HIGHER LEVEL I/O INTERFACES

- Provide structure to files
  - Well-defined, portable formats
  - Self-describing
  - Organization of data in file
  - Interfaces for discovering contents

- Present APIs are more appropriate for computational science
  - Typed data
  - Noncontiguous regions in memory and file
  - Multidimensional arrays and I/O on subsets of these arrays

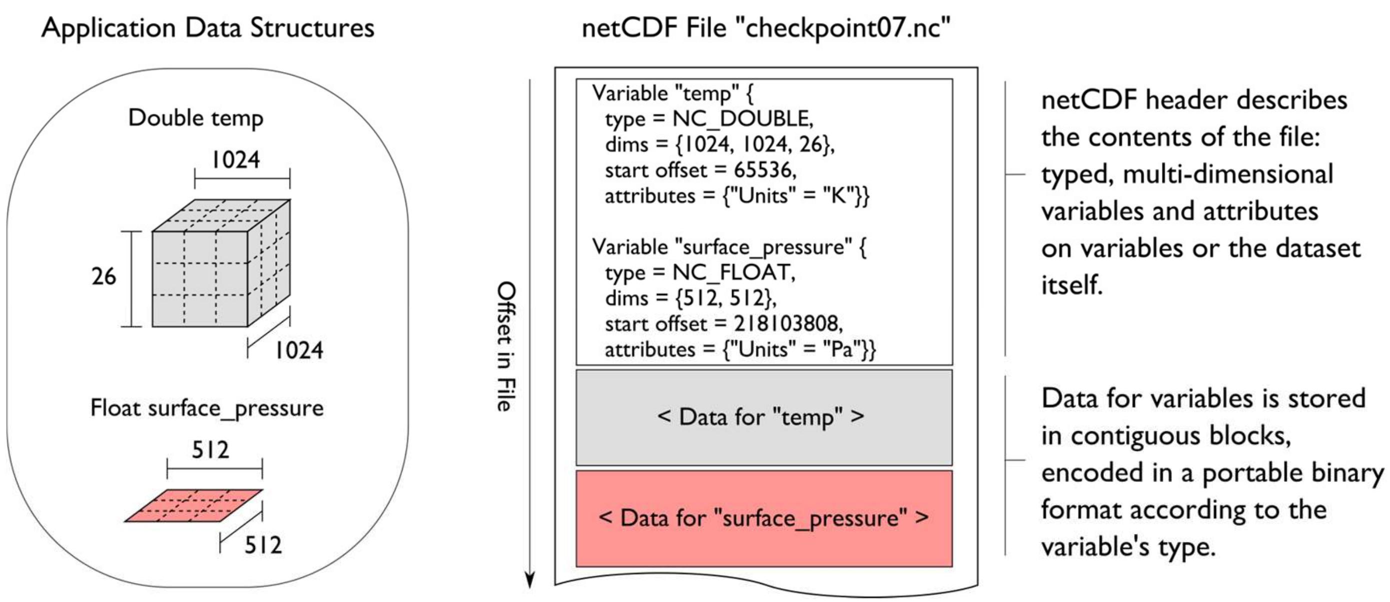
- Both of our example interfaces are implemented on top of MPI-IO

PARALLEL NETCDF (PNETCDF)

- Based on original “Network Common Data Format” (netCDF) work from Unidata
  - Derived from their source code
- Data Model:
  - Collection of variables in single file
  - Typed, multidimensional array variables
  - Attributes on file and variables
- Features:
  - C, Fortran, and F90 interfaces
  - Portable data format (identical to netCDF)
  - Noncontiguous I/O in memory using MPI datatypes
  - Noncontiguous I/O in file using sub-arrays
  - Collective I/O
  - Non-blocking I/O
- Unrelated to netCDF-4 work
- Parallel-NetCDF tutorial:
  - https://parallel-netcdf.github.io/wiki/QuickTutorial.html

