Argonne NATIONAL LABORATORY

... for a brighter future

University of California
Lawrence Livermore National Laboratory

U.S. Department of Energy

UChicago Argonne LLC

Office of Science
U.S. DEPARTMENT OF ENERGY

A U.S. Department of Energy laboratory managed by UChicago Argonne, LLC

Parallel NetCDF

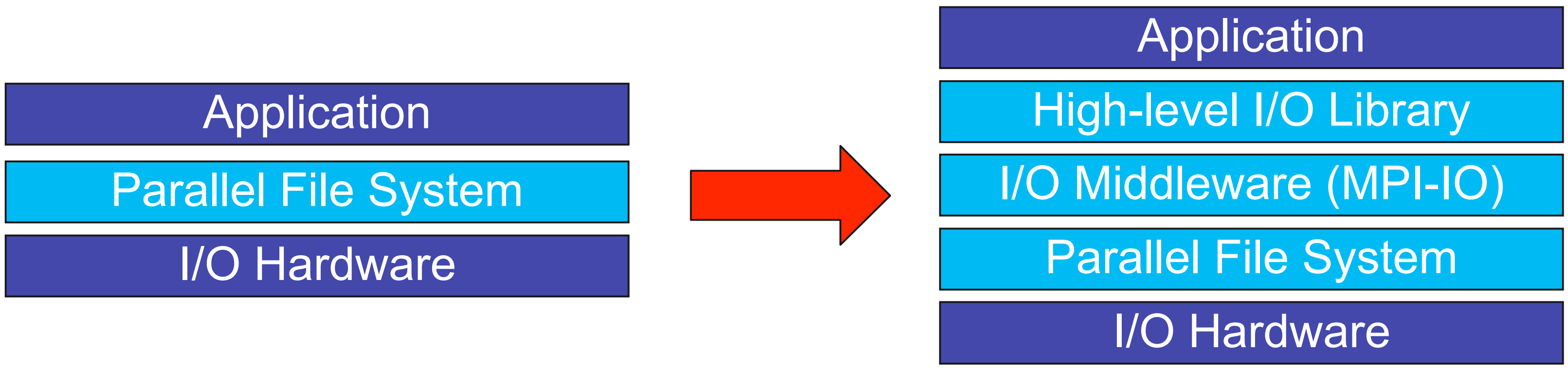
Rob Latham

Mathematics and Computer Science Division

Argonne National Laboratory

robl@mcs.anl.gov

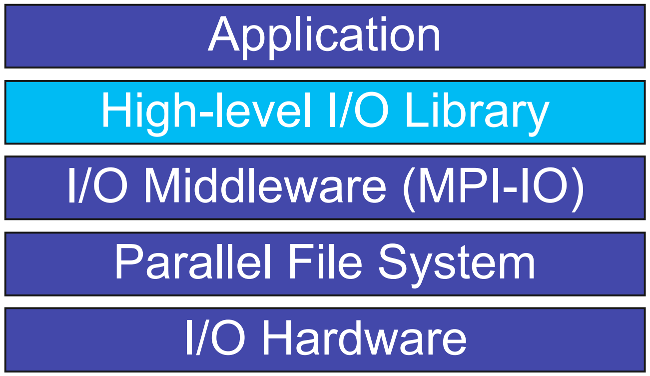
I/O for Computational Science



- Application require more software than just a parallel file system
- Break up support into multiple layers with distinct roles:
  - Parallel file system maintains logical space, provides efficient access to data (e.g. PVFS, GPFS, Lustre)
  - Middleware layer deals with organizing access by many processes (e.g. MPI-IO, UPC-IO)
  - High level I/O library maps app. abstractions to a structured, portable file format (e.g. HDF5, Parallel netCDF)

High Level Libraries

- Match storage abstraction to domain
  - Multidimensional datasets
  - Typed variables
  - Attributes
- Provide self-describing, structured files
- Map to middleware interface
  - Encourage collective I/O
- Implement optimizations that middleware cannot, such as
  - Caching attributes of variables
  - Chunking of datasets



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 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

