Date: 18 May, 2017	# M8.3 ACME Migration to DAOS

Government Purpose Rights

Prime Contract No.: DE-AC52-07NA27344
LLNL Subcontract No.: B613306
Subcontractor Name: Intel Federal LLC, on behalf of itself, its parent and AffiliatesSubcontractor Address: 4100 Monument Corner Dr, Ste 540, Fairfax, VA 22030

The Government's rights to use, modify, reproduce, release, perform, display, or disclose this technical data are restricted by the above agreement.

Limited Rights

Prime Contract No.: DE-AC52-07NA27344
LLNL Subcontract No.: B613306
Subcontractor Name: Intel Federal LLC, on behalf of itself, its parent and AffiliatesSubcontractor Address: 4100 Monument Corner Dr, Ste 540, Fairfax, VA 22030

The Government's rights to use, modify, reproduce, release, perform, display, or disclose this technical data are restricted by the above agreement.

NOTICES

Acknowledgment: This material is based upon work supported by Lawrence Livermore National Laboratory subcontract B613306.

USG Disclaimer: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Intel Disclaimer: Intel makes available this document and the information contained herein in furtherance of extreme-scale storage and I/O research and development. None of the information contained therein is, or should be construed, as advice. While Intel makes every effort to present accurate and reliable information, Intel does not guarantee the accuracy, completeness, efficacy, or timeliness of such information. Use of such information is voluntary, and reliance on it should only be undertaken after an independent review by qualified experts.

Access to this document is with the understanding that Intel is not engaged in rendering advice or other professional services. Information in this document may be changed or updated without notice by Intel.

This document contains copyright information, the terms of which must be observed and followed.

Reference herein to any specific commercial product, process or service does not constitute or imply endorsement, recommendation, or favoring by Intel or the US Government.

Intel makes no representations whatsoever about this document or the information contained herein.

IN NO EVENT SHALL INTEL BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES FOR ANY USE OF THIS DOCUMENT, INCLUDING, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, OR OTHERWISE, EVEN IF INTEL IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ACME Migration to DAOS

M. Scot Breitenfeld[1] and Neil Fortner[2]

1 brtnfld@hdfgroup.org, [2]nfortne2@hdfgroup.org

I.NetCDF Overview

NetCDF is a set of software libraries used to facilitate the creation, access, and sharing of array-oriented scientific data in self-describing, machine-independent data formats. The version of NetCDF for this project uses a DAOS version of HDF5 found in the *hdf5_daosm* branch at

https://bitbucket.hdfgroup.org/users/nfortne2/repos/hdf5_nf/browse.

This implementation of HDF5, in comparison to the inaugural IOD/DAOS version, removes access to the DAOS variables from the user (i.e. the read context identifier, the event stack identifier, the transaction identifier and version number) by handled them internally within the HDF5 library. Therefore, there is no longer a need for separate DOAS HDF5 APIs, which were distinguished by a *_ff* prefix in the original implementation of HDF5 for IOD/DAOS. This greatly simplifies porting an application which is already using the standard HDF5 library to work with DAOS. The DAOS HDF5 APIs used by NetCDF are,

- **H5Acreate**
- **H5Awrite**
- **H5Aopen**
- **H5Aclose**
- **H5Aiterate**
- **H5Aget_name**
- **H5Aget_space**
- **H5Aget_type**
- **H5Dcreate**
- **H5Dopen**
- **H5Dwrite**
- **H5Dread**
- **H5Fopen**
- **H5Fcreate**
- **H5Gopen**
- **H5Gcreate**
- **H5Gopen**
- **H5Gclose**
- **H5Literate**
- **H5Oget_info**
- **H5Oclose**
- **H5Topen**

Furthermore, this project uses a unique DAOS version of NetCDF, meaning a NetCDF version built using the DAOS HDF5 implementation mention above, found at

https://hdfgit.hdfgroup.org/scm/ffwd2/NetCDF-c.git.

