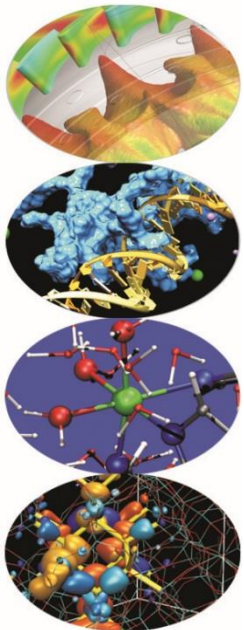


# HDF5: theory & practice/2

Giorgio Amati  
SCAI Dept.

14 May 2015



# Agenda

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

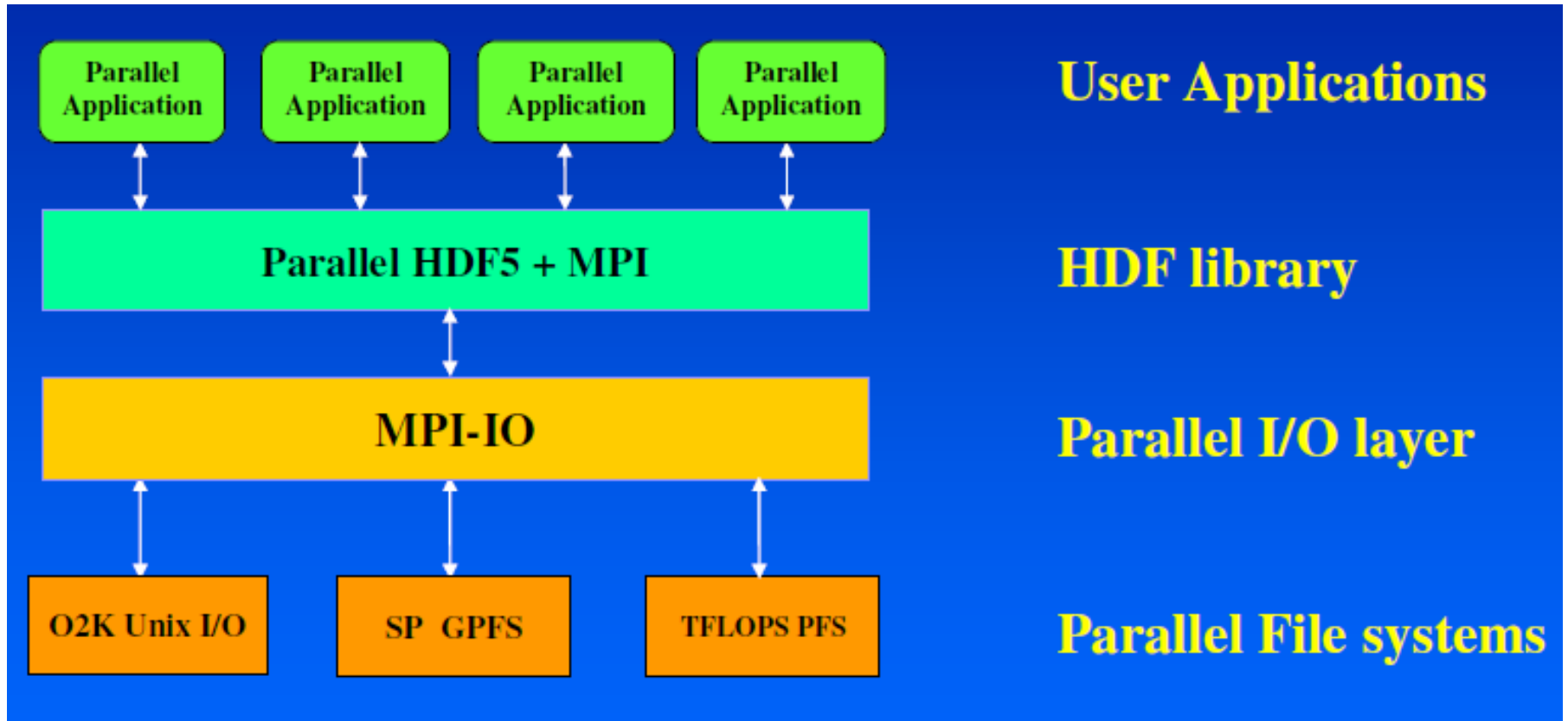
# PHDF5 Initial Target

- 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 useable by different number of processes
- Standard parallel I/O interface
  - Must be portable to different platforms

