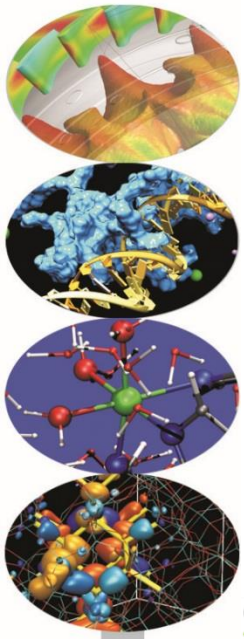
CINECA SCAI
SuperComputing Applications and Innovation

HDF5: theory & practice/2

Giorgio Amati, Mario Tacconi
SCAI Dept.



Cineca
TRAINING
High Performance
Computing 2016

CINECA

Agenda

- ✓ HDF5: introduction
- ✓ Using the API (serial)
- ✓ Using the API (parallel)
- ✓ Tools
- ✓ Some comments

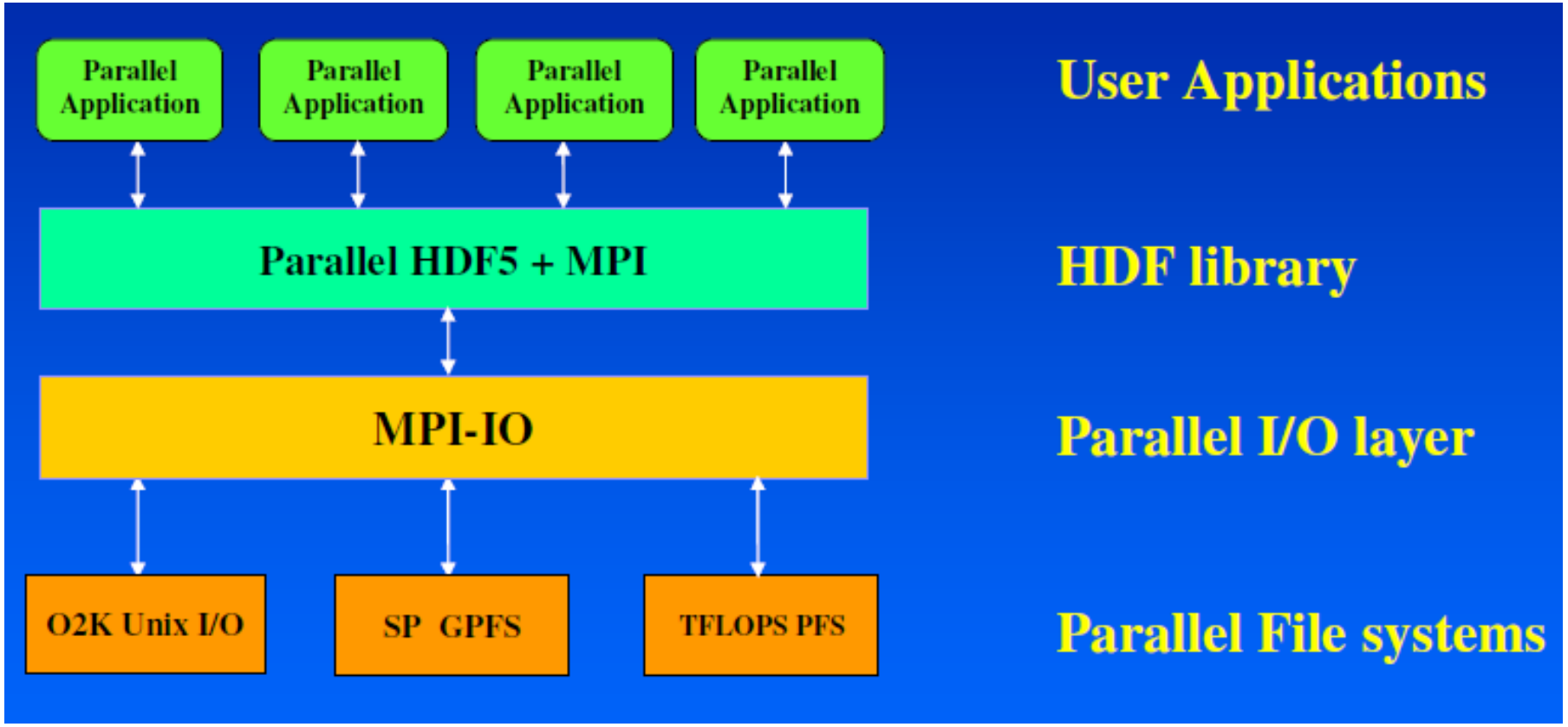
PHDF5 Initial Target

- Support for MPI programming
- Not for shared memory programming
  - Threads
  - OpenMP
- Has some experiments with
  - Thread-safe support for Pthreads;
  - OpenMP if called “correctly”.

PHDF5 Requirements

- PHDF5 files compatible with serial HDF5 files
  - Shareable between different serial or parallel platforms
- Single file image to all processes
  - One file per process design is undesirable
    - ✓ Expensive post processing
    - ✓ Not usable by different number of processes
- Standard parallel I/O interface
  - Must be portable to different platforms

PHDF5 Implementation Layers



PHDF5 open issues

The parallel version of the HDF5 library has some limitations in respect to the serial version:

- C++ bindings are not available;
- The HDF5 data filtering pipeline is not supported: i.e. you cannot use compression while using the parallel version of the HDF5 library.

