

# Parallel IO concepts (MPI-IO and pHDF5)

Matthieu Haefele



Saclay, 6-7 March 2017,  
Parallel filesystems and parallel IO libraries PATC@Mds



# Outline Day 1

## Morning:

- HDF5 in the context of Input/Output (IO)
- HDF5 Application Programming Interface (API)
- Playing with Dataspace
- Hands on session

## Afternoon:

- Basics on HPC, MPI and parallel file systems
- Parallel IO with POSIX, MPI-IO and **Parallel HDF5**
- Hands on session (pHDF5)



# HPC machine architecture

**An HPC machine is composed of processing elements or cores which**

- Can access a central memory
- Can communicate through a high performance network
- Are connected to a high performance storage system

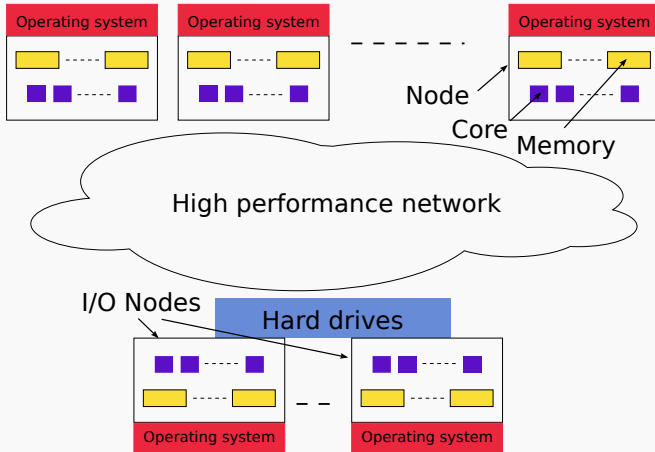
**Until now, two major families of HPC machines existed:**

- Shared memory machines
- Distributed memory machines

**New architectures like GPGPUs, Cell, FPGAs, ... are not covered here**



# Distributed memory machines





# MPI: Message Passing Interface

## **MPI is an Application Programming Interface**

- Defines a standard for developing parallel applications
- Several implementations exists (openmpi, mpich, IBM, Par-Tec...)

## **It is composed of**

- A parallel execution environment
- A library to link the application with



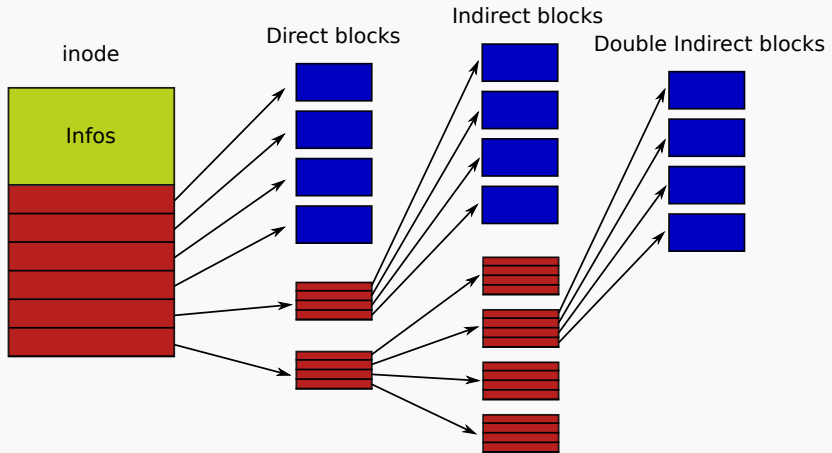
# MPI communications

## Four classes of communications

- **Collective**: all processes belonging to a same MPI communicator communicates together according to a defined pattern (scatter, gather, reduce, ...)
- **Point-to-Point**: one process sends a message to another one (send, receive)
- For both Collective or Point-to-Point, **blocking and non-blocking** functions are available