# PHDF5 Implementation Layers



# Collective vs. Independent Calls

- MPI definition of collective call
  - All processes of the communicator must participate in the right order
- Independent means not collective
- Collective is not necessarily synchronous

# Programming Restrictions

- Most PHDF5 APIs are collective
- PHDF5 opens a parallel file with a communicator
  - ✓ Returns a file-handle
  - ✓ Future access to the file via the file-handle
  - ✓ All processes must participate in collective PHDF5 APIs
  - ✓ Different files can be opened via different communicators

# Examples of PHDF5 API

- Examples of PHDF5 collective API
  - ✓ File operations: **H5Fcreate**, **H5Fopen**, **H5Fclose**
  - ✓ Objects creation: **H5Dcreate**, **H5Dopen**, **H5Dclose**
  - ✓ Objects structure: **H5Dextend** (increase dimension sizes)
- Array data transfer
  - ✓ Dataset operations: **H5Dwrite**, **H5Dread**



# What Does PHDF5 Support?

- After a file is opened by the processes of a communicator
  - ✓ All parts of file are accessible by all processes
  - ✓ All objects in the file are accessible by all processes
  - ✓ Multiple processes write to the same data array
  - ✓ Each process writes to individual data array

# Programming Model

1. Create or open a Parallel HDF5 file with a collective call to:
  - ✓ **H5Dcreate**, **H5Dopen**
2. Obtain a copy of the file transfer property list and set it to use collective or independent I/O.
  - ✓ First passing a data transfer property list class type to: **H5Pcreate**
  - ✓ Set the data transfer mode to either use *independent I/O* access or to use *collective I/O*, with a call to: **H5Pset\_dxpl\_mpio**
3. Access the dataset with 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**

# More info

- `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.

# Setup access template

Each process of the MPI communicator creates an access template and sets it up with MPI parallel access information

Using C:

```
herr_t H5Pset_fapl_mpio(hid_t plist_id, MPI_Comm comm,  
MPI_Info info);
```

Using F90:

```
integer(hid_t) :: plist_id  
integer :: comm, info  
call h5pset_fapl_mpio_f(plist_id, comm, info)
```

**plist\_id** is a file access property list identifier

# 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 */
plist_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(plist_id, MPI_COMM_WORLD, MPI_INFO_NULL);
file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);

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

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

# 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 and Opening Dataset

- All processes of the MPI communicator open/close a dataset by a collective call
  - ✓ C: **H5Dcreate** or **H5Dopen**; **H5Dclose**
  - ✓ F90: **h5dcreate\_f** or **h5dopen\_f**; **h5dclose\_f**
  
- All processes of the MPI communicator extend dataset with unlimited dimensions before writing to it
  - ✓ C: **H5Dextend**
  - ✓ F90: **h5dextend\_f**

# 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 collective */
```

```
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 set 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)
- Access dataset with the defined transfer property