Collective vs. Independent Calls

- MPI definition of collective call
  - ✓ All processes of the communicator must participate in the right order
- Independent means not collective, i.e. a process may or may not participate to the call
- Collective is not necessarily synchronous

PHDF5 collective calls

- All processes must participate to a collective call
- Examples of PHDF5 collective API:
  - ✓ File operations: **`H5Fcreate`**, **`H5Fopen`**, **`H5Fclose`**
  - ✓ Objects creation: **`H5Dcreate`**
  - ✓ Object access: **`H5Dopen`**, **`H5Dclose`**
  - ✓ Objects structure: **`H5Dextend`** (increase dimension sizes)
  - ✓ Dataset operations: **`H5Dwrite`**, **`H5Dread`**
- Rule of thumb: routines that create or modify HDF5 objects need to be called collectively
- Actual rules: www.hdfgroup.org > HDF5 > Documentation > HDF5 Reference Manual > Collective Calls in Parallel

PHDF5 independent calls

- A process may or may not participate to an independent call
- Examples of PHDF5 independent API:
  - ✓ Dataset operations: **`H5Dwrite`**, **`H5Dread`**
- What you can and cannot do independently:
  - You can have a few processes reading from or writing to a collectively open dataset;
  - You can have a process reading from an independently open dataset;
  - You cannot write onto an independently open dataset: object modifications require a collective call: all processes must be aware of the object modifications.

PHDF5 collective calls: file creation and file opening

- The opening of an existing file or the creation of a new file in parallel is a collective action: all processes must participate;
- PHDF5 opens (creates) a parallel file with a MPI communicator:
  - ✓ Returns a handle to the file object
  - ✓ The parallel application accesses the file via the handle
  - ✓ Different files can be opened by using different communicators

What does PHDF5 file support?

After a file is opened by the processes of a communicator

- ✓ All parts of the file are accessible to all processes
- ✓ All objects in the file are accessible to all processes
- ✓ Multiple processes can write arbitrary* amount of data to the same dataset

* Provided the data to be written don't exceed the dataset dimensions

The transfer and access property lists API

- **`H5Pset_dxpl_mpio`**
- **`herr_t H5Pset_dxpl_mpio( hid_t dxpl_id, H5FD_mpio_xfer_t xfer_mode )`**

✓ Sets data transfer mode.

- **`H5Pset_fapl_mpio`**
- **`herr_t H5Pset_fapl_mpio( hid_t fapl_id, MPI_Comm comm, MPI_Info info )`**

✓ Stores MPI IO communicator information to the file access property list.

Creating and accessing a file in parallel

- HDF5 uses the file access property list object to control the file access mechanism
- General model to access HDF5 file in parallel:
  - ✓ Setup the MPI-IO access template by configuring the file access property list with the MPI communicator of choice;
  - ✓ Open the file by using the file access property list;
  - ✓ Do something with the file…
  - ✓ Close the file.

Configuring the file access property list

Each process of the MPI communicator creates a file access property list and configures it with the MPI communicator:

Using C:

```
MPI_Comm comm = MPI_COMM_WORLD;
MPI_Info info;
hid_t apl_id = H5Pcreate(H5P_FILE_ACCESS);
herr_t hdferr = H5Pset_fapl_mpio(apl_id, comm, info);
```

Using F90:

```
integer(hid_t) :: apl_id
integer :: comm, info, ierr
comm = MPI_COMM_WORLD
call h5pcreate_f(H5P_FILE_ACCESS_F, apl_id, ierr)
call h5pset_fapl_mpio_f(apl_id, comm, info)
```

Example 1

```
/* Initialize MPI */
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);
MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
..
/* Set up file access property list for MPI-IO access */
apl_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(apl_id, MPI_COMM_WORLD, MPI_INFO_NULL);
file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT, apl_id);

/* release property list identifier */
H5Pclose(apl_id);

/* Close the file. */
H5Fclose(file_id);
```

Compiling

- C wrapper: **h5pcc**
- Fortran wrapper: **h5pfc**

```
h5pcc -show
mpicc -D_LARGEFILE64_SOURCE -D_LARGEFILE_SOURCE -
   I/cineca/prod/libraries/zlib/1.2.8/gnu--4.8.3/include -
   I/cineca/prod/libraries/szip/2.1/gnu--4.8.3/include -m64 -fPIC -
   L/cineca/prod/libraries/hdf5/1.8.15_par/intelmpi--5.0.1--binary/lib
   /cineca/prod/libraries/hdf5/1.8.15_par/intelmpi--5.0.1--
   binary/lib/libhdf5_hl.a
   /cineca/prod/libraries/hdf5/1.8.15_par/intelmpi--5.0.1--
   binary/lib/libhdf5.a -L/cineca/prod/libraries/zlib/1.2.8/gnu--
   4.8.3/lib -L/cineca/prod/libraries/szip/2.1/gnu--4.8.3/lib -lsz -lz
   -ldl -lm -Wl,-rpath -
   Wl,/cineca/prod/libraries/hdf5/1.8.15_par/intelmpi--5.0.1--bina
```

H5dump: example 1

```
h5pcc parallel_ex1.c
mpirun -np 4 ./a.out
I'm task 1 of 4
I'm task 2 of 4
I'm task 0 of 4
I'm task 3 of 4
creating h5 file RUN/my_first_parallel_file.h5......all done
….
h5dump -H my_first_parallel_file.h5
HDF5 "my_first_parallel_file.h5" {
GROUP "/" {
}
}
```

Creating, Opening, Extending, a Dataset

- All processes of the MPI communicator used in the file access property list create/open/close a dataset by a collective call
  - ✓ C: **`H5Dcreate`** or **`H5Dopen`**; **`H5Dclose`**
  - ✓ F90: **`h5dcreate_f`** or **`h5dopen_f`**; **`h5dclose_f`**
- A process can independently open a dataset if the dataset will not be modified
- All processes of the MPI communicator extend the dataset dimension before writing to it:
  - ✓ C: **`H5Dextend`**, F90: **`h5dextend_f`**

