Физический факультет МГУ имени М.В. Ломоносова

Параллельное программирование
для ресурсоёмких задач численного моделирования в физике

В.О. Милицин, Д.Н. Янышев, И.А. Буткарев

Лекция № 11

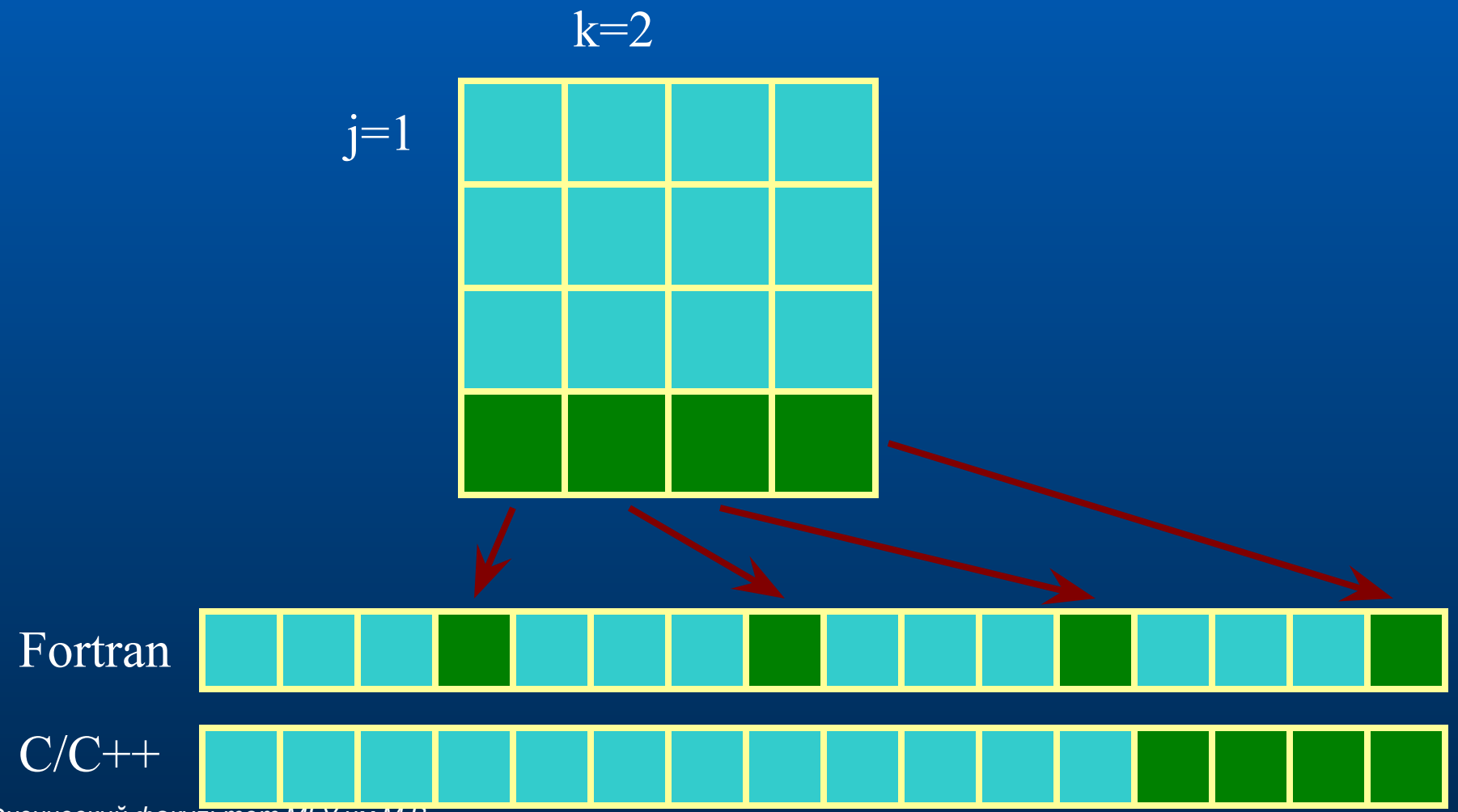
Содержание лекции

- Производные типы данных
- Передача упакованных данных

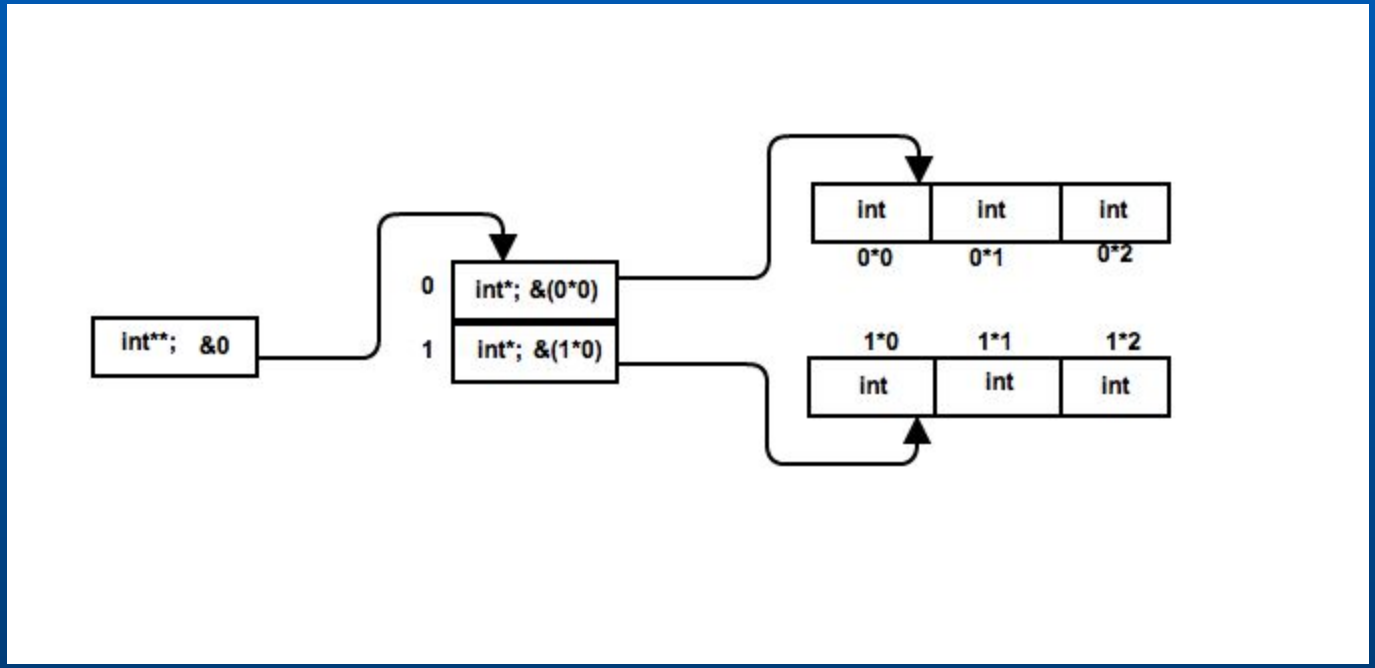
Структура сообщений MPI

- *Сообщение* – массив **однотипных** данных, расположенных в последовательных ячейках памяти

Расположение элементов матрицы в линейной модели памяти



Динамические массивы C/C++



Структуры и объекты в C/C++

- Структуры
  - размер – sizeof
  - выравнивание – align
  - порядок байтов (LE, BE)
  - указатели в структуре
- Объекты (экземпляры классов)
  - предки, виртуальные методы, …
  - блоки public/private/protected
  - приведение к структуре
  - **serialization**

```
struct Foo {
  char ch;
  short id;
  short opt;
  int value;
};

#pragma pack(push,1)
struct Foo {
…
#pragma pack(pop)
```

Механизмы эффективной пересылки данных в MPI

- создания производных типов вместо предопределенных типов MPI
- пересылку упакованных данных

Производные типы данных

- создаются во время выполнения программы с помощью подпрограмм-конструкторов

- Определения и использования типа:
  - Конструирование типа
  - Регистрация типа
  - Уничтожение типа

Производный тип данных

- последовательность базовых типов
- последовательность смещений

  - отображение (карта) типа
    - Typemap = {($type_0$, $disp_0$),
      … ,
      ($type_{n-1}$,$disp_{n-1}$)}.

