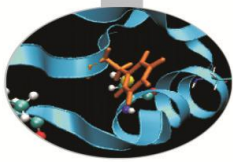
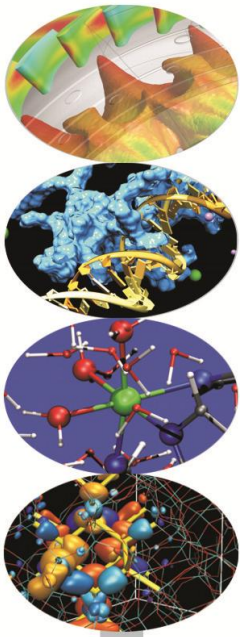
CINECA SCAI
SuperComputing Applications and Innovation



Parallel HDF5 (PHDF5)

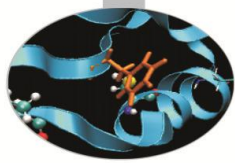


Giusy Muscianisi – g.muscianisi@cineca.it
SuperComputing Applications and Innovation Department

May 17th, 2013

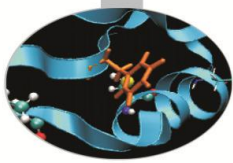
CINECA

Outline



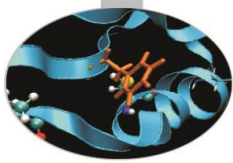
- Overview of Parallel HDF5 design
- Programming model for
  - Creating and accessing a File
  - Creating and accessing a Dataset
  - Writing and reading Hyperslabs

Outline



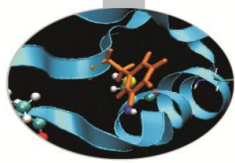
- Overview of Parallel HDF5 design
- Programming model for
  - Creating and accessing a File
  - Creating and accessing a Dataset
  - Writing and reading Hyperslabs

PHDF5 Initial Target



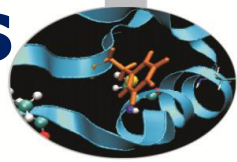
- Support for MPI programming
- Not for shared memory programming
  - Threads
  - OpenMP
- Has some experiments with
  - Thread-safe support for Pthreads
  - OpenMP if called “correctly”

PHDF5 Requirements



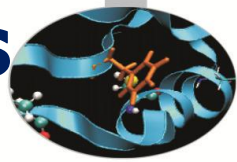
- PHDF5 files compatible with serial HDF5 files
  - Shareable between different serial or parallel platforms
- Single file image to all processes
  - One file per process design is undesirable
    - Expensive post processing
    - Not useable by different number of processes
- Standard parallel I/O interface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

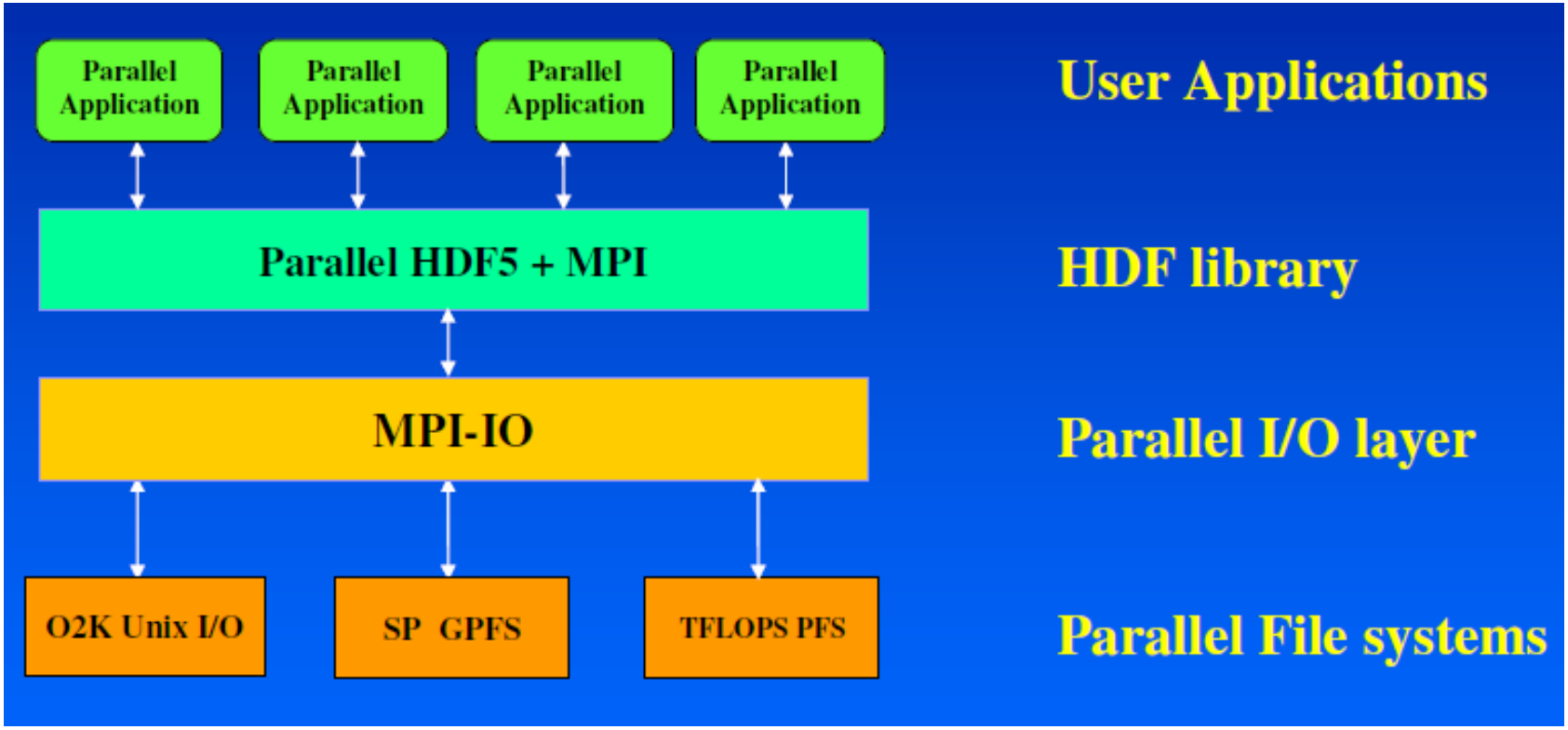
Implementation Requirements

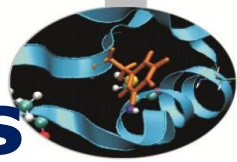


- No use of Threads
  - Not commonly supported (1998)
- No reserved process
  - May interfere with parallel algorithms
- No spawn process
  - Not commonly supported even now

PHDF5 Implementation Layers



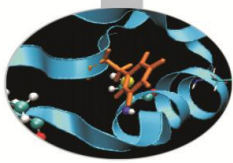




Collective vs. Independent Calls

- 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

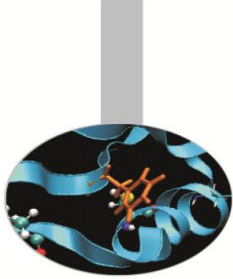
Programming Restrictions



- 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

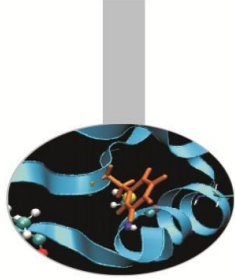
Examples of PHDF5 API



- Examples of PHDF5 collective API
  - File operations: H5Fcreate, H5Fopen, H5Fclose
  - Objects creation: H5Dcreate, H5Dopen, H5Dclose
  - Objects structure: H5Dextend (increase dimension sizes)

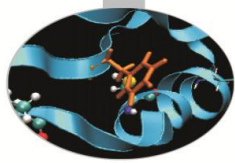
- Array data transfer can be collective or independent
  - Dataset operations: H5Dwrite, H5Dread

What Does PHDF5 Support ?



