Post-processing issue
Introduction to HDF5 and XDMF

Matthieu Haefele

High Level Support Team

Max-Planck-Institut für Plasmaphysik, München, Germany

Lyon, 13 Janvier 2011

Outline

Post-processing definition

Post-processing is a treatment of numerical data that comes from either experiment measurements or numerical simulation.

- Signal processing (noise reduction, measures correction...)
- Diagnostics computing (features extraction)
- Visualization
- ...
- Anything that can improve the understanding of the data

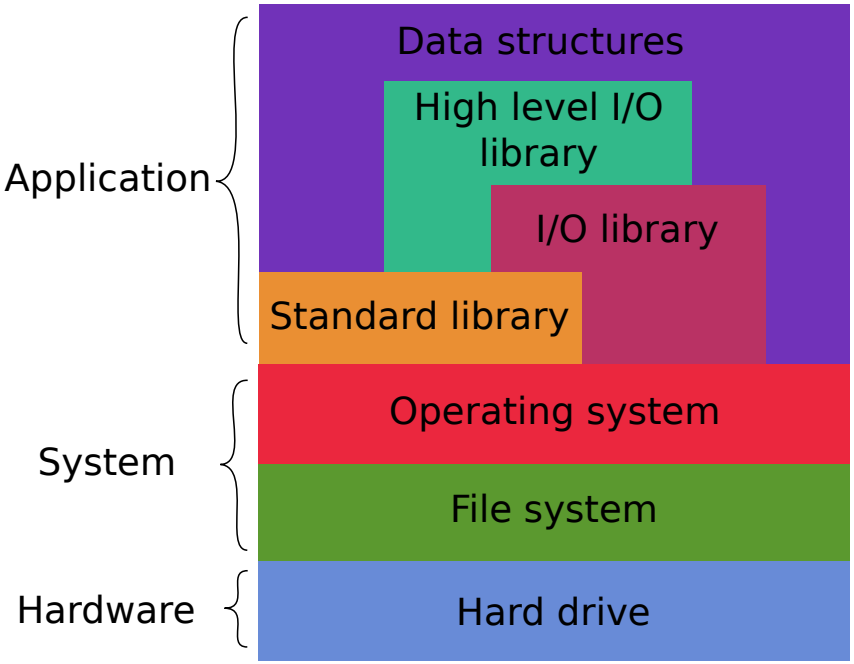
Identify technological requirements, constraints and choices

- How much can the **data source** be modified ?
- What are the **hardware** requirements/constraints/choices:
  - CPU
  - Memory
  - Network
  - Storage capacity
  - Storage system bandwidth
- What are the **software** requirements/constraints/choices:
  - Operating systems
  - Grid middleware
  - I/O library
  - Programing language

Post-processing general rules

- It involves read/write accesses from/to a storage system
- These Input/Output (I/O) accesses generally represent a large part of the post-processing
  - Execution time: bottleneck is often the storage system bandwidth
  - Development/maintenance time: file format design and implementation

Hardware/Software stack



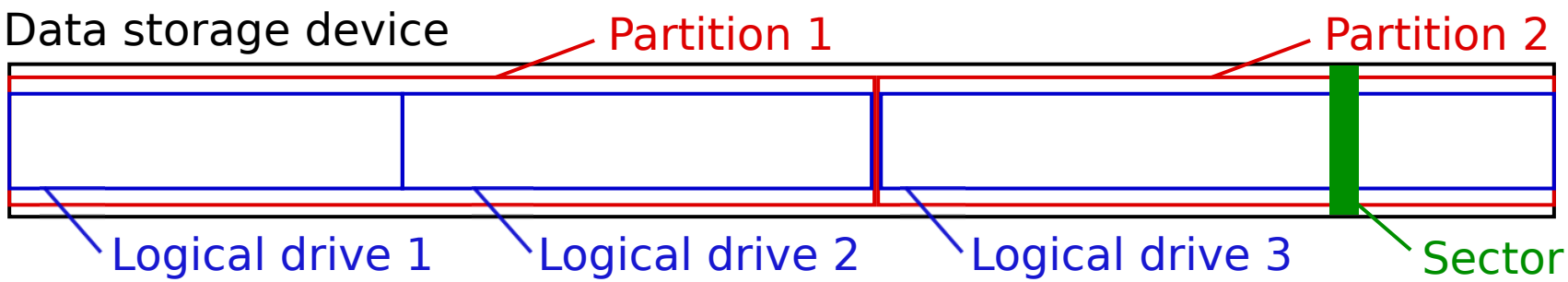
From the application level

- One file ⇔ one sequence of bytes
- These bytes flow through the operating system layer

