Методы и средства параллельного программирования

3 курс кафедры СКИ
сентябрь – декабрь 2016 г.

Лектор доцент Н.Н.Попова

Лекция 10
21 ноября 2016 г.

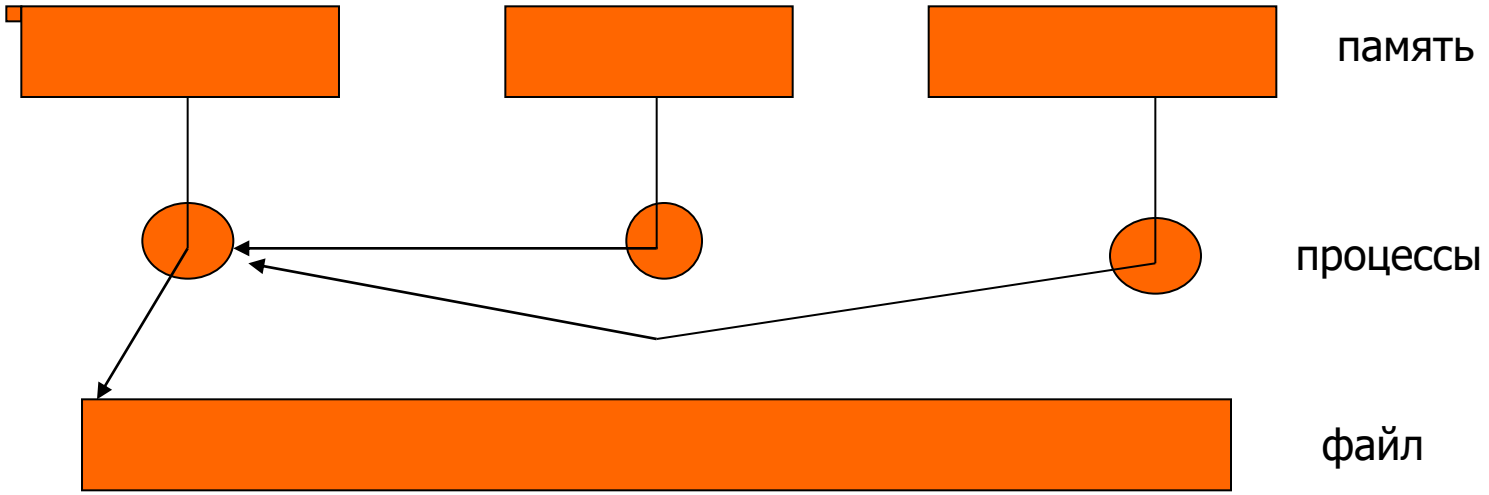
MPI-2.

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 подсказок (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

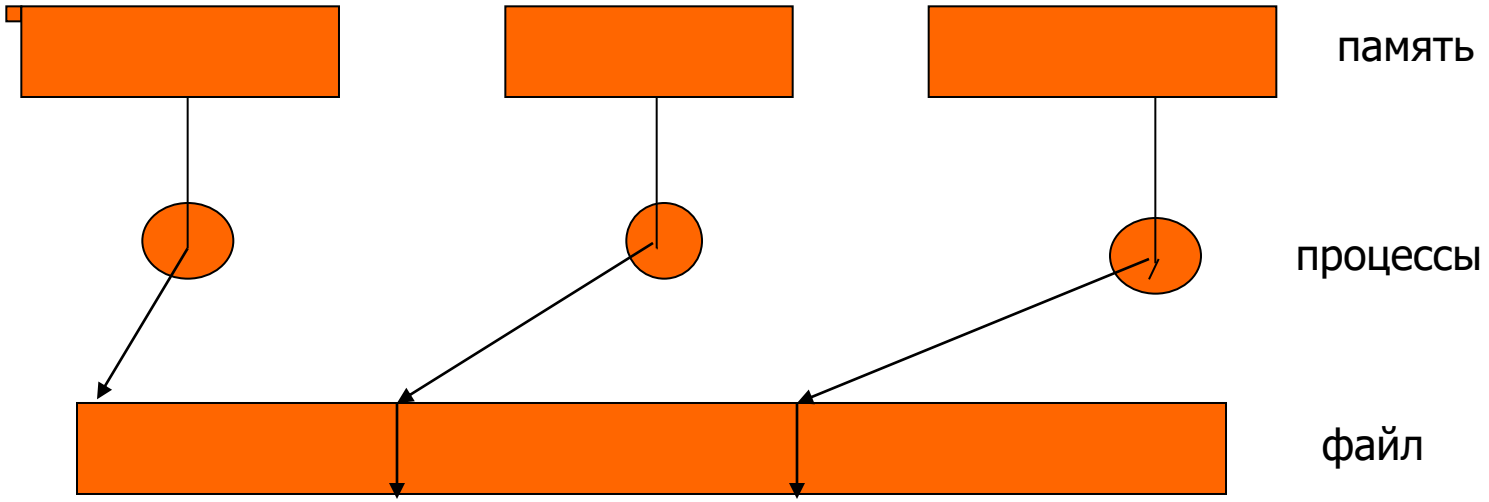
```
/* не MPI параллельная запись в разные файлы */
#include "mpi.h"
#include <stdio.h>
#define BUFSIZE 100