# 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);
```

...



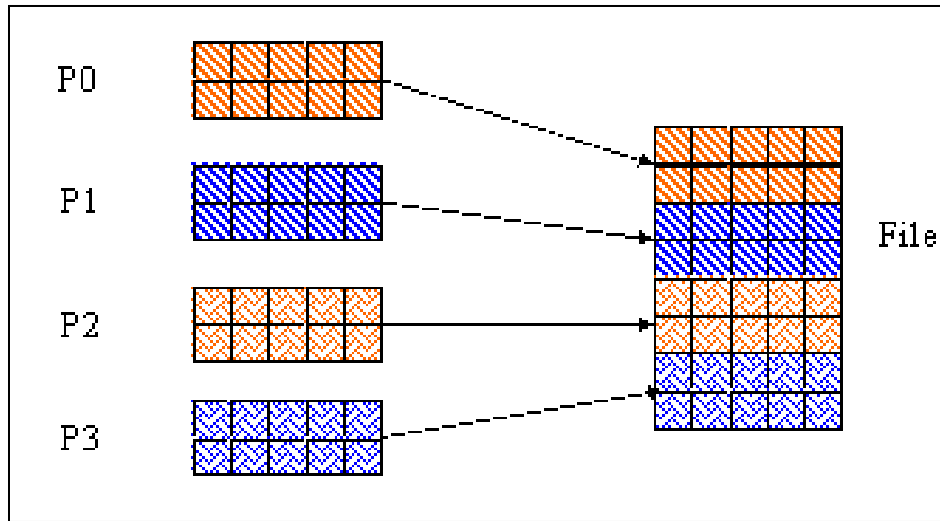
# Writing Hyperslabs

- This is a distributed memory model: data is split among processes
- PHDF5 uses hyperslab model
  - Each process defines memory and file hyperslabs
  - Each process executes partial write/read call
    - ✓ Collective calls
    - ✓ Independent calls
- The memory and file hyperslabs in the first step are defined with the **H5Sselect\_hyperslab**

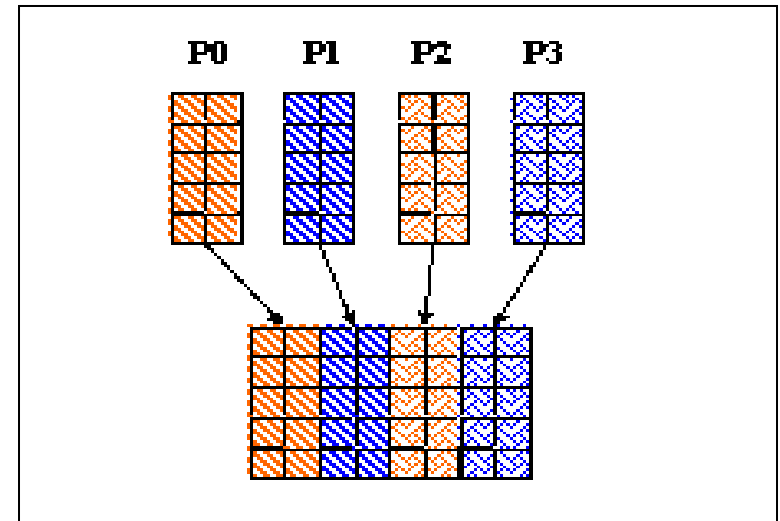
# Writing Hyperslabs

- The start (or offset), count, stride, and block parameters define the portion of the dataset to write to.
- By changing the values of these parameters you can write hyperslabs with Parallel HDF5 by
  - ✓ contiguous hyperslab,
  - ✓ regularly spaced data in a column/row,
  - ✓ by patterns,
  - ✓ by chunks.

# 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)

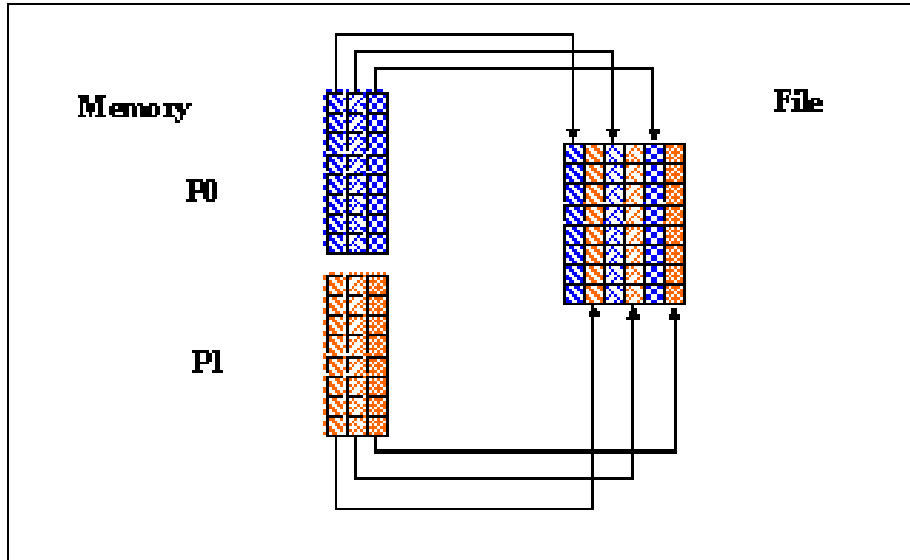
```
count[0] = dimsf[0]/proc_x;  
count[1] = dimsf[1]/proc_y;  
  