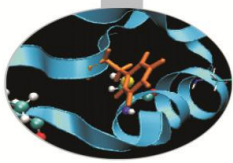
- After a file is opened by the processes of a communicator
  - All parts of file are accessible by all processes
  - All objects in the file are accessible by all processes
  - Multiple processes write to the same data array
  - Each process writes to individual data array

Outline



- Overview of Parallel HDF5 design
- Programming model for
  - Creating and accessing a File
  - Creating and accessing a Dataset
  - Writing and reading Hyperslabs

Programming Model

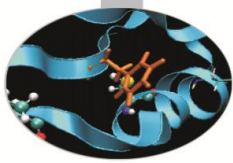


1. Create or open a Parallel HDF5 file with a collective call to:
   - **H5Dcreate** ; **H5Dopen**

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**

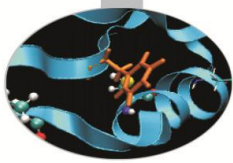
3. Access the dataset with the defined transfer property list.
   - All processes that have opened a dataset may do collective I/O.
   - Each process may do an independent and arbitrary number of data I/O access calls, using: **H5Dwrite** and/or **H5Dread**

Creating and Accessing a File



- HDF5 uses access template object (property list) to control the file access mechanism
- General model to access HDF5 file in parallel:
  - Setup MPI-IO access template (access property list)
  - Open File
  - Close File

Setup access template



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

C:

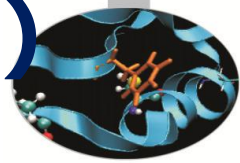
```
herr_t H5Pset_fapl_mpio(hid_t plist_id, MPI_Comm
   comm, MPI_Info info);
```

F90:

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

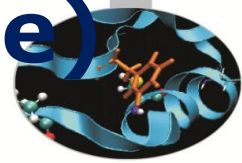
plist_id is a file access property list identifier

Parallel File Create (C Example)



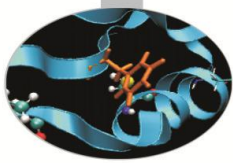
```
comm = MPI_COMM_WORLD;
info = MPI_INFO_NULL;

/* Initialize MPI */
MPI_Init(&argc, &argv);

/* Set up file access property list for MPI-IO access */
plist_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(plist_id, comm, info);

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

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

Parallel File Create (F90 Example)



```
comm = MPI_COMM_WORLD
info = MPI_INFO_NULL

CALL MPI_INIT(mpierror)
! Initialize FORTRAN predefined datatypes
CALL h5open_f(error)

! Setup file access property list for MPI-IO access.
CALL h5pcreate_f(H5P_FILE_ACCESS_F, plist_id, error)
CALL h5pset_fapl_mpio_f(plist_id, comm, info, error)

CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id, error,
   access_prp = plist_id)

! Close the file.
CALL h5fclose_f(file_id, error)
! Close FORTRAN interface
CALL h5close_f(error)
CALL MPI_FINALIZE(mpierror)
```

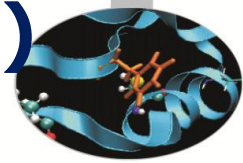
Creating and Opening Dataset



- 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

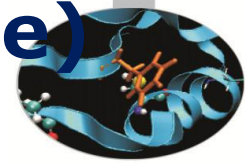
- All processes of the MPI communicator extend dataset with unlimited dimensions before writing to it
  - C: H5Dextend
  - F90: h5dextend_f

Parallel Dataset Create (C Example)



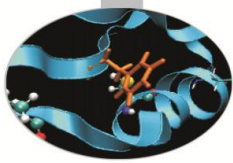
```
file_id = H5Fcreate(…);

/* 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);

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

Parallel Dataset Create (F90 Example)



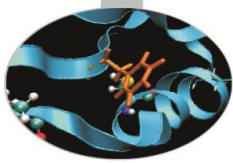
```
CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id,
   error, access_prp = plist_id)
CALL h5screate_simple_f(rank, dimsf, filespace,
   error)
! Create the dataset with default properties.

CALL h5dcreate_f(file_id, "dataset1",
   H5T_NATIVE_INTEGER, filespace, dset_id, error)

! Close the dataset.
CALL h5dclose_f(dset_id, error)
! Close the file.
CALL h5fclose_f(file_id, error)
```

Accessing a Dataset



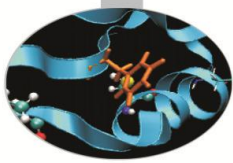
- All processes that have opened dataset may do collective I/O
- Each process may do independent and arbitrary number of data I/O access calls
  - C: H5Dwrite and H5Dread
  - F90: h5dwrite_f and h5dread_f

Accessing a Dataset



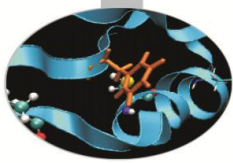
- Create and set dataset transfer property
  - C: H5Pset_dxpl_mpio
    - H5FD_MPIO_COLLECTIVE
    - H5FD_MPIO_INDEPENDENT (default)
  - F90: h5pset_dxpl_mpio_f
    - H5FD_MPIO_COLLECTIVE_F
    - H5FD_MPIO_INDEPENDENT_F (default)
- Access dataset with the defined transfer property

C Example: Collective write



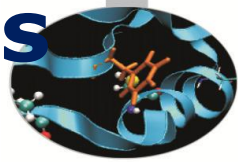
```
/* Create property list for collective dataset
   write. */
plist_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);

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

F90 Example: Collective write



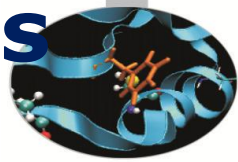
```
! Create property list for collective dataset write
CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id,
   error)
CALL h5pset_dxpl_mpio_f(plist_id, &
   H5FD_MPIO_COLLECTIVE_F, error)

! Write the dataset collectively.