Dataset programming Model

1. Create or open a HDF5 dataset with a collective call to:
   - ✓ **`H5Dcreate, H5Dopen`**
2. Create a file transfer property list object and configure it to use collective or independent I/O.
   - ✓ Use **`H5Pcreate`** to create the transfer property list object;
   - ✓ use **`H5Pset_dxpl_mpio`** to set the data transfer mode to either use *independent I/O* access or to use *collective I/O.*
3. Access the dataset by using the defined transfer property list.
   - ✓ All processes that have opened a dataset may do collective I/O.
   - ✓ Each process may do an independent and arbitrary number of data I/O access calls, using: **`H5Dwrite`** and/or **`H5Dread`**

Example 2

```
file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT,
                    plist_id);

/* Create the dataspace for the dataset.  */
dimsf[0] = NX;
dimsf[1] = NY;
filespace = H5Screate_simple(RANK, dimsf, NULL);

/* Create the dataset with default properties */
dset_id = H5Dcreate(file_id,"dataset1",H5T_NATIVE_INT,
                    filespace,H5P_DEFAULT);
H5Dclose(dset_id);
H5Sclose(filespace);
H5Fclose(file_id);
```

H5dump output

```
h5dump my_second_parallel_file.h5
HDF5 "my_second_parallel_file.h5" {
GROUP "/" {
   DATASET "dataset1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 10, 8 ) / ( 10, 8 ) }
      DATA {
      (0,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (1,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (2,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (3,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (4,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (5,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (6,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (7,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (8,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (9,0): 0, 0, 0, 0, 0, 0, 0, 0
...
```

Accessing a Dataset

- All processes that have opened the dataset may do collective I/O
- Each process may do independent and arbitrary number of data I/O access calls
  - ✓ C: **`H5Dwrite`** and **`H5Dread`**
  - ✓ F90: **`h5dwrite_f`** and **`h5dread_f`**

Accessing a Dataset

- Create and configure dataset transfer property
  - C: **`H5Pset_dxpl_mpio`**
    - ✓**`H5FD_MPIO_COLLECTIVE`**
    - ✓**`H5FD_MPIO_INDEPENDENT`** (default)
  - F90: **`h5pset_dxpl_mpio_f`**
    - ✓**`H5FD_MPIO_COLLECTIVE_F`**
    - ✓**`H5FD_MPIO_INDEPENDENT_F`** (default)
- write to or read from the dataset with the defined transfer property list

Collective write (C)

```
…
/* Create property list for collective dataset write */
plist_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);

status = H5Dwrite(dset_id, H5T_NATIVE_INT, memspace,
  filespace, plist_id, data);
…
```

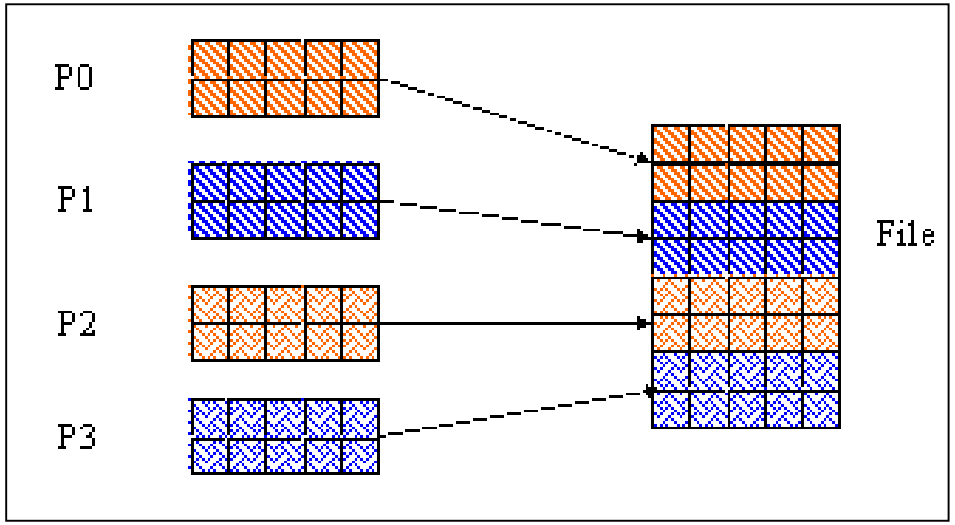
Writing Hyperslabs

- This is a distributed memory model: data is split among processes
- PHDF5 uses hyperslab model
  - Each process defines the appropriate hyperslab selection onto a file and memory dataspace
  - Each process executes partial H5Dwrite/H5Dread call by using the file and memory dataspaces. Depending on the setting of the transfer property list the end result could be:
    - ✓ A collective data transfer
    - ✓ An independent data transfer
- The memory and file hyperslabs selections are defined using the **`H5Sselect_hyperslab`** routine

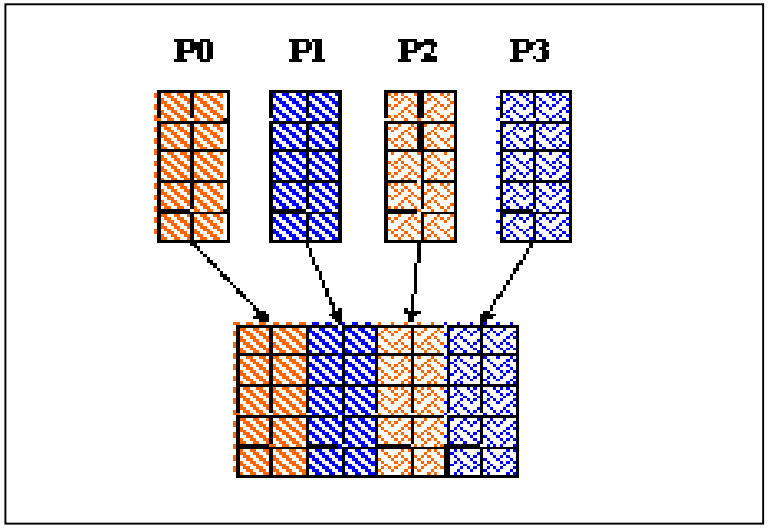
Writing Hyperslabs

- The start (or offset), count, stride, and block parameters define the portion of the dataset to write to.
- Each processes by changing the values of these parameters can write hyperslabs into a specific region of the dataset.
- Different access pattern are possible:
  - ✓ contiguous hyperslab,
  - ✓ regularly spaced data in a column/row,
  - ✓ more complicated patterns by combining multiple hyperslab selections