/* set coordinates... */  
if(mpi_rank == 0) { x = 0; y = 0;}  
if(mpi_rank == 1) { x = 1; y = 0;}  
if(mpi_rank == 2) { x = 0; y = 1;}  
if(mpi_rank == 3) { x = 1; y = 1;}  
  
memspace = H5Screate_simple(rank, count, NULL);  
  
/* set offset for each task */  
offset[0] = x*count[0];  
offset[1] = y*count[1];
```



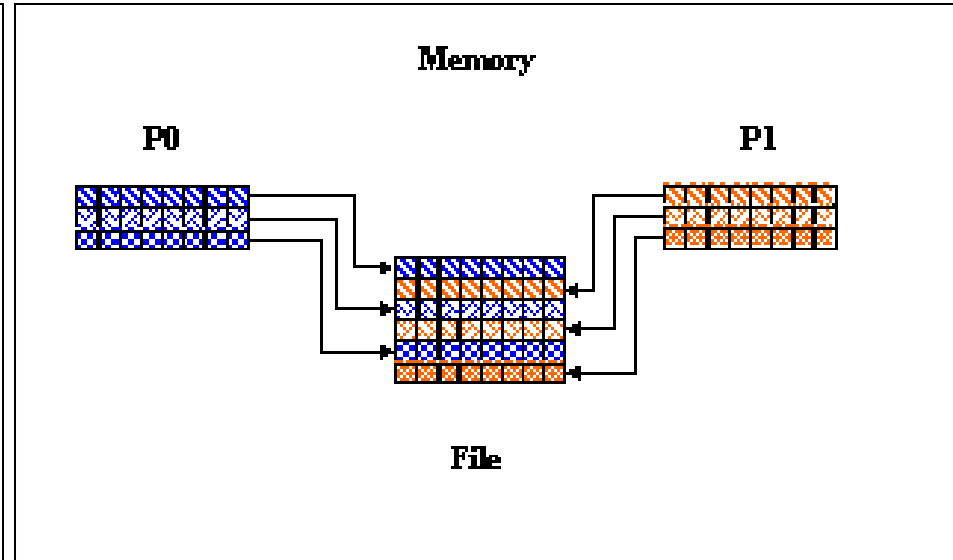
# 2D Decomposition

```
[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

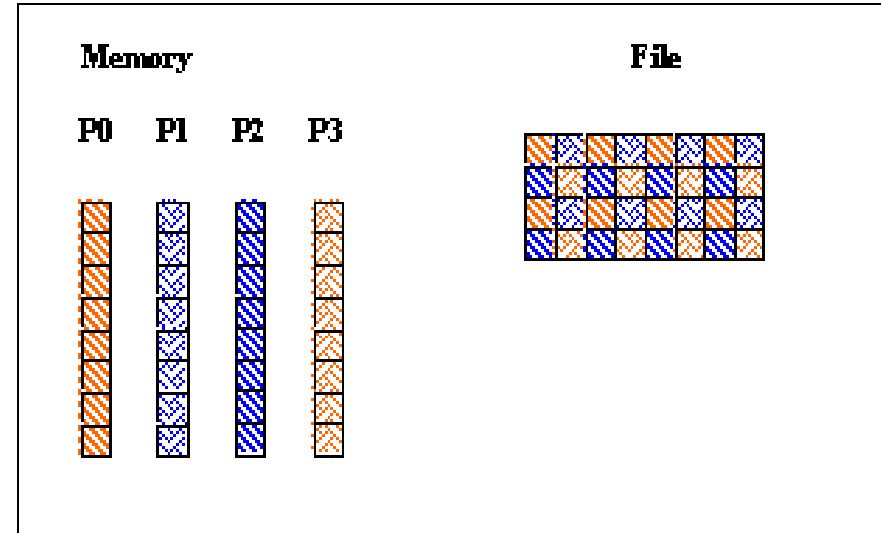
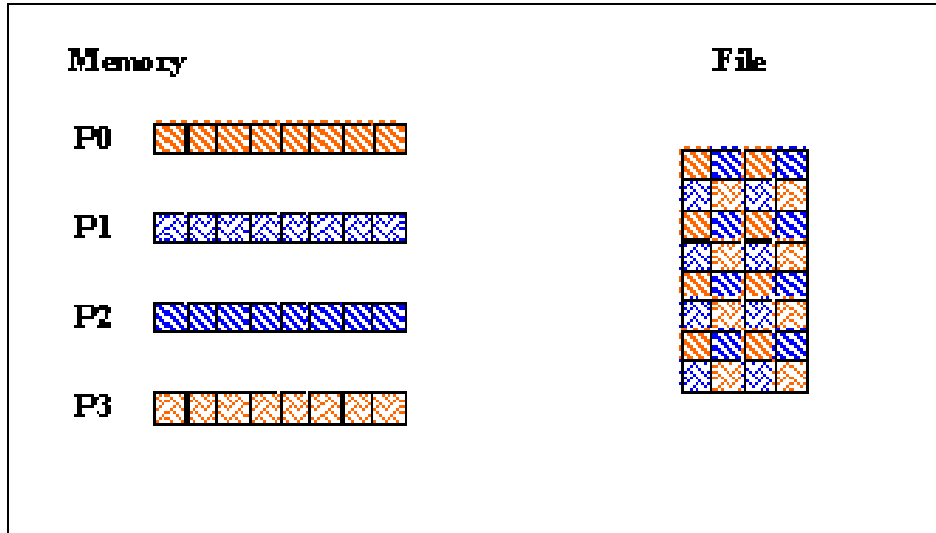