CALL h5dwrite_f(dset_id, H5T_NATIVE_INTEGER, data, &
 error, filespace, memspace, & xfer_prp = plist_id)
```

Writing and Reading Hyperslabs



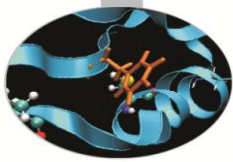
- Distributed memory model: data is split among processes
- PHDF5 uses hyperslab model
  - Each process defines memory and file hyperslabs
  - Each process executes partial write/read call
    - Collective calls
    - Independent calls
- The memory and file hyperslabs in the first step are defined with the H5Sselect_hyperslab

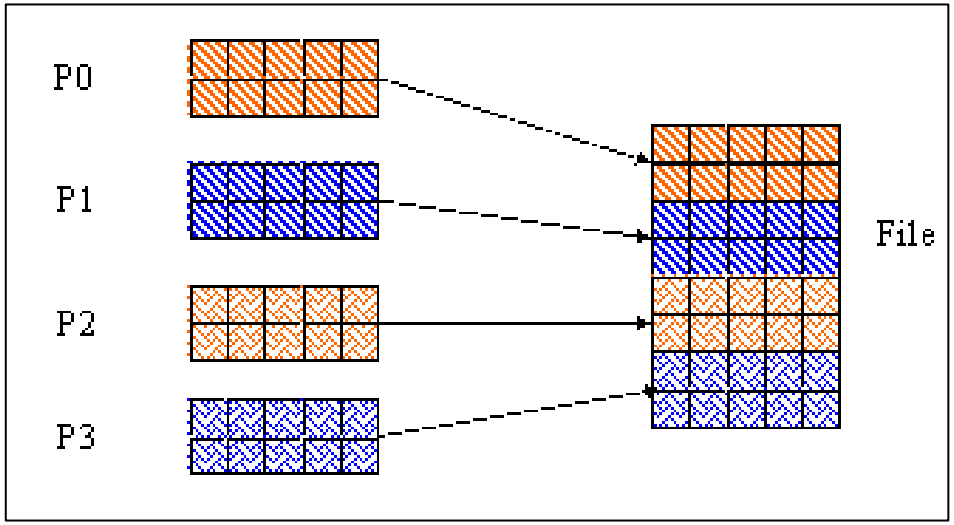
Writing and Reading Hyperslabs



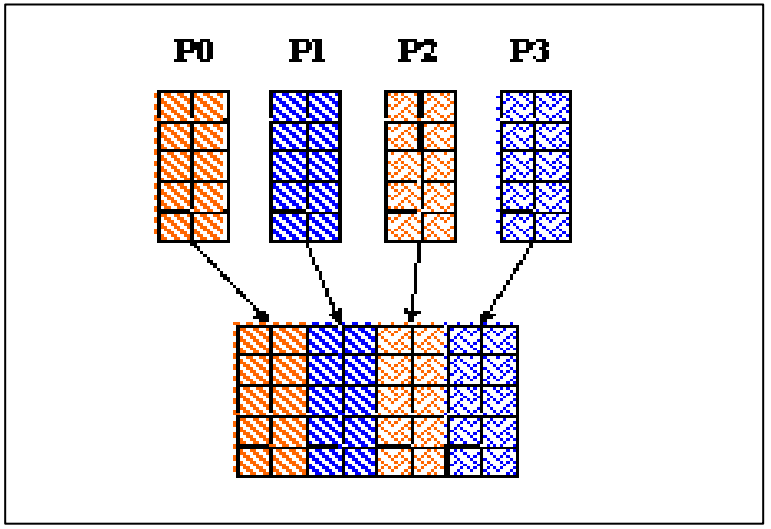
- The *start* (or *offset*), *count*, *stride*, and *block* parameters define the portion of the dataset to write to.
- By changing the values of these parameters you can write hyperslabs with Parallel HDF5 by
  - contiguous hyperslab,
  - regularly spaced data in a column/row,
  - by patterns,
  - by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

Contiguous Hyperslab



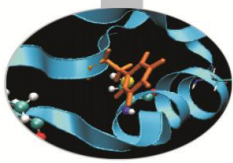


C example



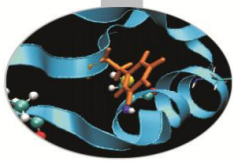
Fortran 90 example

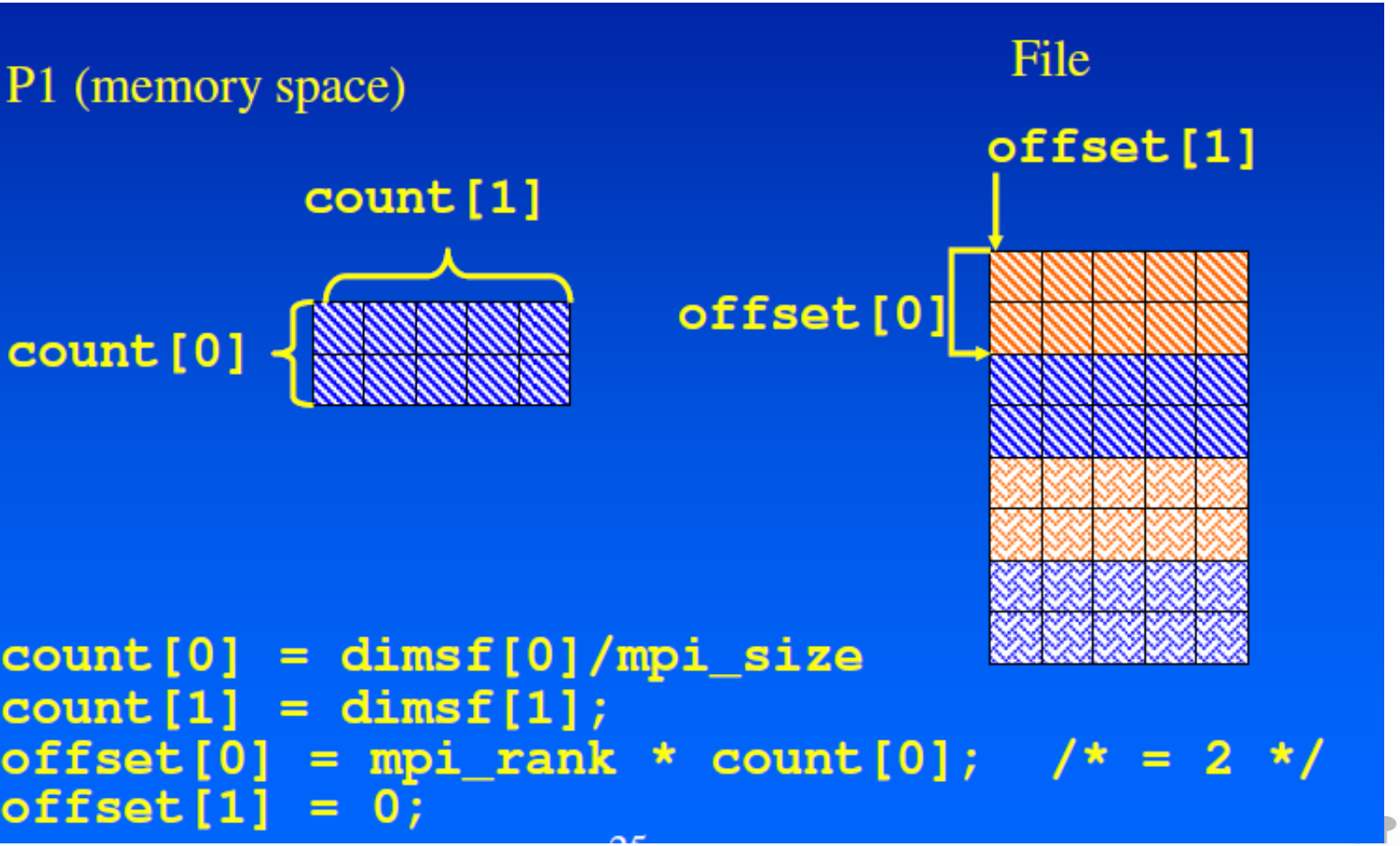
Contiguous Hyperslab



