CSE 5449: Intermediate Studies in Scientific Data Management

Lecture 7: MPI-IO, NetCDF, PnetCDF

Dr. Suren Byna

The Ohio State University

E-mail: byna.1@osu.edu

https://sbyna.github.io

02/02/2023

Today’s class

- Any questions about hyperslabs and MPI-IO?

- Revised class project execution plan

- Class presentation topic
  - Parallel I/O performance
  - Metadata management in scientific data
  - Provenance of scientific data
  - Quality of scientific data

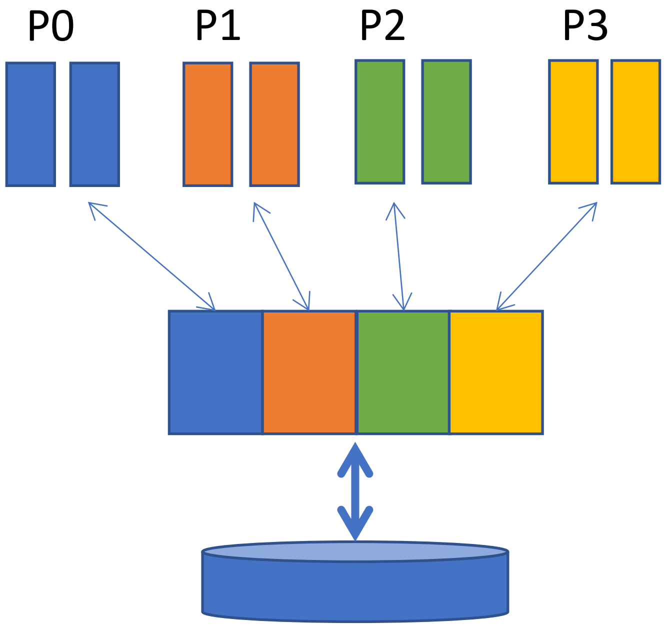
- This Class –
  - MPI-IO optimizations
  - NetCDF and PnetCDF

MPI-IO performance optimizations

- Too many I/O requests to storage devices hurt performance
  - Each request has a start up cost and data transfer cost
  - While data transfer cost is dependent on the amount of data, start up cost is typically the same for all requests
  - Less number of requests avoids some of the start up cost
- Optimizations in MPI-IO
  - Collective buffering (two-phase I/O)
  - Data sieving

MPI-IO performance optimizations – Collective buffering

- Also known as two-phase I/O
- A few processes aggregate data to temporary buffers and the data is then written to file (collective write operations)



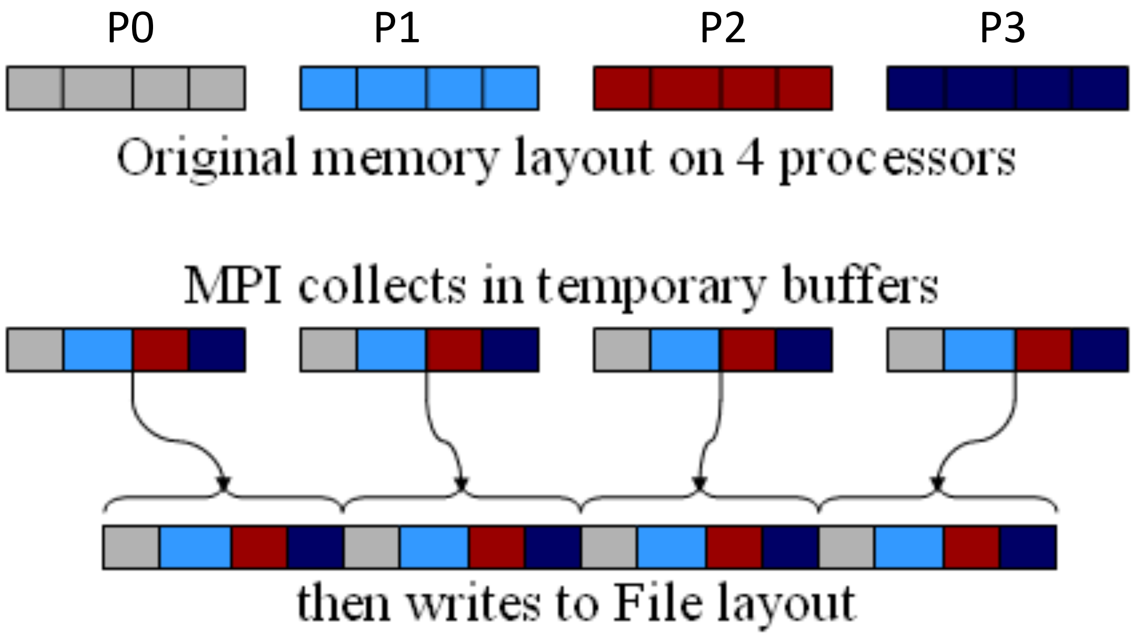

P0
P1
P2
P3
Original memory layout on 4 processors
MPI collects in temporary buffers
then writes to File layout