int main(int argc, char *argv[]) {
    int i, myrank, buf[BUFSIZE];
    char filename[128];
    FILE *myfile;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    for (i=0; 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, bufsize, *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

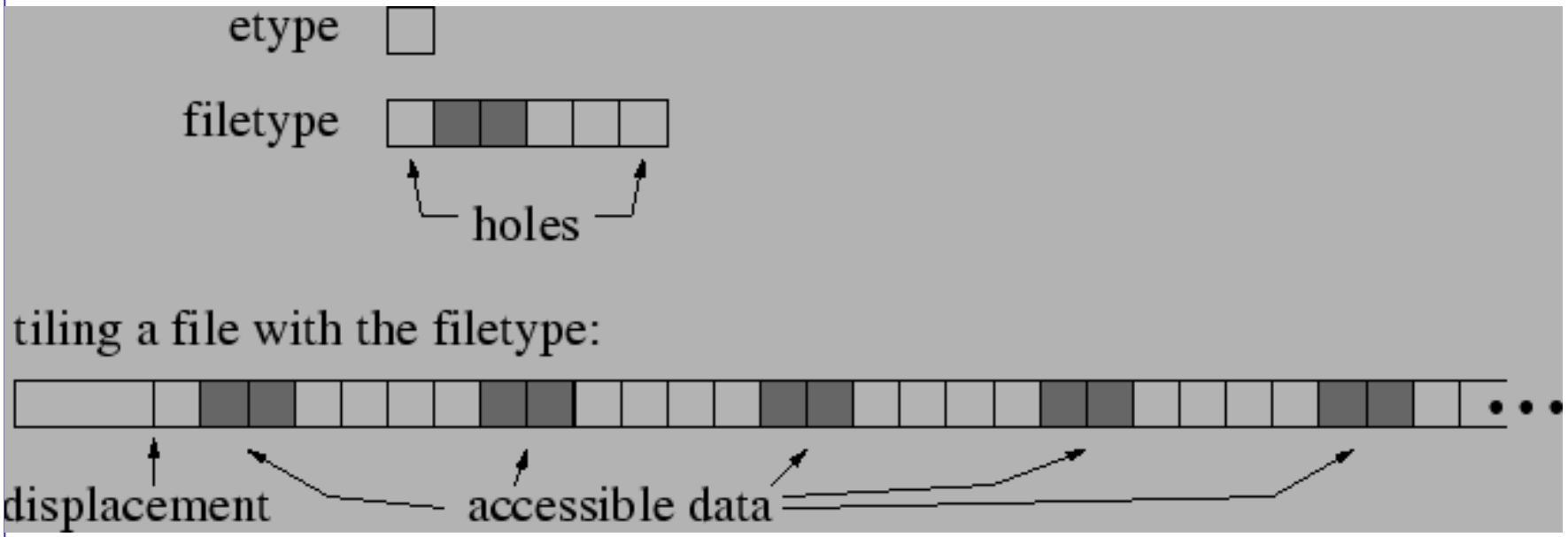
- ***E-тип (элементарный тип данных)*** - единица доступа к данным и позиционирования. Это может быть любой определенный в *MPI* или производный тип данных.