## inode pointer structure (ext3)





## **“Serial” file system**

**Meta-data, block address and file blocks are stored a single logical drive with a “serial” file system**

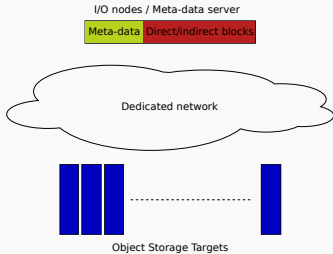
Logical drive







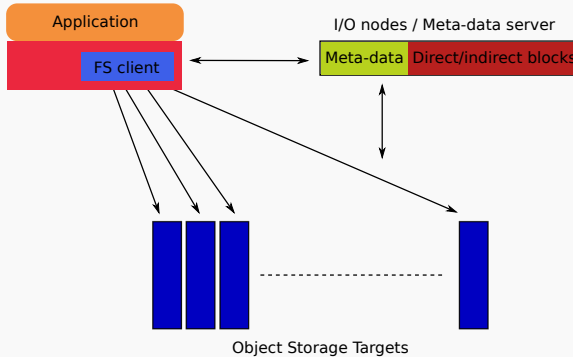
# Parallel file system architecture



- Meta-data and file blocks are stored on separate devices
- Several devices are used
- Bandwidth is aggregated
- A file is **striped** across different object storage targets.



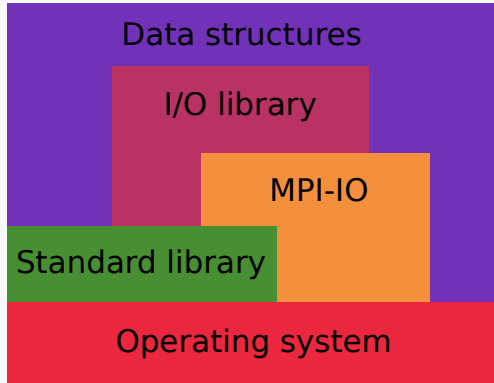
# Parallel file system usage



The file system client gives to the application the view of a "serial" file system

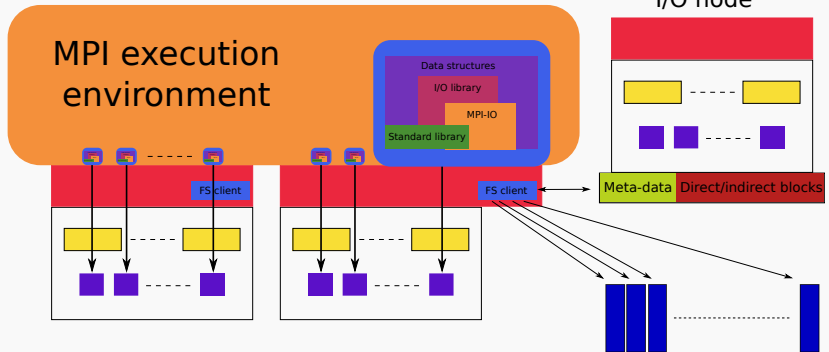


# The software stack



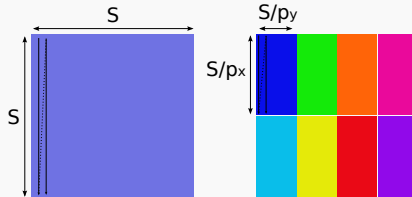


# Let us put everything together





## Test case to illustrate strategies



Let us consider:

- A 2D structured array
- The array is of size  $S \times S$
- A block-block distribution is used
- With  $P = p_x p_y$  cores



## Multiple POSIX files

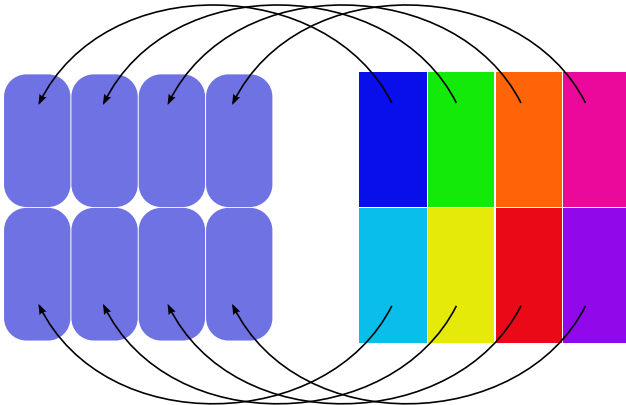
### Each MPI process writes its own file