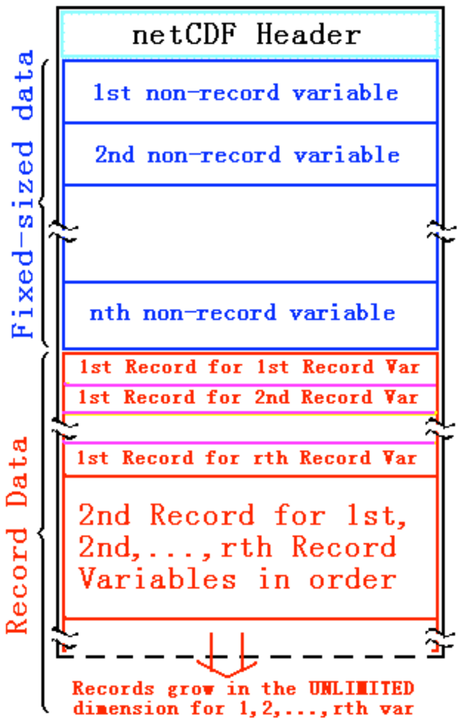
PnetCDF Interface and File Format

Parallel netCDF (PnetCDF)

- Based on original “Network Common Data Format” (netCDF) work from Unidata
  - Derived from their source code
  - Argonne, Northwestern, and community
- Data Model:
  - Collection of variables in single file
  - Typed, multidimensional array variables
  - Attributes on file and variables
- Features:
  - C and Fortran 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
- Unrelated to netCDF-4 work (more later)

netCDF/PnetCDF Files

- PnetCDF files consist of three regions
  - Header
  - Non-record variables (all dimensions specified)
  - Record variables (ones with an unlimited dimension)
- Record variables are interleaved, so using more than one in a file is likely to result in poor performance due to noncontiguous accesses
- Data is always written in a big-endian format



Storing Data in PnetCDF

- Create a dataset (file)
  - Puts dataset in define mode
  - Allows us to describe the contents
    - *Define dimensions for variables*
    - *Define variables using dimensions*
    - *Store attributes if desired (for variable or dataset)*
- Switch from define mode to data mode to write variables
- Store variable data
- Close the dataset

Simple PnetCDF Examples

- Simplest possible PnetCDF version of “Hello World”
- First program creates a dataset with a single attribute
- Second program reads the attribute and prints it
- Shows very basic API use and error checking

Simple PnetCDF: Writing (1)

Integers used for references to datasets, variables, etc.

```
#include <mpi.h>
#include <pnetcdf.h>
int main(int argc, char **argv)
{
    int ncfile, ret, count;
    char buf[13] = "Hello World\n";
    MPI_Init(&argc, &argv);
    ret = ncmpi_create(MPI_COMM_WORLD, "myfile.nc",
        NC_CLOBBER, MPI_INFO_NULL, &ncfile);
    if (ret != NC_NOERR) return 1;

    /* continues on next slide */
```

Simple PnetCDF: Writing (2)

```
    ret = ncmpi_put_att_text(ncfile, NC_GLOBAL,
        "string", 13, buf);
    if (ret != NC_NOERR) return 1;
    ncmpi_enddef(ncfile);

    /* entered data mode – but nothing to do */

    ncmpi_close(ncfile);
    MPI_Finalize();
    return 0;
}
```

Storing value while in define mode as an attribute

Retrieving Data in PnetCDF

- Open a dataset in read-only mode (NC_NOWRITE)
- Obtain identifiers for dimensions
- Obtain identifiers for variables
- Read variable data
- Close the dataset

Simple PnetCDF: Reading (1)

```
#include <mpi.h>
#include <pnetcdf.h>
int main(int argc, char **argv)
{
        int ncfile, ret, count;
        char buf[13];
        MPI_Init(&argc, &argv);
        ret = ncmpi_open(MPI_COMM_WORLD, "myfile.nc",
            NC_NOWRITE, MPI_INFO_NULL, &ncfile);
        if (ret != NC_NOERR) return 1;

        /* continues on next slide */
```

Simple PnetCDF: Reading (2)

```
    /* verify attribute exists and is expected size */
    ret = ncmpi_inq_attlen(ncfile, NC_GLOBAL, "string", &count);
    if (ret != NC_NOERR || count != 13) return 1;

    /* retrieve stored attribute */
    ret = ncmpi_get_att_text(ncfile, NC_GLOBAL, "string", buf);
    if (ret != NC_NOERR) return 1;
    printf("%s", buf);

    ncmpi_close(ncfile);
    MPI_Finalize();
    return 0;
}
```