MPI_INT ~ {(integer,0)}

Типы данных в MPI

- Протяженность [байт]
  - = адрес последней ячейки данных - адрес первой ячейки данных + +длина последней ячейки данных
  - MPI_TYPE_EXTENT(DATATYPE, EXTENT, IERR)
- Размер [байт]
  - = сумме длин всех базовых элементов определяемого типа
  - MPI_TYPE_SIZE(DATATYPE, SIZE, IERR)

Функция MPI_Type_contiguous

- MPI_TYPE_CONTIGUOUS (COUNT, OLDTYPE, NEWTYPE, IERR)

  - INTEGER COUNT, OLDTYPE
    - COUNT число элементов базового типа
    - OLDTYPE базовый тип

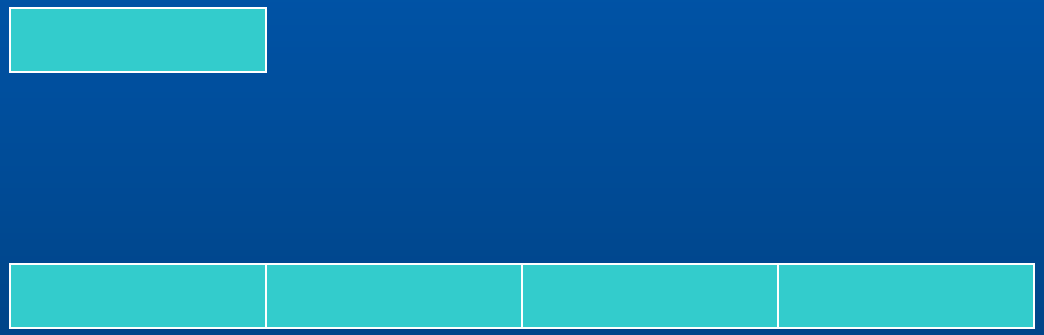
  - INTEGER NEWTYPE новый производный тип данных

MPI_Type_contiguous

OldType

Count =4

NewType



Функция-конструктор *MPI_Type_vector*

- MPI_TYPE_VECTOR(COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE, IERR)

  - INTEGER COUNT, BLOCKLENGTH, STRIDE, OLDTYPE
    - COUNT число блоков
    - BLOCKLENGTH число элементов базового типа в каждом блоке
    - STRIDE шаг между началами соседних блоков, измеренный числом элементов базового типа
    - OLDTYPE базовый тип данных
  - INTEGER NEWTYPE новый производный тип данных

MPI_Type_vector

OldType 

Count =3, blocklength=2, stride=3

NewType 

Функция-конструктор *MPI_Type_hvector*

- **MPI_TYPE_HVECTOR(COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE, IERR)**
  - Данная функция устарела и заменяется на MPI_TYPE_CREATE_HVECTOR в MPI-2.0 (изменился только тип **STRIDE** в FORTRAN).

  - **INTEGER COUNT, BLOCKLENGTH, STRIDE, OLDTYPE**
    - **COUNT число блоков;**
    - **BLOCKLENGTH число элементов базового типа в каждом блоке**
    - **STRIDE шаг между началами соседних блоков в байтах** (в C тип MPI_Aint)
    - **OLDTYPE базовый тип данных**

  - **INTEGER NEWTYPE новый производный тип данных**

MPI_Type_hvector

OldType 

Count =3, blocklength=2, stride=7

NewType 

Функция-конструктор *MPI_Type_indexed*

- MPI_TYPE_INDEXED(COUNT, ARRAY_OF_BLOCKLENGTHS, ARRAY_OF_DISPLACEMENTS, OLDTYPE, NEWTYPE, IERR)
  - INTEGER COUNT, ARRAY_OF_BLOCKLENGTHS(*), ARRAY_OF_DISPLACEMENTS(*), OLDTYPE,
    - COUNT число блоков
    - ARRAY_OF_BLOCKLENGTHS массив, содержащий число элементов базового типа в каждом блоке
    - ARRAY_OF_DISPLACEMENTS массив смещений каждого блока от начала размещения элемента нового типа, смещения измеряются числом элементов базового типа
    - OLDTYPE базовый тип данных
  - INTEGER NEWTYPE новый производный тип данных

