

# ***Параллельное программирование для высокопроизводительных систем***

сентябрь – декабрь 2018 г.

Лектор доцент Н.Н.Попова

---

Лекция 10  
26 ноября 2018 г.

---

**MPI-2.**

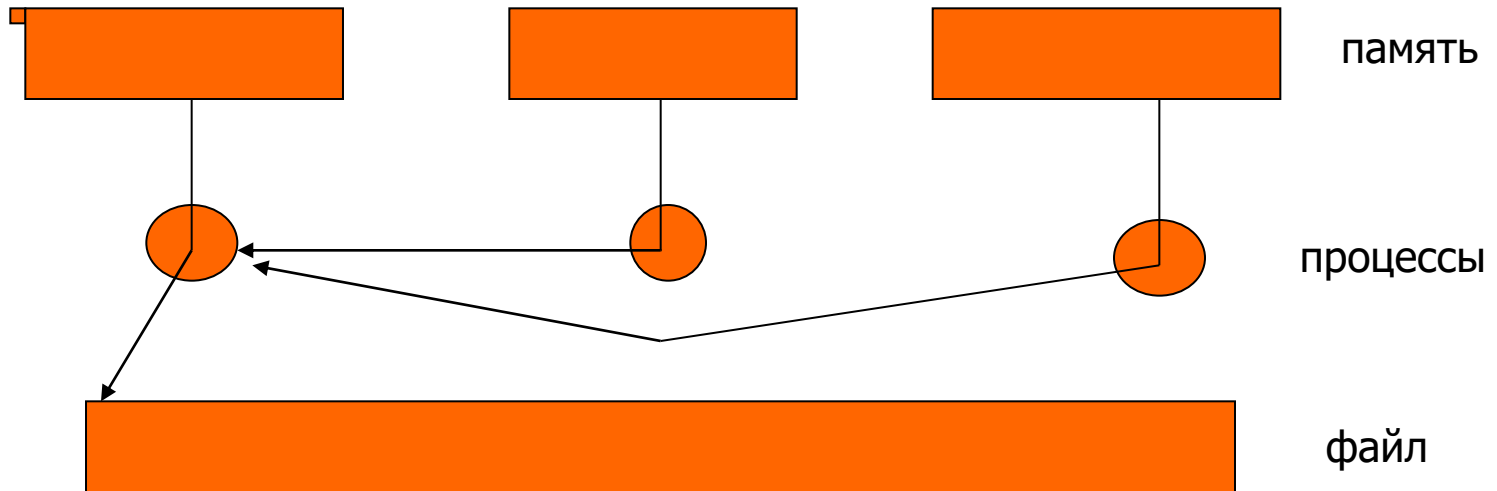
**Параллельная работа с файлами.**

# Основные возможности

---

- Произвольный доступ к файлам
- Коллективные операции ввода/вывода
- Индивидуальные и разделяемые файловые указатели
- Неблокирующий I/O
- Переносимое представление данных
- Использование рекомендаций пользователя (hints)

# MPI-2 I/O: непараллельный I/O



# Непараллельный I/O из MPI прогр. (2)

```
if (myrank != 0)
    MPI_Send(buf, BUFSIZE, MPI_INT, 0, 99, MPI_COMM_WORLD);
else {
    myfile = fopen("testfile", "w");
    fwrite(buf, sizeof(int), BUFSIZE, myfile);
    for (i=1; i<numprocs; i++) {
        MPI_Recv(buf, BUFSIZE, MPI_INT, i, 99, MPI_COMM_WORLD,
                &status);
        fwrite(buf, sizeof(int), BUFSIZE, myfile);
    }
    fclose(myfile);
}
MPI_Finalize();
return 0;
}
```

# Непараллельный I/O из MPI прогр. (3)

## — *Когда имеет смысл:*

- I/O поддерживается только на определенном узле многопроцессорной системы
- В программе используется высокоуровневая библиотека, не поддерживающая параллельный ввод-вывод
- Результирующий файл должен обрабатываться последовательным ПО
- Возможное повышение эффективности за счет буферизации данных

## *Почему надо использовать параллельный ввод-вывод:*

- Масштабируемость, эффективность при увеличении числа процессоров

# MPI-2 I/O: не MPI параллельный I/O

**/\* не MPI параллельная запись в разные файлы \*/**

**#include "mpi.h"**

**#include <stdio.h>**

**#define BUFSIZE 100**

**int main(int argc, char \*argv[]) {**

**int i, myrank, buf[BUFSIZE];**

**char filename[128];**

**FILE \*myfile;**

**MPI\_Init(&argc, &argv);**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank);**

**for (i=0; i<BUFSIZE; i++)**

**buf[i] = myrank \* BUFSIZE + i;**

**sprintf(filename, "testfile.%d", myrank);**

**myfile = fopen(filename, "w");**

**fwrite(buf, sizeof(int), BUFSIZE, myfile);**

**fclose(myfile);**

**MPI\_Finalize();**

**return 0; }**

Преимущества:

-параллельный доступ

Недостатки:

-много файлов

-использование

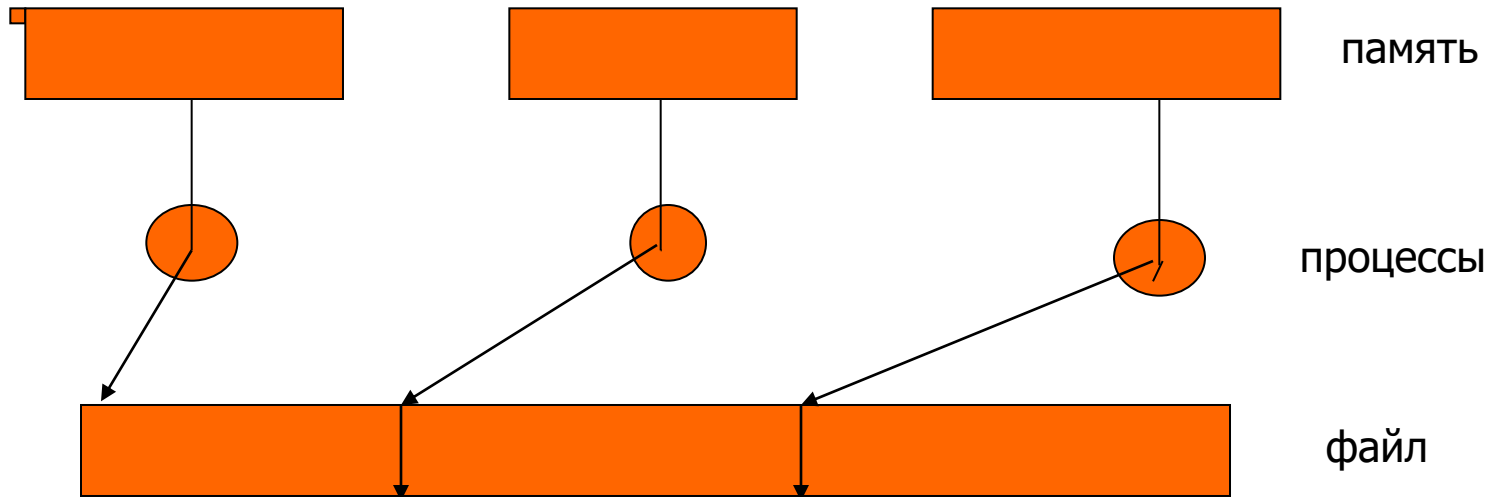
файлов при повторном  
запуске (то же число  
процессов)

# MPI-2 I/O: MPI I/O в разные файлы

```
/* Параллельная MPI-запись в разные файлы */
#include "mpi.h"
#include <stdio.h>
#define BUFSIZE 100
int main(int argc, char *argv[]){
    int i, myrank, buf[BUFSIZE];
    char filename[128];
    MPI_File myfile;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    for (i=0; i<BUFSIZE; i++)
        buf[i] = myrank * BUFSIZE + i;
    sprintf(filename, "testfile.%d", myrank);
    MPI_File_open(MPI_COMM_SELF, filename,
                  MPI_MODE_WRONLY | MPI_MODE_CREATE,
                  MPI_INFO_NULL, &myfile);
    MPI_File_write(myfile, buf, BUFSIZE, MPI_INT,
                  MPI_STATUS_IGNORE);
    MPI_File_close(&myfile);
    MPI_Finalize(); return 0; }
```



# Параллельный I/O в один файл



# Параллельный MPI I/O: запись в один файл

```
/* параллельный MPI вывод в файл */
```

```
#include "mpi.h"
```

```
#include <stdio.h>
```

```
#define BUFSIZE 100
```

```
int main(int argc, char *argv[]){
```

```
    int i, myrank, buf[BUFSIZE];
```

```
    MPI_File thefile;
```

```
    MPI_Init(&argc, &argv);
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

```
    for (i=0; i<BUFSIZE; i++)
```

```
        buf[i] = myrank * BUFSIZE + i;
```

```
    MPI_File_open(MPI_COMM_WORLD, "testfile",  
                  MPI_MODE_CREATE | MPI_MODE_WRONLY,  
                  MPI_INFO_NULL, &thefile);
```

```
    MPI_File_set_view(thefile, myrank * BUFSIZE * sizeof(int),  
                      MPI_INT, MPI_INT, "native", MPI_INFO_NULL);
```

```
    MPI_File_write(thefile, buf, BUFSIZE, MPI_INT,  
                   MPI_STATUS_IGNORE);
```

```
    MPI_File_close(&thefile);
```

```
    MPI_Finalize();
```

```
    return 0;
```

```
}
```

Data representation

Displacement

Тип: MPI\_Offset

etype

ftype

# Параллельный I/O : один файл-чтение (1)

```
/* параллельное чтение из файла произвольным числом процессов*/  
#include "mpi.h"  
#include <stdio.h>  
int main(int argc, char *argv[]){  
    int myrank, numprocs, bufsizе, *buf, count;  
    MPI_File thefile;  
    MPI_Status status;  
    MPI_Offset filesize;  
    MPI_Init(&argc, &argv);  
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);  
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);  
    MPI_File_open(MPI_COMM_WORLD, "testfile", MPI_MODE_RDONLY,  
        MPI_INFO_NULL, &thefile);
```

## Параллельный I/O: один файл-чтение (2)

