

Суперкомпьютерные вычислительные технологии.

Лекционно-практический курс
для студентов 5 курса факультета ВМиК МГУ
сентябрь – декабрь 2013 г.

Лектор доцент Н.Н.Попова

Лекция 6
11 октября 2013 г.

Тема

- Группы и коммутаторы
- Виртуальные топологии
- Задание 2: 2D уравнение Лапласа

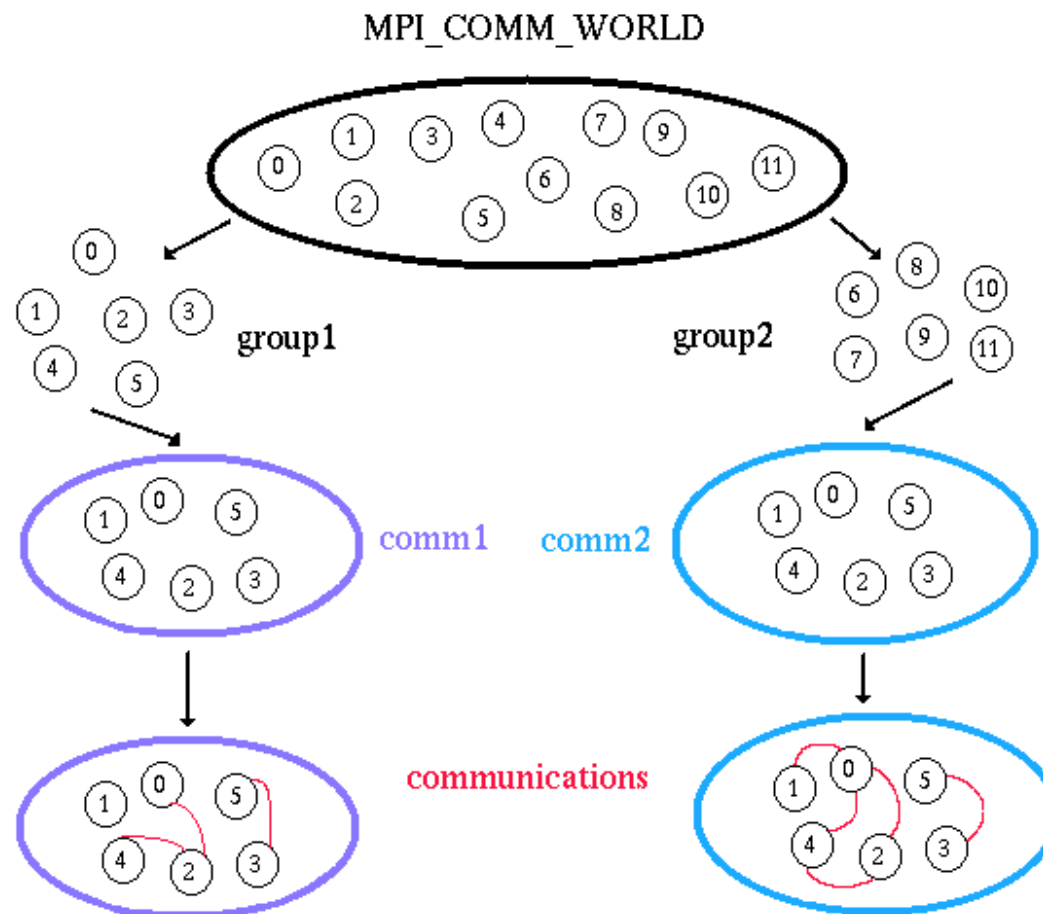
Пример: вычисление π (1)

```
#include "mpi.h"
#include <math.h>
int main(int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i, rc;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x, a;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    while (!done) {
        if (myid == 0) {
            printf("Enter the number of intervals: (0 quits) ");
            scanf("%d",&n);
            MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
            if (n == 0) break;
```

