Post-processing issue
Introduction to HDF5

Matthieu Haefele

High Level Support Team

Max-Planck-Institut für Plasmaphysik, München, Germany

Autrans, 26-30 Septembre 2011,

École d'été Masse de données : structuration, visualisation

Monday: Serial IO, HDF5 and XDMF

- Monday 1 MH : Post-processing and introduction to HDF5
- Monday 2 MH : Hands on session on HDF5
- Monday 3 MH : Advanced HDF5 and XDMF
- Monday 4 MH : Hands on session on HDF5 and XDMF

Tuesday: Parallel IO, MPI-IO and parallel HDF5

- Tuesday 1 PW : Introduction super computer architectures and MPI
- Tuesday 2 PW : Parallel file system
- Tuesday 3 MH : Parallel IO methods
- Tuesday 4 PW+MH : Hands on session on MPI-IO and parallel HDF5

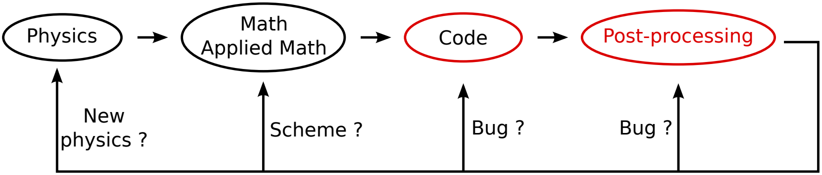
Wednesday: Your day !

- Wednesday 1 : Presentation of your problematic
- Wednesday 2 MH : Post-processing chain
- Wednesday 3-4 MH : Multiple choices
  - inkscape (small presentation + hands on)
  - python basic (presentation)
  - python numpy (small presentation + hands on)
  - python matplotlib (small presentation + hands on)
  - application of XDMF/HDF5 to user projects

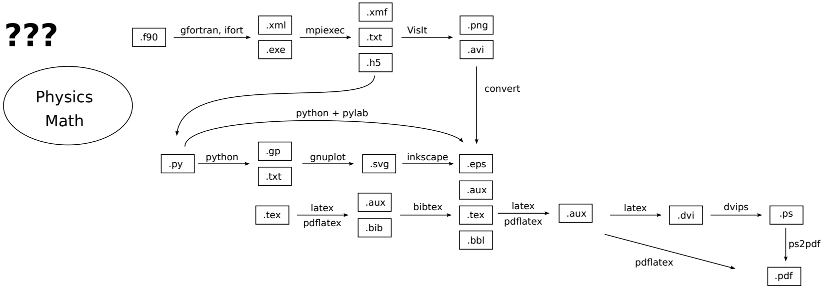
Outline for this morning

1. Introduction to post-processing
2. Starting from file systems and operating systems
   - Hardware → Operating System
   - Operating System → Application
   - IO libraries
3. HDF5 library
   - Concepts and API
   - Detailed example
   - Hands on

Numerical science “process”



Part of the process in numerical science



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

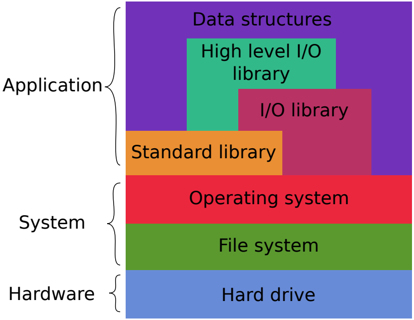
Identify technological requirements, constraints and choices

- 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



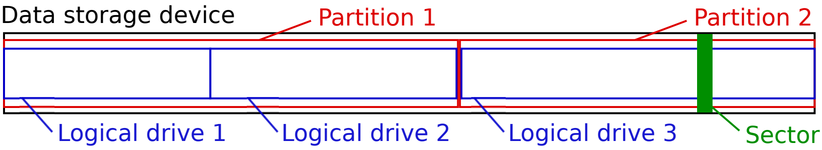
From the application level

- 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:

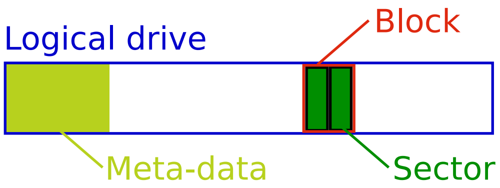
- A set of Bytes
- Organized as a 1D sequence
- Grouped by sectors (512 B, 1, 2, 4 KB)
- The sequence is cut into partitions
- Partitions can be cut into logical drives



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.