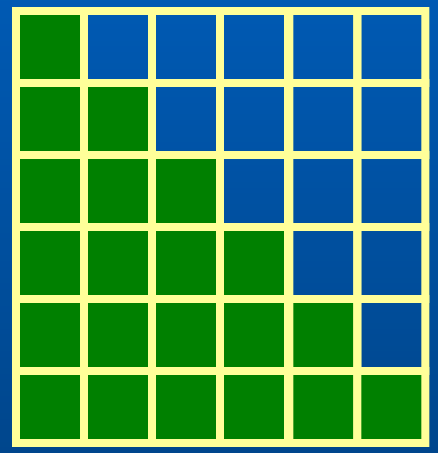
MPI_Type_indexed

OldType 

Count =3, blocklength=(3, 5, 10), displacements=(0, 4, 10)

NewType 

Пример *MPI_Type_indexed*



```
do i=1,N
   blocklens(i)=N-i+1
   displs(i)=N*(i-1)+i-1
end do
call MPI_TYPE_INDEXED(N, blocklens,
&     displs, MPI_DOUBLE_PRECISION,
&     newtype, ierr)
```

Функция-конструктор *MPI_Type_hindexed*

- **MPI_TYPE_HINDEXED(COUNT, ARRAY_OF_BLOCKLENGTHS, ARRAY_OF_DISPLACEMENTS, OLDTYPE, NEWTYPE, IERR)**
  - Данная функция устарела и заменяется на MPI_TYPE_CREATE_HINDEXED в MPI-2.0 (изменился только тип **ARRAY_OF_DISPLACEMENTS** в FORTRAN)

  - **INTEGER COUNT, ARRAY_OF_BLOCKLENGTHS(*), ARRAY_OF_DISPLACEMENTS(*), OLDTYPE,**
    - **COUNT число блоков**
    - **ARRAY_OF_BLOCKLENGTHS массив, содержащий число элементов базового типа в каждом блоке**
    - **ARRAY_OF_DISPLACEMENTS массив смещений каждого блока от начала размещения элемента нового типа, смещения измеряются в байтах** (в C тип MPI_Aint)
    - **OLDTYPE базовый тип данных**

  - **INTEGER NEWTYPE новый производный тип данных**

MPI_Type_hindexed

OldType 

Count =3, blocklength=(2, 3, 1), displacements=(0, 7, 18)

NewType 

Функция-конструктор *MPI_Type_struct*

- **MPI_TYPE_STRUCT(COUNT, ARRAY_OF_BLOCKLENGTHS, ARRAY_OF_DISPLACEMENTS, ARRAY_OF_TYPES, NEWTYPE, IERR)**
  - Данная функция устарела и заменяется на MPI_TYPE_CREATE_STRUCT в MPI-2.0 (изменился только тип **ARRAY_OF_DISPLACEMENTS** в FORTRAN).

  - **INTEGER COUNT, ARRAY_OF_BLOCKLENGTHS(*), ARRAY_OF_DISPLACEMENTS(*), ARRAY_OF_TYPES(*)**
    - **COUNT число блоков**
    - **ARRAY_OF_BLOCKLENGTHS массив, содержащий число элементов одного из базовых типов в каждом блоке**
    - **ARRAY_OF_DISPLACEMENTS массив смещений каждого блока от начала размещения структуры, смещения измеряются в байтах** (в C тип MPI_Aint)
    - **ARRAY_OF_TYPES массив, содержащий тип элементов в каждом блоке**
  - **INTEGER NEWTYPE новый производный тип данных**

MPI_Type_struct

OldType 

Count =3, blocklength=(2, 3, 4), displacements=(0, 7, 16)

NewType 

Пример создания нового типа данных

```
blocklens(1)=3
blocklens(2)=2
types(1)=MPI_DOUBLE_PRECISION
types(2)=MPI_CHAR
displs(1)=0
displs(2)=24
call MPI_TYPE_STRUCT(2, blocklens, displs, types, newtype, ierr)
```

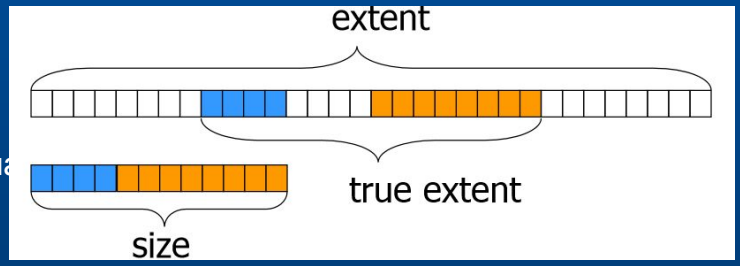
- Создание нового типа данных (тип элемента, количество байт от начала буфера)