Data storage device

A data storage device is a device for recording (storing) information (data). In the context of computer science:

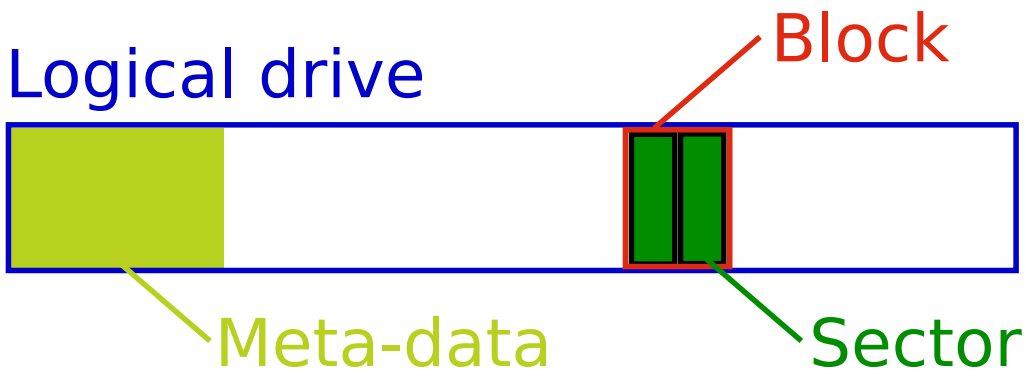
- A set of Bytes
- Organized as a 1D sequence
- Grouped by sectors (512 B, 1, 2, 4 KB)
- The sequence is cut into partitions
- Partitions can be cut into logical drives



File system

A file system is a method of storing and organizing computer files and their data.

- Meta-data
- Sectors are gathered in blocks or sectors (1-64)
- The block is the smallest amount of disk space that can be allocated to hold a file.