```
MPI_File_get_size(thefile, &filesize); /* in bytes */  
filesize = filesize / sizeof(int);    /* in number of ints */  
bufsize = filesize / numprocs + 1;    /* local number to read */  
buf = (int *) malloc (bufsize * sizeof(int));  
MPI_File_set_view(thefile, myrank * bufsize * sizeof(int),  
                  MPI_INT, MPI_INT, "native", MPI_INFO_NULL);  
MPI_File_read(thefile, buf, bufsize, MPI_INT, &status);  
MPI_Get_count(&status, MPI_INT, &count);  
printf("process %d read %d ints\n", myrank, count);  
MPI_File_close(&thefile);  
MPI_Finalize();  
return 0;  
}
```

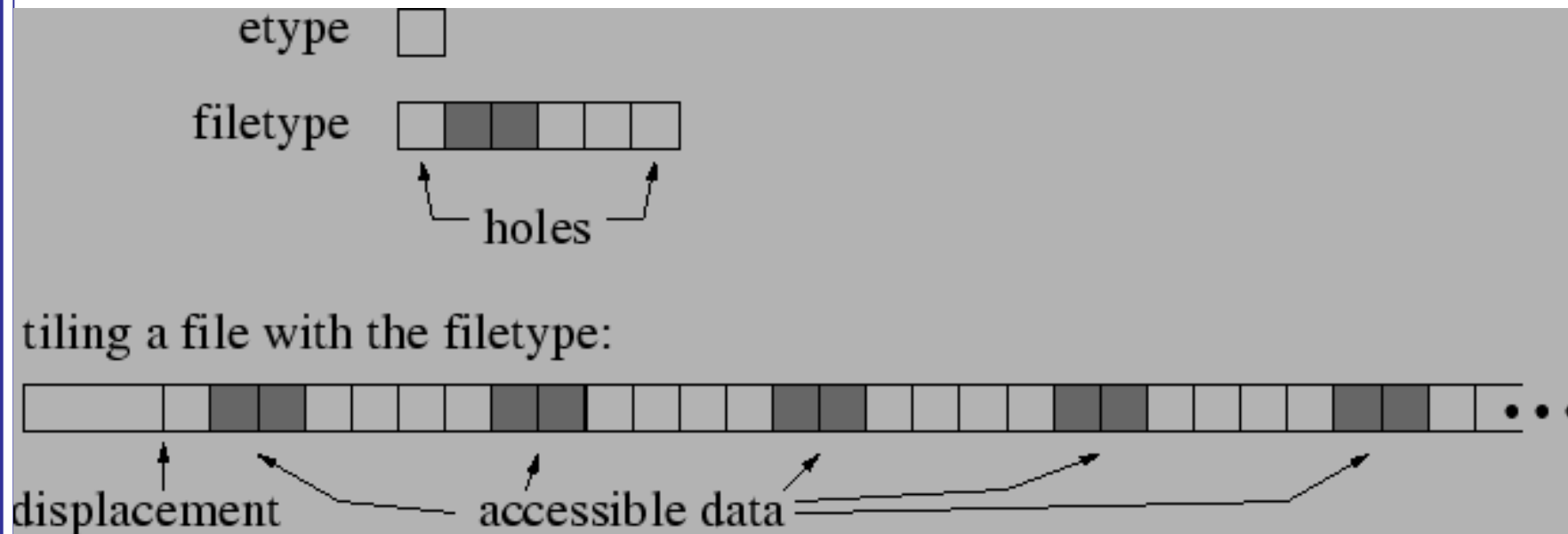
# Чтение из общего файла с использованием индивидуального файлового указателя