In the initial implementation of DAOS NetCDF, the DAOS variables were exposed to the calling program using the alternate, DAOS specific, *_ff* prefix HDF5 APIs. Furthermore, in the initial implementation, the NetCDF APIs managed the transactions within NetCDF, from file creation to file closing. In the current NetCDF implementation, since DAOS control is

now shifted to within HDF5, the use of DAOS specific HDF5 APIs were replaced with standard HDF5 APIs, and the DAOS specific variables were for the most part removed from NetCDF.

However, the current DAOS version of NetCDF still contains a modified DAOS compatible schema. Namely, the original NetCDF implementation of dimensions used the HDF5 Dimension Scale API to add dimensions to variables (analogous to HDF5 datasets). This works well as a way to store dimensions with coordinate variables (i.e. variables used as a dimension scale for a dimension of the same name) in a way that is intuitive and self-describing for applications that access the file directly through HDF5. However, this approach introduces several dependencies between datasets and attributes that do not fit well with the DAOS transaction model. Also, it presents many special cases that must be handled, increasing the difficulty of implementation. Finally, there is currently no DAOS implementation of the Dimension Scale APIs and implementing them would be difficult due to the transaction model.

Since this project is not concerned about NetCDF/DAOS datasets (containers/files) being accessed independently of the NetCDF API, the existing NetCDF scheme was abandoned to simplify the implementation. All variables and dimensions are implemented as HDF5 datasets, all groups as HDF5 groups, and all attributes as HDF5 attributes. All dimensions have the string *DIM_* prepended to the name in HDF5, all variables have the string *VAR_* prepended, and all attributes have the string *ATT_* prepended. Variables have an HDF5 attribute *DIMENSION_LIST*, invisible to the NetCDF API, that stores references to the dimensions for the variable. Dimensions are implemented as a scalar dataset of type *H5T_STD_U64LE*, where the value indicates the dimension length, or all *1*s (i.e. *(uint64_t)(int64_t)-1*) to indicate an unlimited dimension. This implementation will avoid all name conflicts without having to add any special cases to the code and allows us to remove code paths for handling coordinate variables as a particular case, instead of treating them like any other variable.

II. ACME/PIO Overview

The end goal of implementing a DAOS version of NetCDF is to demonstrate an application which uses NetCDF. The NetCDF application identified for this project is software associated with the Accelerated Climate Modeling for Energy (ACME) program. Their software uses the package Parallel I/O (PIO) to perform I/O which, in turn, uses as its backend the NetCDF file format. PIO uses a large subset (211) of the NetCDF functions (not all of which need to be DAOS compatible), and a complete list of those functions is given in Appendix A.

The PIO performance testing program *pioperformance.F90* is an ACME I/O stand-alone driver program which closely duplicates the I/O pattern from an actual ACME application. Therefore, this project implemented the program *pioperformance.F90* within the DAOS framework via a DAOS version of PIO. *Pioperformance.F90* uses the following PIO functions,

- PIO_init
- PIO_Readdof
- **PIO_CreateFile**
- PIO_InitDecomp
- PIO_setframe
- **PIO_write_darray**
- **PIO_read_darray**
- PIO_freedecomp
- **PIO_OpenFile**
- **PIO_CloseFile**
- PIO_def_var
- PIO_def_dim
- PIO_def_att
- **PIO_enddef**

The PIO APIs in **blue** utilize DOAS compatible NetCDF APIs. Since no NetCDF APIs were changed as result of porting NetCDF to DAOS, the actual code modifications in PIO were minimal. For the most part, the changes simply included the addition of a call to initiating DAOS (*H5VLdaosm_init*) and a call for terminating DAOS (*H5VLdaosm_term*).

The DAOS PIO source files and test code can be downloaded from

https://hdfgit.hdfgroup.org/scm/ffwd2/parallelio.git

PIO expects as input from the application the partitioned data arrays for each process. Additionally, PIO has the option for requesting a subset of the CN that will perform the IO. Hence, PIO aggregates the IO from each process to only a subset of processes for IO. The IO processes then use NetCDF APIs to carry out the IO. PIO implements two methods for aggregating the IO from all the processes to the subset of IO processes. In the *box* method, each compute task will transfer data to one or more of the IO processes. For the *subset* method, each IO process is associated with a unique subset of computing processes for which each compute process transfers data to only one IO process [1]. In general, the *subset* method reduces the overall communication cost when compared to the *box* method. All the demonstrations use the *box* method.