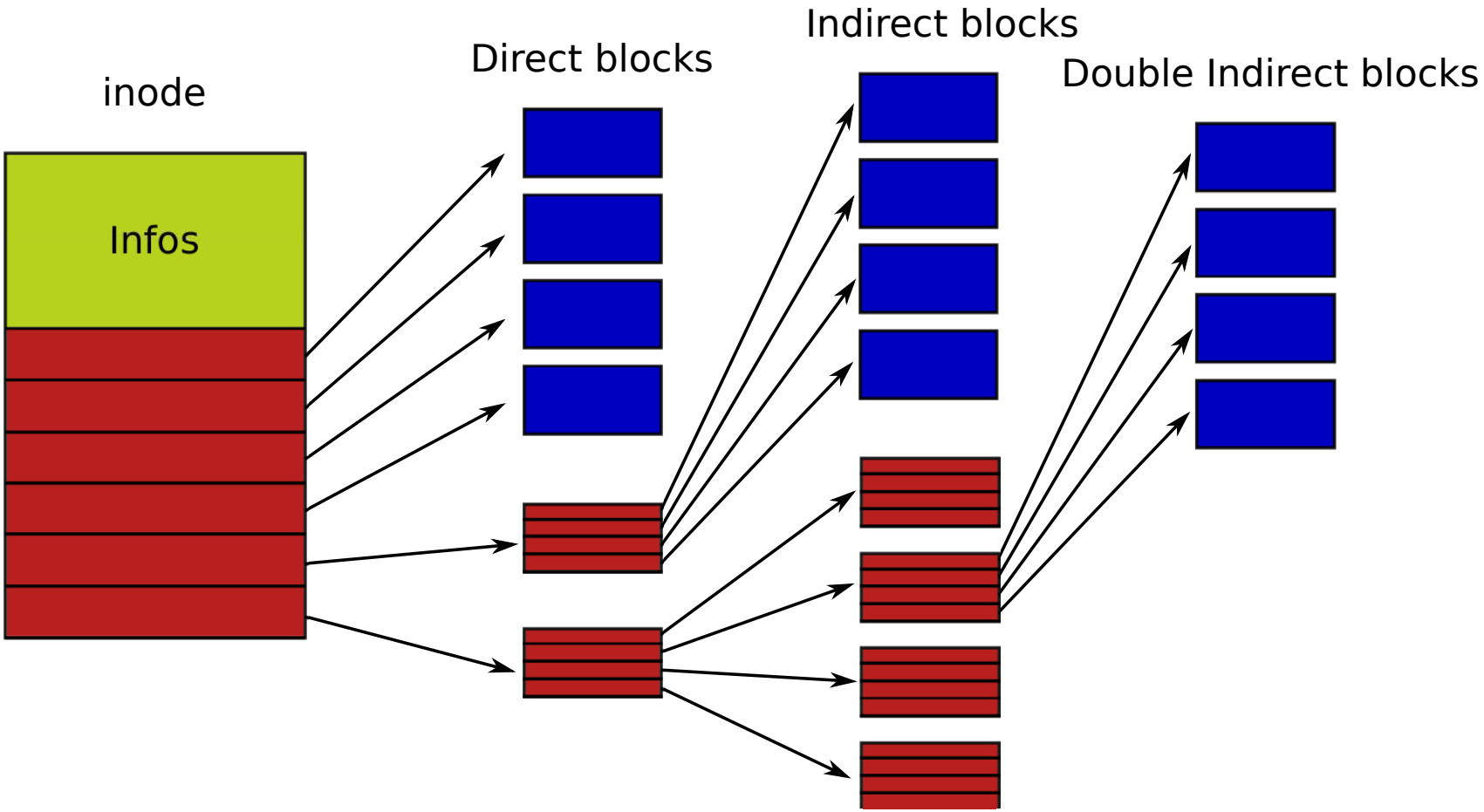


File (ext3)

A file is an inode in the file system. The inodes are stored in the file system meta-data and contain:

- File size
- Owner and Access rights
- Timestamps
- Link counts
- Pointers to the disk blocks that store the file's contents

inode pointer structure (ext3)



Kernel calls

I/O are performed through 3 functions:

```
off_t lseek(int fildes, off_t offset, int whence);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

Additional functions to manipulate the file system:

- *readdir, mkdir*, ...: Manipulating directories
- *link, symlink, unlink*, ...: Manipulating links
- *open, dup, close*, ...: Manipulating files
- *fcntl, flock, stat*, ...: Manipulating files cont.
- ...

Standard library

I/O are performed through 5 functions:

```
int fseek (FILE *stream, long offset, int whence);
size_t fread (void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite (const void *ptr, size_t size, size_t nmemb, \
               FILE *stream);
int fscanf (FILE *stream, const char *format, ...);
int fprintf (FILE *stream, const char *format, ...);
```

Additional functions to manipulate the file system:

- *opendir*, ...: Manipulating directories
- *fopen, fdup, fclose*, ...: Manipulating files
- ...

Two main representations of floating point numbers

ASCII representation: array of characters

- One byte per digit
- Minus, plus sign, comma, e signs and carriage return take also 1 byte each

IEEE 754 representation: $m \times 2^e$

- m: significand or mantissa
- e: exponent

Type	Sign	Exponent	Significand	Total bits
Half	1	5	10	16
Single	1	8	23	**32**
Double	1	11	52	**64**
Quad	1	15	112	128

ASCII I/O

```
int fscanf (FILE *stream, const char *format, ...);
int fprintf (FILE *stream, const char *format, ...);
```

Read: Disk content is turned into the memory number representation and dumped in memory

Write: Memory content is turned into an array of characters and dumped on disk

- Non optimal performance
  - CPU involved in the translation
  - Several calls are needed to read/write the whole data
- Storage overhead: each stored character takes a Byte of memory
- Machine independent
- Human readable files

Binary I/O

```
size_t fread (void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite (const void *ptr, size_t size, size_t nmemb, \
               FILE *stream);
```

Read: Memory content is dumped on disk
Write: Disk content is dumped into memory

- Most efficient method (no CPU, 1 single call if contiguous data)
- No storage overhead
- Can be machine dependent
  - Floating point data are now normalized by IEEE
  - Only endianness portability issues remain
- Non human readable files

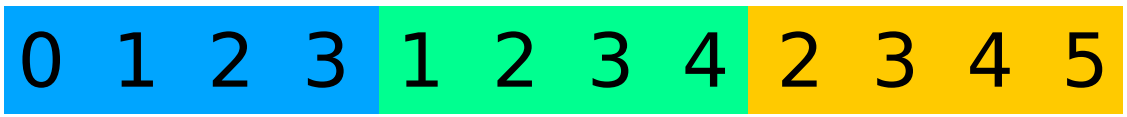
C order versus Fortran order