```
HDF5 "SDS_row.h5" {
GROUP "/" {
   DATASET "IntArray" {
         DATATYPE H5T_STD_I32BE
         DATASPACE SIMPLE { ( 8, 5 ) / ( 8, 5 ) }
         DATA {
                  10, 10, 10, 10, 10,
                  10, 10, 10, 10, 10,
                  11, 11, 11, 11, 11,
                  11, 11, 11, 11, 11,
                  12, 12, 12, 12, 12,
                  12, 12, 12, 12, 12,
                  13, 13, 13, 13, 13,
                  13, 13, 13, 13, 13
         }
   }
}
}
```

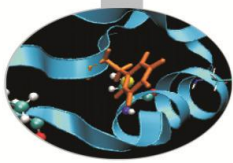
Contiguous Hyperslab






```
count[0] = dimsf[0]/mpi_size
count[1] = dimsf[1];
offset[0] = mpi_rank * count[0];   /* = 2 */
offset[1] = 0;
```

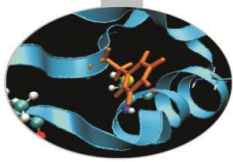
Contiguous Hyperslab

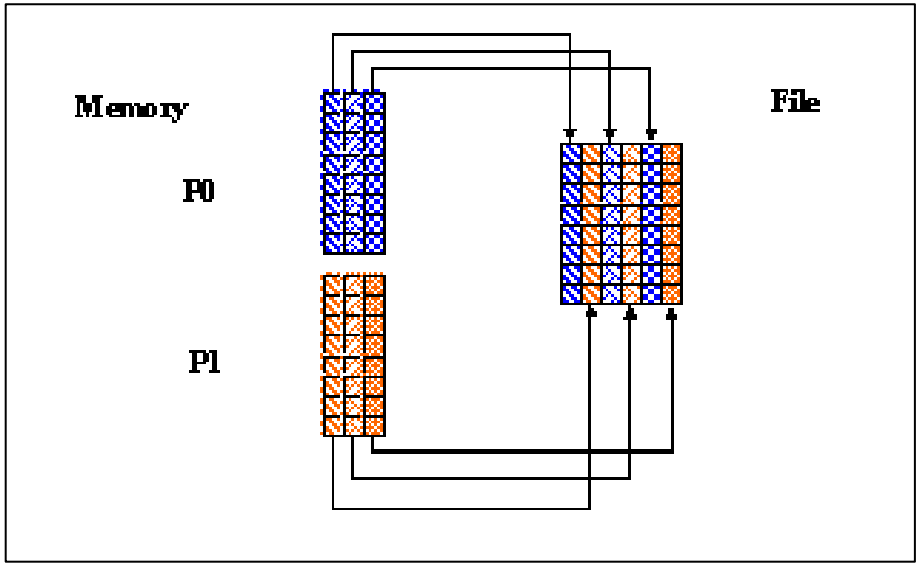


```
/* Each process defines dataset in memory and writes
   it to the hyperslab in the file. */
count[0] = dimsf[0]/mpi_size;
count[1] = dimsf[1];
offset[0] = mpi_rank * count[0];
offset[1] = 0;
memspace = H5Screate_simple(RANK,count,NULL);