{(**MPI_DOUBLE**, 0), (**MPI_DOUBLE**, 8), (**MPI_DOUBLE**, 16), (**MPI_CHAR**, 24), (**MPI_CHAR**, 25)}

Адресные функции

- **int MPI_Address(void *location, MPI_Aint *address)**
- **MPI_ADDRESS(LOCATION, ADDRESS, IERR)**
  - Данная функция устарела и заменяется на MPI_GET_ADDRESS в MPI-2.0 (изменился только тип ADDRESS в FORTRAN)

  - **<type> LOCATION(*) INTEGER ADDRESS, IERR**
  - **Определение абсолютного байт-адреса ADDRESS размещения массива LOCATION**



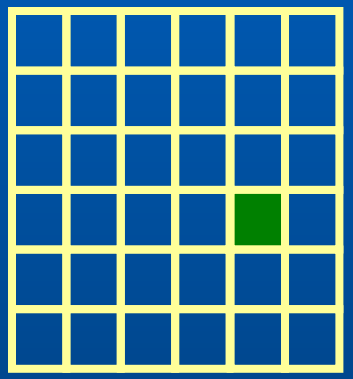
- **MPI_TYPE_SIZE(DATATYPE, SIZE, IERR)**
  - Определение размера SIZE типа данных DATATYPE в байтах (объема памяти, занимаемого одним элементом данного типа
- **MPI_TYPE_GET_EXTENT(DATATYPE, LB, EXTENT, IERROR)**
  - LB = lower bound; UP = upper bound
- **MPI_TYPE_GET_TRUE_EXTENT(DATATYPE, EXTENT, IERROR)**
- **MPI_TYPE_LB(DATATYPE, LB, IERR)**
  - Определение смещения **LB** в байтах нижней границы элемента типа данных DATATYPE от начала буфера данных.
- **MPI_TYPE_UB(DATATYPE, UB, IERR)**
  - Определение смещения **UB** в байтах верхней границы элемента типа данных DATATYPE от начала буфера данных.
- **MPI_TYPE_EXTENT(DATATYPE, EXTENT, IERR)**
  - Определение диапазона EXTENT (разницы между верхней и нижней границами элемента данного типа) типа данных DATATYPE В байтах.

Определение длины сообщения MPI

- MPI_Get_count
- MPI_Get_elements
  - **MPI_GET_ELEMENTS(STATUS, DATATYPE, COUNT, IERR)**
    - **INTEGER STATUS(MPI_STATUS_SIZE), DATATYPE**
      - **STATUS** статус сообщения
      - **DATATYPE** тип элементов сообщения
    - **INTEGER COUNT** число элементов простых типов, содержащихся в сообщении

Пример создания нового типа данных (C)

```
typedef struct {
   int row;
   int column;
   double value;
}  Matrix_Entry
```




```
Matrix_Entry matentry;
MPI_Datatype matrix_component; // Matrix_Entry
int block_lengths[3];
MPI_Aint displacements[3]; // MPI_Aint – это не pointer/& в C/C++
MPI_Aint addresses[4];
MPI_Datatype list_of_types[3];
```

Пример создания нового типа данных (C)

```
for (k =0; k<3; k++) block_lengths[k] = 1;
list_of_types[0] = list_of_types[1] = MPI_INT;
list_of_types[2] = MPI_DOUBLE;
MPI_Address(&matentry, &addresses[0]);
MPI_Address(&(matentry.row), &addresses[1]);
MPI_Address(&(matentry.column), &addresses[2]);
MPI_Address(&(matentry.value), &addresses[3]);
for (k=0; k < 3; k++) displacements[k] = addresses[k+1] - addresses[0];
MPI_Type_struct (3, block_lengths, displacements, list_of_types,
        &matrix_component);
MPI_Type_commit(&matrix_component);
```

Регистрация созданного производного типа

- MPI_TYPE_COMMIT(DATATYPE, IERR)
  - INTEGER DATATYPE
    новый производный тип данных

Уничтожение описателя производного типа

- MPI_TYPE_FREE(DATATYPE, IERR)
  - INTEGER DATATYPE
    уничтожаемый производный тип данных
    - DATATYPE = MPI_DATATYPE_NULL

