Системы и средства параллельного программирования

4 курс кафедры СКИ
сентябрь – декабрь 2016 г.

Лектор доцент Н.Н.Попова

Лекция 11
28 ноября 2016 г.

Тема

- Параллельный ввод/вывод (продолжение).

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

Разделенные коллективные упорядоченные операции

- Коллективные (блокирующие) упорядоченные операции:
  -MPI_File_read_orderded
  -MPI_File_write_orderded

……..

- Разделенные коллективные неблокирующие упорядоченные операции:
  - MPI_File_read_orderded_begin
  - MPI_File_read_orderded_end

……

Обработка ошибок IO

- Возвращаемое значение всех функций MPI_SUCCESS в случае нормального завершения
- В случае ошибки возвращается код (целое число), зависящий от реализации
- Возвращаемое значение может быть приведено к «стандартному» значению функцией MPI_Error_class
- MPI_Error_string может быть использована для получения текстового сообщения

MPI_Error_class

- int MPI_Error_class(*int errorcode, int *errorclass*);

MPI_Error_string

int MPI_Error_string(*int errorcode*, *char *string*, *int *resultlen*)

(IN) errorcode код ошибки, возвращаемой функцией MPI

(OUT) string текстовая строка, соответствующая коду ошибки **errorcode**

(OUT) resultlen длина строки **string**

Память для **string** должна быть не меньше MPI_MAX_ERROR_STRING. Реальная длина - **resultlen**.

Обработка ошибок IO:пример

```
    errcode = MPI_File_open(MPI_COMM_WORLD,
     “/gpfs/data/user_login/file.txt”,MPI_MODE_RDONLY,
     MPI_INFO_NULL,&fh);
     if (errcode != MPI_SUCCESS) {
    MPI_Error_class(errcode, &errclass);
     if (errclass == MPI_ERR_NO_SUCH_FILE)
            printf (“File doeas not exist \n”);
     else{
     MPI_Error_string(errcode, str, &len);
     printf ( “%s \n”, str);
     }
}
```

Error classes MPI

MPI_ERR_FILE Invalid file handle
MPI_ERR_NOT_SAME Collective argument not identical on all processes,
or collective routines called in a different order by different processes
MPI_ERR_AMODE Error related to the amode passed to MPI_FILE_OPEN
MPI_ERR_UNSUPPORTED_DATAREP Unsupported datarep passed to MPI_FILE_SET_VIEW
MPI_ERR_UNSUPPORTED_OPERATION Unsupported operation, such as seeking on a file which supports sequential access only
MPI_ERR_NO_SUCH_FILE File does not exist
MPI_ERR_FILE_EXISTS File exists
MPI_ERR_BAD_FILE Invalid file name (e.g., path name too long)
MPI_ERR_ACCESS Permission denied
MPI_ERR_NO_SPACE Not enough space
MPI_ERR_QUOTA Quota exceeded
MPI_ERR_READ_ONLY Read-only file or file system
MPI_ERR_FILE_IN_USE File operation could not be completed, as the file is currently open by some process
MPI_ERR_DUP_DATAREP Conversion functions could not be registered because a data representation identifier that was already defined was passed to MPI_REGISTER_DATAREP
MPI_ERR_CONVERSION An error occurred in a user supplied data conversion function
MPI_ERR_IO Other I/O error

Поддержка согласованности доступа к файлу

- MPI_FILE_SET_ATOMICITY(fh, flag)
- MPI_FILE_GET_ATOMICITY(fh, flag)
- MPI_FILE_SYNC(fh)

Поддержка согласованности MPI_File_sync

- int MPI_File_Sync (MPI_File fh)

  INOUT fh Дескриптор файла (дескриптор)

  Сброс всех буферов в файл. Операция коллективная! Должна выполняться ВСЕМИ процессами, входящими в коммуникатор, указанный при открытии файла.

Поддержка согласованности. Атомарность.