Contiguous Hyperslab



C example



Fortran 90 example

By rows (C)

```
dimsf[0] = 10; /* set global size */
dimsf[1] = 8;

/* Each task defines dataset in memory and writes it to the
   hyperslab in the file */
count[0] = dimsf[0]/mpi_size;
count[1] = dimsf[1];
memspace = H5Screate_simple(rank, count, NULL);

/* set offset for each task */
offset[0] = mpi_rank * count[0];
offset[1] = 0;

/* Initialize data buffer  */
data = (int *) malloc(sizeof(int)*count[0]*count[1]);
for (i=0; i < count[0]*count[1]; i++) {
     data[i] = mpi_rank*1000 + i;
}
```

Example 3 (by rows)

```
[gamati01@node342 PARALLEL]$ mpirun -np 5 ./a.out
[gamati01@node342 PARALLEL]$ h5dump my_third_parallel_file.h5
HDF5 "my_third_parallel_file.h5" {
GROUP "/" {
   DATASET "dataset1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 10, 8 ) / ( 10, 8 ) }
      DATA {
      (0,0): 0, 1, 2, 3, 4, 5, 6, 7,
      (1,0): 8, 9, 10, 11, 12, 13, 14, 15,
      (2,0): 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007,
      (3,0): 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015,
      (4,0): 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007,
      (5,0): 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015,
      (6,0): 3000, 3001, 3002, 3003, 3004, 3005, 3006, 3007,
      (7,0): 3008, 3009, 3010, 3011, 3012, 3013, 3014, 3015,
      (8,0): 4000, 4001, 4002, 4003, 4004, 4005, 4006, 4007,
      (9,0): 4008, 4009, 4010, 4011, 4012, 4013, 4014, 4015
      }
```

Example 3 (by row)

```
[gamati01@node342 PARALLEL]$ mpirun -np 10 ./a.out
[gamati01@node342 PARALLEL]$ h5dump my_third_parallel_file.h5
HDF5 "my_third_parallel_file.h5" {
GROUP "/" {
   DATASET "dataset1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 10, 8 ) / ( 10, 8 ) }
      DATA {
      (0,0): 0, 1, 2, 3, 4, 5, 6, 7,
      (1,0): 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007,
      (2,0): 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007,
      (3,0): 3000, 3001, 3002, 3003, 3004, 3005, 3006, 3007,
      (4,0): 4000, 4001, 4002, 4003, 4004, 4005, 4006, 4007,
      (5,0): 5000, 5001, 5002, 5003, 5004, 5005, 5006, 5007,
      (6,0): 6000, 6001, 6002, 6003, 6004, 6005, 6006, 6007,
      (7,0): 7000, 7001, 7002, 7003, 7004, 7005, 7006, 7007,
      (8,0): 8000, 8001, 8002, 8003, 8004, 8005, 8006, 8007,
      (9,0): 9000, 9001, 9002, 9003, 9004, 9005, 9006, 9007
      }
```

Example 3 (by row)

```
[gamati01@node342 PARALLEL]$ mpirun -np 3 ./a.out
[gamati01@node342 PARALLEL]$ h5dump my_third_parallel_file.h5
HDF5 "my_third_parallel_file.h5" {
GROUP "/" {
   DATASET "dataset1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 10, 8 ) / ( 10, 8 ) }
      DATA {
      (0,0): 0, 1, 2, 3, 4, 5, 6, 7,
      (1,0): 8, 9, 10, 11, 12, 13, 14, 15,
      (2,0): 16, 17, 18, 19, 20, 21, 22, 23,
      (3,0): 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007,
      (4,0): 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015,
      (5,0): 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023,
      (6,0): 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007,
      (7,0): 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015,
      (8,0): 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023,
      (9,0): 0, 0, 0, 0, 0, 0, 0, 0
      }
   }
}
```

Example 3 (by row)

```
[gamati01@node342 PARALLEL]$ mpirun -np 11 ./a.out
[gamati01@node342 PARALLEL]$ h5dump my_third_parallel_file.h5
HDF5 "my_third_parallel_file.h5" {
GROUP "/" {
   DATASET "dataset1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 10, 8 ) / ( 10, 8 ) }
      DATA {
      (0,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (1,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (2,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (3,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (4,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (5,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (6,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (7,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (8,0): 0, 0, 0, 0, 0, 0, 0, 0,
      (9,0): 0, 0, 0, 0, 0, 0, 0, 0
      }
   }
}
```

By columns (C)

```
dimsf[0] = 10; /* set global size */
dimsf[1] = 8;

/* Each task defines dataset in memory and writes it to the hyperslab
   in the file */
count[0] = dimsf[0];
count[1] = dimsf[1]/mpi_size;
memspace = H5Screate_simple(rank, count, NULL);

/* set offset for each task */
offset[1] = 0;
offset[1] = mpi_rank * count[1];

/* Initialize data buffer  */
data = (int *) malloc(sizeof(int)*count[0]*count[1]);
for (i=0; i < count[0]*count[1]; i++) {
     data[i] = mpi_rank*1000 + i;
}
```