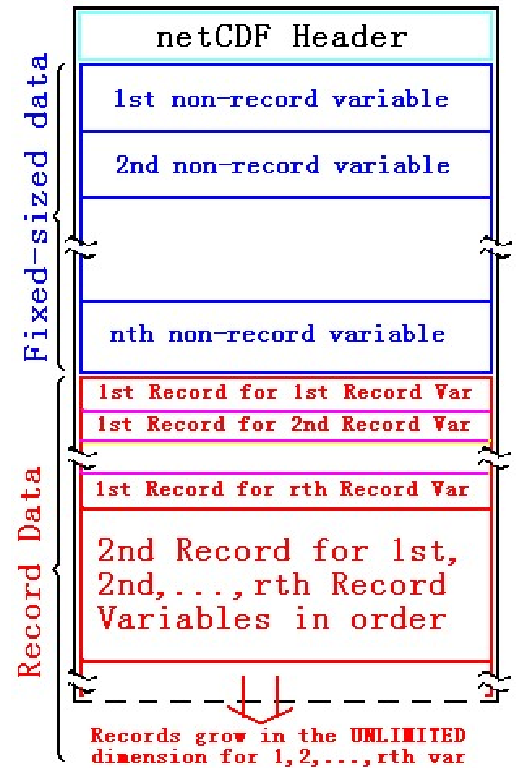
NETCDF DATA MODEL

The netCDF model provides a means for storing multiple, multi-dimensional arrays in a single file.



RECORD VARIABLES IN NETCDF

- Record variables are defined to have a single "unlimited" dimension
  - Convenient when a dimension size is unknown at time of variable creation
- Record variables are stored after all the other variables in an interleaved format
  - Using more than one in a file is likely to result in poor performance due to number of noncontiguous accesses



TUNING FOR PARALLEL-NETCDF

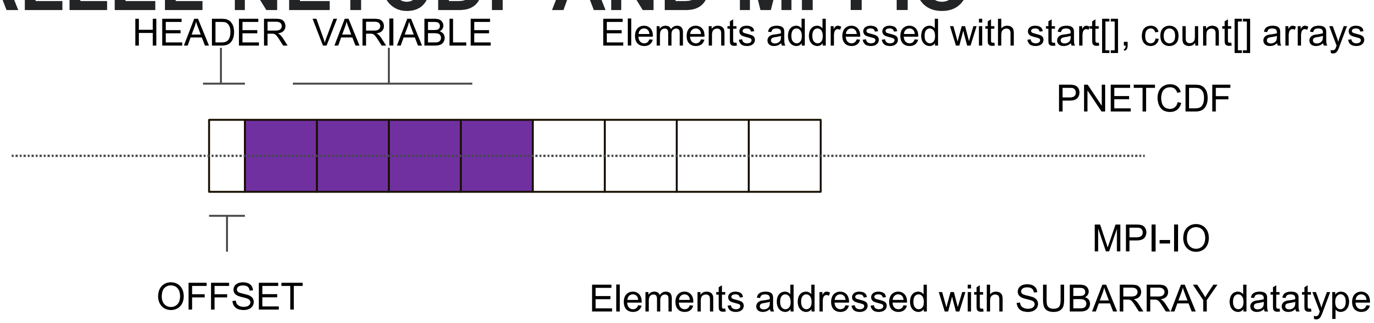
- Not a lot of tuning parameters in pnetcdf itself
  – All the MPI-IO tuning parameters apply
  – All the file system tuning parameters apply
- API choices can have large impact
- Record variables require careful consideration

PNETCDF TUNING: INFO OBJECT

- Create an Info object as you would for MPI codes
- Add key/value strings to info object
- Pass that to create or open routine
- A few hints are pnetcdf specific
  – Alignment of header, variables
  – Gating experimental features

```
MPI_Info_create(&info);
MPI_Info_set(info, "striping_factor", "-1");
MPI_Info_set(info, "romio_ds_write", "disable");
ncmpi_create(comm, filename, NC_CLOBBER,
      info, &ncfile)
```