- int MPI_File_set_atomicity(MPI_File fh, int flag) - все записи в файл немедленно записываются на диск; коллективная операция
  - INOUT fh Дескриптор файла (дескриптор)
  - IN flag true для установки атомарного режима, false для отмены атомарного режима (логическая)
- int MPI_File_get_atomicity(MPI_File fh, int *flag) - возвращает текущее значение семантики непротиворечивости для операций доступа к данным
  - IN fh Дескриптор файла (дескриптор)
  - INOUT flag true при атомарном режиме, false при неатомарном режиме (логическая)

Операция должна выполняться во **ВСЕХ** процессах коммуникатора, указанного при открытии файла, параметр flag должен быть **ОДИНАКОВЫМ** у всех процессов.

Пример 1

- Файл открыт с использованием **MPI_COMM_WORLD**. Каждый процесс пишет в свою область в файле и читает то, что сам записал.

Process 0	Process 1
`MPI_File_open(MPI_COMM_WORLD,…)`	`MPI_File_open(MPI_COMM_WORLD,…)`
`MPI_File_write_at(off=0,cnt=100)`	`MPI_File_write_at(off=100,cnt=100)`
`MPI_File_read_at(off=0,cnt=100)`	`MPI_File_read_at(off=100,cnt=100)`

- MPI гарантирует, что данные будут считаны корректно.

Пример 2

- Аналогично примеру 1, за исключением того, что каждый процесс читает то, что записал другой. (overlapping accesses)
- В этом случае MPI не гарантирует, что данные автоматически будут считаны верно.

Process 0

```
/* incorrect program */
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=0,cnt=100)
MPI_Barrier
MPI_File_read_at(off=100,cnt=100)
```

Process 1

```
/* incorrect program */
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=100,cnt=100)
MPI_Barrier
MPI_File_read_at(off=0,cnt=100)
```

Пример 2 (продолж.)

- Пользователь должен предпринять специальные действия для того, чтобы обеспечить корректность.
- 3 возможных варианта действий:
  - установка atomicity в true
  - Закрыть и снова открыть файл
  - Убедиться, что никакая записывающая последовательность в каком-либо процессе не пересекается во времени ни с какой –либо другой последовательностью (чтения или записи) в других последовательностях

Пример 2, вариант1
Установка atomicity в true

Process 0

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_set_atomicity(fh1,1)
MPI_File_write_at(off=0,cnt=100)
MPI_Barrier
MPI_File_read_at(off=100,cnt=100)
```

Process 1

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_set_atomicity(fh2,1)
MPI_File_write_at(off=100,cnt=100)
MPI_Barrier
MPI_File_read_at(off=0,cnt=100)
```

Пример 2, вариант 2.Закрыть и снова открыть файл.

Process 0	Process 1
`MPI_File_open(MPI_COMM_WORLD,…)`	`MPI_File_open(MPI_COMM_WORLD,…)`
`MPI_File_write_at(off=0,cnt=100)`	`MPI_File_write_at(off=100,cnt=100)`
`MPI_File_close`	`MPI_File_close`
`MPI_Barrier`	`MPI_Barrier`
`MPI_File_open(MPI_COMM_WORLD,…)`	`MPI_File_open(MPI_COMM_WORLD,…)`
`MPI_File_read_at(off=100,cnt=100)`	`MPI_File_read_at(off=0,cnt=100)`

Поддержка согласованности.

- Критерий: последовательность операций записи в любом процессе не пересекается с любой последовательностью операций чтения в других процессах.
- *Последовательность* – множество операций IO, заключенная в пару операций MPI_File_sync, MPI_File_open, MPI_File_close. Последовательность записи – если в последовательности есть операция записи.
- Пример: sync-write-read-sync, open-write-close, sync-read-read-sync.
- MPI гарантирует, что данные, записанные одним процессом могут быть считаны другим процессом, если последовательность записи в одном процессе не пересекается ни с одной другой последовательностью любого другого процесса.

