.

Понятие коммуникатора MPI

- **Все** обращения к MPI функциям содержат коммуникатор, как параметр.
- Наиболее часто используемый коммуникатор **`MPI_COMM_WORLD:`**
  - определяется при вызове **`MPI_Init`**
  - содержит ВСЕ процессы программы
- Другие предопределенные коммуникаторы:
  - **MPI_COMM_SELF** – только один (собственный) процесс
  - **MPI_COMM_NULL** – пустой коммуникатор

Типы данных MPI

- Данные в сообщении описываются тройкой: (***address, count, datatype***), где
- ***datatype*** определяется рекурсивно как :
  - предопределенный базовый тип, соответствующий типу данных в базовом языке (например, MPI_INT, MPI_DOUBLE_PRECISION)
  - Непрерывный массив MPI типов
  - Векторный тип
  - Индексированыый тип
  - Произвольные структуры
- MPI включает функции для построения пользовательских типов данных, например, типа данных, описывающих пары (int, float).

Базовые MPI-типы данных

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Специальные типы MPI

- MPI_Comm
- MPI_Status
- MPI_datatype

Понятие тэга

- Сообщение сопровождается определяемым пользователем признаком – целым числом – ***тэгом*** для идентификации принимаемого сообщения

- Теги сообщений у отправителя и получателя должны быть согласованы. Можно указать в качестве значения тэга константу **`MPI_ANY_TAG`**.

MPI helloworld.c

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv){
    MPI_Init(&argc, &argv);
    printf("Hello, MPI world\n");
    MPI_Finalize();
    return 0; }
```

Функции определения среды

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

должна первым вызовом, вызывается только один раз

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

число процессов в коммуникаторе

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

номер процесса в коммуникаторе (нумерация с 0)

```
int MPI_Finalize()
```

завершает работу процесса

```
int MPI_Abort (MPI_Comm_size(MPI_Comm comm,
int*errorcode)
```

завершает работу программы

Инициализация MPI

- ***MPI_Init*** должна быть первым вызовом, вызывается только один раз

C:

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

http://www.mcs.anl.gov/research/projects/mpi/www/www3/MPI_Init.html

Обработка ошибок MPI-функций

Определяется константой ***MPI_SUCCESS***

```
I
int error;
……
error = MPI_Init(&argc, &argv));
If (error != MPI_SUCCESS)
{
 fprintf (stderr, “ MPI_Init error \n”);
return 1;

}
```

MPI_Comm_size
Количество процессов в коммуникаторе

- **Размер коммуникатора**

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

Результат – число процессов

http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Comm_size.html

MPI_Comm_rank
номер процесса (process rank)

- Process ID в коммуникаторе
  - Начинается с 0 до (*n*-1), где *n* – число процессов
- Используется для определения номера процесса-отправителя и получателя

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

Результат – номер процесса

Завершение MPI-процессов

- Никаких вызовов MPI функций после

int MPI_Finalize()

int MPI_Abort (MPI_Comm_size(MPI_Comm comm, int*errorcode)

Hello, MPI world!

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv){
   int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello, MPI world! I am %d of %d\n“,rank,size);
    MPI_Finalize();
    return 0; }
```

Трансляция MPI-программ

- Трансляция

  mpicc –o <имя_программы> <имя>.c <опции>

  Например:

  mpicc –o hw helloworld.c
- Запуск в интерактивном режиме

  mpirun –np 128 hw

Взаимодействие «точка-точка»

- Самая простая форма обмена сообщением
- Один процесс посылает сообщения другому
- Несколько вариантов реализации того, как пересылка и выполнение программы совмещаются

Варианты передачи «точка-точка»

- Синхронные пересылки
- Асинронные передачи
- Блокирующие передачи
- Неблокирующие передачи

Функции MPI передачи «точка-точка»

Point-to-Point Communication Routines		
MPI_Bsend	MPI_Bsend_init	MPI_Buffer_attach
MPI_Buffer_detach	MPI_Cancel	MPI_Get_count
MPI_Get_elements	MPI_Ibsend	MPI_Iprobe
MPI_Irecv	MPI_Irsend	MPI_Isend
MPI_Issend	MPI_Probe	MPI_Recv
MPI_Recv_init	MPI_Request_free	MPI_Rsend
MPI_Rsend_init	MPI_Send	MPI_Send_init
MPI_Sendrecv	MPI_Sendrecv_replace	MPI_Ssend
MPI_Ssend_init	MPI_Start	MPI_Startall
MPI_Test	MPI_Test_cancelled	MPI_Testall
MPI_Testany	MPI_Testsome	MPI_Wait
MPI_Waitall	MPI_Waitany	MPI_Waitsome