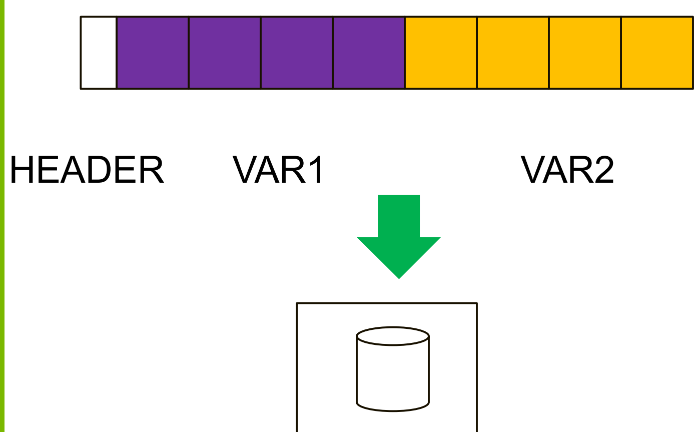
PARALLEL-NETCDF AND MPI-IO



- `ncmpi_put_vara_all` describes access in terms of arrays, elements of arrays
  - For example, “Give me a 3x3 subcube of this larger 1024x1024 array”
- Library translates into MPI-IO calls
  - `MPI_Type_create_subarray`
  - `MPI_File_set_view`
  - `MPI_File_write_all`

PARALLEL-NETCDF WRITE-COMBINING OPTIMIZATION

```
ncmpi_iput_vara(ncfile, varid1,
     &start, &count, &buffer1,
     count, MPI_INT, &requests[0]);
ncmpi_iput_vara(ncfile, varid2,
     &start,&count, &buffer2,
     count, MPI_INT, &requests[1]);
ncmpi_wait_all(ncfile, 2, requests, statuses);
```



- netCDF variables laid out contiguously
- Applications typically store data in separate variables
  - temperature(lat, long, elevation)
  - Velocity_x(x, y, z, timestep)
- Operations posted independently, completed collectively
  - Defer, coalesce synchronization
  - Increase average request size

	Separate	Combined
POSIX writes	161	2
POSIX reads	0	1
MPI-IO indep writes	1	1
MPI-IO coll. writes	160	16

Selected Darshan stats from 16 processes each writing 10 variables to a dataset: Note effects of data sieving and deciding against collective I/O

https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on/blob/master/array/solutions/10-array-pnetcdf-op-combine-compare.c

PNETCDF EXAMPLE

- Write five variables to a file
- Do some "work"

Case 1: one collective at a time

```
srand(10+rank);
for (int j=0; j< NVARS; j++)  {
    /* mimic doing some computation
     * additionally, pretend the computation is unevenly distributed across
     * processes */
    usleep(rand()%5000000);

    ret = ncmpi_put_vara_all(ncfile, varid[j], start, count,
            write_buf, count[0], MPI_INT);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);
}
```

Case 2: non-blocking operations

```
srand(10+rank);
for (int j=0; j< NVARS; j++)  {
    /* mimic doing some computation
     * additionally, pretend the computation is unevenly distributed across
     * processes */
    usleep(rand()%5000000);
    /* the non-blockig operations don't actually do any i/o, so we can issue
     * them non-collectively */
    ret = ncmpi_iput_vara(ncfile, varid[j], start, count,
            write_buf, count[0], MPI_INT,
            &requests[j]);
}
/* in the non-blocking case, this collective wait routine is where all the
 * work happens: there is no background i/o thread  */
ret = ncmpi_wait_all(ncfile, NVARS, requests, statuses);
if (ret != NC_NOERR) handle_error(ret, __LINE__);

/* check status of each nonblocking call */
for (int j=0; j< NVARS; j++)
    if (statuses[j] != NC_NOERR) handle_error(statuses[j], __LINE__);
```

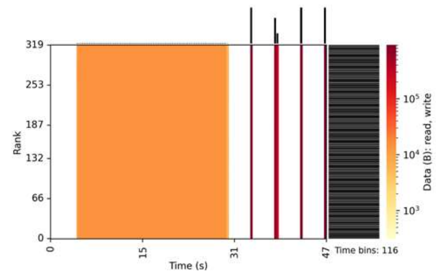
https://github.com/radix-io/io-sleuthing/tree/main/examples/pnetcdf

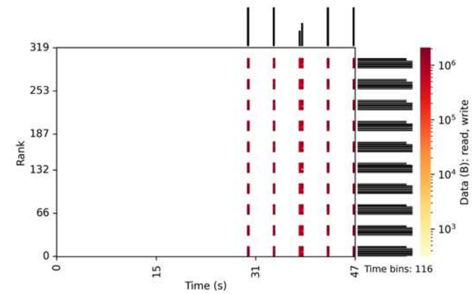
COMPARING APPROACHES WITH DARSHAN

MPI-IO

POSIX

blocking





Non-blocking