```c
/* C language */
#define NX 4
#define NY 3
int x,y;
int f[NY][NX];

for (y=0;y<NY;y++)
  for (x=0;x<NX;x++)
    f[y][x] = x+y;
```

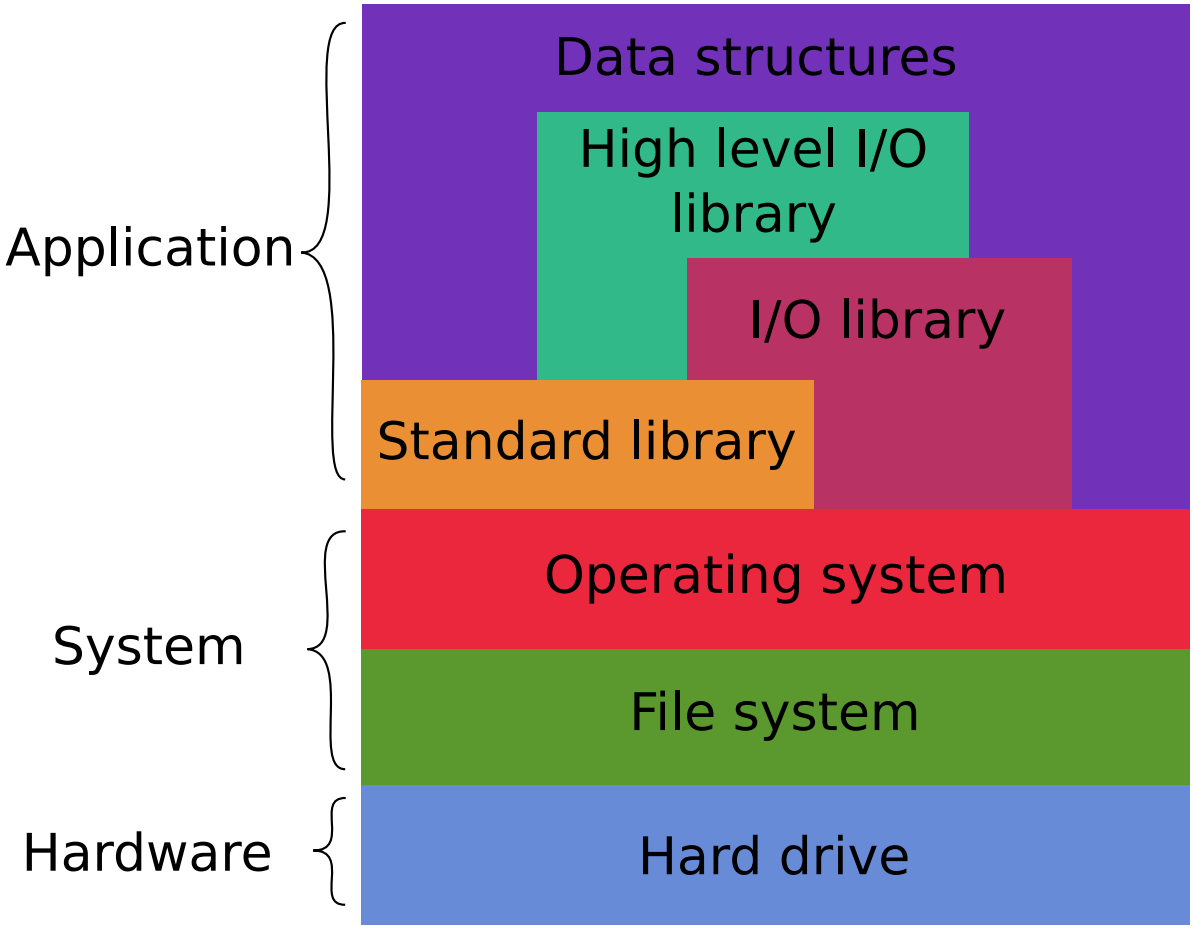
```
! Fortran language
integer, parameter :: NX=4
integer, parameter :: NY=3
integer                    :: x,y
integer, dimension(NX,NY) :: f

do y=1,NY
  do x=1,NX
    f(x,y) = (x-1) + (y-1)
  enddo
enddo
```



The memory mapping is identical, the language semantic is different !!

Hardware/Software stack



I/O libraries

The purpose of I/O libraries is to provide:

- Efficient I/O
- Portable binary files
- Higher level of abstraction for the developer

Two main existing libraries:

- Hierarchical Data Format: HDF5
- Network Common Data Form: NetCDF

HDF5 is becoming a standard and parallel NetCDF is built on top of parallel HDF5

HDF5 library

An HDF5 file consists of:

- HDF5 group: a grouping structure containing instances of zero or more groups or datasets
- HDF5 dataset: a multidimensional array of data elements

An HDF5 dataset is a multidimensional array and consists of:

- Name
- Datatype (Atomic, NATIVE, Compound)
- Dataspace (rank, sizes, max sizes)
- Storage layout (contiguous, compact, chunked)

HDF5 library API

- **H5F**: File-level access routines
- **H5G**: Group functions, for creating and operating on groups of objects
- **H5S**: Dataspace functions, which create and manipulate the dataspace in which the elements of a data array are stored
- **H5D**: Dataset functions, which manipulate the data within datasets and determine how the data is to be stored in the file
- . . .

HDF5 High Level APIs

- HDF5 **Lite** API (H5LT): Enables to write simple dataset in one call
- HDF5 **Image** API (H5IM): Enables to write images in one call
- HDF5 **Table** API (H5TB): Hides the compound types needed for writing tables
- . . .

HDF5 Tools

- h5ls: List the groups and dataset of a file
- h5dump: Dump the content of an HDF5 file on the standard output
- h5diff: Compare two hdf5 files
- hdfview: Spreadsheet representation of a HDF5 file
- ...

HDF5 first example