- ***Файловый тип*** –

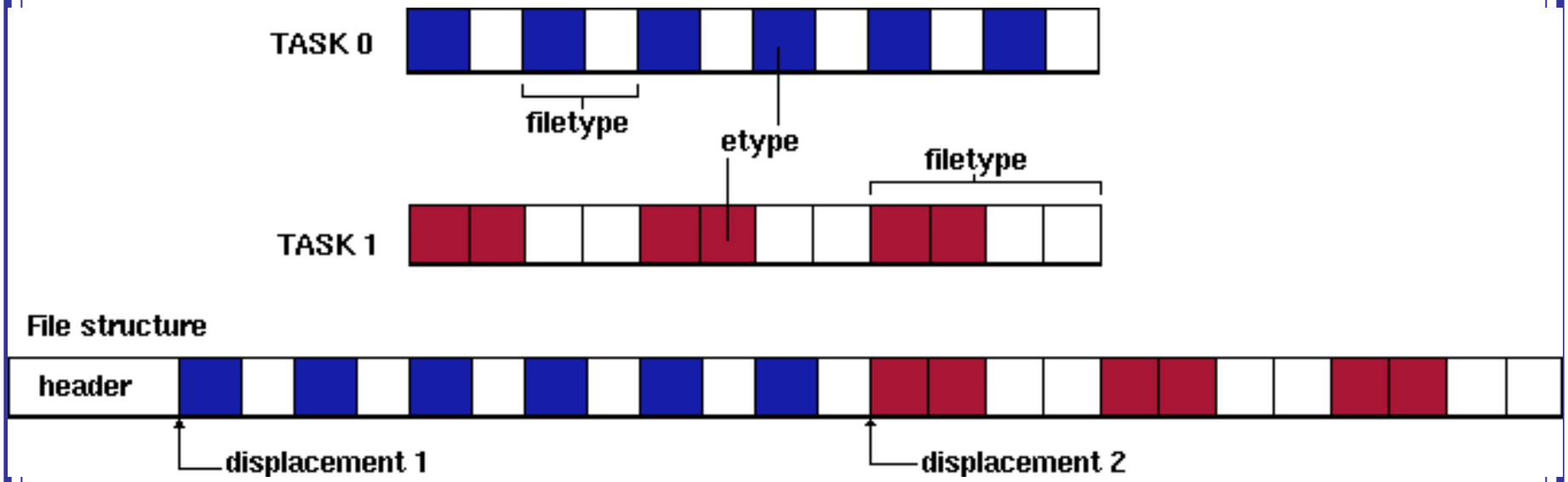
  базис для разбиения файла в среде процессов, определяет шаблон доступа к файлу.

  Обычный e-тип или производный тип данных *MPI*, состоящий из нескольких элементов одного и того же e-типа.

E-типы и Файловые типы



Файловые виды и структура файла



MPI-2 I/O: Терминология

- ***Вид*** (file view)- набор данных, видимый и доступный из открытого файла как упорядоченный набор е-типов. Каждый процесс имеет свой вид файла, определенный тремя параметрами: смещением, е-типом и файловым типом. Шаблон, описанный в файловом типе, повторяется, начиная со смещения.