```
/* чтение из общего файла с использованием индивид. Файлового указателя*/
#include "mpi.h"
#define FILESIZE (1024 * 1024)
int main(int argc, char *argv[]){
    int rank, nprocs, bufsize, *buf, nints;
    MPI_File fh;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    bufsize = FILESIZE/nprocs;
    buf = (int *) malloc(bufsize);
    nints = bufsize/ sizeof(int);
    MPI_File_open(MPI_COMM_WORLD, "testfile", MPI_MODE_RDONLY,
        MPI_INFO_NULL, &fh);
    MPI_File_seek(fh, rank*bufsize, MPI_SEEK_SET);
    MPI_File_read (fh,buf, nints, MPI_INT, &status);
    MPI_File_close(&fh);
    free (buf);
    MPI_Finalize();
    return 0; }
```

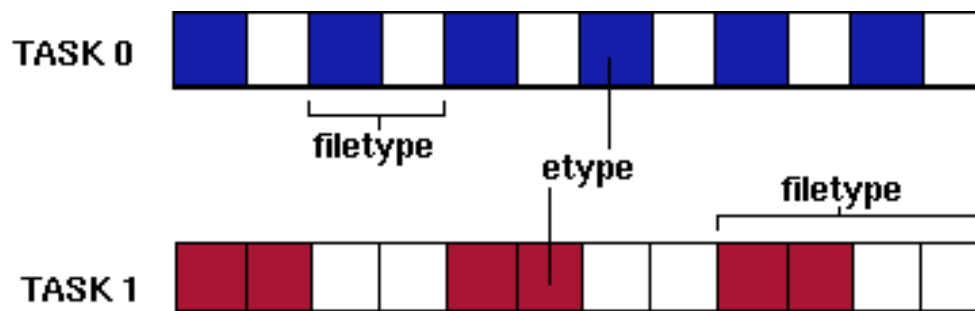
# MPI-2 I/O: Терминология

- ***Е-тип (элементарный тип данных)*** - единица доступа к данным и позиционирования. Это может быть любой определенный в *MPI* базовый тип или производный тип данных.
- ***Файловый тип*** – базис для разбиения файла в среде процессов, определяет шаблон доступа к файлу.  
Обычный е-тип или производный тип данных *MPI*, состоящий из нескольких элементов одного и того же е-типа.  
Ограничения: смещения в производном типе должны быть упорядочены по возрастанию! Например, производный тип со смещениями: {2, 6, 5, 7, 4} нельзя использовать для построения файлового типа

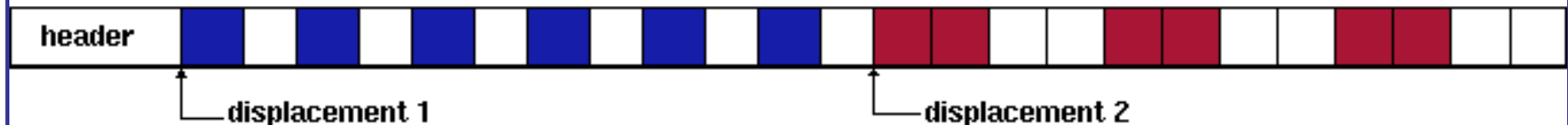
# Е-типы и Файловые типы



# Файловые виды и структура файла



File structure





# MPI-2 I/O: Терминология

---

- **View** (file view)- набор данных, видимый и доступный из открытого файла как упорядоченный набор e-типов. Каждый процесс имеет свой вид файла, определенный тремя параметрами: смещением, e-типом и файловым типом. Шаблон, описанный в файловом типе, повторяется, начиная со смещения.
- **Смещение** - это позиция в файле относительно текущего вида, представленная как число e-типов. ``Дыры" в файловом типе вида пропускаются при подсчете номера этой позиции. Нулевое смещение - это позиция первого видимого e-типа в виде (после пропуска смещения и начальных ``дыр" в виде).

# MPI-2 I/O: Терминология

---

- **Размер MPI файла** измеряется в байтах от начала файла
- **Конец файла** - это смещение первого e-типа, доступного в данном виде, начинающегося после последнего байта в файле
- **Индивидуальные файловые указатели** - файловые указатели, локальные для каждого процесса, открытого файл.
- **Общие файловые указатели** - файловые указатели, которые используются одновременно группой процессов, открывающих файл.
- **Дескриптор файла** - это закрытый объект, создаваемый MPI\_FILE\_OPEN и уничтожаемый MPI\_FILE\_CLOSE. Все операции над открытым файлом работают с файлом через его дескриптор

# MPI-2 I/O: Базовый алгоритм работы

---

- Определение необходимых переменных и типов данных
- Открытие файла (`MPI_File_open`)
- Установка вида файла (`MPI_File_set_view`)
- Запись/чтение (`MPI_File_write`, `MPI_File_read`)
- Для неблокирующих операций, ожидание их завершения (напр., `MPI_Wait`)
- Заккрытие файла (`MPI_File_close`)

# Базовый алгоритм для работы с файлами