C example



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: main issues
- ✓ 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
```

# HFD5: Tools/1

- There are many useful tools (read manual please)
  - ✓ h5ls
  - ✓ h5dump
  - ✓ h5debug
  - ✓ h5repart
  - ✓ h5mkgrp
  - ✓ h5redeploy
  - ✓ h5import
  - ✓ h5repack
  - ✓ ...

# HFD5: Tools/2

- There are many tools useful (read manual please)
  - ✓ `h5jam`
  - ✓ `h5unjam`
  - ✓ `h5copy`
  - ✓ `h5stat`
  - ✓ `gif2h5`
  - ✓ `h52gif`
  - ✓ `h5perf_serial`
  - ✓ `h5perf`
  - ✓ `hdfview`

# Agenda

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

# API 1.6 vs API 1.8

- Moving from release 1.6.x to 1.8.0 some interfaces were modified
  - ✓ If your program uses 1.6/1.8 API and HDF5 library was compiled using 1.6/1.8 API → no problem
  - ✓ If your program uses 1.8 API and HDF5 library was compiled using 1.6 API → use `"-DH5Dcreate_vers=2, ..."`
  - ✓ If your program uses 1.6 API and HDF5 library was compiled using 1.8 API → use `"-DH5_USE_16_API"`

```
[amati@droemu PARALLEL]$ h5pcc -showconfig
```

```
...
```

```
Default API Mapping: v18
```

```
With Deprecated Public Symbols: yes
```

```
...
```

# API 1.6 vs API 1.8

```
[amati@droemu PARALLEL]$ h5pcc parallel_ex4.c
parallel_ex4.c: In function 'main':
parallel_ex4.c:97: error: too few arguments to function 'H5Dcreate2'
[amati@droemu PARALLEL]$ h5pcc parallel_ex5.c
parallel_ex5.c: In function 'main':
parallel_ex5.c:64: error: too few arguments to function 'H5Dopen2'
[amati@droemu PARALLEL]$ h5pcc -DH5_USE_16_API parallel_ex4.c
[amati@droemu PARALLEL]$ h5pcc -DH5_USE_16_API parallel_ex5.c
```

- 1.6 API ( → H5Dopen1)

```
H5Dcreate(file_id,"dset",H5T_NATIVE_INT,filespace,H5P_DEFAULT);
H5Dopen(file_id, "dataset1");
```

- 1.8 API ( → H5Dopen2)

```
H5Dcreate(file_id,"dset",H5T_NATIVE_INT,filespace,H5P_DEFAULT,
          H5P_DEFAULT,H5P_DEFAULT);
H5Dopen(file_id, "dataset1",H5P_DEFAULT);
```



# 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 (from a numerical point of view)!!!
  - ✓ But as memory occupation (in RAM) is a double precision simulation!!!

# 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](mailto: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!!!!