- ***Смещение*** - это позиция в файле относительно текущего вида, представленная как число е-типов. ``Дыры'' в файловом типе вида пропускаются при подсчете номера этой позиции. Нулевое смещение - это позиция первого видимого е-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 е-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)
- Закрытие 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)**
  - IN *comm* коммуникатор (дескриптор)
  - IN *filename*имя открываемого файла (строка)
  - IN *amode* тип доступа к файлу (целое)
  - IN *info* информационный объект (дескриптор)
  - OUT *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.

Закрытие файла

- int MPI_File_close(MPI_File *fh)
  - INOUT *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)

- IN *fh* дескриптор файла (дескриптор)
- offset
- whence – MPI_SEEK_SET – начало файла

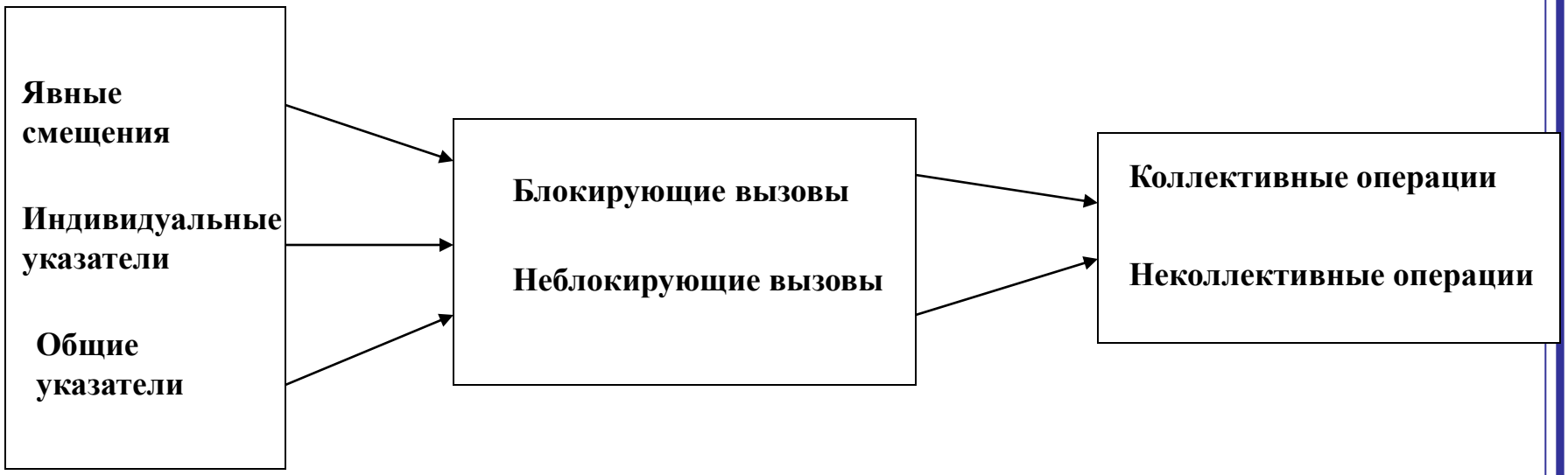
Файловые виды (1)

- **int MPI_File_set_view(MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, char *datarep, MPI_Info info)**
  - INOUT *fh* дескриптор файла (дескриптор)
  - IN *disp* смещение (целое)
  - IN *etype* элементарный тип данных (дескриптор)
  - IN *filetype* тип файла (дескриптор)
  - IN *datarep* представление данных (строка)
  - IN *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)