Блокирующие и неблокирующие передачи

- Определяют, при каких условиях операции передачи завершаются:

  - **Блокирующие**: возврат из функций передачи сообщений только по завершению передачи
  - **Неблокирующие** : немедленный возврат из функций, пользователь должен контролировать завершение передач

Основа 2-точечных обменов

*int MPI_Send(void *buf,int count, MPI_Datatype datatype,int dest, int tag,*
MPI_Comm comm)

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

MPI_Send

*int MPI_Send(void *buf,int count, MPI_Datatype datatype,int dest,*
int tag, MPI_Comm comm)

buf		адрес буфера
count	–	число пересылаемых элементов
Datatype	–	MPI datatype
dest	-	rank процесса-получателя
tag	-	определяемый пользователем параметр,
comm	–	MPI-коммуникатор

Пример:

```
MPI_Send(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD)
```

http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Send.html

MPI_Recv

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

buf		адрес буфера
count	–	число пересылаемых элементов
Datatype	–	MPI datatype
source	-	rank процесса-отправителя
tag	-	определяемый пользователем параметр,
comm	–	MPI-коммуникатор,
status	-	статус

Пример:

MPI_Send(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD)

http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Send.html

Wildcarding (джокеры)

- Получатель может использовать джокер
  для получения сообщения от **ЛЮБОГО** процесса
  `MPI_ANY_SOURCE`

- Для получения сообщения с ЛЮБЫМ тэгом
  `MPI_ANY_TAG`

- Реальные номер процесса-отправителя и тэг возвращаются через параметр ***`status`***

Информация о завершившемся приеме сообщения

- Возвращается функцией **`MPI_Recv`** через параметр **`status`**
- Содержит:
  - Source: ***`status.MPI_SOURCE`***
  - Tag: ***`status.MPI_TAG`***
  - Count: ***`MPI_Get_count`***

Полученное сообщение

- Может быть меньшего размера, чем указано в функции MPI_Recv
- `count` – число реально полученных элементов

C:

int ***MPI_Get_count*** (MPI_Status *status,
MPI_Datatype datatype, int *count)

MPI_Probe

int MPI_Probe(int *source*, int *tag*, MPI_Comm *comm*, MPI_Status* *status*)

Проверка статуса операции приема сообщения.

Параметры аналогичны функции MPI_Recv

Пример

```
if (rank == 0) {
// Send size  of integers to process 1
    MPI_Send(buf, size, MPI_INT, 1, 0,
      MPI_COMM_WORLD);
    printf("0 sent %d numbers to 1\n",
      size);
 } else if (rank == 1) {
    MPI_Status status;
    // Probe for an incoming message from
      process
    MPI_Probe (0, 0, MPI_COMM_WORLD,
      &status);
    MPI_Get_count (&status, MPI_INT,
      &size);
```

```
int* number_buf =
(int*)malloc(sizeof(int) * size);
 // Now receive the message with the
allocated buffer
 MPI_Recv (number_buf, size, MPI_INT,
                0, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
printf("1 dynamically received %d
numbers from 0.\n",
          number_amount);
    free(number_buf);
 }
```

Совмещение передач типа «отсылка-прием»

```
int MPI_Sendrecv (void *sendbuf,
        int sendcount, MPI_Datatype sendtype,
        int dest, int sendtag,
        void *rcvbuf, int rcvcount, MPI_Datatype rcvtype,
         int source, int rcvtag,
        MPI_Comm comm,
        MPI_Status *status)
```

Обмен данными одного типа с замещением

```
int MPI_Sendrecv_replace
(void* buf, int count,
MPI_Datatype datatype,
int dest, int sendtag,
 int source, int recvtag,
 MPI_Comm comm, MPI_Status *status)
```

Неблокирующие коммуникации

Цель – уменьшение времени работы параллельной программы за счет совмещения вычислений и обменов.

Неблокирующие операции завершаются, не дожидаясь окончания передачи данных.

Проверка состояния передач и ожидание завершение передач выполняются специальными функциями.

Форматы неблокирующих функций

```
MPI_Isend(buf,count,datatype,dest,tag,comm,request)
```

```
MPI_Irecv(buf,count,datatype,source,tag,comm, request)
```

Проверка завершения операций `MPI_Wait()` and `MPI_Test()`.

`MPI_Wait()` ожидание завершения.

`MPI_Test()` проверка завершения. Возвращается флаг, указывающий на результат завершения.