/* Select hyperslab in the file. */
filespace = H5Dget_space(dset_id);
H5Sselect_hyperslab(filespace,
   H5S_SELECT_SET,offset,NULL,count,NULL);
```

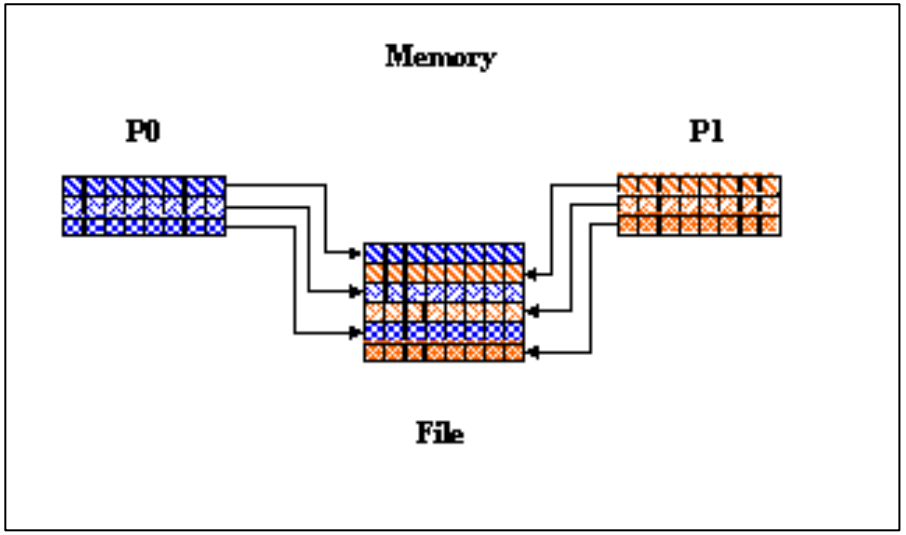
http://www.hdfgroup.org/ftp/HDF5/examples/parallel/Hyperslab_by_row.c
http://www.hdfgroup.org/ftp/HDF5/examples/parallel/hyperslab_by_col.f90

Regularly spaced data



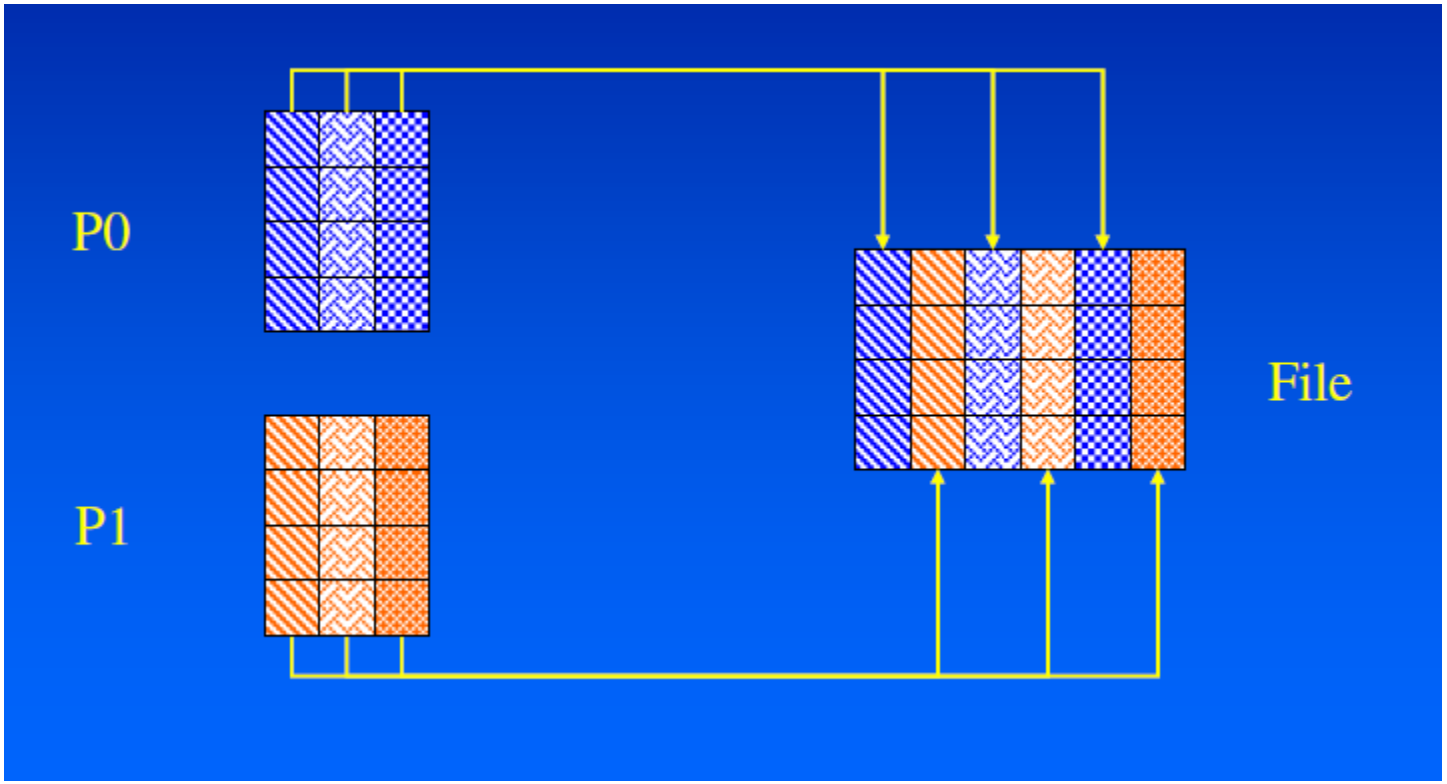


C example

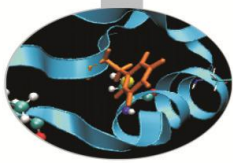


Fortran 90 example

Regularly spaced data

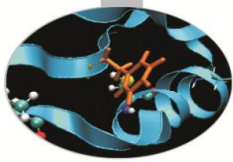


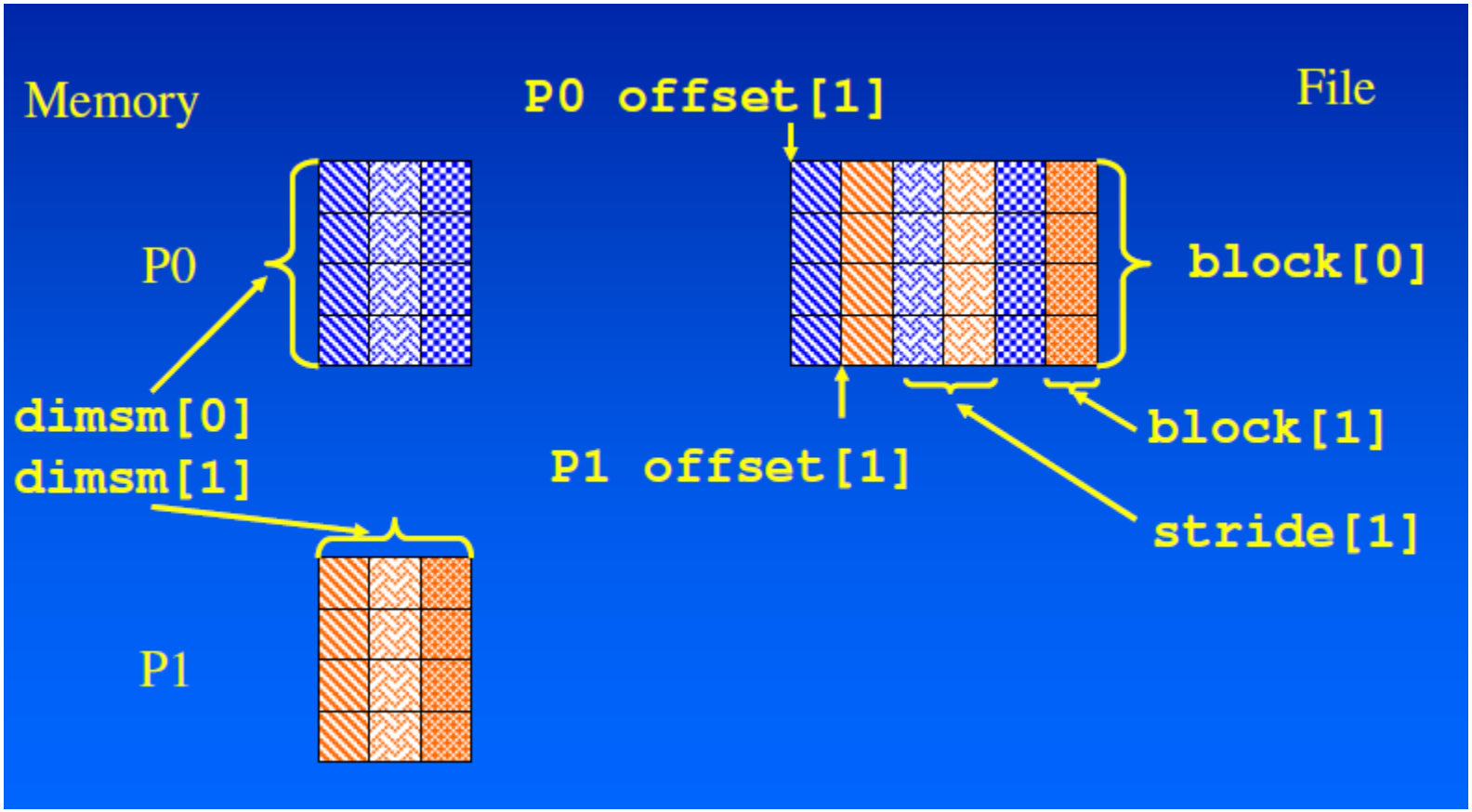
Regularly spaced data



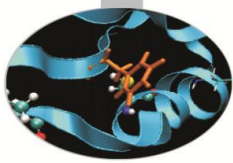
```
HDF5 "SDS_col.h5" {
GROUP "/" {
   DATASET "IntArray" {
        DATATYPE H5T_STD_I32BE
        DATASPACE SIMPLE { ( 8, 6 ) / ( 8, 6 ) }
        DATA {
                1, 2, 10, 20, 100, 200,
                1, 2, 10, 20, 100, 200,
                1, 2, 10, 20, 100, 200,
                1, 2, 10, 20, 100, 200,
                1, 2, 10, 20, 100, 200,
                1, 2, 10, 20, 100, 200,
                1, 2, 10, 20, 100, 200,
                1, 2, 10, 20, 100, 200
        }
   }
}
}
```

Regularly spaced data (C)





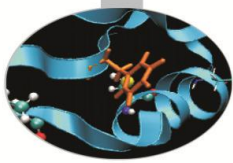
Regularly spaced data

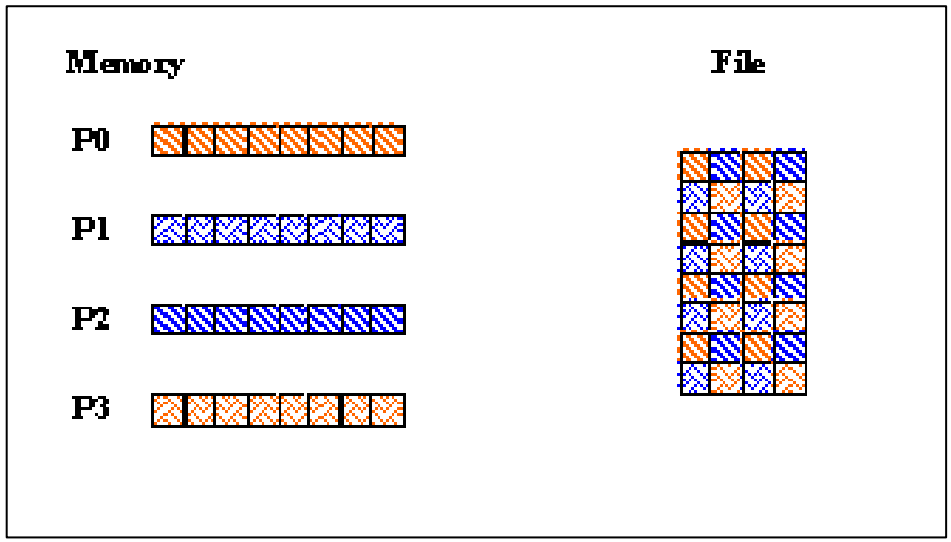