Пример: PI (2)

```
h    = 1.0 / (double) n;
sum  = 0.0;
for (i = myid + 1; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += 4.0 / (1.0 + x*x);
}
mypi = h * sum;
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
           MPI_COMM_WORLD);
if (myid == 0)
    printf("pi is approximately %.16f, Error is %.16f\n",
           pi, fabs(pi - PI25DT));
}
MPI_Finalize();
return 0;}
```

Группы и коммунитаторы



Понятие коммуникатора MPI

- Коммуникатор - управляющий объект, представляющий группу процессов, которые могут взаимодействовать друг с другом

Группы и коммутаторы

- Группа:
 - Упорядоченное множество процессов
 - Каждый процесс в группе имеет уникальный номер
 - Процесс может принадлежать нескольким группам
 - rank всегда относителен группы
- Коммутаторы:
 - Все обмены сообщений всегда проходят в рамках коммутатора
 - С точки зрения программирования группы и коммутаторы эквивалентны
 - communicators are also “opaque objects”
- Группы и коммутаторы – динамические объекты, должны создаваться и уничтожаться в процессе работы программы

Типичный шаблон работы

1. Извлечение глобальной группы из коммуникатора `MPI_COMM_WORLD`, используя функцию `MPI_Comm_group`
1. Формирование новой группы как подмножества глобальной группы, используя `MPI_Group_incl` или `MPI_Group_excl`
2. Создание новый коммуникатор для новой группы, используя `MPI_Comm_create`
3. Определение номера процесса в новом коммуникаторе, используя `MPI_Comm_rank`
4. Обмен сообщениями, используя функции MPI
5. По окончании освобождение созданных коммуникатора и группы, используя `MPI_Comm_free` и `MPI_Group_free`

Специальные типы MPI

- MPI_Comm
- MPI_group

Создание новых коммуникаторов

2 способа создания новых коммуникаторов:

- Использовать функции для работы с группами и коммуникаторами (создать новую группу процессов и по новой группе создать коммуникатор, разделить коммуникатор и т.п.)
- Использовать встроенные в MPI виртуальные топологии

Количество процессов в группе

- Размер группы (число процессов в группе)

```
int MPI_Group_size(MPI_Group comm, int *size)
```

Результат – число процессов

Если указать MPI_GROUP_EMPTY, то size=0

Номер процесса в группе

- Номер процесса в группе

```
int MPI_Group_rank(MPI_Group comm, int *rank)
```

Результат – номер процессов или MPI_UNDEFINED

Определение группы по коммуникатору

- Группа по коммуникатору

```
int MPI_Comm_group(MPI_Comm comm, MPI_Group  
*group)
```

Пример:

```
MPI_Group commGroup;
```

```
MPI_Comm_group (MPI_COMM_WORLD, &commGroup);
```

Включение процессов в группу

■ Включение процессов в группу

*int MPI_Group_incl(MPI_Group comm, , int n, int *ranks, MPI_Group *newgroup)*

n – число процессов в новой группе

ranks – номера процессов в группе group, которые будут составлять группу newgroup (выходной параметр);

newgroup – новая группа, составленная из процессов из ranks, в порядке, определенном ranks (выходной параметр).

В случае *n=0 MPI_Group_incl* вернет **MPI_GROUP_EMPTY**.

Функция может применяться для перенумерации процессов в группе.

Исключение процессов из группы

- Номер процесса в группе

*int MPI_Group_excl(MPI_Group oldgroup, , int n, int *ranks, MPI_Group *newgroup)*

n - число процессов в массиве ranks

ranks – номера процессов в группе oldgroup, которые будут исключаться из группы oldgroup;

newgroup – новая группа , не содержащая процессов с номерами из ranks, порядок процессов такой же, как в группе group (выходной параметр).

Каждый из *n* процессов с номерами из массива ranks должен существовать, иначе функция вернет ошибку. В случае *n*=0 MPI_Group_excl вернет группу group.

Создание коммуникатора по группе

int MPI_Comm_create (*MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm*)

comm – коммуникатор;

group – группа, представляющая собой подмножество процессов, ассоциированное с коммуникатором ***comm***;

newcomm – новый коммуникатор (выходной параметр).