Замер времени MPI_Wtime

- Время замеряется в секундах
- Выделяется интервал в программе

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

Пример.

```
double start, finish, elapsed, time ;
start=-MPI_Wtime;
MPI_Send(…);
finish = MPI_Wtime();
time= start+finish;
```

Функции коллективных передач

Collective Communication Routines		
MPI_Allgather	MPI_Allgatherv	MPI_Allreduce
MPI_Alltoall	MPI_Alltoallv	MPI_Barrier
MPI_Bcast	MPI_Gather	MPI_Gatherv
MPI_Op_create	MPI_Op_free	MPI_Reduce
MPI_Reduce_scatter	MPI_Scan	MPI_Scatter
MPI_Scatterv		

Характеристики коллективных передач

- Коллективные операции не являются помехой операциям типа «точка-точка» и наоборот
- Все процессы коммуникатора должны вызывать коллективную операцию
- Синхронизация не гарантируется (за исключением барьера)
- Нет неблокирующих коллективных операций
- Нет тэгов
- Принимающий буфер должен точно соответствовать размеру отсылаемого буфера

Широковещательная рассылка

- One-to-all передача: один и тот же буфер отсылается от процесса root всем остальным процессам в коммуникаторе
- ```
  int MPI_Bcast (void *buffer, int, count,
    MPI_Datatype datatype,int root, MPI_Comm comm)
  ```
- Все процессы должны указать один тот же root и communicator
  ```

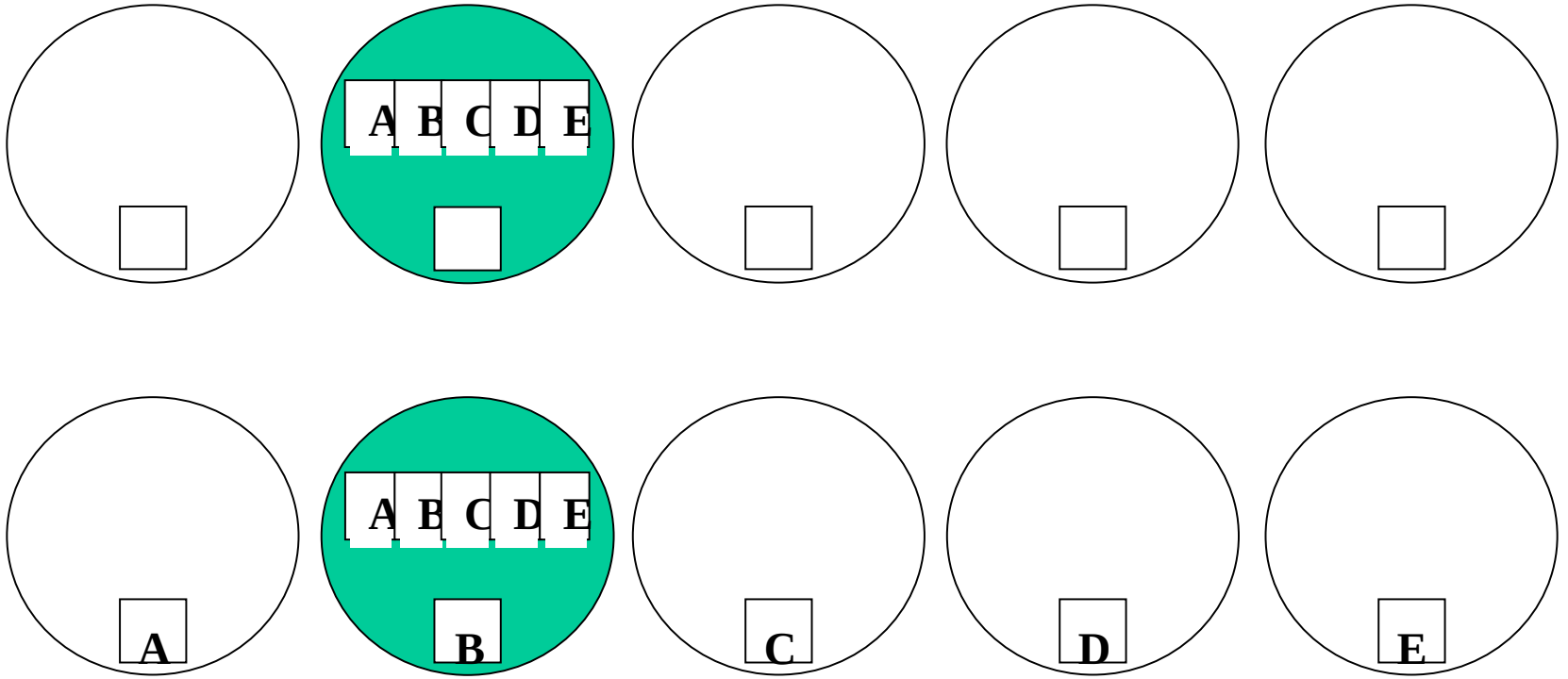
Scatter

- One-to-all communication: различные данные из одного процесса рассылаются всем процессам коммуникатора (в порядке их номеров)