Example 3 (by Column)

```
[gamati01@node342 PARALLEL]$ mpirun -np 2 ./a.out
[gamati01@node342 PARALLEL]$ h5dump my_forth_parallel_file.h5
HDF5 "my_forth_parallel_file.h5" {
GROUP "/" {
   DATASET "dataset1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 10, 8 ) / ( 10, 8 ) }
      DATA {
      (0,0): 0, 1, 2, 3, 1000, 1001, 1002, 1003,
      (1,0): 4, 5, 6, 7, 1004, 1005, 1006, 1007,
      (2,0): 8, 9, 10, 11, 1008, 1009, 1010, 1011,
      (3,0): 12, 13, 14, 15, 1012, 1013, 1014, 1015,
      (4,0): 16, 17, 18, 19, 1016, 1017, 1018, 1019,
      (5,0): 20, 21, 22, 23, 1020, 1021, 1022, 1023,
      (6,0): 24, 25, 26, 27, 1024, 1025, 1026, 1027,
      (7,0): 28, 29, 30, 31, 1028, 1029, 1030, 1031,
      (8,0): 32, 33, 34, 35, 1032, 1033, 1034, 1035,
      (9,0): 36, 37, 38, 39, 1036, 1037, 1038, 1039
      }
```

Example 3 (by Column)

```
[gamati01@node342 PARALLEL]$ mpirun -np 8 ./a.out
[gamati01@node342 PARALLEL]$ h5dump my_forth_parallel_file.h5
HDF5 "my_forth_parallel_file.h5" {
GROUP "/" {
   DATASET "dataset1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 10, 8 ) / ( 10, 8 ) }
      DATA {
      (0,0): 0, 1000, 2000, 3000, 4000, 5000, 6000, 7000,
      (1,0): 1, 1001, 2001, 3001, 4001, 5001, 6001, 7001,
      (2,0): 2, 1002, 2002, 3002, 4002, 5002, 6002, 7002,
      (3,0): 3, 1003, 2003, 3003, 4003, 5003, 6003, 7003,
      (4,0): 4, 1004, 2004, 3004, 4004, 5004, 6004, 7004,
      (5,0): 5, 1005, 2005, 3005, 4005, 5005, 6005, 7005,
      (6,0): 6, 1006, 2006, 3006, 4006, 5006, 6006, 7006,
      (7,0): 7, 1007, 2007, 3007, 4007, 5007, 6007, 7007,
      (8,0): 8, 1008, 2008, 3008, 4008, 5008, 6008, 7008,
      (9,0): 9, 1009, 2009, 3009, 4009, 5009, 6009, 7009
      }
```

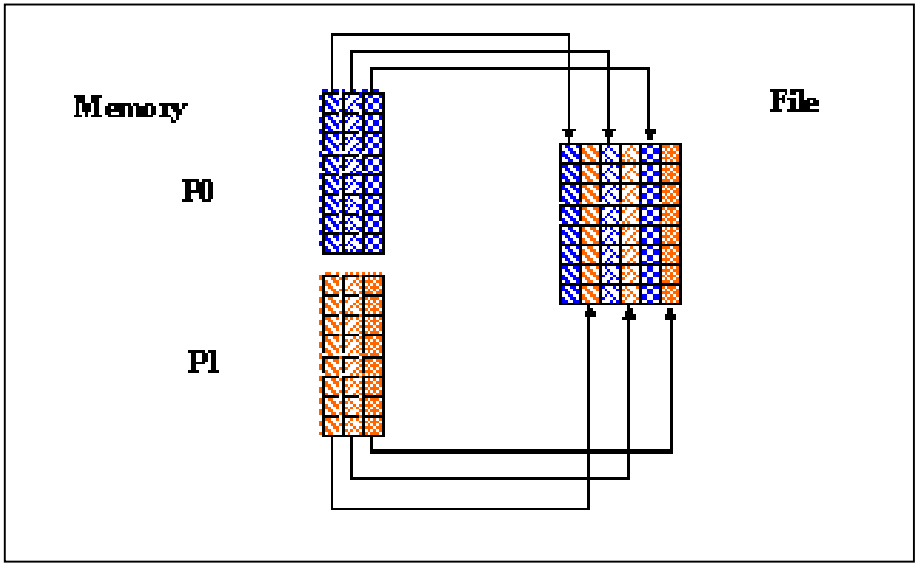
Example 4 (2D decomp)

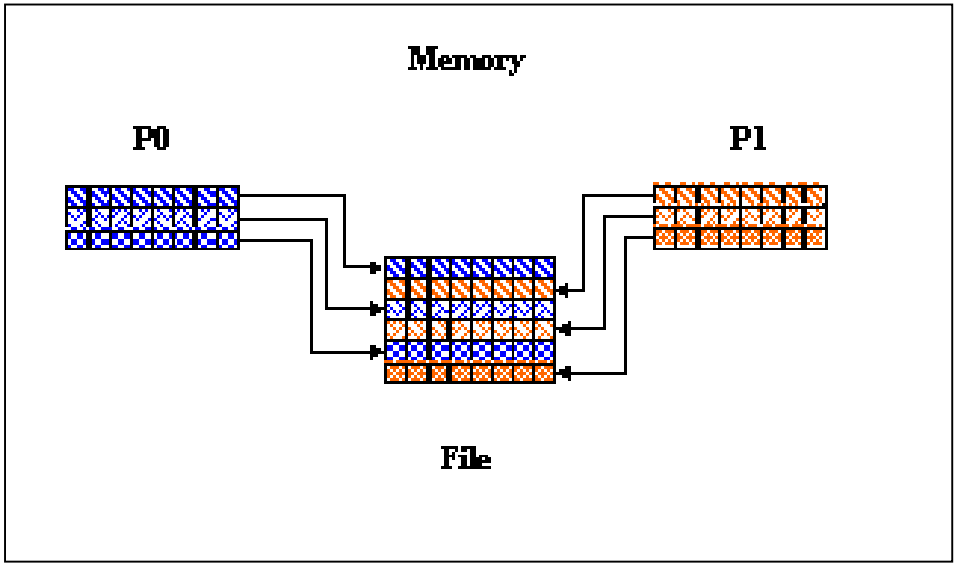
```
count[0] = dimsf[0]/proc_x;
count[1] = dimsf[1]/proc_y;

/* set coordinates... */
if(mpi_rank == 0) { x = 0; y = 0;}
if(mpi_rank == 1) { x = 1; y = 0;}
if(mpi_rank == 2) { x = 0; y = 1;}
if(mpi_rank == 3) { x = 1; y = 1;}

memspace = H5Screate_simple(rank, count, NULL);

/* set offset for each task */
offset[0] = x*count[0];
offset[1] = y*count[1];
```