```
/* Each process defines hyperslab in the file  */
count[0] = 1;  count[1] = dimsm[1];
offset[0] = 0;  offset[1] = mpi_rank;
stride[0] = 1;  stride[1] = 2;
block[0] = dimsf[0];  block[1] = 1;

/* Each process selects hyperslab.  */
filespace = H5Dget_space(dset_id);
H5Sselect_hyperslab(filespace, H5S_SELECT_SET,
   offset, stride,
count, block);
```

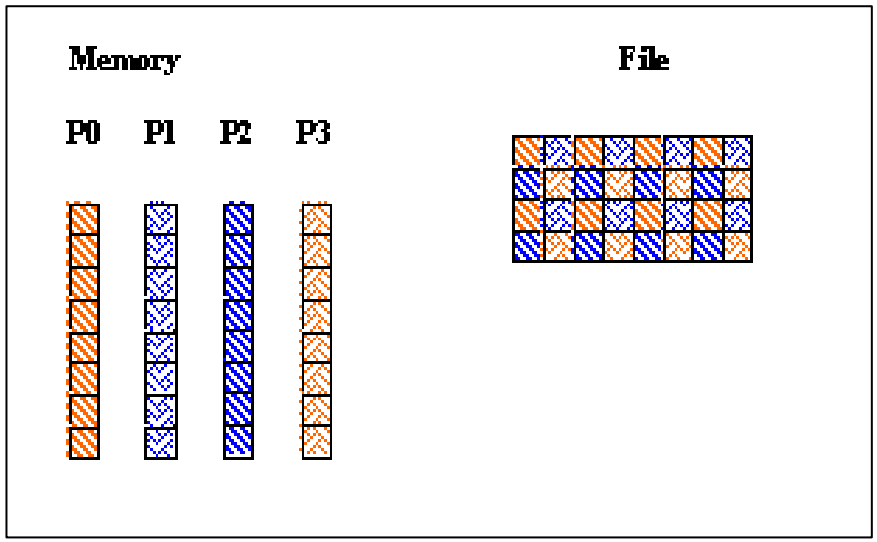
http://www.hdfgroup.org/ftp/HDF5/examples/parallel/Hyperslab_by_col.c
http://www.hdfgroup.org/ftp/HDF5/examples/parallel/hyperslab_by_col.f90

By patterns



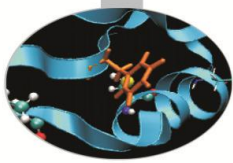


C example



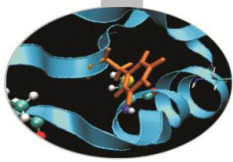
Fortran 90 example

By patterns

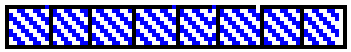


```
HDF5 "SDS_pat.h5" {
GROUP "/" {
   DATASET "IntArray" {
        DATATYPE H5T_STD_I32BE
        DATASPACE SIMPLE { ( 8, 4 ) / ( 8, 4 ) }
        DATA {
               1, 3, 1, 3,
               2, 4, 2, 4,
               1, 3, 1, 3,
               2, 4, 2, 4,
               1, 3, 1, 3,
               2, 4, 2, 4,
               1, 3, 1, 3,
               2, 4, 2, 4
        }
   }
}
}
```

By patterns

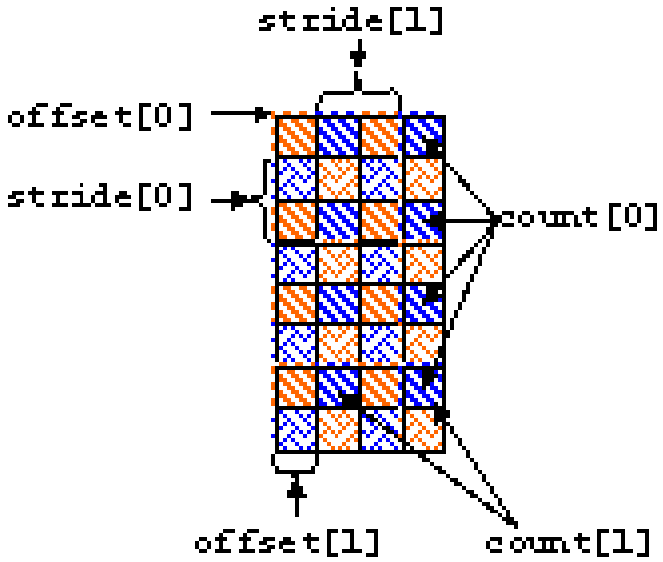


P2 Memory

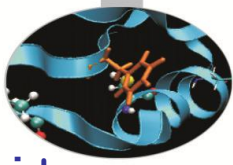