```
MPI_File fh;
MPI_Datatype filetype;
MPI_Status status;
MPI_Offset offset;
int mode;
float data[100];
/* other code */
/* set offset and filetype */
mode = MPI_MODE_CREATE | MPI_MODE_RDWR;
MPI_File_open(MPI_COMM_WORLD, "myfile", mode, MPI_INFO_NULL,
    &fh);
MPI_File_set_view(fh, offset, MPI_FLOAT, filetype, "native",
    MPI_INFO_NULL);
MPI_File_write(fh, data, 100, MPI_FLOAT, &status);
MPI_File_close(&fh);
```

# Открытие файла

---

- **int MPI\_File\_open(MPI\_Comm comm, char \*filename, int amode, MPI\_Info info, MPI\_File \*fh)**
  - *comm* коммуникатор (дескриптор)
  - *filename* имя открываемого файла (строка)
  - *amode* тип доступа к файлу (целое)
  - *info* информационный объект (дескриптор)
  - *fh* новый дескриптор файла (дескриптор)
- открывает файл с именем *filename* для всех процессов из группы коммуникатора *comm*
- все процессы должны обеспечивать одинаковое значение *amode* и имена файлов, указывающие на один и тот же файл
- *info* используется как «подсказка» (шаблоны доступа)

# Типы доступа

---

- `MPI_MODE_RDONLY` -- только чтение,
- `MPI_MODE_RDWR` -- чтение и запись,
- `MPI_MODE_WRONLY` -- только запись,
- `MPI_MODE_CREATE` -- создавать файл, если он не существует,
- `MPI_MODE_EXCL` -- ошибка, если создаваемый файл уже существует,
- `MPI_MODE_DELETE_ON_CLOSE` -- удалять файл при закрытии,
- `MPI_MODE_UNIQUE_OPEN` -- файл не будет параллельно открыт где-либо еще,
- `MPI_MODE_SEQUENTIAL` -- файл будет доступен лишь последовательно,
- `MPI_MODE_APPEND` -- установить начальную позицию всех файловых указателей на конец файла.

# Заккрытие файла

---

- `int MPI_File_close(MPI_File *fh)`
  - *fh* дескриптор файла (дескриптор)
- сначала синхронизирует состояние файла затем закрывает файл, ассоциированный с *fh*
- пользователь должен обеспечить условие, чтобы все ожидающие обработки неблокирующие запросы и разделенные коллективные операции над *fh*, производимые процессом, были выполнены до вызова `MPI_FILE_CLOSE`

## Установка индивидуального указателя файла

---

`int MPI_File_seek(MPI_File fh, MPI_Offset offset, int whence)`

- *fh* дескриптор файла (дескриптор)
- *offset* – смещение от начала файла (всегда в БАЙТАХ!)
- *whence*:
  - `MPI_SEEK_SET` – указатель файла устанавливается на *offset*
  - `MPI_SEEK_CUR`: указатель файла на текущую позицию плюс *offset*
  - `MPI_SEEK_END`: указатель файла на конец файла плюс *offset*

По умолчанию при открытии файла:  
*offset*=0, всегда измеряется в *etype*,  
*etype* и *filetype* = `MPI_BYTE`



# Файловые виды (1)

---

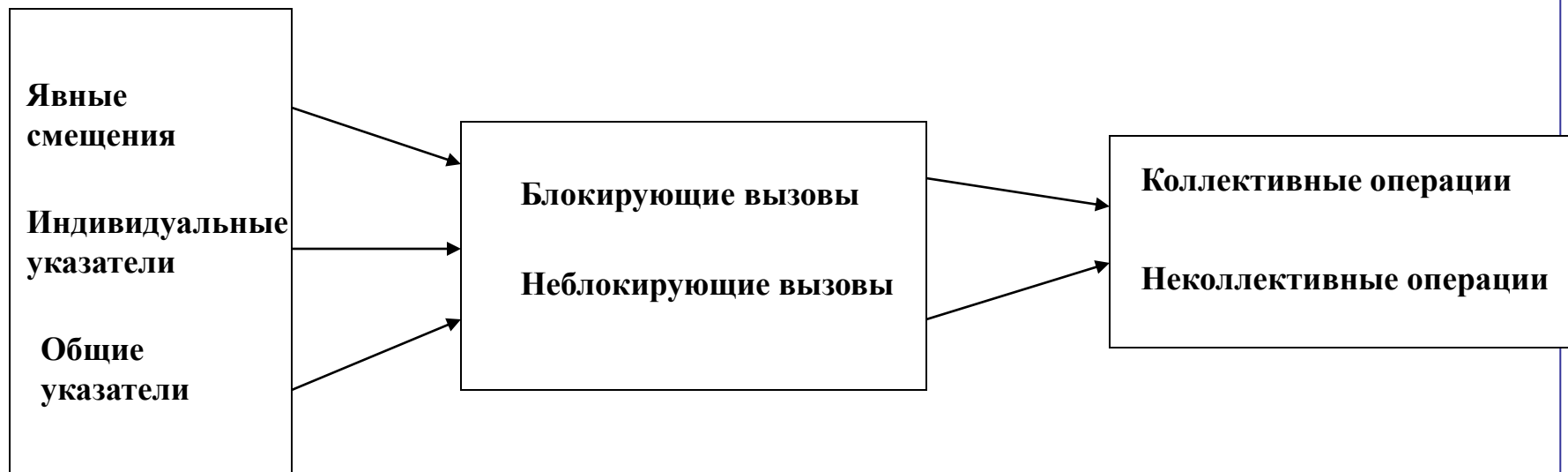
- **int MPI\_File\_set\_view(MPI\_File fh, MPI\_Offset disp, MPI\_Datatype etype, MPI\_Datatype filetype, char \*datarep, MPI\_Info info)**
  - *fh* дескриптор файла (дескриптор)
  - *disp* смещение (целое)
  - *etype* элементарный тип данных (дескриптор)
  - *filetype* тип файла (дескриптор)
  - *datarep* представление данных (строка)
  - *info* информационный объект (дескриптор)

# Файловые виды (2)

- 
- `int MPI_File_get_view(MPI_File fh, MPI_Offset *disp, MPI_Datatype *etype, MPI_Datatype *filetype, char *datarep)`
    - *fh* дескриптор файла (дескриптор)
    - *disp* смещение (целое)
    - *etype* элементарный тип данных (дескриптор)
    - *filetype* тип файла (дескриптор)
    - *datarep* представление данных (строка)

# Доступ к данным

---



Позиционирование	Синхронизация	Координация	
		неколлективные	коллективные
Явные смещения	блокирующие	<b>MPI_FILE_READ_AT MPI_FILE_WRITE_AT</b>	<b>MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL</b>
	неблокирующие и расщепленные коллективные	<b>MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT</b>	<b>MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END</b>
Индивидуальные указатели	блокирующие	<b>MPI_FILE_READ MPI_FILE_WRITE</b>	<b>MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL</b>
	неблокирующие и расщепленные коллективные	<b>MPI_FILE_IREAD MPI_FILE_IWRITE</b>	<b>MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END</b>
Общие указатели	блокирующие	<b>MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED</b>	<b>MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED</b>
	неблокирующие и расщепленные коллективные	<b>MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED</b>	<b>MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END</b>

# Доступ к данным

---

- `int MPI_File_read(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)`
  - *fh* - дескриптор файла (дескриптор)
  - *buf* - начальный адрес буфера (выбор)
  - *count* - количество элементов в буфере (целое)
  - *datatype* - тип данных каждого элемента буфера (дескриптор)
  - *status* - объект состояния (Status)
- `int MPI_File_write(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)`
  - *fh* - дескриптор файла (дескриптор)
  - *buf* - начальный адрес буфера (выбор)
  - *count* - количество элементов в буфере (целое)
  - *datatype* - тип данных каждого элемента буфера (дескриптор)
  - *status* - объект состояния (Status)

# Доступ к данным

- `int MPI_File_read_at(MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)`
  - INOUT *fh* - дескриптор файла (дескриптор)
  - OUT *buf* - начальный адрес буфера (выбор)
  - IN *count* - количество элементов в буфере (целое)
  - IN *datatype* - тип данных каждого элемента буфера (дескриптор)
  - OUT *status* - объект состояния (Status)
  
- `int MPI_File_write_at(MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)`
  - INOUT *fh* - дескриптор файла (дескриптор)
  - IN *buf* - начальный адрес буфера (выбор)
  - IN *count* - количество элементов в буфере (целое)
  - IN *datatype* - тип данных каждого элемента буфера (дескриптор)
  - OUT *status* - объект состояния (Status)

# Коллективные операции I/O

---

- Используя коллективные функции I/O, пользователь полагается на оптимизацию их выполнения
- Реализация коллективных функций I/O может быть выполнена с условием объединения
- Рекомендуется использовать в случае, когда доступ к файлу из разных процессов производится в произвольном порядке и может пересекаться по времени

# Коллективные операции I/O MPI-2

---

- `MPI_File_read_all`, `MPI_File_read_at_all`, etc
- `_all` означает, что все процессы, входящие в группу, заданную коммуникатором при открытии файла, должны вызвать эту функцию
- Каждый процесс определяет свою собственную информацию для выполнения этой функции -- список параметров такой же, как и для неколлективных операций



# Пример: коллективные операции