2D Decomposition

```
[gamati01@node342 PARALLEL]$ mpirun -np 4 ./a.out
[gamati01@node342 PARALLEL]$ h5dump my_fifth_parallel_file.h5
HDF5 "my_third_parallel_file.h5" {
GROUP "/" {
   DATASET "dataset1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 10, 8 ) / ( 10, 8 ) }
      DATA {
      (0,0): 0, 1, 2, 3, 2000, 2001, 2002, 2003,
      (1,0): 4, 5, 6, 7, 2004, 2005, 2006, 2007,
      (2,0): 8, 9, 10, 11, 2008, 2009, 2010, 2011,
      (3,0): 12, 13, 14, 15, 2012, 2013, 2014, 2015,
      (4,0): 16, 17, 18, 19, 2016, 2017, 2018, 2019,
      (5,0): 1000, 1001, 1002, 1003, 3000, 3001, 3002, 3003,
      (6,0): 1004, 1005, 1006, 1007, 3004, 3005, 3006, 3007,
      (7,0): 1008, 1009, 1010, 1011, 3008, 3009, 3010, 3011,
      (8,0): 1012, 1013, 1014, 1015, 3012, 3013, 3014, 3015,
      (9,0): 1016, 1017, 1018, 1019, 3016, 3017, 3018, 3019
      }
```

Regularly spaced data

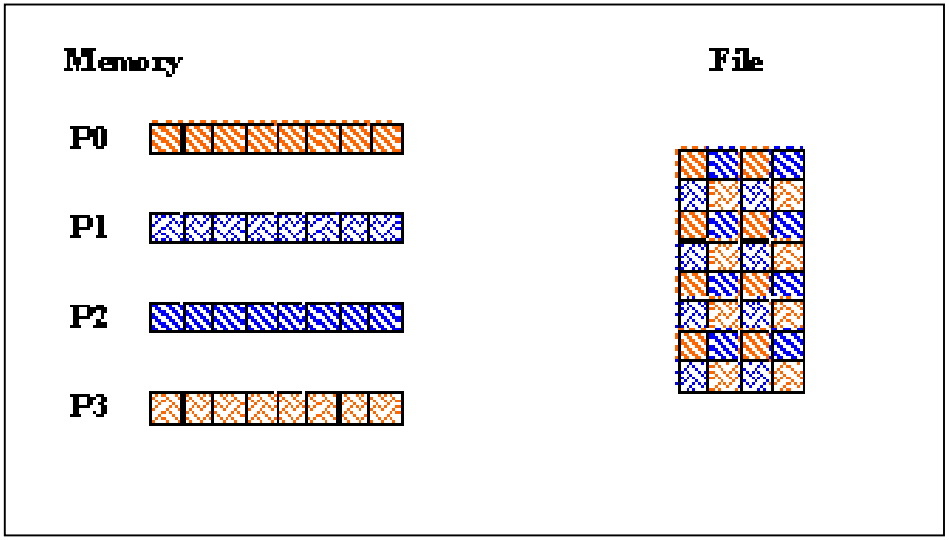




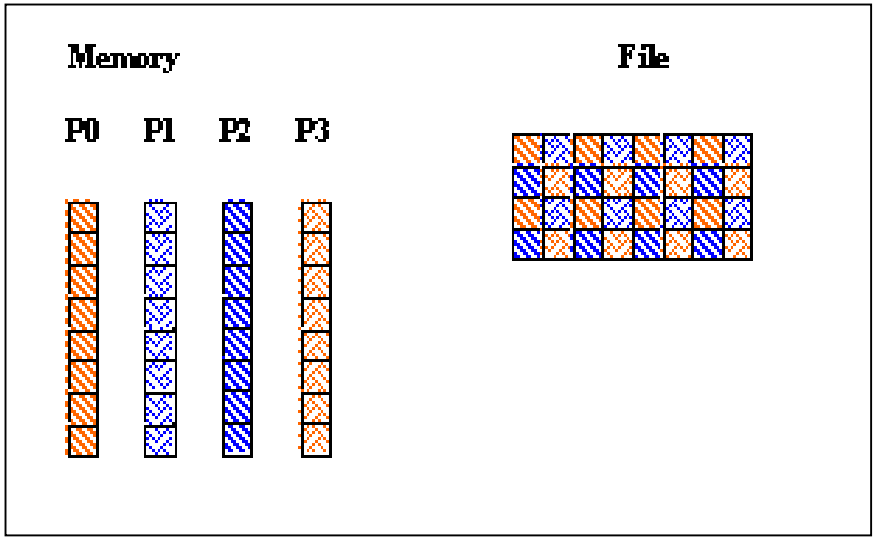
C example

Fortran 90 example

By patterns



C example



Fortran 90 example

Reading Hyperslabs

- The start (or offset), count, stride, and block parameters define the portion of the dataset to read from