Пример 2, вариант 3

Process 0

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=0,cnt=100)
MPI_File_sync

MPI_Barrier

MPI_File_sync /*collective*/

MPI_File_sync /*collective*/

MPI_Barrier

MPI_File_sync
MPI_File_read_at(off=100,cnt=100)
MPI_File_close
```

Process 1

```
MPI_File_open(MPI_COMM_WORLD,…)

MPI_File_sync /*collective*/

MPI_Barrier

MPI_File_sync
MPI_File_write_at(off=100,cnt=100)
MPI_File_sync

MPI_Barrier

MPI_File_sync /*collective*/
MPI_File_read_at(off=0,cnt=100)
MPI_File_close
```

Пример 3

- Аналогично примеру 2, за исключением того, что каждый процесс использует **`MPI_COMM_SELF`** при открытии общего файла
- Единственный путь – убедиться, что ни одна записывающая последовательность в любом из процессов не пересекается с читающей последовательностью в любом другом процессе.

Пример 3

Process 0	Process 1
`MPI_File_open(MPI_COMM_SELF,…)`	`MPI_File_open(MPI_COMM_SELF,…)`
`MPI_File_write_at(off=0,cnt=100)`	
`MPI_File_sync`	
`MPI_Barrier`	`MPI_Barrier`
	`MPI_File_sync`
	`MPI_File_write_at(off=100,cnt=100)`