- Pure “non-portable” binary files
  - A single distributed data is spread out in different files
  - The way it is spread out depends on the number of MPI processes
- ⇒ More work at post-processing level
- ⇒ May lead to huge amount of files (forbidden)
- ⇒ Very easy to implement



# Multiple POSIX files

## POSIX IO operations





## MPI gather + single POSIX file

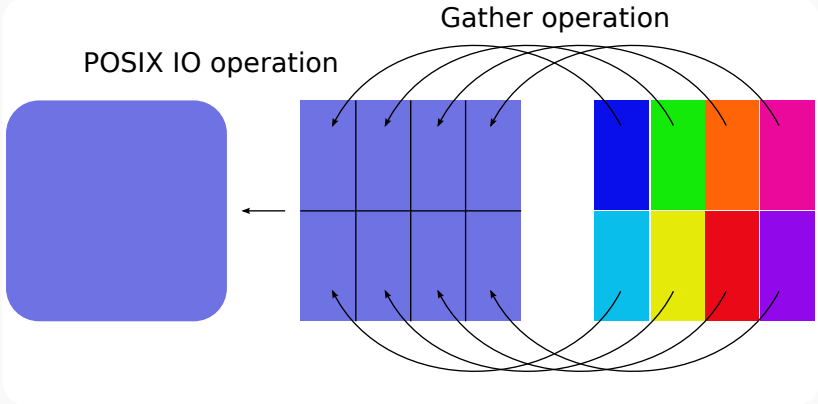
**A collective MPI call is first performed to gather the data on one MPI process. Then, this process writes a single file**

- Single pure “non-portable” binary file
  - The memory of a single node can be a limitation
- ⇒ Single resulting file





## MPI Gather + single POSIX file





## MPI-IO concept

- I/O part of the MPI specification
- Provide a set of read/write methods
- Allow one to describe how a data is distributed among the processes (thanks to MPI derived types)
- MPI implementation takes care of actually writing a single contiguous file on disk from the distributed data
- Result is identical as the gather + POSIX file

**MPI-IO performs the gather operation within the MPI implementation**

- No more memory limitation
- Single resulting file
- Definition of MPI derived types
- Performance linked to MPI library



# MPI-IO API

Level 0

Level 1

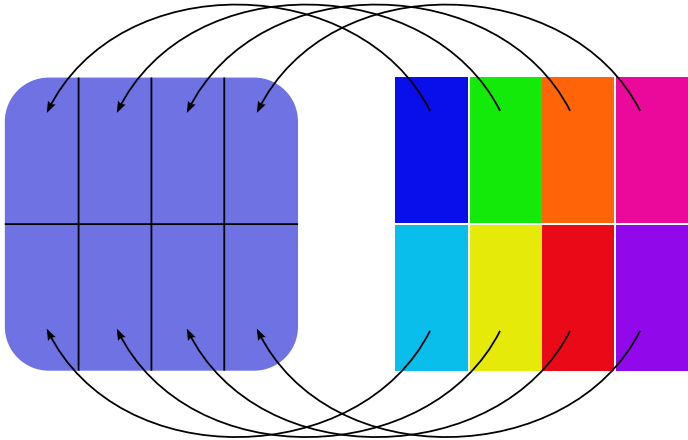
Positioning	Synchronism	Coordination	
		Non collective	Collective
Explicit offsets	Blocking	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	Non blocking & Split call	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
Individual file pointers	Blocking	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	Non blocking & Split call	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
Shared file pointers	Blocking	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	Non blocking & Split call	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