```
/* noncontiguous access with a single collective I/O function */
#include "mpi.h"

#define FILESIZE 1048576
#define INTS_PER_BLK 16

int main(int argc, char **argv)
{
    int *buf, rank, nprocs, nints, bufsize;
    MPI_File fh;
    MPI_Datatype filetype;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

    bufsize = FILESIZE/nprocs;
    buf = (int *) malloc(bufsize);
    nints = bufsize/sizeof(int);

    MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile", MPI_MODE_RDONLY,
        MPI_INFO_NULL, &fh);
```

## Пример: коллективные операции

```
MPI_Type_vector(nints/INTS_PER_BLK, INTS_PER_BLK,
                INTS_PER_BLK*nprocs, MPI_INT, &filetype);
MPI_Type_commit(&filetype);
MPI_File_set_view(fh, INTS_PER_BLK*sizeof(int)*rank, MPI_INT,
                  filetype, "native", MPI_INFO_NULL);

MPI_File_read_all(fh, buf, nints, MPI_INT, MPI_STATUS_IGNORE);
MPI_File_close(&fh);

MPI_Type_free(&filetype);
free(buf);
MPI_Finalize();
return 0;
}
```

# Неблокирующие коллективные операции I/O

- Ограниченная форма неблокирующих коллективных операций I/O
- Только **ОДНА** активная коллективная неблокирующая операция может выполняться в данный момент времени над заданным файловым указателем
- Не требуется request