Функция возвращает MPI_COMM_NULL процессам, не входящим в group. Завершится с ошибкой, если не все аргументы group будут одинаковыми в различных вызывающих функцию процессах, или если group не является подмножеством группы, ассоциированной с коммуникатором comm.

Вызвать функцию должны все процессы, входящие в comm, даже если они не принадлежат новой группе.

Создание новых коммуникаторов

int MPI_Comm_split (MPI_Comm comm, int color, int key, MPI_Comm *newcomm)

comm - коммуникатор;

color - признак разделения на группы;

key - параметр, определяющий нумерацию в новых коммуникаторах;

newcomm – новый коммуникатор (выходной параметр).

Функция разбивает все множество процессов, входящих в коммуникатор *comm*, на непересекающиеся подгруппы - одну подгруппу на каждое значение параметра *color* (неотрицательное число). Каждая новая подгруппа содержит все процессы одного цвета. Если в качестве *color* указано значение MPI_UNDEFINED, то в *newcomm* будет возвращено значение MPI_COMM_NULL. Это коллективная функция, но каждый процесс может указывать свои значения для параметров *color* и *key*. Значение *color* должно быть неотрицательным.

Удаление коммуникатора

int MPI_Comm_free (MPI_Comm comm)

comm – удаляемый коммуникатор.

Функция уничтожает коммуникатор, ассоциированный с идентификатором *comm*, который после возвращения устанавливается в MPI_COMM_NULL. Все незаконченные операции, использующие этот коммуникатор, завершатся нормально.

Пример (1)

```
#include "mpi.h"
#include <stdio.h>
#define NPROCS 8
#define MASTER 0
#define MSGSIZE 7
int main(argc,argv)
int argc;
char *argv[]; {
    int    rank, new_rank,
           ranks1[4]={0,1,2,3},
           ranks2[4]={4,5,6,7};
    char    *msg;
    MPI_Group orig_group, new_group;
    MPI_Comm new_comm;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

Пример (2)

```
/* Extract the original group handle */  
MPI_Comm_group(MPI_COMM_WORLD, &orig_group);  
  
/* Divide tasks into two distinct groups. First */  
/* create new group and then a new communicator. */  
/* Find new rank in new group and setup for the */  
  
if (rank < NPROCS/2) {  
    MPI_Group_incl(orig_group, NPROCS/2, ranks1, &new_group);  
  
    MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);  
    MPI_Group_rank (new_group, &new_rank);  
}
```

Пример (3)

```
if (new_rank == MASTER) msg="Group 1"; }

else {

    MPI_Group_incl(orig_group, NPROCS/2, ranks2, &new_group);
    MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);
    MPI_Group_rank (new_group, &new_rank);
    if (new_rank == MASTER) msg="Group 2";
    }
    MPI_Bcast(&msg,MSGSIZE,MPI_CHAR,MASTER,new_comm);

    printf("rank= %d newrank= %d msg= %s\n",rank,new_rank,msg);