```
offset[0] = 0;
offset[1] = 1;
count[0]  = 4;
count[1]  = 2;
stride[0] = 2;
stride[1] = 2;
```

File



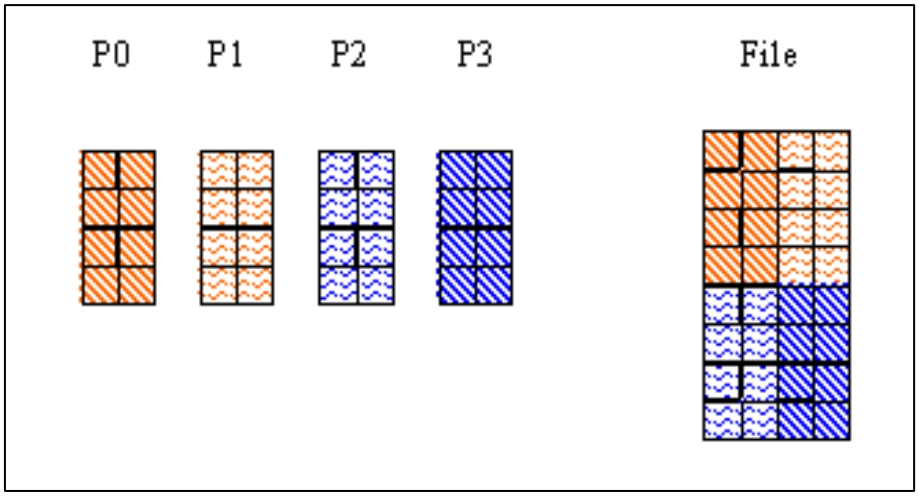
By pattern



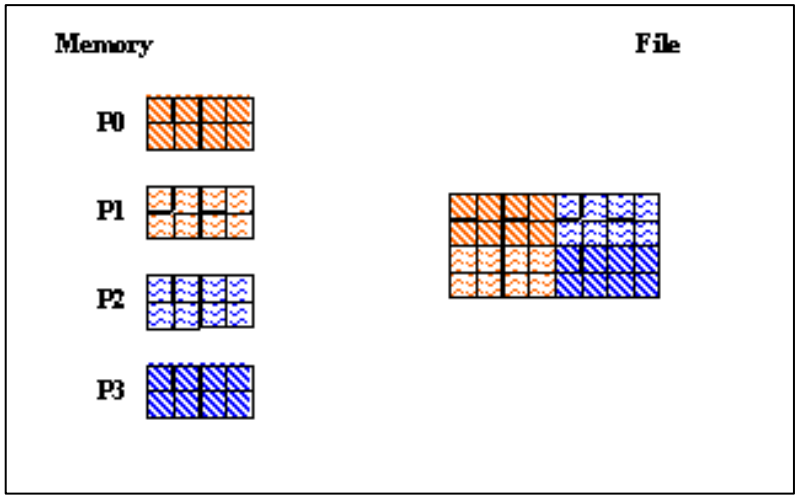
```
/* * Each process defines dataset in memory and writes it
   to the hyperslab * in the file. */
count[0] = 4; count[1] = 2;
stride[0] = 2; stride[1] = 2;
if(mpi_rank == 0) { offset[0] = 0; offset[1] = 0; }
if(mpi_rank == 1) { offset[0] = 1; offset[1] = 0; }
if(mpi_rank == 2) { offset[0] = 0; offset[1] = 1; }
if(mpi_rank == 3) { offset[0] = 1; offset[1] = 1; }