	`MPI_File_sync`
`MPI_Barrier`	`MPI_Barrier`
`MPI_File_sync`	
`MPI_File_read_at(off=100,cnt=100)`	`MPI_File_read_at(off=0,cnt=100)`
`MPI_File_close`	`MPI_File_close`

Переносимость (interoperability)

Означает:

- MPI-файл может использоваться обычной файловой системой
- MPI-файл может переноситься с одной вычислительной системы на другую
- MPI-файл, записанный на одной системе, может быть прочитан на другой, используя различные способы представления данных

Переносимость

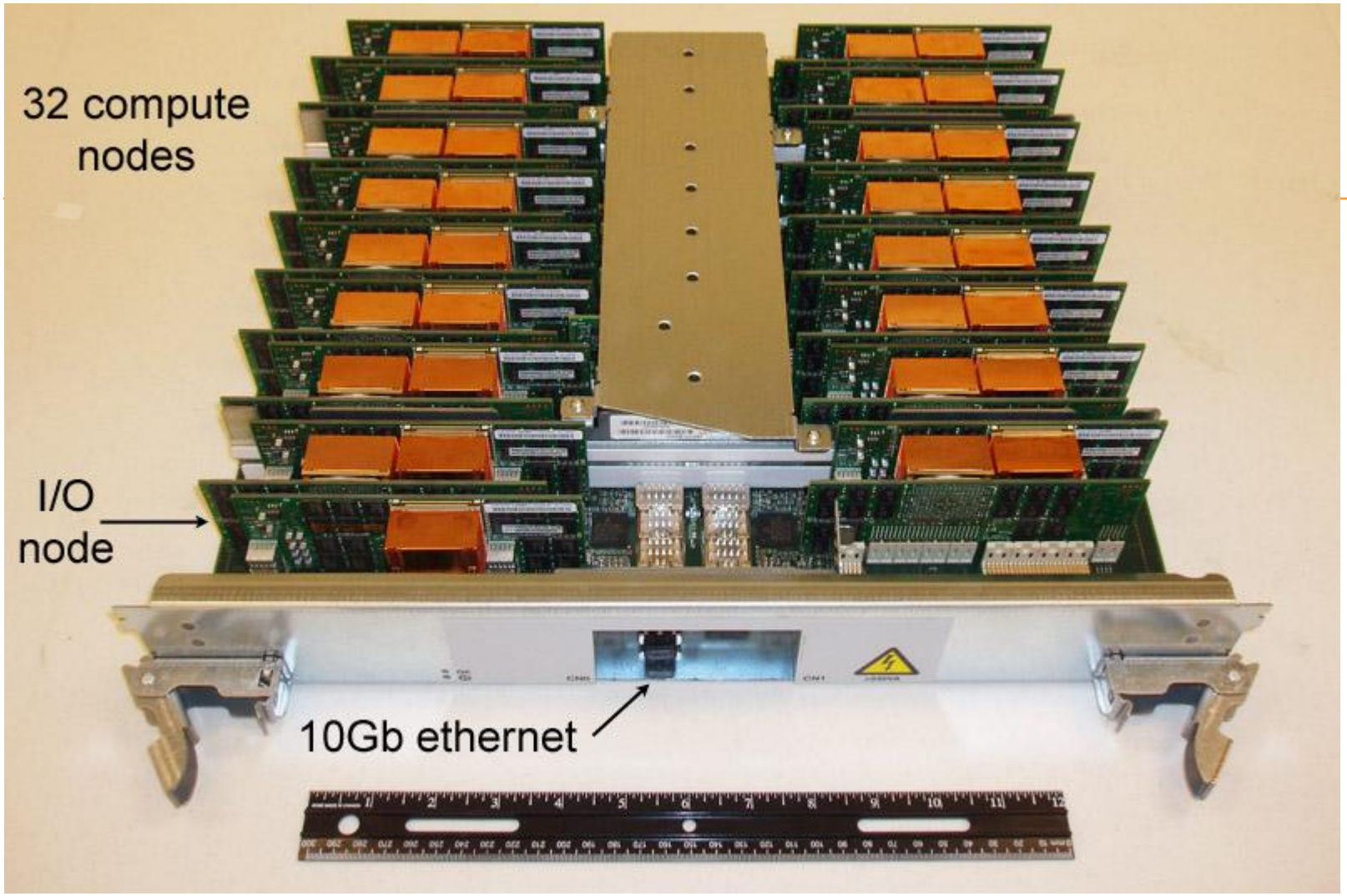
Обеспечивается:

- Использование datarep параметра функции MPI_File_set_view:

  - native
  - internal
  - external32 – 32-bit big-endian IEEE формат

- Использование производных MPI-типов

Особенности организации ввода-вывода суперкомпьютера IBM Blue Gene/P

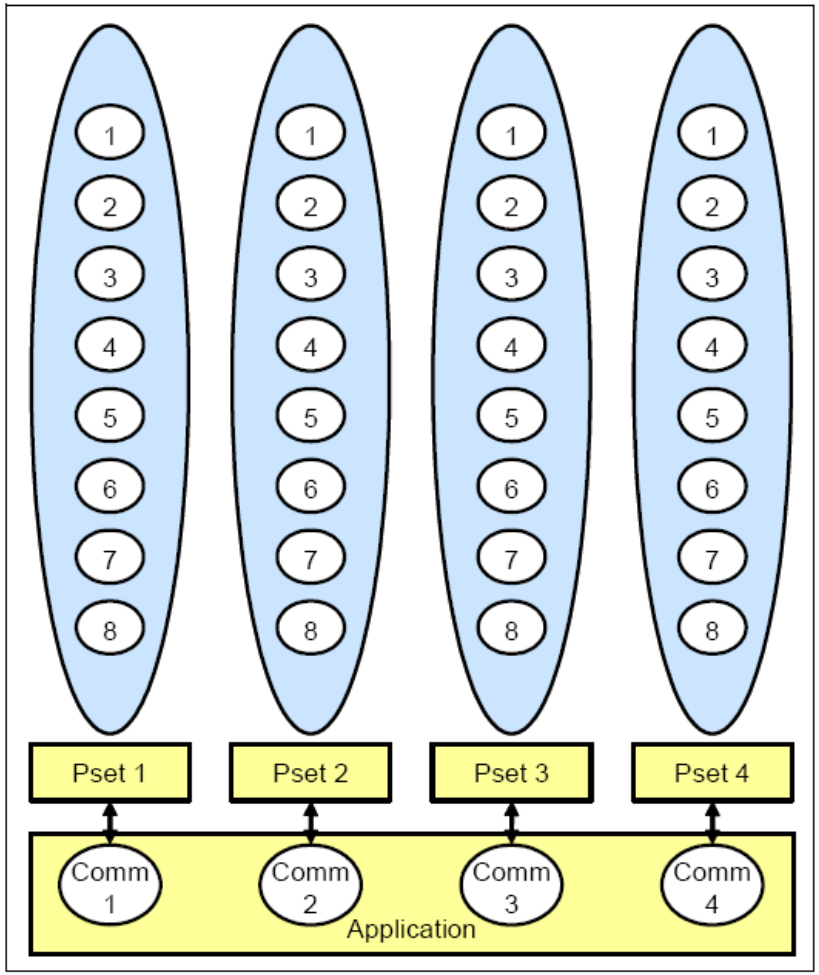

32 compute nodes
I/O node
10Gb ethernet

MPIX_Pset_same_comm_create

int MPIX_Pset_same_comm_create (MPI_Comm *pset_comm);

Коллективная функция. Создает множество коммуникаторов таких, что все узлы в коммуникаторе имеют один и тот же узел ввода-вывода.