Пример

```
typedef struct {
   char a;
   int b;
   double c;
} mystruct;

mystruct mydata;
MPI_Get_address ( &mydata, &baseaddr);
MPI_Get_address ( &mydata.b, &addr1);
MPI_Get_address ( &mydata.c, &addr2);
displ[0] = 0;
displ[1] = addr1 – baseaddr;
displ[2] = addr2 – baseaddr;
dtype[0] = MPI_CHAR;                    blength[0] = 1;
dtype[1] = MPI_INT;                     blength[1] = 1;
dtype[2] = MPI_DOUBLE;                  blength[2] = 1;
MPI_Type_create_struct ( 3, blength, displ, dtype, &newtype );
MPI_Type_commit ( &newtype );
```

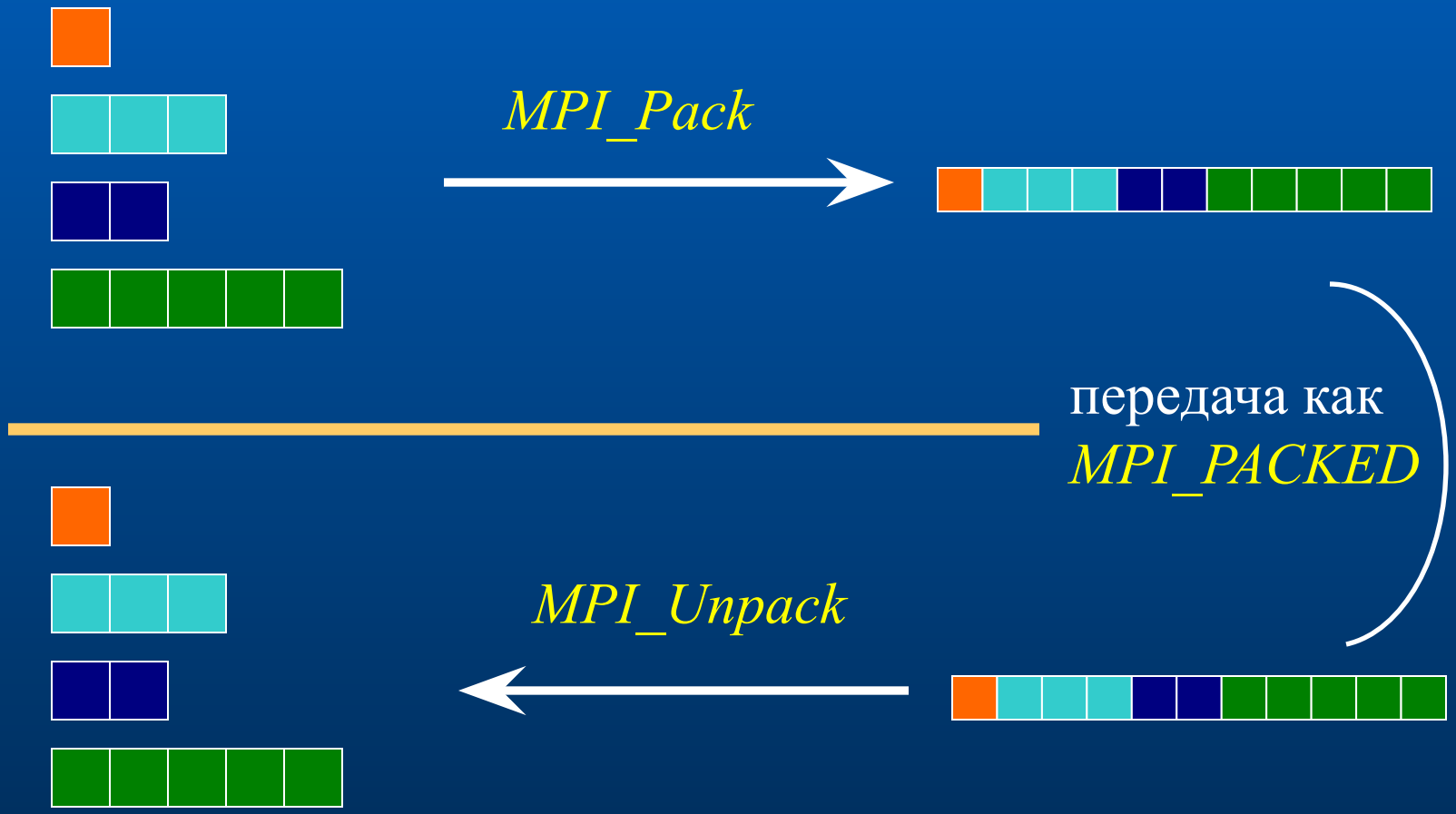
Содержание лекции

- Производные типы данных
- Передача упакованных данных

Передача упакованных данных

- *Функция MPI_Pack* упаковывает элементы предопределенного или производного типа MPI, помещая их побайтное представление в выходной буфер
- *Функция MPI_Unpack* извлекает заданное число элементов некоторого типа из побайтного представления элементов во входном массиве

Передача упакованных данных



Упаковка элементов

- **MPI_PACK(INBUF, INCOUNT, DATATYPE, OUTBUF, OUTSIZE, POSITION, COMM, IERR)**
  - **<TYPE> INBUF(*) адрес начала области памяти с элементами, которые требуется упаковать**
  - **INTEGER INCOUNT, DATATYPE, OUTSIZE, COMM**
    - **INCOUNT число упаковываемых элементов**