Image from https://cvw.cac.cornell.edu/ParallelIO/choreography

MPI-IO performance optimizations – Collective reads

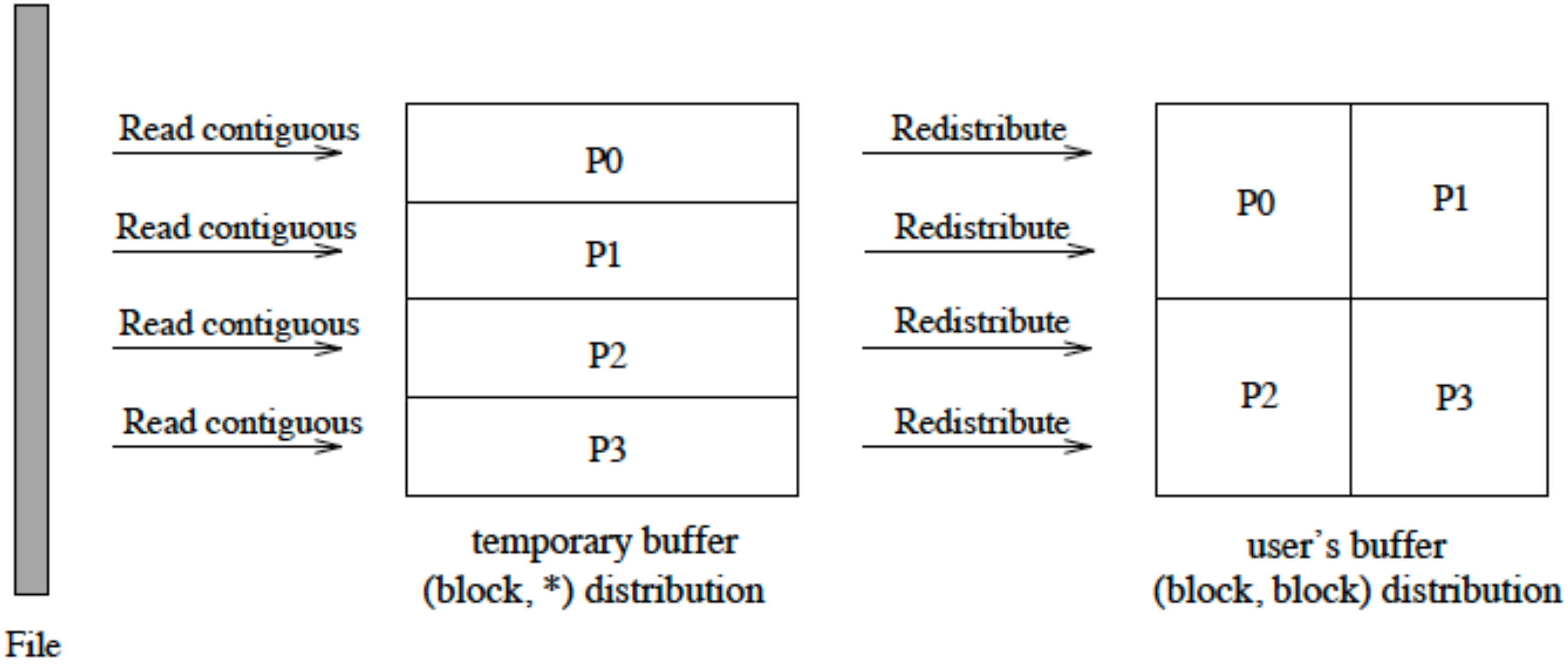


Image from Rajeev Thakur, et al., “Data Sieving and Collective I/O in ROMIO”, Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation, February 1999