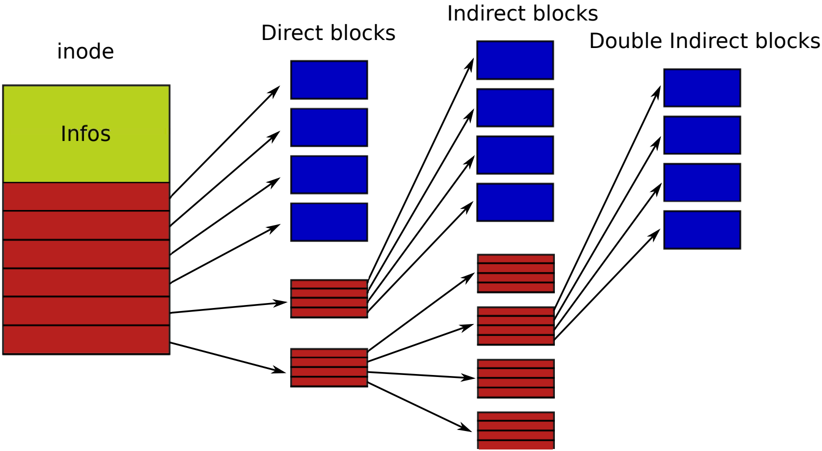


File (ext3)

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

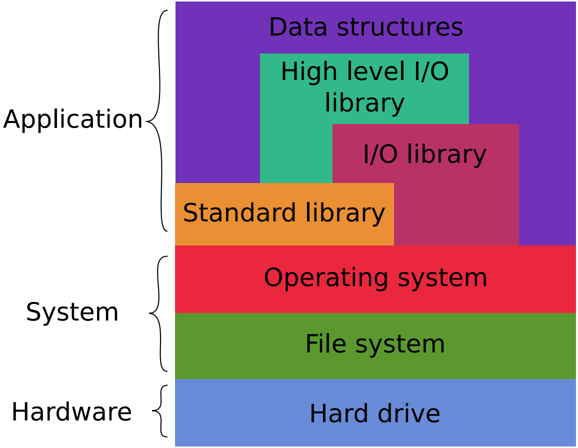
Binary I/O

```
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

Hardware/Software stack



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
! 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
```

0 1 2 3 1 2 3 4 2 3 4 5

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

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

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)

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 **Dimension Scale** API (H5DS): Enables to attach dataset dimension to scales
- 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 **Packet Table** API (H5PT): Almost H5TB but without record insertion/deletion but supports variable length records
- ...

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

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;
}
```

Variable C type

```
hid_t    file, dataset, dataspace;
hsize_t dimsf[2];
herr_t   status;
```

- hid_t: handler for any HDF5 objects (file, groups, dataset, dataspace, datatypes. . .)
- hsize_t: C type used for number of elements of a dataset (in each dimension)
- herr_t: C type used for getting error status of HDF5 functions

File creation

```
file = H5Fcreate("example.h5", H5F_ACC_TRUNC, H5P_DEFAULT,\
                 H5P_DEFAULT);
```

- "example.h5": file name
- H5F_ACC_TRUNC: File creation and suppress it if it exists already
- H5P_DEFAULT: file creation property list
- H5P_DEFAULT: file access property list (needed for MPI-IO)

Dataspace creation

```
dimsf[0] = NX;
dimsf[1] = NY;
dataspace = H5Screate_simple(RANK, dimsf, NULL);
```

- RANK: dataset dimensionality
- dimsf: size of the dataspace in each dimension
- NULL: specify max size of the dataset being fixed to the size

Dataset creation

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

- file: HDF5 objects where to create the dataset. Should be a file or a group.
- ”IntArray”: dataset name
- H5T_NATIVE_INT: type of the data the dataset will contain
- dataspace: size of the dataset
- H5P_DEFAULT: default option for property list.

Datatype

- Pre-Defined Datatypes: created by HDF5.
- Derived Datatypes: created or derived from the pre-defined datatypes.

There are two types of pre-defined datatypes:

- **STANDARD**: They defined standard ways of representing data. Ex: H5T_IEEE_F32BE means IEEE representation of 32 bit floating point number in big endian.
- **NATIVE**: Alias to standard datatypes according to the platform where the program is compiled. Ex: on an Intel based PC, H5T_NATIVE_INT is aliased to the standard pre-defined type, H5T_STD_32LE.

Datatype cont.

A datatype can be:

- ATOMIC: cannot be decomposed into smaller datatype units at the API level. Ex: integer
- COMPOSITE: An aggregation of one or more datatypes. Ex: compound datatype, array, enumeration

Dataset writing

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

- dataset: HDF5 objects representing the dataset to write
- H5T_NATIVE_INT: Type of the data in memory
- H5S_ALL: dataspace specifying the portion of memory that needs be read (in order to be written)
- H5S_ALL: dataspace specifying the portion of the file dataset that needs to be written
- H5P_DEFAULT: default option for property list (needed for MPI-IO).
- data: buffer containing the data to write

Closing HDF5 objects

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

Opened/created HDF5 objects are closed.

HDF5 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 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;
}
```

Some comments

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

  H5Fclose(file);

  return 0;
}
```

This example is almost a **fwrite**, but:

- The generated file is portable
- The generated file can be accessed with HDF5 tools
- Attributes can be added on datasets or groups
- The type of the data can be fixed
- The storage layout can be modified
- Portion of the dataset can be written
- ...

Hands on

1. Correct the program
2. Correct the Makefile to compile the program
3. Execute the program and examine the result with HDF5 tools
4. Modify the program to add an attribute to the main dataset (use the high level Lite library)
5. Modify the program to create the dataset within a group instead at the root
6. Modify the program to write the data in chunks
7. Modify the program to compress the dataset