Example 5 (2D decomp.)

```
proc_x = mpi_size; proc_y = 1;
dimsf[0] = 10; dimsf[1] = 8;
count[0] = dimsf[0]/proc_x; count[1] = dimsf[1];
offset[0] = x*count[0]; offset[1] = y*count[1];

/* Set up file access property list for MPI-IO access */
plist_id = H5Pcreate(H5P_FILE_ACCESS);
status = H5Pset_fapl_mpio(plist_id, MPI_COMM_WORLD, MPI_INFO_NULL);
file_id = H5Fopen(H5FILE_NAME, H5F_ACC_RDWR, plist_id);

/* Create property list for collective dataset read */
plist2_id = H5Pcreate(H5P_DATASET_XFER);
status = H5Pset_dxpl_mpio(plist2_id, H5FD_MPIO_COLLECTIVE);

status = H5Dread(dset_id, H5T_NATIVE_INT, memspace, filespace1,
   plist2_id, data);
```

The file to read…

```
[gamati01@node342 PARALLEL]$ h5dump my_fifth_parallel_file.h5
HDF5 "my_third_parallel_file.h5" {
GROUP "/" {
   DATASET "dataset1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 10, 8 ) / ( 10, 8 ) }
      DATA {
      (0,0): 0, 1, 2, 3, 2000, 2001, 2002, 2003,
      (1,0): 4, 5, 6, 7, 2004, 2005, 2006, 2007,
      (2,0): 8, 9, 10, 11, 2008, 2009, 2010, 2011,
      (3,0): 12, 13, 14, 15, 2012, 2013, 2014, 2015,
      (4,0): 16, 17, 18, 19, 2016, 2017, 2018, 2019,
      (5,0): 1000, 1001, 1002, 1003, 3000, 3001, 3002, 3003,
      (6,0): 1004, 1005, 1006, 1007, 3004, 3005, 3006, 3007,
      (7,0): 1008, 1009, 1010, 1011, 3008, 3009, 3010, 3011,
      (8,0): 1012, 1013, 1014, 1015, 3012, 3013, 3014, 3015,
      (9,0): 1016, 1017, 1018, 1019, 3016, 3017, 3018, 3019
      }
   }
```

Output

```
[gamati01@node342 PARALLEL]$ cat RUN/task.0.dat
i =  0, value --> 0
i =  1, value --> 1
i =  2, value --> 2
i =  3, value --> 3
i =  4, value --> 2000
i =  5, value --> 2001
i =  6, value --> 2002
i =  7, value --> 2003
i =  8, value --> 4
i =  9, value --> 5
i = 10, value --> 6
i = 11, value --> 7
i = 12, value --> 2004
i = 13, value --> 2005
i = 14, value --> 2006
i = 15, value --> 2007
```

Agenda

- ✓ HDF5: introduction
- ✓ Using the API (serial)
- ✓ Using the API (parallel)
- ✓ Tools
- ✓ Some comments

H5diff/1

- It make diff between two h5 files
- Can compare single directory…
- …but can be misleading
- Example:

```
h5diff my_third_file.h5 my_fourth_file.h5
```

- Are this files the same?

- … Obviously not!!!

H5diff/2

```
h5diff -v my_third_file.h5 my_fourth_file.h5
file1     file2
------------------------------------
    x       x     /
    x       x     /MyGroup
    x       x     /MyGroup/Group_A
    x       x     /MyGroup/Group_B
            x     /dset
group   : </> and </>
0 differences found
group   : </MyGroup> and </MyGroup>
0 differences found
group   : </MyGroup/Group_A> and </MyGroup/Group_A>
0 differences found
group   : </MyGroup/Group_B> and </MyGroup/Group_B>
0 differences found
```

H5diff/3

```
h5diff my_fourth_file.after.h5 my_fourth_file.before.h5
dataset: </dset> and </dset>
8 differences found

h5diff -v my_fourth_file.after.h5 my_fourth_file.before.h5
…
dataset: </dset> and </dset>
size:              [4x6]            [4x6]
position          dset              dset              difference
----------------------------------------------------------------
[ 1 1 ]            1000              8                  992
[ 1 2 ]            1001              9                  992
[ 1 3 ]            1002              10                 992
[ 1 4 ]            1003              11                 992
[ 2 1 ]            1004              14                 990
[ 2 2 ]            1005              15                 990
[ 2 3 ]            1006              16                 990
[ 2 4 ]            1007              17                 990
8 differences found
```

H5diff/4

Very useful feature

- **Strict equality**: The default comparison mode is to check for strict equality of data values.
- **Fixed difference:** With the '-d delta' or '--delta=delta' option, h5diff considers two data values to be equal if the absolute value of the difference is less than the specified delta.
- **Relative difference:** With the '-p relative' or '--relative=relative' option, h5diff considers two data values to be equal if the absolute value of the relative difference is less than the value specified in relative.
- **System epsilon:** With the '--use-system-epsilon' option, h5diff considers two data values to be equal if the absolute value of the difference is less than the computing platform's system epsilon (or a pre-determined value if no system epsilon is defined).