MPI-IO performance optimizations – collective read operation

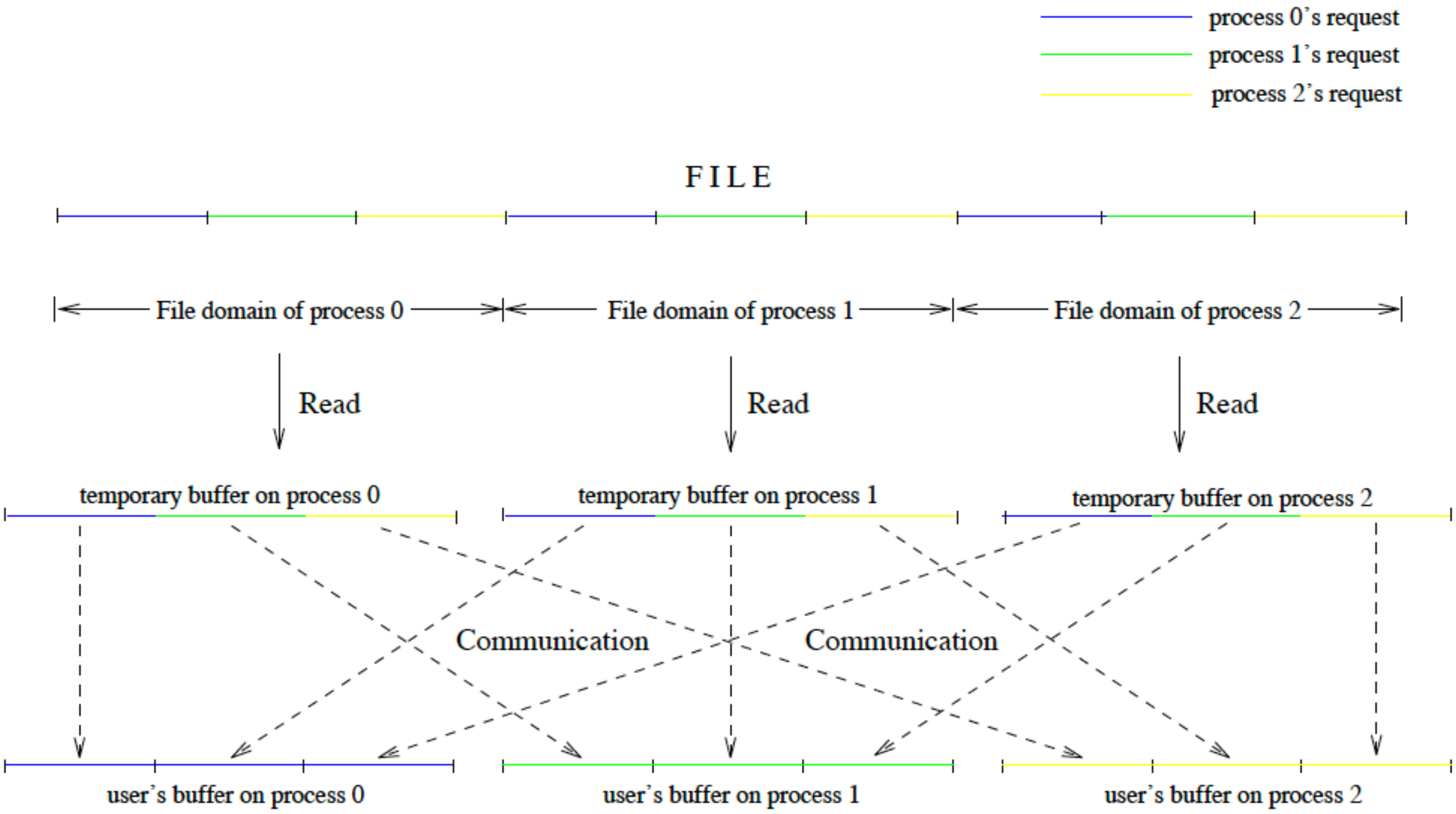


Image from Rajeev Thakur, et al., "Data Sieving and Collective I/O in ROMIO", Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation, February 1999

MPI-IO performance optimizations – Data Sieving

- Data sieving to reduce the number of non-contiguous operations
- Read more than needed – including the data between non-contiguous accesses
  - Copy requested portions to user buffer