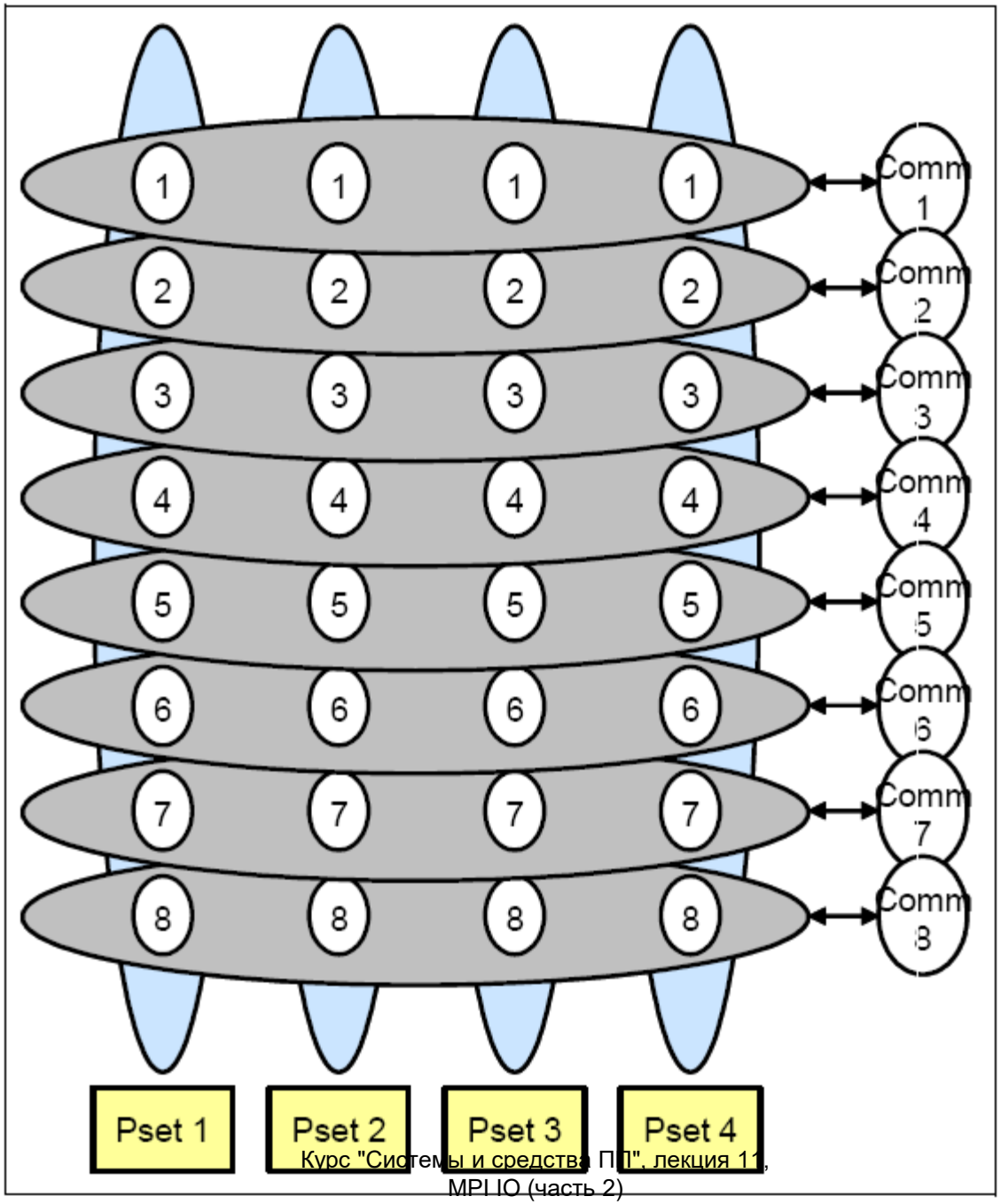
Пример: координация доступа к узлам ввода-вывода. Процесс 0 в коммуникаторе будет осуществлять ввод-вывод, все остальные передают ему данные для ввода-вывода.


1
2
3
4
5
6
7
8
Pset 1
Pset 2
Pset 3
Pset 4
Comm 1
Comm 2
Comm 3
Comm 4
Application

int MPIX_Pset_diff_comm_create

int MPIX_Pset_diff_comm_create (MPI_Comm *pset_comm);

Коллективная функция. Создает множество коммуникаторов таким образом, что любые 2 процесса в одном коммуникаторе имеют доступ к разным узлам ввода-вывода.


1
1
1
1
Comm 1
2
2
2
2
Comm 2
3
3
3
3
Comm 3
4
4
4
4
Comm 4
5
5
5
5
Comm 5
6
6
6
6
Comm 6
7
7
7
7
Comm 7
8
8
8
8
Comm 8
Pset 1
Pset 2
Pset 3
Pset 4

Пример (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <mpix.h>

int main(int argc, char **argv)
    {
        const MPI_Comm world_comm = MPI_COMM_WORLD;
        int world_rank;

        MPI_Comm pset_comm;
        int pset_rank, pset_size, pset_root = 0;

        MPI_Comm io_comm;
        int io_rank;

        int *buf = NULL;
        int buf_size;
```