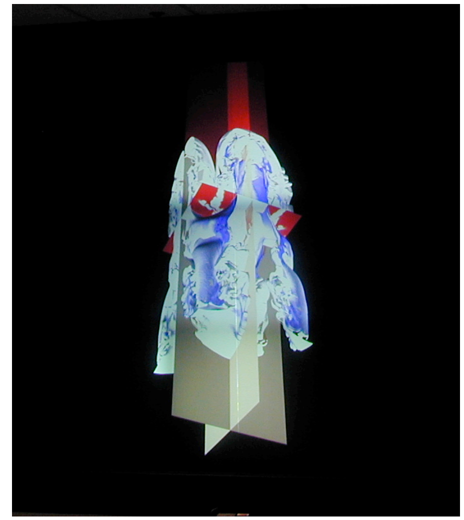
Compiling and Running

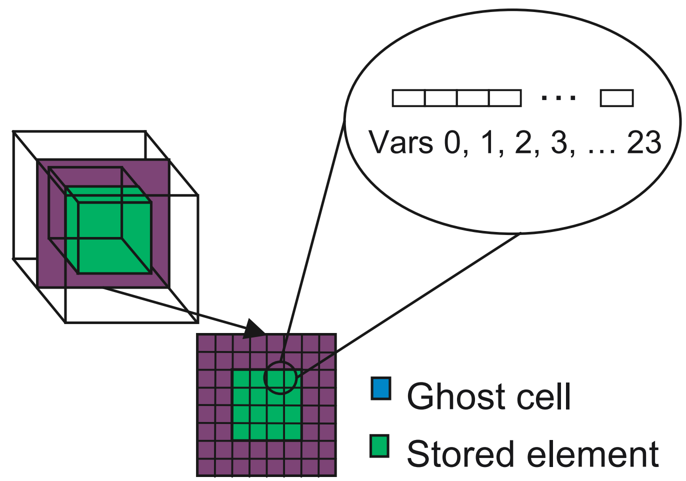
```
;mpicc pnetcdf-hello-write.c -I /usr/local/pnetcdf/include/ -L /usr/local/pnetcdf/lib
    -lpnetcdf -o pnetcdf-hello-write
;mpicc pnetcdf-hello-read.c -I /usr/local/pnetcdf/include/ -L /usr/local/pnetcdf/lib
    -lpnetcdf -o pnetcdf-hello-read
;mpiexec -n 1 pnetcdf-hello-write
;mpiexec -n 1 pnetcdf-hello-read
Hello World

;ls -l myfile.nc
-rw-r--r--    1 rross    rross       68 Mar 26 10:00 myfile.nc

;strings myfile.nc
string
Hello World
```

File size is 68 bytes; extra data (the header) in file.

Example: FLASH Astrophysics

- FLASH is an astrophysics code for studying events such as supernovae
  - Adaptive-mesh hydrodynamics
  - Scales to 1000s of processors
  - MPI for communication
- Frequently checkpoints:
  - Large blocks of typed variables from all processes
  - Portable format
  - Canonical ordering (different than in memory)
  - Skipping ghost cells





Example: FLASH with PnetCDF

- FLASH AMR structures do not map directly to netCDF multidimensional arrays
- Must create mapping of the in-memory FLASH data structures into a representation in netCDF multidimensional arrays
- Chose to
  - Place all checkpoint data in a single file
  - Impose a linear ordering on the AMR blocks
    - *Use 1D variables*
  - Store each FLASH variable in its own netCDF variable
    - *Skip ghost cells*
  - Record attributes describing run time, total blocks, etc.