    MPI_Finalize();
}
```

Виртуальные топологии

- **Топология** – механизм сопоставления процессам альтернативной схемы адресации. В MPI топологии виртуальны, не связаны с физической топологией сети.
- Два типа топологий:
 - **декартова** (прямоугольная решетка произвольной размерности)
 - **топология графа**.

Виртуальные топологии

- Удобный способ именования процессов
- Упрощение написания параллельных программ
- Оптимизация передач
- Возможность выбора топологии, соответствующей логической структуре задачи

Виртуальные топологии

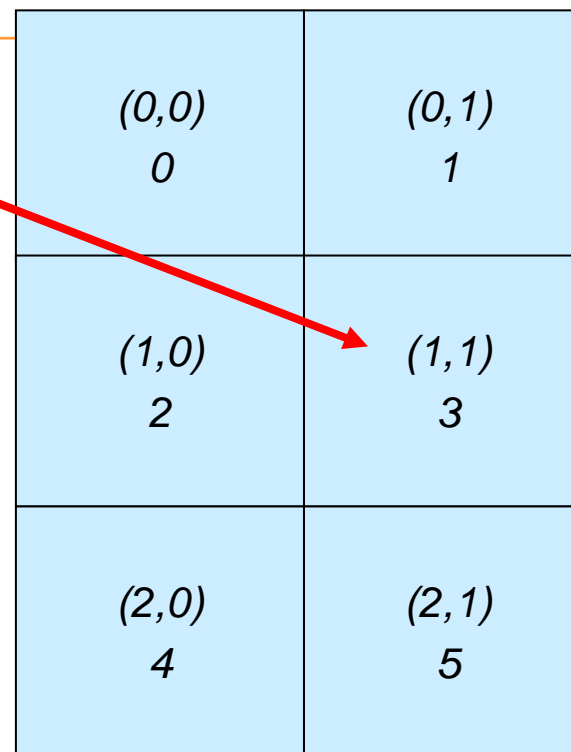
- Основные функции:
 - MPI_CART_CREATE
 - MPI_CART_COORDS
 - MPI_CART_RANK
 - MPI_CART_SUB
 - MPI_CARTDIM_GET
 - MPI_CART_GET
 - MPI_CART_SHIFT

Как использовать виртуальные ТОПОЛОГИИ

- Создание топологии – новый коммуникатор
- MPI обеспечивает “mapping functions”
- Mapping функции вычисляют ранг процессов, базирясь на топологии

2D решетка

- Отображает линейно упорядоченный массив в 2-мерную решетку (2D Cartesian topology),
- Пример: номер 3 адресуется координатами (1,1).
- Каждая клетка представляет элемент 3x2 матрицы.
- Нумерация начинается с 0.
- Нумерация построчная.



(0,0) 0	(0,1) 1
(1,0) 2	(1,1) 3
(2,0) 4	(2,1) 5

Создание виртуальной топологии решетки

```
int MPI_Cart_create (MPI_Comm comm_old,  
                    int ndims,  
                    int *dims,  
                    int *periods,  
                    int reorder,  
                    MPI_Comm *comm_cart)
```

Параметры

comm_old

старый коммуникатор

ndims

размерность

periods

логический массив, указывающий на
циклическое замыкание:

TRUE/FALSE => циклическое замыкание на границе

reorder

возможная перенумерация

comm_cart

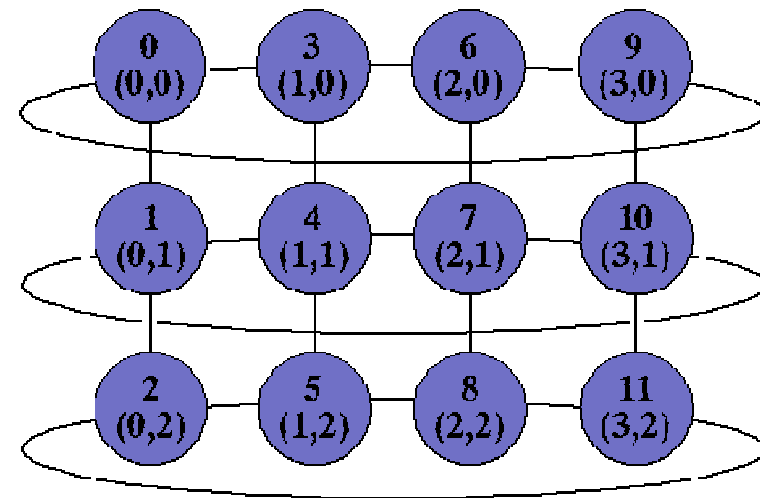
новый коммуникатор

Пример виртуальной топологии решетки

```
MPI_Comm vu;  
int dim[2], period[2], reorder;
```

```
dim[0]=4; dim[1]=3;  
period[0]=TRUE; period[1]=FALSE;  
reorder=TRUE;
```

```
MPI_Cart_create(MPI_COMM_WORLD,2,  
               dim,period,reorder,&vu);
```



Пример (решетка)