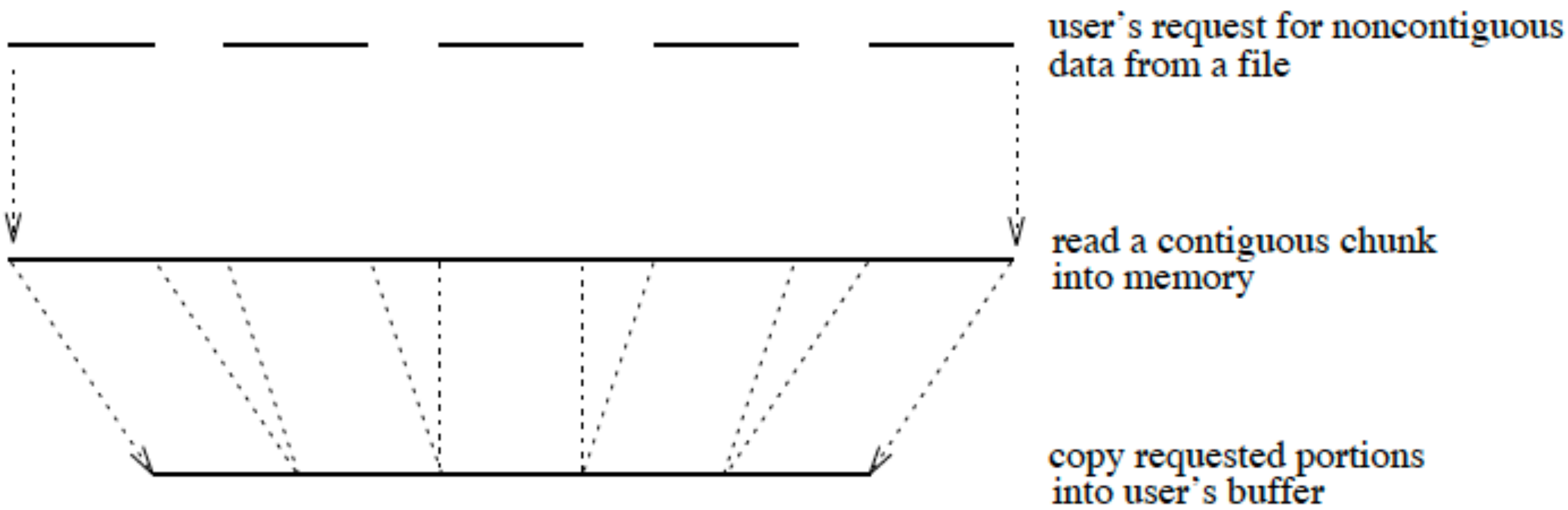

user's request for noncontiguous data from a file
read a contiguous chunk into memory
copy requested portions into user's buffer

Image from Rajeev Thakur, et al., "Data Sieving and Collective I/O in ROMIO", Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation, February 1999

MPI_Info

- Learned about this briefly when we talked about
  - Setting file access property list (FAPL) to use MPI communicator
  - H5Pset_fapl_mpio(hid_t fapl_id, MPI_Comm comm, MPI_Info info);
    - *MPI_Comm* → MPI communicator
      - If all processes will access the file, use MPI_COMM_WORLD
    - *MPI_Info* → MPI Info object for passing hints about I/O to the MPI-IO layer
      - E.g., buffer sizes, MPI-IO concurrency, contiguity, etc.

MPI_Info – for passing hints

- MPI_Info provides an extensible list of (key, value) pairs
- Often used in sending I/O hints to MPI to perform optimizations
- Also used to set values to various communication parameters
- Recommendations only, not commands to MPI

```
striping_unit
striping_factor
cb_buffer_size
cb_nodes
ind_rd_buffer_size
ind_wr_buffer_size
start_iodevice
pfs_svr_buf
direct_read
direct_write
```

Passing hints with MPI_info

```
MPI_Info info;
MPI_Info_create (&info);
/* no. of I/O devices to be used for file striping */
MPI_Info_set (info, "striping_factor", "4");

/* the striping unit in bytes */
MPI_Info_set (info, "striping_unit", "65536");
```

`striping_factor` - size of "strips" on I/O servers
`striping_unit` - number of I/O servers to stripe across
`start_iodevice` - which I/O server to start with
`cb_config_list` - list of aggregators
`cb_nodes` - number of aggregators (upper bound)
`romio_cb_read`, `romio_cb_write` - aggregation on/off
`romio_ds_read`, `romio_ds_write` - data sieving on/off