  - IN *fh* дескриптор файла (дескриптор)
  - OUT *disp* смещение (целое)
  - OUT *etype* элементарный тип данных (дескриптор)
  - OUT *filetype* тип файла (дескриптор)
  - OUT *datarep* представление данных (строка)

Доступ к данным



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

Доступ к данным

- int MPI_File_read(MPI_File fh, 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(MPI_File fh, 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)

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

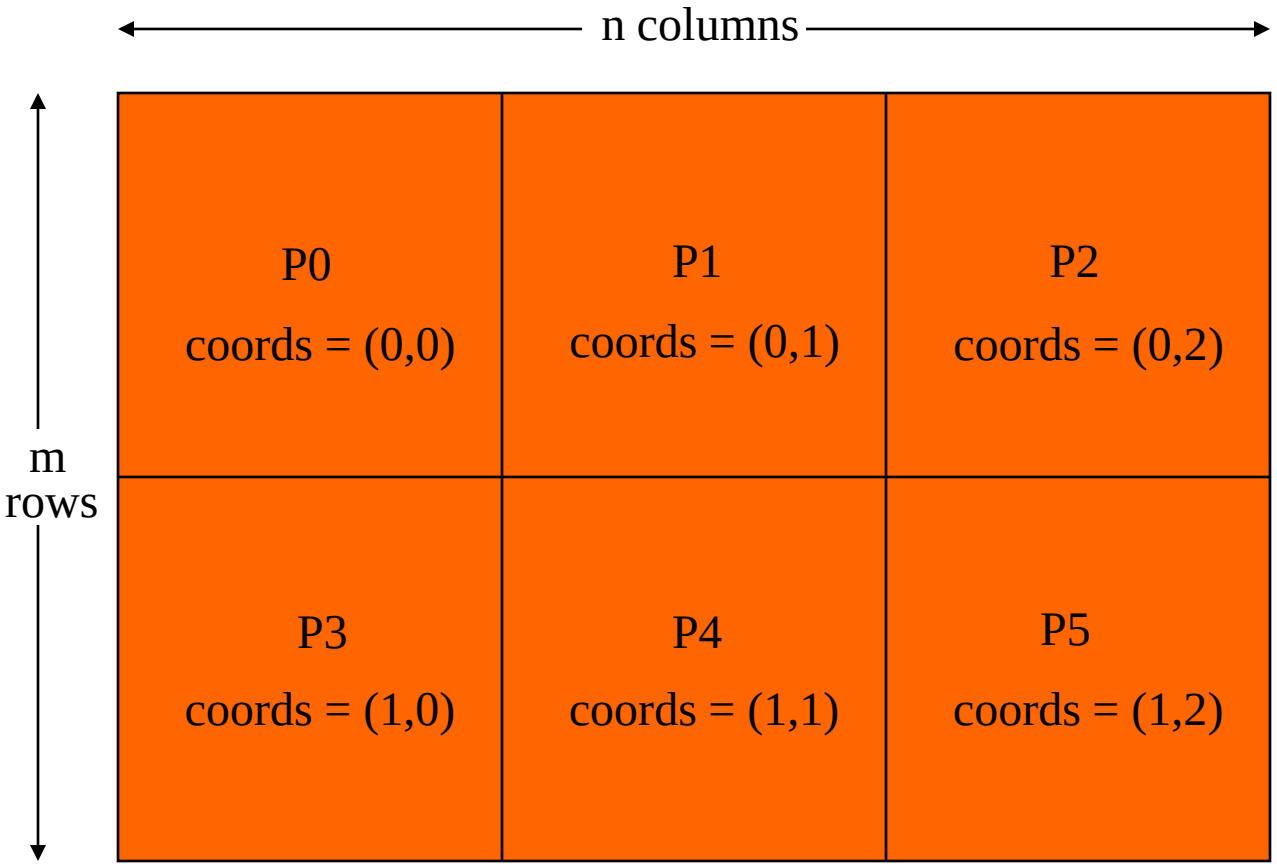
- Ограниченная форма неблокирующих коллективных операций I/O
- Только **ОДНА** активная коллективная неблокирующая операция может выполняться в данный момет времени над заданным файловым указателем
- Не требуется request

```
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);
```

Доступ к массивам, хранящимся в файлах



прос(1) = 2, прос(2) = 3

“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_psizes[], /* 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 построчное 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 Subarray Datatype

```
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 Datatype contd.

```
/* 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);
```