```
#include<mpi.h>
/* Run with 12 processes */
int main(int argc, char *argv[]) {
    int rank;
    MPI_Comm vu;
    int dim[2],period[2],reorder;
    int coord[2];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    dim[0]=4; dim[1]=3;
    period[0]=TRUE; period[1]=FALSE;
    reorder=TRUE;
    MPI_Cart_create(MPI_COMM_WORLD,2,dim,period,reorder,&vu);
```

Координаты процесса в виртуальной решетке

```
int MPI_Cart_coords (MPI_Comm comm,  
    int rank,  
    int numb_of_dims,  
    int  coords[])
```

MPI_CART_RANK

```
int MPI_Cart_rank( MPI_Comm comm, int *coords, int *rank )
```

Перевод логических координат процесса в решетке в ранг процесса.

MPI_CART_SHIFT

- Получение номеров посылающего (`source`) и принимающего (`dest`) процессов в декартовой топологии коммутатора `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

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`.

Пример MPI_Cart_shift

```
#include<mpi.h>
#define TRUE 1
#define FALSE 0
void main(int argc, char *argv[]) {
    int rank; MPI_Comm vu;
    int dim[2],period[2],reorder;
    int up,down,right,left;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    dim[0]=4; dim[1]=3; period[0]=TRUE; period[1]=FALSE; reorder=TRUE;
    MPI_Cart_create(MPI_COMM_WORLD,2,dim,period,reorder,&vu);
    if(rank==9){
        MPI_Cart_shift(vu,0,1,&left,&right);
        MPI_Cart_shift(vu,1,1,&up,&down);
        printf("P:%d My neighbors are r: %d d:%d 1:%d
        :%d\n",rank,right,down,left,up); }
    MPI_Finalize();}
```

Создание подрешетки

```
int MPI_Cart_sub (MPI_Comm comm_old,  
                  int    remain_dims[],  
                  MPI_Comm *new_comm)
```

MPI_CARTDIM_GET

- Определение числа измерений в решетке.

```
int MPI_Cartdim_get( MPI_Comm comm, int* ndims )
```

- comm коммуникатор (решетка)
- ndims число измерений

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;
```

```
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[DOW
N],nbrs[LEFT],inbuf[UP],inbuf[DOWN],inbuf[L
EFT],inbuf[RIGHT]);
}
else
    printf("Must specify %d tasks.
Terminating.\n",SIZE);
MPI_Finalize();
}
```

Задание . Исследование эффективности решения задачи Дирихле для уравнения Лапласа. MPI-реализация

- Задача Дирихле для уравнения Лапласа (1).