```
int MPI_Scatter(void* sendbuf, int sendcount,
    MPI_Datatype sendtype, void* recvbuf,
    int recvcount,MPI_Datatype recvtype, int root,
    MPI_Comm comm)
```

- *sendcount* – число элементов, посланных каждому процессу, не общее число отосланных элементов;
- send параметры имеют смысл только для процесса root

Scatter – графическая иллюстрация



Глобальные операции редукции

- Операции выполняются над данными, распределенными по процессам коммуникатора
- Примеры:
  - Глобальная сумма или произведение
  - Глобальный максимум (минимум)
  - Глобальная операция, определенная пользователем

Общая форма

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

- **count** число операций “*op*” выполняемых над последовательными элементами буфера **sendbuf**
- (также размер **recvbuf**)

- **op** является ассоциативной операцией, которая выполняется над парой операндов типа **datatype** и возвращает результат того же типа

Предопределенные операции редукции

MPI Name	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location
MPI_MINLOC	Minimum and location

Виртуальные топологии

Virtual Topology Routines		
MPI_Cart_coords	MPI_Cart_create	MPI_Cart_get
MPI_Cart_map	MPI_Cart_rank	MPI_Cart_shift
MPI_Cart_sub	MPI_Cartdim_get	MPI_Dims_create
MPI_Graph_create	MPI_Graph_get	MPI_Graph_map
MPI_Graph_neighbors	MPI_Graph_neighbors_count	MPI_Graphdims_get
MPI_Topo_test		

Декартова топология

- Логическая топология, определяемая многомерной решеткой.
- Обобщение линейной и матричной топологий на произвольное число измерений.
- Для создания коммуникатора с декартовой топологией используется функция MPI_Cart_create. С помощью этой функции можно создавать топологии с произвольным числом измерений, причем по каждому измерению в отдельности можно накладывать периодические граничные условия.

Виртуальные топологии

- Основные функции:
  - MPI_CART_CREATE
  - MPI_CART_COORDS
  - MPI_CART_RANK
  - MPI_CART_SUB
  - MPI_CARTDIM_GET
  - MPI_CART_GET
  - MPI_CART_SHIFT

MPI_CART_CREATE

Создает структуру «прямоугольная решетка» произвольной размерности.

```
int MPI_Cart_create(MPI_Comm old_comm, int ndims, int
  *dim_size, int *periods, int reorder, MPI_Comm *new_comm)
```

.

MPI_CART_SUB

Используется для разделения коммуникатора на подгруппы . MPI_CART_SUB создает новый коммуникатор меньшей размерности

*int MPI_Cart_sub(MPI_Comm old_comm, int *belongs, MPI_Comm *new_comm)*

MPI_CART_SHIFT

- Получение номеров посылающего (**`source`**) и принимающего (**`dest`**) процессов в декартовой топологии коммуникатора **`comm`** для осуществления сдвига вдоль измерения **`direction`** на величину **`disp`**.