Defining Dimensions

```
int status, ncid, dim_tot_blks, dim_nxb,
    dim_nyb, dim_nzb;
MPI_Info hints;
/* create dataset (file) */
status = ncmpi_create(MPI_COMM_WORLD, filename,
    NC_CLOBBER, hints, &file_id);
/* define dimensions */
status = ncmpi_def_dim(ncid, "dim_tot_blks",
    tot_blks, &dim_tot_blks);
status = ncmpi_def_dim(ncid, "dim_nxb",
    nzones_block[0], &dim_nxb);
status = ncmpi_def_dim(ncid, "dim_nyb",
    nzones_block[1], &dim_nyb);
status = ncmpi_def_dim(ncid, "dim_nzb",
    nzones_block[2], &dim_nzb);
```

Each dimension gets a unique reference

Creating Variables

```
int dims = 4, dimids[4];
int varids[NVARS];
/* define variables (X changes most quickly) */
dimids[0] = dim_tot_blks;
dimids[1] = dim_nzb;
dimids[2] = dim_nyb;
dimids[3] = dim_nxb;
for (i=0; i < NVARS; i++) {
      status = ncmpi_def_var(ncid, unk_label[i],
         NC_DOUBLE, dims, dimids, &varids[i]);
}
```

Same dimensions used for all variables

Storing Attributes

```
/* store attributes of checkpoint */
status = ncmpi_put_att_text(ncid, NC_GLOBAL, "file_creation_time",
    string_size, file_creation_time);
status = ncmpi_put_att_int(ncid, NC_GLOBAL, "total_blocks", NC_INT, 1,
    tot_blks);
status = ncmpi_enddef(file_id);

/* now in data mode … */
```

Writing Variables

```
double *unknowns; /* unknowns[blk][nzb][nyb][nxb] */
size_t start_4d[4], count_4d[4];
start_4d[0] = global_offset; /* different for each process */
start_4d[1] = start_4d[2] = start_4d[3] = 0;
count_4d[0] = local_blocks;
count_4d[1] = nzb;  count_4d[2] = nyb;  count_4d[3] = nxb;
for (i=0; i < NVARS; i++) {
        /* ... build datatype “mpi_type” describing values of a single variable ... */
        /* collectively write out all values of a single variable */
        ncmpi_put_vara_all(ncid, varids[i], start_4d, count_4d, unknowns, 1, mpi_type);
}
status = ncmpi_close(file_id);
```

Typical MPI buffer-count-type tuple

Inside PnetCDF Define Mode

- In define mode (collective)
  - Use MPI_File_open to create file at create time
  - Set hints as appropriate (more later)
  - Locally cache header information in memory
    - *All changes are made to local copies at each process*
- At ncmpi_enddef
  - Process 0 writes header with MPI_File_write_at
  - MPI_Bcast result to others
  - Everyone has header data in memory, understands placement of all variables
    - *No need for any additional header I/O during data mode!*

Inside PnetCDF Data Mode

- Inside ncmpi_put_vara_all (once per variable)
  - Each process performs data conversion into internal buffer
  - Uses MPI_File_set_view to define file region
    - *Contiguous region for each process in FLASH case*
  - MPI_File_write_all collectively writes data
- At ncmpi_close
  - MPI_File_close ensures data is written to storage

- MPI-IO performs optimizations
  - Two-phase possibly applied when writing variables

Tuning PnetCDF: Hints

- Uses MPI_Info, so identical to straight MPI-IO hin
- For example, turning off two-phase writes, in case you're doing large contiguous collective I/O on Lustre:

```
MPI_Info info;
MPI_File fh;
MPI_Info_create(&info);
MPI_Info_set(info, ”romio_cb_write", “disable”);
ncmpi_open(comm, filename, NC_NOWRITE, info, &ncfile);
MPI_Info_free(&info);
```

Wrapping Up:

- PnetCDF gives us
  - Simple, portable, self-describing container for data
  - Collective I/O
  - Data structures closely mapping to the variables described
- Easy – though not automatic – transition from serial NetCDF
- Datasets Interchangeable with serial NetCDF
- If PnetCDF meets application needs, it is likely to give good performance
  - Type conversion to portable format does add overhead
- Complimentary, not predatory
  - Research
  - Friendly, healthy competition

References

- PnetCDF
  http://www.mcs.anl.gov/parallel-netcdf/
  - Mailing list, SVN

- netCDF
  http://www.unidata.ucar.edu/packages/netcdf
- ROMIO MPI-IO
  http://www.mcs.anl.gov/romio/

- Shameless plug: Parallel-I/O tutorial at SC2007

Acknowledgements

This work is supported in part by U.S. Department of Energy Grant DE-FC02-01ER25506, by National Science Foundation Grants EIA-9986052, CCR-0204429, and CCR-0311542, and by the U.S. Department of Energy under Contract W-31-109-ENG-38.

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.