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

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

Уравнение Лапласа

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

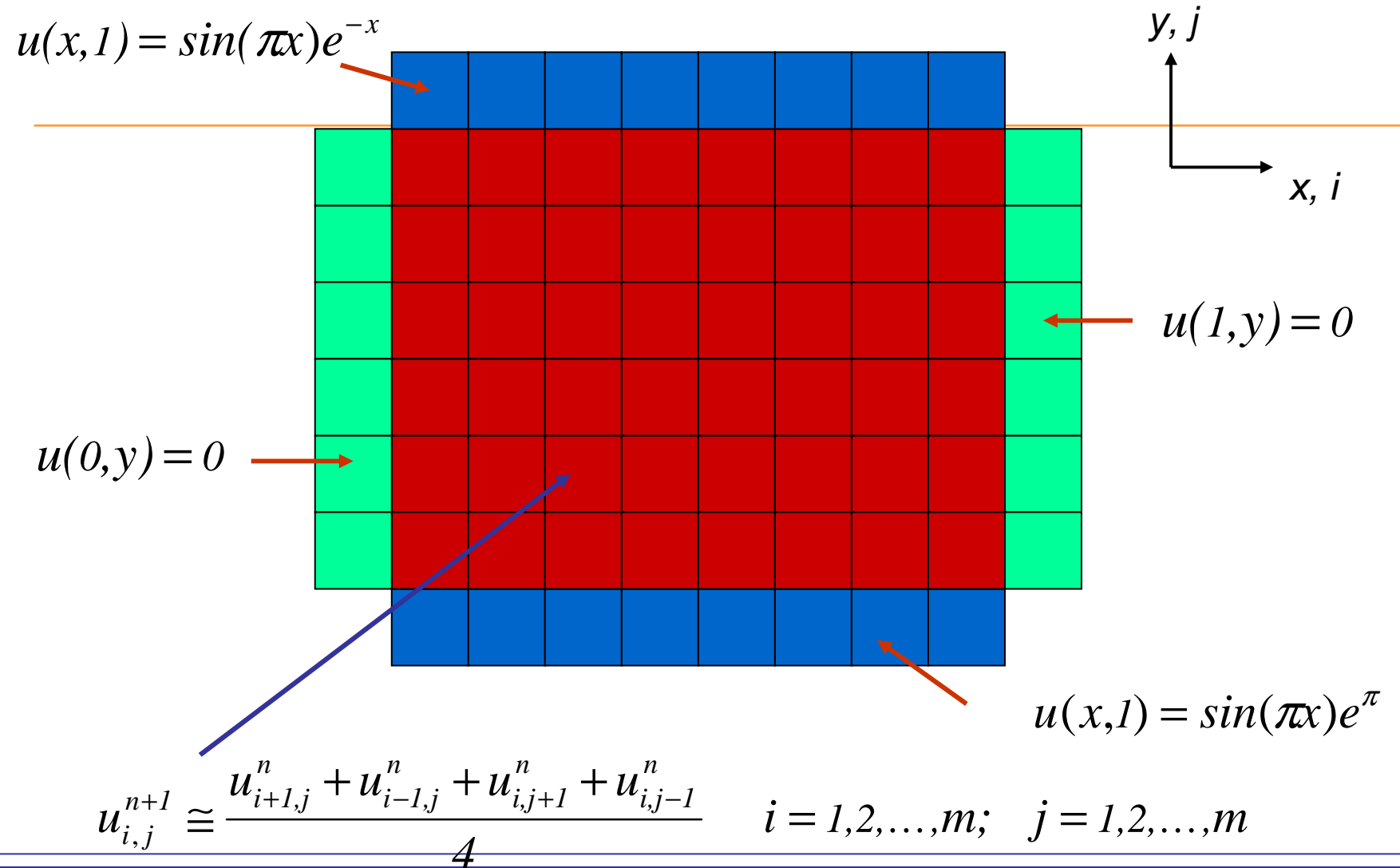
Краевые условия:

$$\begin{aligned} u(x,0) &= \sin(\pi x) & 0 \leq x \leq 1 \\ u(x,1) &= \sin(\pi x)e^{-x} & 0 \leq x \leq 1 \\ u(0,y) &= u(1,y) = 0 & 0 \leq y \leq 1 \end{aligned} \quad (2)$$

Аналитическое решение:

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

Вычислительная область



Распределение данных – 1D

5-точечная схема требует значений от соседних процессов

						X	
					X	O	X
		X				X	

Процесс 2

Необходим обмен данными на границах области

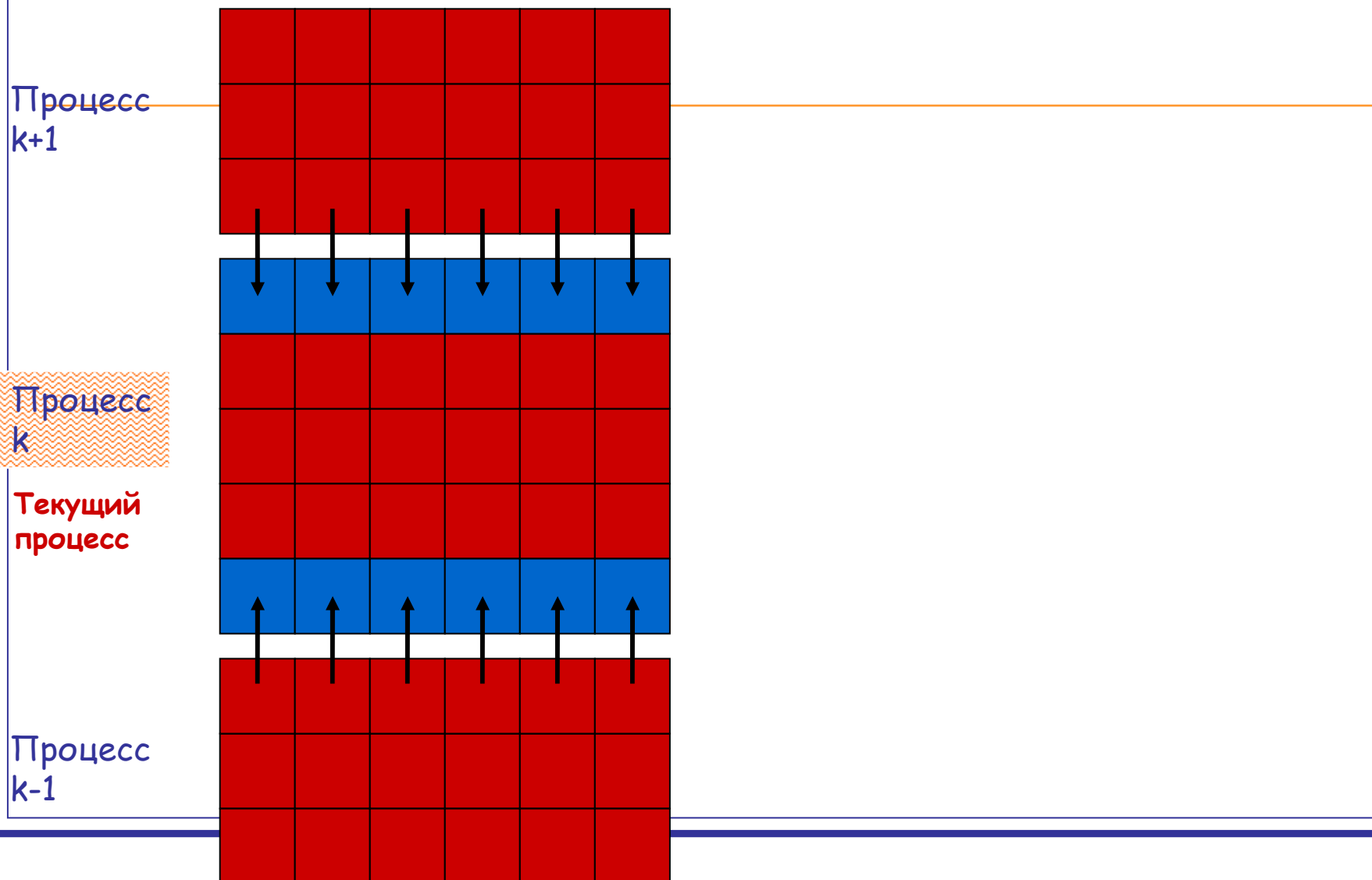
	X	O	X				
		X					

Процесс 1

					X		
				X	O	X	
				X			

Процесс 0

Схема передачи данных



Метод последовательной верхней релаксации (Successive Over Relaxation)

1. Определяем начальное значение u во всех внутренних точках (i,j) .
2. Определяем ω_n ($0 < \omega_n < 2$)
3. Используя 5-точечный шаблон вычисляем $u'_{i,j}$ во всех внутренних точках (i,j) .
4. Вычисляем $u_{i,j}^{n+1} = \omega_n u'_{i,j} + (1 - \omega_n) u_{i,j}^n$
5. Завершаем процесс, если точность достигнута, иначе
6. Обновляем: $u_{i,j}^n = u_{i,j}^{n+1} \quad \forall i, j$
7. Переход на пункт 2.

$$\omega_0 = 0 \quad ; \quad \omega_1 = \frac{1}{1-\rho^2/2} \quad ; \quad \omega_2 = \frac{1}{1-\rho^2\omega_1/4}$$

$$\omega_n = \frac{1}{1-\rho^2\omega_{n-1}/4} \quad ; \quad n > 2$$

$$\rho = 1 - \left(\frac{\pi}{2(m+1)} \right)^2$$

На шаге 3 вычисляем u' ,
используя u в момент $n+1$.