    - **DATATYPE тип упаковываемых элементов**
    - **OUTSIZE размер выходного буфера в байтах**
    - **COMM коммуникатор**
  - **<TYPE> OUTBUF(*)** адрес начала выходного буфера для упакованных данных
  - **INTEGER POSITION** текущая позиция в выходном буфере в байтах

Определение необходимого объема памяти

- **MPI_PACK_SIZE(INCOUNT, DATATYPE, COMM, SIZE, IERROR)**
  - **INTEGER INCOUNT, DATATYPE, COMM**
    - **INCOUNT** число элементов, подлежащих упаковке
    - **DATATYPE** тип элементов, подлежащих упаковке
    - **COMM** коммуникатор
  - **INTEGER SIZE** размер сообщения в байтах после его упаковки

Распаковка элементов

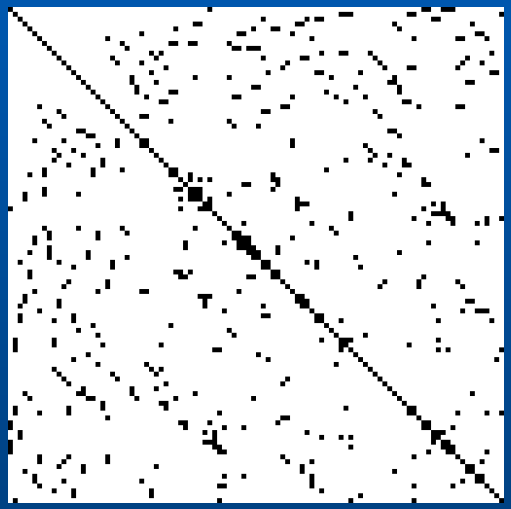
- MPI_UNPACK(INBUF, INSIZE, POSITION, OUTBUF, OUTCOUNT, DATATYPE, COMM, IERR)
  - <TYPE> INBUF(*) адрес начала входного буфера с упакованными данными
  - INTEGER INSIZE, POSITION, OUTCOUNT, DATATYPE, COMM
    - INSIZE размер входного буфера в байтах
    - POSITION текущая позиция во входном буфере в байтах
    - OUTCOUNT число извлекаемых элементов
    - DATATYPE тип извлекаемых элементов
    - COMM коммуникатор
  - <TYPE> OUTBUF(*) адрес начала области памяти для размещения распакованных элементов
  - INTEGER POSITION текущая позиция в входном буфере в байтах