```
#define NX      5
#define NY      6
#define RANK    2

int main (void)
{
    hid_t       file, dataset, dataspace;
    hsize_t     dimsf[2];
    herr_t      status;
    int         data[NX][NY];

    init(data);
    file = H5Fcreate("example.h5", H5F_ACC_TRUNC, H5P_DEFAULT,\
                     H5P_DEFAULT);
    dimsf[0] = NX;
    dimsf[1] = NY;
```

HDF5 first example cont.

```
    dataspace = H5Screate_simple(RANK, dimsf, NULL);

    dataset = H5Dcreate(file, "IntArray", H5T_NATIVE_INT, \
                        dataspace, H5P_DEFAULT);

    status = H5Dwrite(dataset, H5T_NATIVE_INT, H5S_ALL, \
                      H5S_ALL,H5P_DEFAULT, data);

    H5Sclose(dataspace);
    H5Dclose(dataset);
    H5Fclose(file);

    return 0;
}
```

HDF5 high level example cont.

```
    status = H5LTmake_dataset_int(file, "IntArray", RANK, dimsf, data);

    H5Fclose(file);

    return 0;
}
```

HDF5 conclusion

HDF5 is not a format. It is an I/O library which:

- Provides efficient I/O
- Creates portable binary files
- Gives the developer an interface to manipulate groups and datasets rather than binary streams
- Allows one to define his own format

High level I/O libraries

The purpose of high level I/O libraries is to provide the developer a higher level of abstraction to manipulate computational modeling objects

- Meshes of various complexity (rectilinear, curvilinear, unstructured. . .)
- Discretized functions on such meshes
- Materials
- . . .

Until now, these libraries are mainly used in the context of visualization

Existing libraries

- Silo
  - Wide range of objects
  - Built on top of HDF5
  - “Native” format for VisIt
- Exodus
  - Focused on unstructured meshes and finite element representations
  - Built on top of NetCDF
- Famous/intensively used codes’ output format
- eXtensible Data Model and Format (XDMF)

XDMF

XDMF is an XML language that allows one to describe complex computational modeling objects from a set of datasets

An XDMF representation consists of:

- **Light data**: An XML file containing XDMF language statements and references to datasets contained in the heavy data
- **Heavy data**: A set of binary or HDF5 files

A flexible design

1. Existing data can be easily brought into the framework
   $\Rightarrow$ **XML file written by hand**
2. Existing I/O procedures can be kept untouched
   $\Rightarrow$ **XML file written in addition within the procedure**
3. I/O procedures are modified to write data through XDMF API
   $\Rightarrow$ **Both heavy and light data written by the XDMF library**

XDMF first example

```
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd">
<Xdmf Version="2.0">
 <Domain>
  <Grid Name="Structured mesh" GridType="Uniform">
   <Topology TopologyType="2DRectMesh" Dimensions="3 4"/>
   <Geometry GeometryType="VXVYVZ">
    <DataItem Format="XML" Dimensions="3" NumberType="Float" Precision="4">
     0.0 0.5 1.0
    </DataItem>
    <DataItem Format="XML" Dimensions="4" NumberType="Float" Precision="4">
     0.0 1.0 2.0 3.0
    </DataItem>
   </Geometry>
   <Attribute Name="Node Centered Values" Center="Node">
    <DataItem Format="HDF" Dimensions="12" NumberType="Int">
     basic_topology2d.h5:/values
    </DataItem>
   </Attribute>
  </Grid>
 </Domain>
</Xdmf>
```

Conclusion

Four levels of interfaces to perform I/O:

- High level I/O libraries
- I/O libraries
- Standard library
- Kernel call

I/O and high level I/O libraries

- need to be mastered
- introduce a software dependency, so portability and durability issues
- provide higher level API, so less code and more maintainable code

Performance is another story. . .