/* * Select hyperslab in the file. */
filespace = H5Dget_space(dset_id);
status = H5Sselect_hyperslab(filespace, H5S_SELECT_SET,
   offset, stride, count, NULL);
```

http://www.hdfgroup.org/ftp/HDF5/examples/parallel/Hyperslab_by_pattern.c
http://www.hdfgroup.org/ftp/HDF5/examples/parallel/hyperslab_by_pattern.f90

By chuncks

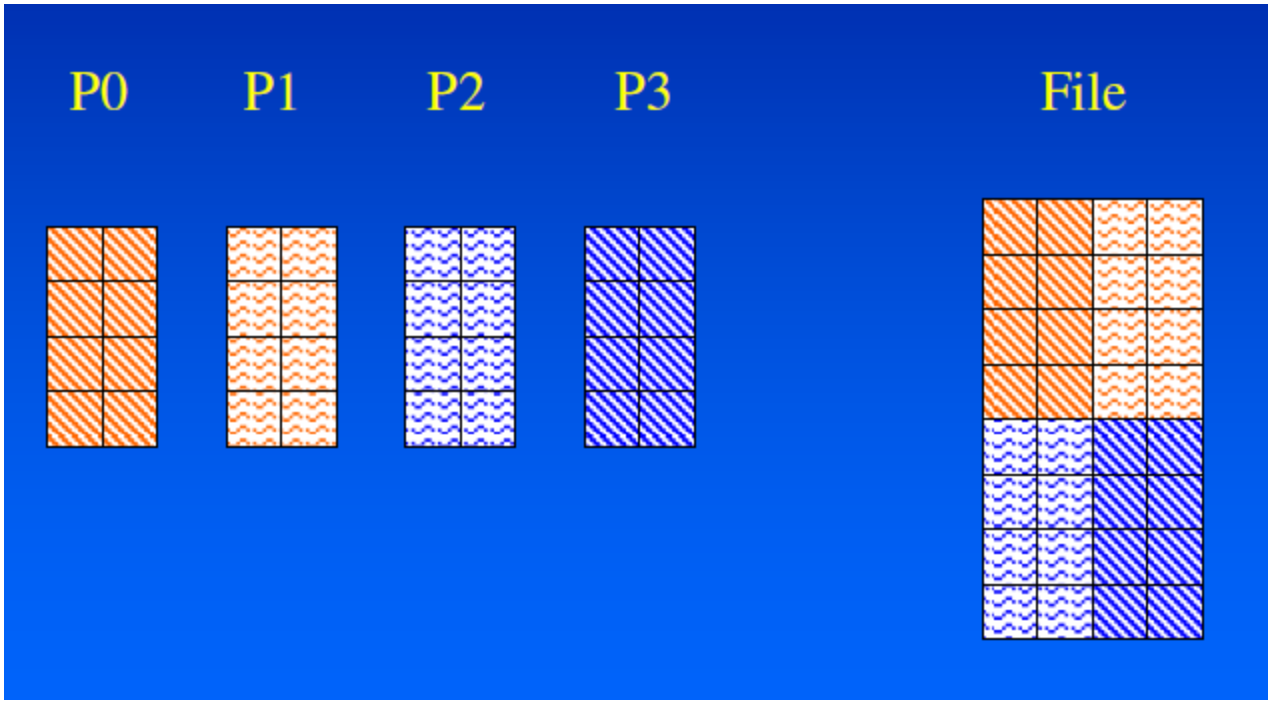


C example

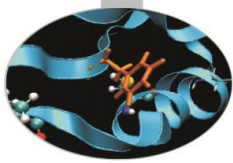


Fortran 90 example

By chunks

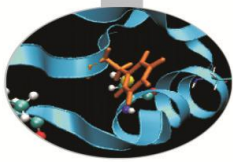


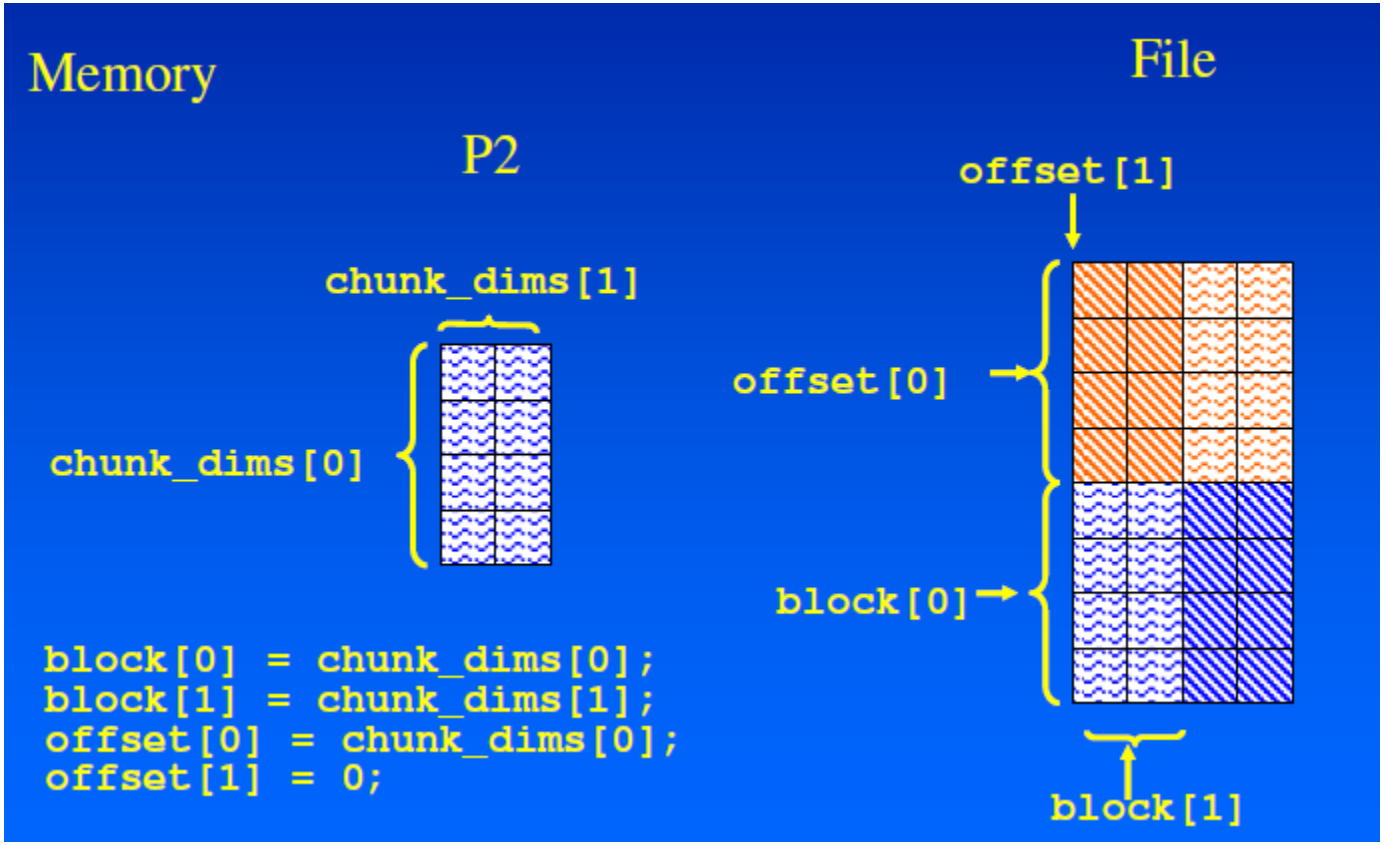
By chunks



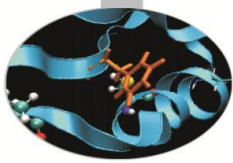
```
HDF5 "SDS_chnk.h5" {
GROUP "/" {
   DATASET "IntArray" {
         DATATYPE H5T_STD_I32BE
         DATASPACE SIMPLE { ( 8, 4 ) / ( 8, 4 ) }
         DATA {
                  1, 1, 2, 2,
                  1, 1, 2, 2,
                  1, 1, 2, 2,
                  1, 1, 2, 2,
                  3, 3, 4, 4,
                  3, 3, 4, 4,
                  3, 3, 4, 4,
                  3, 3, 4, 4
         }
   }
}
}
```

By chunks (C)




Memory
P2
chunk_dims[1]
chunk_dims[0]
block[0] = chunk_dims[0];
block[1] = chunk_dims[1];
offset[0] = chunk_dims[0];
offset[1] = 0;
File
offset[1]
offset[0]
block[0]
block[1]

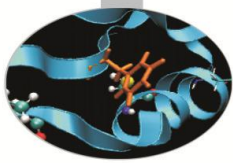
By chunks



```
count[0] = 1; count[1] = 1 ;
stride[0] = 1; stride[1] = 1;
block[0] = chunk_dims[0];  block[1] = chunk_dims[1];
if(mpi_rank == 0) { offset[0] = 0; offset[1] = 0; }
if(mpi_rank == 1) { offset[0] = 0; offset[1] = chunk_dims[1]; }
if(mpi_rank == 2) { offset[0] = chunk_dims[0]; offset[1] =  0; }
if(mpi_rank == 3) { offset[0] = chunk_dims[0];
        offset[1] = chunk_dims[1]; }

/* * Select hyperslab in the file. */
filespace = H5Dget_space(dset_id);
status = H5Sselect_hyperslab(filespace, H5S_SELECT_SET, offset,
   stride, count, block);
```

http://www.hdfgroup.org/ftp/HDF5/examples/parallel/Hyperslab_by_chunk.c
http://www.hdfgroup.org/ftp/HDF5/examples/parallel/hyperslab_by_chunk.f90

Usefull links

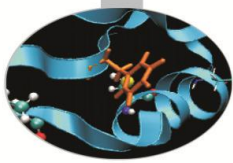


The HDF Group Page: http://hdfgroup.org/
HDF5 Home Page: http://hdfgroup.org/HDF5/

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

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

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



QUESTIONS ???