Пример

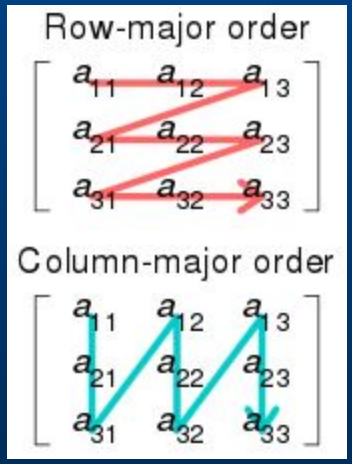
```
real a(10)
character b(10), buf(100)

position=0
if(rank .eq. 0) then
   call MPI_PACK(a, 10, MPI_REAL, buf, 100,
          &         position, MPI_COMM_WORLD, ierr)
   call MPI_PACK(b, 10, MPI_CHAR, buf, 100,
          &         position, MPI_COMM_WORLD, ierr)
   call MPI_BCAST(buf, 100,
          &         MPI_PACKED, 0,MPI_COMM_WORLD, ierr)
 else
   call MPI_BCAST(buf, 100, MPI_PACKED, 0,
          &         MPI_COMM_WORLD, ierr)
   call MPI_UNPACK(buf, 100, position, a,
          &         10, MPI_REAL, MPI_COMM_WORLD, ierr)
   call MPI_UNPACK(buf, 100, position, b,
          &         10, MPI_CHAR, MPI_COMM_WORLD, ierr)
end if
```

Разряженные матрицы (Sparse Matrix)

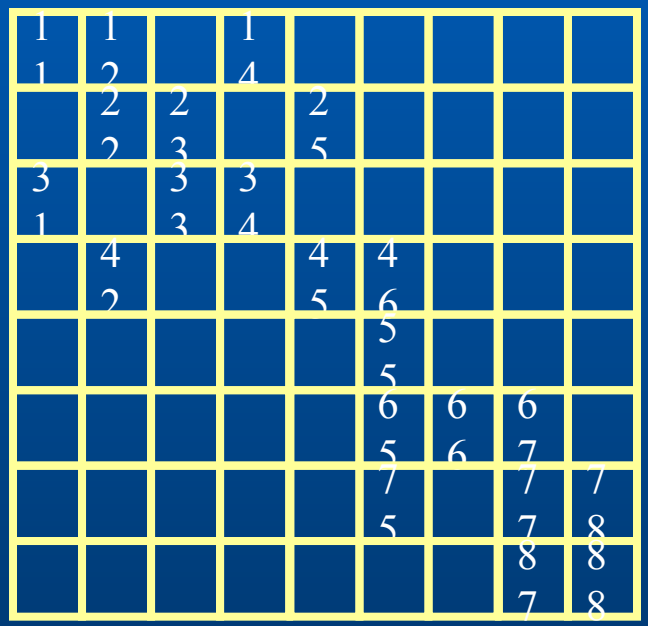
- **Эффективное изменение**
- **Эффективный доступ**
- **Специальная структура**



- Координатный формат
  = COO (Coordinate list)
- Разреженный строчный формат
  = CSR (compressed sparse row)
  = CRS (compressed row storage)
  = Yale format
- Compressed sparse column (CSC)



Разряженная матрица (CSR формат)



- CSR data
  - vals = [11 12 14 22 23 25 31 33 34 42 45 46 55 65 66 67 75 77 78 87 88]
  - columns = [0 1 3 1 2 4 0 2 3 1 4 5 4 4 5 6 4 6 7 6 7]
  - rowindx = [0 3 6 9 12 13 16 19 21]

Пересылка строки разряженной матрицы

```
char buffer[HUGE]; //буфер
//отправка k-ой строки
pos = 0; indx = rowindx[k]; nnz= rowindx[k+1] - rowindx[k];
MPI_Pack(&nnz, 1, MPI_INT, buffer, HUGE, &pos, comm);
MPI_Pack(&colunms[indx], nnz, MPI_INT, buffer, HUGE, &pos, comm);
MPI_Pack(&vals[indx], nnz, MPI_DOUBLE, buffer, HUGE, &pos, comm);
MPI_Send(buffer, pos, MPI_PACKED, dest, tag, comm); //send the row via usual MPI methods


//получение k-ой строки
MPI_Recv(buffer, HUGE, MPI_PACKED, from, tag, comm, &status);
pos = 0;
MPI_Unpack(buffer, HUGE, &pos, &nnz, 1, MPI_INT, comm);
rowvals = (double *) malloc(nnz*sizeof(double)); cols = (int *) malloc(nnz*sizeof(int ));
MPI_Unpack(buffer, HUGE, &pos, cols, nnz, MPI_INT, comm);
MPI_Unpack(buffer, HUGE, &pos, rowvals, nnz, MPI_DOUBLE, comm);
//Note: pos is an in/out argument, incremented by Pack/Unpack
```