Level 2

Level 3

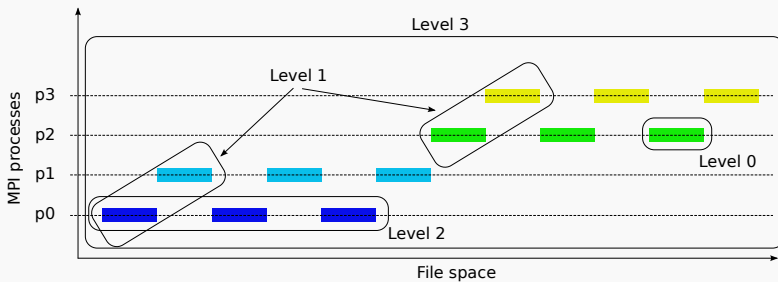


# MPI-IO





# MPI-IO level illustration





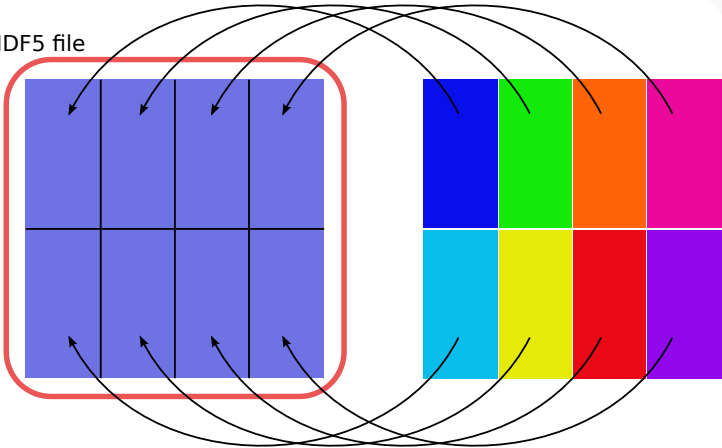
## Parallel HDF5

- Built on top of MPI-IO
- Must follow some restrictions to enable underlying collective calls of MPI-IO
- From the programming point of view, only few parameters have to be given to the HDF5 library
- Data distribution is described thanks to HDF5 hyper-slices
- Result is a single portable HDF5 file
- Easy to develop
- Single portable file
- Maybe some performance issues



# Parallel HDF5

HDF5 file





# Parallel HDF5 implementation

```
INTEGER(HSIZE_T) :: array_size(2), array_subsize(2), array_start(2)
INTEGER(HID_T) :: plist_id1, plist_id2, file_id, filespace, dset_id, memspace
array_size(1) = S
array_size(2) = S
array_subsize(1) = local_nx
array_subsize(2) = local_ny
array_start(1) = proc_x * array_subsize(1)
array_start(2) = proc_y * array_subsize(2)
```

## !Allocate and fill the tab array

```
CALL h5open_f(ierr)
CALL h5pcreate_f(H5P_FILE_ACCESS_F, plist_id1, ierr)
CALL h5pset_fapl_mpio_f(plist_id1, MPI_COMM_WORLD, MPI_INFO_NULL, ierr)
CALL h5fcreate_f('res.h5', H5F_ACC_TRUNC_F, file_id, ierr, access_prp = plist_id1)
```

## ! Set collective call

```
CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id2, ierr)
CALL h5pset_dxpl_mpio_f(plist_id2, H5FD_MPIO_COLLECTIVE_F, ierr)
```

```
CALL h5screate_simple_f(2, array_size, filespace, ierr)
CALL h5screate_simple_f(2, array_subsize, memspace, ierr)
```

```
CALL h5dcreate_f(file_id, 'pi_array', H5T_NATIVE_REAL, filespace, dset_id, ierr)
CALL h5sselect_hyperslab_f(filespace, H5S_SELECT_SET_F, array_start, array_subsize, ierr)
CALL h5dwrite_f(dset_id, H5T_NATIVE_REAL, tab, array_subsize, ierr, memspace, filespace, plist_id2)
```

## ! Close HDF5 objects





# IO technology comparison

## **Scientific results / diagnostics**






- Multiple POSIX files in ASCII or binary
- MPI-IO
- pHDF5
- XIOS

## **Restart files**

- SIONlib
- ADIOS



## IO technology comparison

	Dev/main cost	Performance	Post-processing
POSIX ASCII			
POSIX binary			
MPI-IO			
pHDF5			
XIOS			
SIONlib			
ADIOS			



# Hands-on parallel HDF5

1. git clone ??
2. **Parallel multi files:** all MPI ranks write their whole memory in separate file (provided)
3. **Serialized:** each rank opens the file and writes its data one after the other
  - 3.1 Data written as separate datasets
  - 3.2 Data written in the same dataset
4. **Parallel single file:** specific HDF5 parameters given at open and write time to let MPI-IO manage the concurrent file access