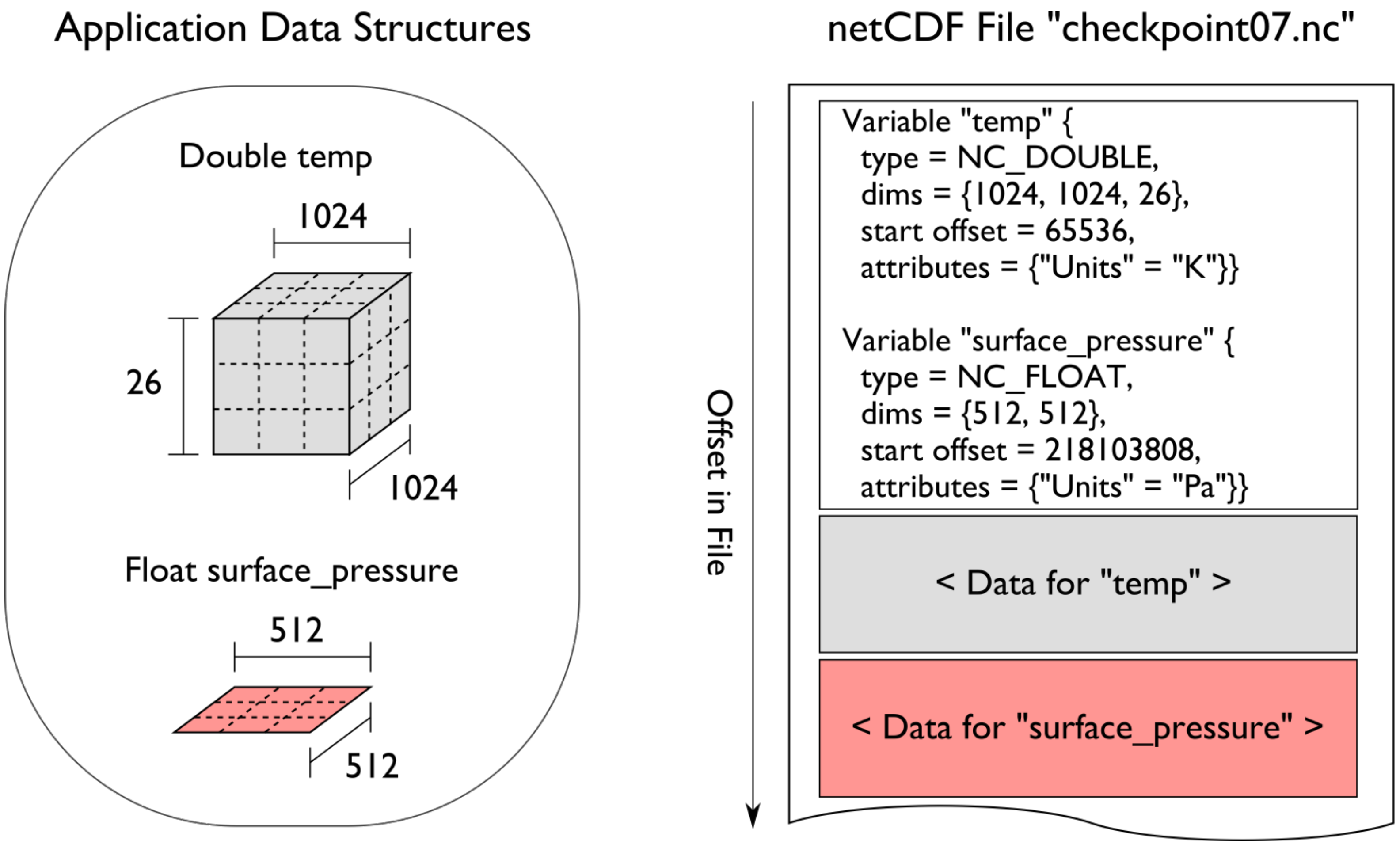
```
MPI_File_open(MPI_COMM_WORLD, ”data.file",
          MPI_MODE_CREATE | MPI_MODE_RDWR, info, &fh);
MPI_Info_free (&info);
```

Adapted from Prof. Bill Gropp's "Introduction to MPI-IO" talk

NetCDF and Parallel netCDF

- Network Common Data Form – NetCDF
- A set of software libraries and self-describing, machine-independent data formats
  - that support the creation, access, and sharing of array-oriented scientific data
- Maintained by Unidata
  - University Corporation for Atmospheric Research (UCAR) Community Programs (UCP)

NetCDF - to store multiple arrays in a single file with metadata



Application Data Structures

netCDF File "checkpoint07.nc"

```
Variable "temp" {
  type = NC_DOUBLE,
  dims = {1024, 1024, 26},
  start offset = 65536,
  attributes = {"Units" = "K"}}

Variable "surface_pressure" {
  type = NC_FLOAT,
  dims = {512, 512},
  start offset = 218103808,
  attributes = {"Units" = "Pa"}}
```