Although PIO has the capability of using a subset of processes for IO, *H5VLdaosm_init* currently needs to be called by all the processes and not only those processes involved in IO. The automatic initialization of DAOS happens when the IO MPI sub-communicator is created in PIO, and it is finalized when this same sub-communicator is freed in PIO.

III Demonstration

The test program uses two input files; the first file contains namelist settings for the testing parameters and the second file contains the decomposition information from a PIO program (e.g. CESM, ACME). Decomposition files are available at,

https://svn-ccsm-piodecomps.cgd.ucar.edu/trunk.

The format of the decomposition file name is,

piodecomp<***NUM_MPI_PROCESSES***>tasks<***NUM_DIMENSIONS***>dims<***COUNTER***>.dat

where *NUM_MPI_TASKS* is the number of MPI tasks/ranks (30, 1024, 2048 and 16384 are available), *NUM_DIMENSIONS* is the number of dimensions in the decomposition (typically corresponding to the variable for which the decomposition was created), and *COUNTER* is a file identifier counter. [2]

III.a Overview of Demonstration

The test program *pioperformance.F90* reads namelist and then generates test data consisting of integers, 4-byte reals and 8-byte reals. It then writes the data using DAOS NetCDF via PIO APIs. It then reads the data back using DAOS PIO, checks for correctness, and outputs the data rate in reading and writing the data.

III.b Summary of the Demonstration on May 18th, 2017

CASE I – This demonstration is for 30 processes, it creates a **limited** dataset, and uses the following parameters:

Decomposition File	*Number of IO tasks*	*Number of variables*	*Number of frames*
piodecomp30tasks03dims07.dat	4	6	1
piodecomp30tasks02dims05.dat	8	6	1

Limited datasets have a fixed initial dataset size, and *unlimited* are extendible datasets and require chunking implemented in the HDF5/DAOS implementation. The milestone for this quarter includes implementing chunking for HDF5/DAOS but does not include a DAOS implementation of H5Sset_extent, which is needed for extending the dimensions of a chunked dataset, which is needed for multiple frames. Thus, this feature will not be demonstrated in PIO.

Procedure

(1) Starting a server

orterun -np 1 --report-uri ~/uri.txt daos_server -c 1

(2) Create pool

orterun -np 1 --ompi-server file:~/uri.txt dmg create

the pool id is stored in environment variable *pid*.

(3) Run PIO

orterun -x DD_MASK="all" -x pid="$pid" --ompi-server file:~/uri.txt -hostfile ~/.host_file -n 30 pioperf pioperf.nl

the input argument for *pioperf* is the namelist control file.

In the demonstration, both instances of the case I completed successfully and the correctness of each program was verified.

III.c Benchmarks for PIO on boro

As mentioned in Section I, this research uses two versions of NetCDF: (1) the "standard" version is the unmodified v4.4.1 of NetCDF available from Unidata and (2) the "DAOS" version, which is a modified v4.4.1 NetCDF as discussed in Section I. There are three combinations of PIO, NetCDF and HDF5 that were investigated, Figure 1,

1. PIO^{standard} – The PIO configuration uses standard HDF5 with the standard version of NetCDF.
2. PIO^{daos_disable} – The PIO configuration uses standard HDF5 with the DAOS version of NetCDF, where the DAOS capabilities in NetCDF are disabled.
3. PIO^{daos} – The PIO configuration used the DAOS version of HDF5 with the DAOS capabilities in NetCDF being enabled.

Thus, for both case 1 and 2, a POSIX file is getting written, via HDF5, to a Lustre backend.