Пример (2)

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(world_comm, &world_rank);

/* Create separate communicator for each pset */
MPIX_Pset_same_comm_create(&pset_comm);
MPI_Comm_rank(pset_comm, &pset_rank);
MPI_Comm_size(pset_comm, &pset_size);

/* Let's gather some data (say, world_rank) from each process of
 * pset_comm on the pset_root */
if (pset_rank == pset_root)
{
    buf_size = pset_size;
    buf = malloc(buf_size * sizeof(buf[0]));
}
```

Пример (3)

```
MPI_Gather(&world_rank, 1, MPI_INT, buf, 1, MPI_INT,
        pset_root, pset_comm);

    /* Create separate communicators where no process
share the same I/O node */
    MPIX_Pset_diff_comm_create(&io_comm);
    MPI_Comm_rank(io_comm, &io_rank);
```

Пример (4)

```
/* In each pset, only pset_root process works; these processes
    are in the same io_comm and use it for MPI-IO collectives */
        if (pset_rank == pset_root)
        {
            char *fname = "mpix.bin";
            MPI_File fout;

            MPI_File_open(io_comm, fname, MPI_MODE_CREATE
    | MPI_MODE_WRONLY, MPI_INFO_NULL, &fout);
```

Пример (5)

```
MPI_File_set_view(fout, io_rank * buf_size * sizeof(buf[0]),
                  MPI_INT, MPI_INT, "native", MPI_INFO_NULL);
        MPI_File_write_all(fout, buf, buf_size, MPI_INT,
  MPI_STATUS_IGNORE);
        MPI_File_close(&fout);
    }

    if (pset_rank == pset_root)
        free(buf);

    MPI_Finalize();

    return 0; }
```