< Data for "temp" >

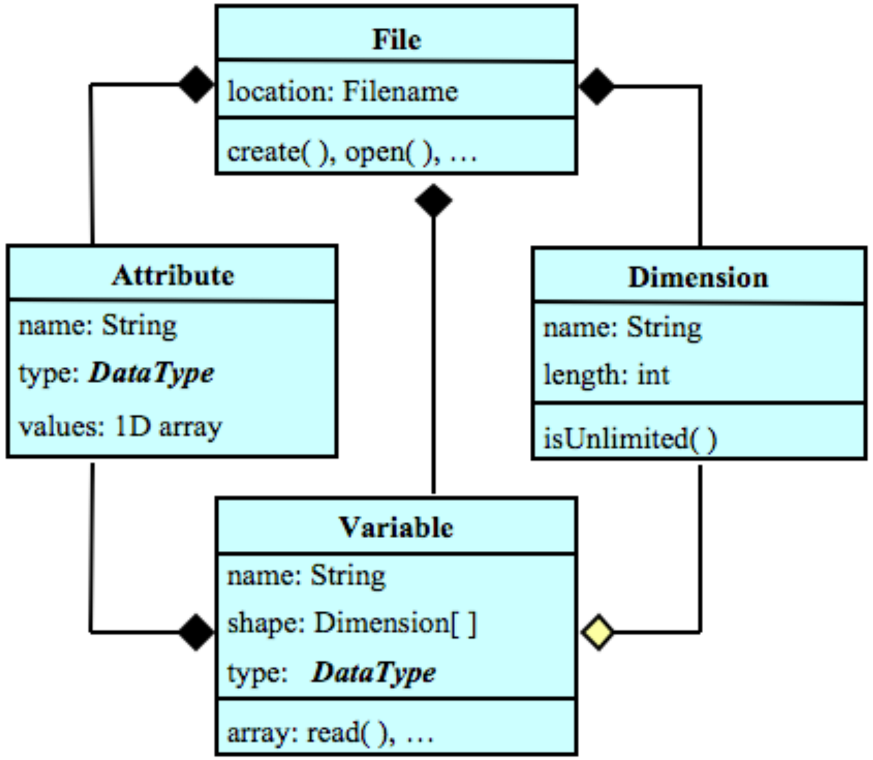
< Data for "surface_pressure" >

netCDF header describes the contents of the file: typed, multi-dimensional variables and attributes on variables or the dataset itself.

Data for variables is stored in contiguous blocks, encoded in a portable binary format according to the variable's type.

Image from Pnetcdf tutorial / Bill Gropp's slides on MPI-IO

NetCDF data model - Classic



File

location: Filename

create( ), open( ), …

Attribute

name: String

type: ***DataType***

values: 1D array

Dimension

name: String

length: int

isUnlimited( )

Variable

name: String

shape: Dimension[ ]

type: ***DataType***

array: read( ), …

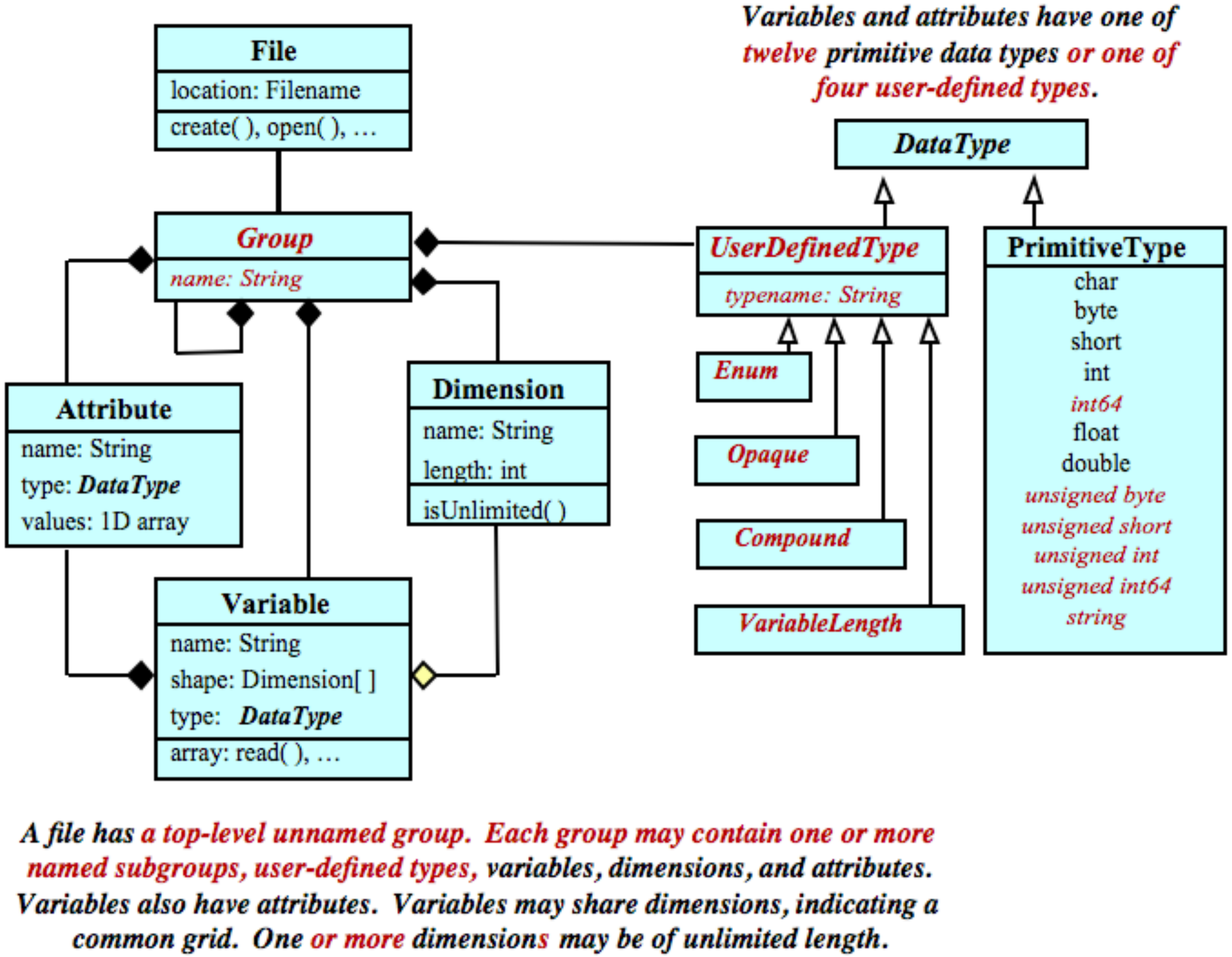
A file has named variables, dimensions, and attributes. Variables also have attributes. Variables may share dimensions, indicating a common grid. One dimension may be of unlimited length.