```
MPI_File_write_all_begin(fh, buf, count, datatype);
```

```
for (i=0; i<1000; i++) {  
    /* perform computation */  
}
```

```
MPI_File_write_all_end(fh, buf, &status);
```

# Разделяемый файловый указатель

---

- Функции для работы с разделяемым указателем:
  - MPI\_File\_read\_shared
  - MPI\_File\_write\_shared
  - MPI\_File\_seek\_shared
  - MPI\_File\_iread\_shared
  - MPI\_File\_iwrite\_shared

Операция начинается над текущим указателем в файле, значение которого меняется после выполнения каждой операции

Процесс может сдвинуть файловый указатель -  
MPI\_File\_seek\_shared

Все процессы должны определить одинаковый file view.

# Разделенные коллективные упорядоченные операции

---

- Коллективные (блокирующие) упорядоченные операции:

- MPI\_File\_read\_ordered
- MPI\_File\_write\_ordered

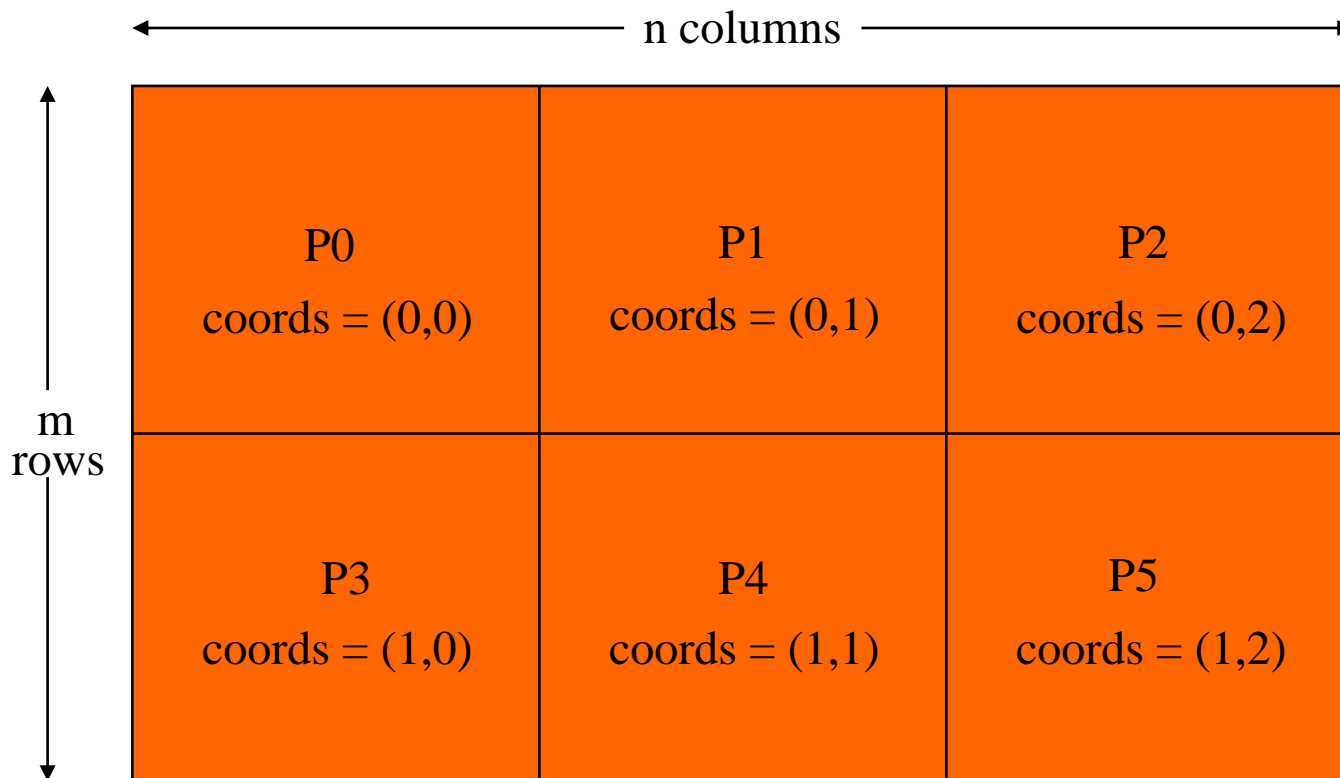
.....

- Разделенные коллективные неблокирующие упорядоченные операции:

- MPI\_File\_read\_ordered\_begin
- MPI\_File\_read\_ordered\_end

.....

# Доступ к массивам, хранящимся в файлах



$\text{nproc}(1) = 2, \text{nproc}(2) = 3$

# Дополнительные типы данных MPI, используемые для работы с файлами

---

- Distributed Array ( аналогично High Performance Fortran)
- Subarray
- INDEXED\_BLOCK

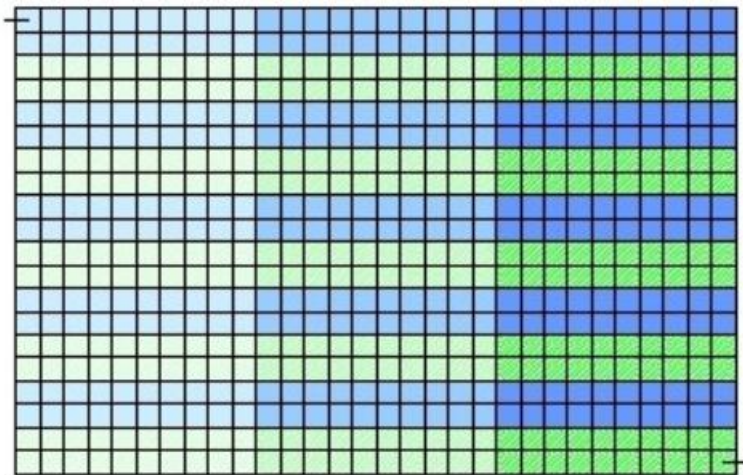
# Пример задания параметров

Пример распределения данных на процессной решетке:

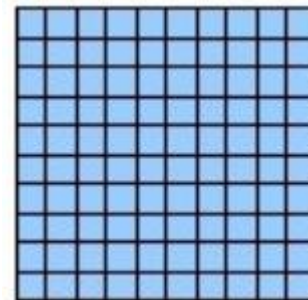
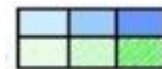
-CYCLIC(2), BLOCK:

- распределение с шагом 2 по первому измерению,
- блочное распределение

Глобальный массив



процессная решетка





# “Distributed Array” (Darray) Datatype

```
int gsizes[2], distribs[2], dargs[2], psizes[2];
gsizes[0] = m; /* no. of rows in global array */
gsizes[1] = n; /* no. of columns in global array */
distribs[0] = MPI_DISTRIBUTE_BLOCK;
distribs[1] = MPI_DISTRIBUTE_BLOCK;
dargs[0] = MPI_DISTRIBUTE_DFLT_DARG;
dargs[1] = MPI_DISTRIBUTE_DFLT_DARG;

psizes[0] = 2; /* no. of processes in vertical dimension
                of process grid */
psizes[1] = 3; /* no. of processes in horizontal dimension
                of process grid */
```

# Darray (2)

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
MPI_Type_create_darray(6, rank, 2, gsizes, distribs, dargs,  
    psizes, MPI_ORDER_C, MPI_FLOAT, &filetype);  
MPI_Type_commit(&filetype);
```

```
MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile",  
    MPI_MODE_CREATE | MPI_MODE_WRONLY,  
    MPI_INFO_NULL, &fh);  
MPI_File_set_view(fh, 0, MPI_FLOAT, filetype, "native",  
    MPI_INFO_NULL);
```

```
local_array_size = num_local_rows * num_local_cols;  
MPI_File_write_all(fh, local_array, local_array_size,  
    MPI_FLOAT, &status);  
MPI_File_close(&fh);
```