H5dump -p

```
gamati01@node013.pico:[SERIAL]$ h5dump -p -H RUN/my_fourth_file.h5
HDF5 "RUN/my_fourth_file.h5" {
…
   DATASET "dset" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 1024, 1024 ) / ( 1024, 1024 ) }
      STORAGE_LAYOUT {
         CHUNKED ( 256, 256 )
         SIZE 86610 (48.427:1 COMPRESSION)
      }
      FILTERS {
         COMPRESSION DEFLATE { LEVEL 9 }
      }
      FILLVALUE {
         FILL_TIME H5D_FILL_TIME_IFSET
         VALUE  0
      }
      ALLOCATION_TIME {
         H5D_ALLOC_TIME_INCR
```

HFD5: Tools/1

- The are many usefull tools (read manual please)
  - ✓ **h5ls**
  - ✓ **h5dump**
  - ✓ **h5debug**
  - ✓ **h5repart**
  - ✓ **h5mkgrp**
  - ✓ **h5redeploy**
  - ✓ **h5import**
  - ✓ **h5repack**
  - ✓ …

HFD5: Tools/2

- The are many tools usefull (read manual please)
  - ✓ **h5jam**
  - ✓ **h5unjam**
  - ✓ **h5copy**
  - ✓ **h5stat**
  - ✓ **gif2h5**
  - ✓ **h52gif**
  - ✓ **h5perf_serial**
  - ✓ **h5perf**
  - ✓ **hdfview**

Agenda

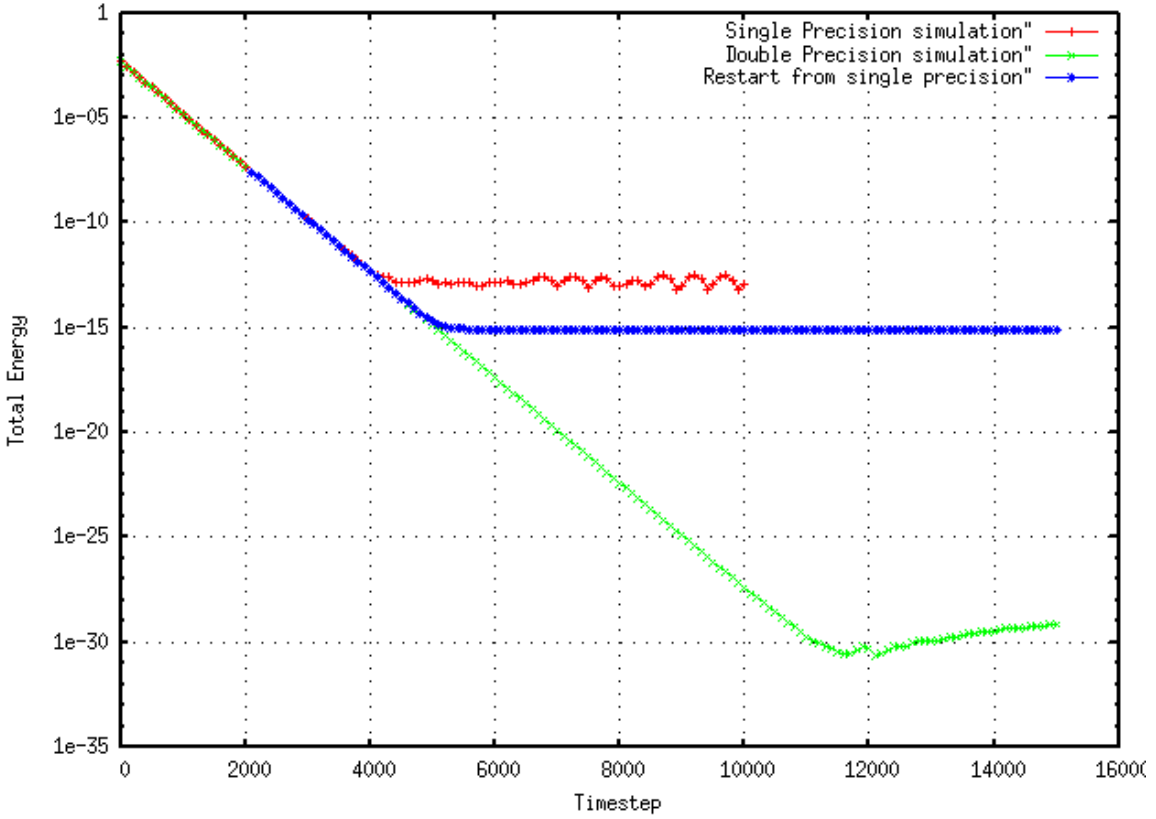
- ✓ HDF5: introduction
- ✓ Using the API (serial)
- ✓ Using the API (parallel)
- ✓ Tools
- ✓ Some (final) comments

Bug or a Feature?

- Try to read a single precision restart from a double precision simulation.
  - ✓ It gives no error
  - ✓ The simulation runs "correctly"
  - ✓ But now it is a single precision simulation (form a numerical point of view)!!!
  - ✓ But as memory occupation (in RAM) is a double precision simulation!!!

Bug or a Feature?

- Taylor Green Vortex, decaying flow
- Analytical solution



4349.87, 0.629130

HFD5: (final) comments

- HDF5 as "robust" API.
  - ✓ It works but take care of warning
  - ✓ Check with h5dump the results
  - ✓ Use the error handle (**`herr_t`**) to verify everthing is ok
  - ✓ Take a look to HFD5 warnings
  - ✓ Can produce an HDF5 file even with lots of warnigs, but it is still a valid file?
  - ✓ Can fail silently….

Usefull links

The HDF Group Page: http://hdfgroup.org/

HDF5 Home Page: http://hdfgroup.org/HDF5/

HDF Helpdesk: help@hdfgroup.org

HDF Mailing Lists: http://hdfgroup.org/services/support.html

Parallel tutorial: http://hdf.ncsa.uiuc.edu/HDF5/doc/Tutor

1.8 vs 1.6:
http://www.hdfgroup.org/HDF5/doc/ADGuide/WhatsNew180.html
http://www.hdfgroup.org/HDF5/doc/ADGuide/Changes.html

That's all folks!!!!