Variables and attributes have one of six primitive data types.

DataType
char
byte
short
int
float
double

Figure source: https://docs.unidata.ucar.edu/netcdf-c/current/netcdf_data_model.html

NetCDF data model - Enhanced



A file has a top-level unnamed group. Each group may contain one or more named subgroups, user-defined types, variables, dimensions, and attributes. Variables also have attributes. Variables may share dimensions, indicating a common grid. One or more dimensions may be of unlimited length.

This format is implemented in NetCDF-4

Stores data in HDF5 files

Supports parallel I/O via parallel HDF5

Figure source: https://docs.unidata.ucar.edu/netcdf-c/current/netcdf_data_model.html

NetCDF API – Write data

- nc_create (FILE_NAME, NC_CLOBBER, &ncid) ;
  - // File create, NC_CLOBBER (overwrite existing file) → similar to file open in HDF5
- nc_def_dim (ncid, "x", NX, &x_dimid);   // Create dimensions → HDF5 dataspaces
- nc_def_dim (ncid, "y", NY, &y_dimid);
  - dimids[0] = x_dimid;
  - dimids[1] = y_dimid;
- nc_def_var (ncid, "data", NC_INT, NDIMS, dimids, &varid); → HDF5 dataset
- nc_enddef (ncid); → to tell NetCDF that metadata definitions are done
- nc_put_var_int (ncid, varid, &data_out[0][0]);   → write function
- nc_close (ncid)

NetCDF API – Read data

- nc_open (FILE_NAME, NC_NOWRITE, &ncid); → Open file for read only
- nc_inq_varid (ncid, "data", &varid); → Get the varid of the data variable, based on its name
- nc_get_var_int (ncid, varid, &data_in[0][0]); → read data
- nc_close (ncid); → close file

More NetCDF functions

- https://docs.unidata.ucar.edu/netcdf-c/current/modules.html

- Modules include
  - File and data I/O → HDF5 files
  - Dimensions → HDF5 dataspaces
  - Variables → HDF5 datasets
  - Attributes → HDF5 attributes
  - Groups → HDF5 groups
  - User-defined types → HDF5 compound datatypes

PnetCDF

- PnetCDF is a high-performance parallel I/O library for accessing NetCDF files
- Parallel I/O library by using MPI-IO