# MPI\_Type\_create\_darray

---

```
int MPI_Type_create_darray(  
    int size, /* число процессов */  
    int rank, /* номер процесса */  
    int ndims, /* размерность массива: и глоб, и локального */  
    int array_of_gsizes[], /* размеры глоб. массива*/  
    int array_of_distribs[], /* способ распределения элементов глоб.  
        массива. Например, MPI_DISTRIBUTE_BLOCK */  
    int array_of_dargs[], /*размер чанка, по умолч.  
        MPI_DISTRIBUTE_DFLT_DARGS */  
    int array_of_psize[], /* число элементов в процессорной решетке по  
        каждому измерению */  
    int order, /* способ представления лок. Массива: MPI_ORDER_C */  
    MPI_Datatype oldtype,  
    MPI_Datatype *newtype)
```

# Замечания по типу Darray

---

- Тип darray предполагает конкретную схему распределения данных – как в HPF
- Например, если размер массива не делится нацело на число процессов, то размер блока определяется округлением до ближайшего целого «сверху» ( $20 / 6 = 4$  )
- darray предполагает строчное распределение процессов в процессорной решетке (так, как это определяется для декартовых топологий в MPI-1)
- Для других схем распределения данных используется тип subarray .

# MPI\_Type\_create\_subarray

---

```
int MPI_Type_create_subarray (  
    int ndims,  
    int array_of_sizes[],  
    int array_of_subsizes[],  
    int array_of_starts[],  
    int order,  
    MPI_Datatype oldtype,  
    MPI_Datatype *newtype)
```

# Использование типа данных Subarray

```
gsizes[0] = m; /* no. of rows in global array */  
gsizes[1] = n; /* no. of columns in global array*/  
psizes[0] = 2; /* no. of procs. in vertical dimension */  
psizes[1] = 3; /* no. of procs. in horizontal dimension */  
  
lsizes[0] = m/psizes[0]; /* no. of rows in local array */  
lsizes[1] = n/psizes[1]; /* no. of columns in local array */  
dims[0] = 2; dims[1] = 3;  
periods[0] = periods[1] = 1;  
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &comm);  
MPI_Comm_rank(comm, &rank);  
MPI_Cart_coords(comm, rank, 2, coords);
```

# Subarray (продолжение)

```
/* global indices of first element of local array */
```

```
start_indices[0] = coords[0] * lsizes[0];
```

```
start_indices[1] = coords[1] * lsizes[1];
```

```
MPI_Type_create_subarray(2, gsizes, lsizes, start_indices,  
    MPI_ORDER_C, MPI_FLOAT, &filetype);
```

```
MPI_Type_commit(&filetype);
```

```
MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile",  
    MPI_MODE_CREATE | MPI_MODE_WRONLY,  
    MPI_INFO_NULL, &fh);
```

```
MPI_File_set_view(fh, 0, MPI_FLOAT, filetype, "native",  
    MPI_INFO_NULL);
```

```
local_array_size = lsizes[0] * lsizes[1];
```

```
MPI_File_write_all(fh, local_array, local_array_size,  
    MPI_FLOAT, &status);
```

# “Подсказки” (hint)

---

- Параметр в функциях: `MPI_File_open`, `MPI_File_set_view`, and `MPI_File_set_info`
- Тип: `MPI_Info`
- Глобальный объект
- Любая пара (key, value) может быть добавлена в `MPI_Info`.  
Типы key и value – строка. Размер строк заданы константами `MPI_MAX_INFO_KEY` and `MPI_MAX_INFO_VAL`,



# “Подсказки” (hint)

```
MPI_File fh; MPI_Info info;  
MPI_Info_create(&info);  
/* FOLLOWING HINTS ARE PREDEFINED IN MPI */  
/* no. of I/O devices across which the file should be striped */  
MPI_Info_set(info, "striping_factor", "16");  
/* the striping unit in bytes */  
MPI_Info_set(info, "striping_unit", "1048576");  
/* buffer size for collective I/O */  
MPI_Info_set(info, "cb_buffer_size", "8388608");  
/* no. of processes that should perform disk accesses during collective I/O */  
MPI_Info_set(info, "cb_nodes", "16");
```

# “Подсказки”

---

```
/* FOLLOWING ARE ADDITIONAL HINTS SUPPORTED BY ROMIO */
```

```
/* buffer size for data sieving in independent reads */
```

```
MPI Info set(info, "ind_rd_buffer_size", "2097152");
```

```
/* buffer size for data sieving in independent writes */
```

```
MPI Info set(info, "ind_wr_buffer_size", "1048576");
```

```
/* NOW OPEN THE FILE WITH THIS INFO OBJECT */
```

```
MPI File open(MPI_COMM_WORLD, "/pfs/datafile",  
MPI_MODE_CREATE | MPI_MODE_RDWR, info, &fh);
```

```
MPI Info free(&info); /* free the info object */
```

# Функции для работы с hint

---

```
int MPI_Info_create(MPI_Info *info)
```

```
int MPI_Info_set(MPI_Info info, const char *key, const char *value)
```

```
int MPI_Info_get(MPI_Info info, const char *key, int valuelen, char  
    *value, int *flag)
```

```
int MPI_Info_get_nkeys(MPI_Info info, int *nkeys)
```

```
int MPI_Info_get_nthkey(MPI_Info info, int n, char *key)
```

```
int MPI_Info_free(MPI_Info *info)
```

```
int MPI_File_get_info(MPI_File fh, MPI_Info *info_used)
```