```
int MPI_Cart_shift( MPI_Comm comm, int direction,
    int displ, int *source, int *dest )
```

MPI_CART_SHIFT

Параметры

comm	MPI_Comm	Input	Коммуникатор
direction	int	Input	Размерность, по которой будет производиться сдвиг
displ	int	Input	Величина и направление сдвига (<0; >0; or 0)
source	int *	Output	Процесс- источник
dest	int *	Output	Процесс- получатель

MPI_CART_SHIFT

Для периодических измерений осуществляется циклический сдвиг, для непериодических – линейный сдвиг.

Для **n**-мерной декартовой решетки значение **direction** должно быть в пределах от **0** до **n-1**.

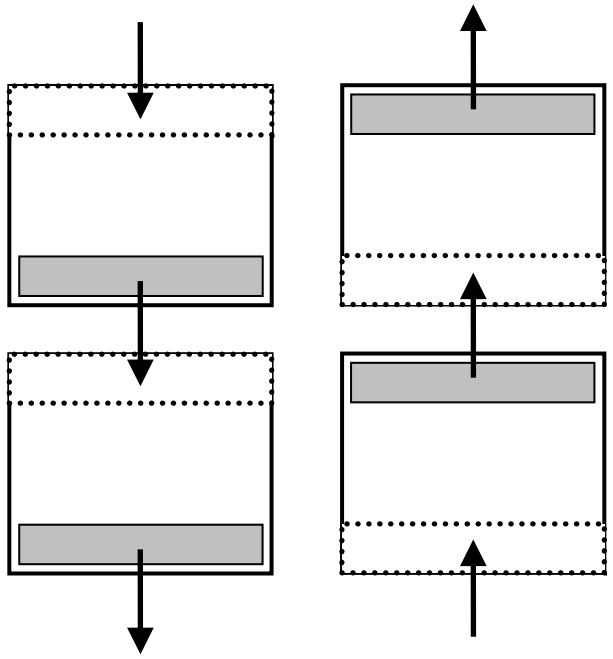
Значения **source** и **dest** можно использовать, например, для обмена функцией **MPI_Sendrecv**.

Не является коллективной операцией!

Обмен данными в программе.

- Процессу необходимо обменяться данными с соседями.
- Из-за того что данные хранятся в памяти не непрерывно, для выполнения межпроцессорных обменов предварительно формируются передаваемые сообщения.

  Потом при помощи функций MPI_Cart_shift происходит определение ранга процесса, которому надо послать данные, и процесса, от которого данные необходимо получить. После чего при помощи MPI_Sendrecv - обмен данными с соседними процессами.



Обмен данными вдоль одного измерения

Фрагменты программы. Обмен данными - влево вдоль оси X.

```
MPI_Cart_shift(cart_comm, 0, -1, &rank_recv, &rank_send);
if (coords[0] != 0 && coords[0] != proc_number[0] - 1) {
        MPI_Sendrecv(buf_left.data, size_y * size_z, MPI_DOUBLE,
        rank_send, TAG_X, right.data, size_y * size_z, MPI_DOUBLE,
        rank_recv, TAG_X, cart_comm, &status);
}
else if (coords[0] == 0 && coords[0] != proc_number[0] – 1){
        MPI_Recv(right.data, size_y * size_z, MPI_DOUBLE,
        rank_recv, TAG_X, cart_comm, &status);
}
else if (coords[0] == proc_number[0] - 1 && coords[0] != 0){
        MPI_Send(buf_left.data, size_y * size_z, MPI_DOUBLE,
        rank_send, TAG_X, cart_comm);
}
```

Пример декартовой решетки (send&recv, mesh)

```
MPI_Request reqs[8];
MPI_Status stats[8];
MPI_Comm cartcomm;

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
if (numtasks == SIZE) {
  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods,
reorder, &cartcomm);
  MPI_Comm_rank(cartcomm, &rank);
  MPI_Cart_coords(cartcomm, rank, 2, coords);
  MPI_Cart_shift(cartcomm, 0, 1, &nbrs[UP], &nbrs[DOWN]);
  MPI_Cart_shift(cartcomm, 1, 1, &nbrs[LEFT], &nbrs[RIGHT]);
  outbuf = rank;
```

Пример декартовой решетки (send&recv, mesh)

```
for (i=0; i<4; i++) {
      dest = nbrs[i];
      source = nbrs[i];
      MPI_Isend(&outbuf, 1, MPI_INT, dest, tag, MPI_COMM_WORLD, &reqs[i]);
      MPI_Irecv(&inbuf[i], 1, MPI_INT, source, tag, MPI_COMM_WORLD,
&reqs[i+4]);
      }
   MPI_Waitall(8, reqs, stats);
   printf("rank= %d coords= %d %d  neighbors(u,d,l,r)= %d %d %d %d
inbuf(u,d,l,r)= %d %d %d %d\n",
rank,coords[0],coords[1],nbrs[UP],nbrs[DOWN],nbrs[LEFT],inbuf[UP],inbuf[DOWN],
inbuf[LEFT],inbuf[RIGHT]);
   }
 else
   printf("Must specify %d tasks. Terminating.\n",SIZE);
 MPI_Finalize(); }
```

Литература

- MPI: A Message-Passing Interface Standard
  http://www.mpi-forum.org/docs/
- Антонов А.С. «Параллельное программирование с использованием технологий MPI и OpenMP» Изд-во МГУ, 2011. 310 с.
- Интернет ресурсы:
- https://computing.llnl.gov/tutorials/mpi, http://parallel.ru/docs/Parallel/mpi1.1/mpi-report.html, **intuit.ru, www.mpi-forum.org, www.open-mpi.org, www.openmp.org**
- **http://www.mcs.anl.gov/research/projects/mpi/www/www3**