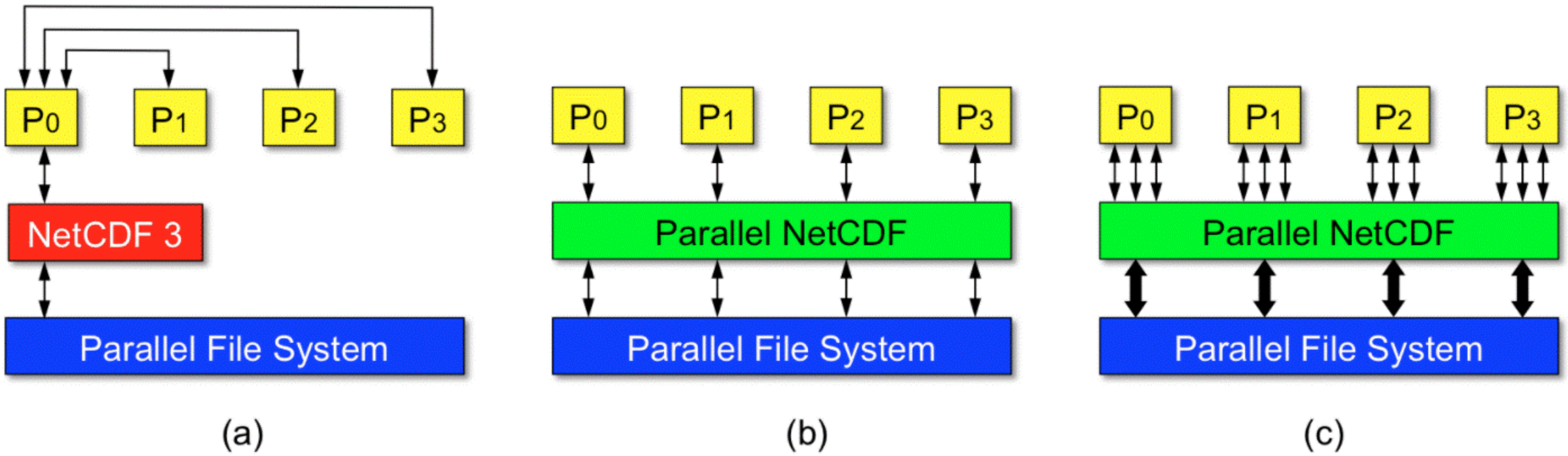


Figure 1. Comparison of data access between using sequential netCDF and PnetCDF. (a) Write operation is carried out through one of the clients when using the sequential netCDF prior to version 4.0. (b) PnetCDF enables concurrent write to parallel file systems. (c) Through nonblocking I/O, PnetCDF can aggregate multiple requests into large ones so a better performance can be achieved.

Image source: cucis.ece.northwestern.edu/projects/PnetCDF

PnetCDF supports MPI-IO optimizations

- Uses ncmpi_ prefix

- Collective I/O
  - By default, collective I/O
  - ncmpi_put_vara_int_all ()
- Independent I/O
  - Specify when to begin and end independent I/O
  - ncmpi_begin_indep_data ()
  - ncmpi_end_indep_data ()

PnetCDF – File operations

- File create:
  - ncmpi_create (MPI_Comm comm, const char* fname, int mode, MPI_Info info, int* id)
- File Open
  - ncmpi_open (MPI_Comm comm, const char* fname, int mode, MPI_Info info, int* id)
- File Close
  - ncmpi_close

PnetCDF – Dimensions and definitions

- ncmpi_def_dim (int id, const char* name, MPI_Offset len, int* dmids)
- ncmpi_def_var (int id, const char* name, nc_type type, int ndims, const int* dmids, int* vid)
- ncmpi_enddef (int id)

PnetCDF – I/O operations

- Read
  - ncmpi_get_vara_<type>_all (int id, int vid, const MPI_Offset start[], const MPI_Offset count[], <type>* var)
- Write
  - ncmpi_put_vara_<type>_all (int id, int vid, const MPI_Offset start[], const MPI_Offset count[], const * var)
- type → int, float, double, …
- Start and count → offsets to be used by MPI processes

- Independent reads and writes
  - ncmpi_get_vara_<type> (...)
  - ncmpi_put_vara_<type> (...)

PnetCDF – Query variables

- ncmpi_inq_varid (int ncid, const char *name, int *varid)
- ncmpi_inq_varname (int ncid, int varid, char *name)
- ncmpi_inq_vartype (int ncid, int varid, nc_type *type)
- ncmpi_inq_vardimid (int ncid, int varid, int dimids[])
- File query
    - ncmpi_inq_<put/get>_size (int ncid, MPI_Offset *size) → how much data was read/written
    - ncmpi_inq_file_info (int ncid, MPI_Info *info) → I/O hints used by PnetCDF

PnetCDF performance hints

- export PNETCDF_HINTS=“romio_cb_write=enable;romio_ds_write=disable;nc_header_align_size=262144”
- More PnetCDF resources
  - https://parallel-netcdf.github.io/
  - https://parallel-netcdf.github.io/wiki/QuickTutorial.html
  - http://cucis.ece.northwestern.edu/projects/PnetCDF/
  - Subfiling: http://cucis.ece.northwestern.edu/projects/PnetCDF/subfiling.html

Summary of today’s class

- MPI-IO optimizations
- NetCDF and PnetCDF
- Next Class – More high-level I/O libraries → ADIOS, VTK, h5py