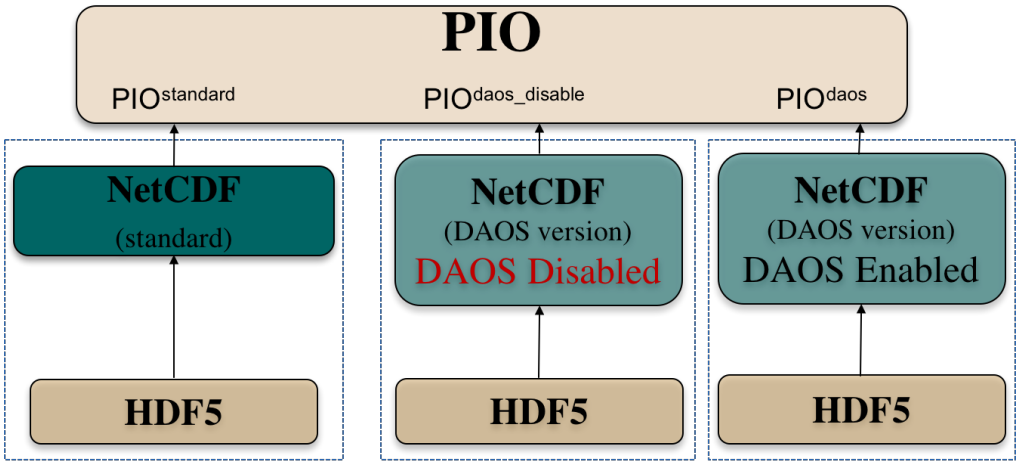

Figure 1. Three combinations of PIO, NetCDF, and HDF5.

All tests were made on Intel's *boro* cluster to compare PIO with Lustre and DAOS, and to verify the DAOS PIO implementation. The Lustre parameters were investigated for the 30 processes case, Figure 1, and accordingly, a stripe count of 1 with a stripe size of 2MB was chosen for all the Lustre benchmark results. The benchmarks were run ten times, and the average IO over all ten runs is plotted as a symbol in the subsequent benchmarking figures. The vertical line segment represents the minimum and maximum of IO over the ten runs.

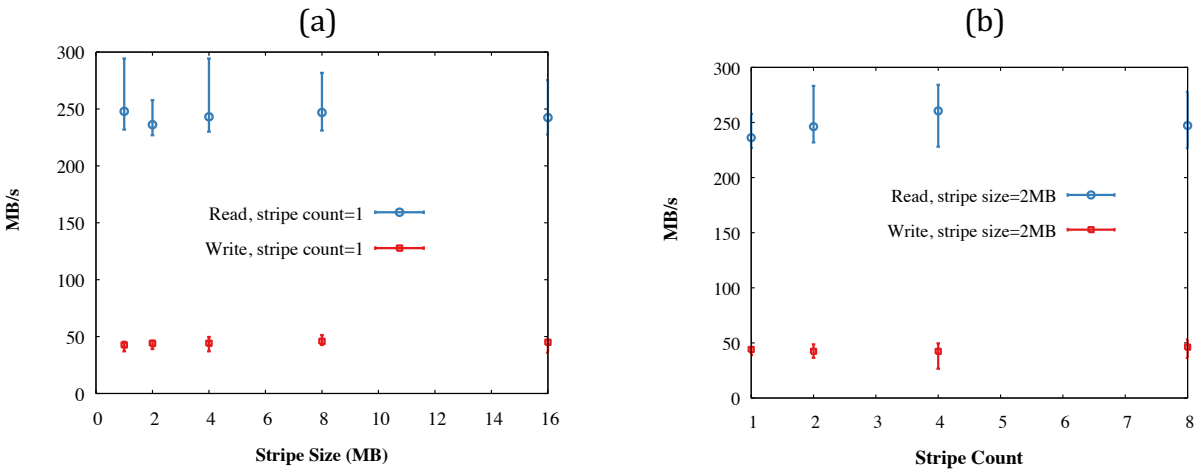

Figure 2. Lustre parameter Stripe Size (a) and Stripe Count (b) effects on reading and write performance.

The first benchmark was with 30 processor and using the decomposition file *piodecomp30tasks03dims07.dat*, Figure 3. The modified DAOS version of netCDF outperforms the standard for both reading and writing. Furthermore, the read performance is nearly twice as fast as the write performance. The results show a decline in performance as the number of IO tasks exceeds 20 processes. This decrease is because of the decomposition; if another decomposition file is used, *piodecomp30tasks03dims15.dat*, then the distinct drop at a certain number of processes is not present, Figure 4.

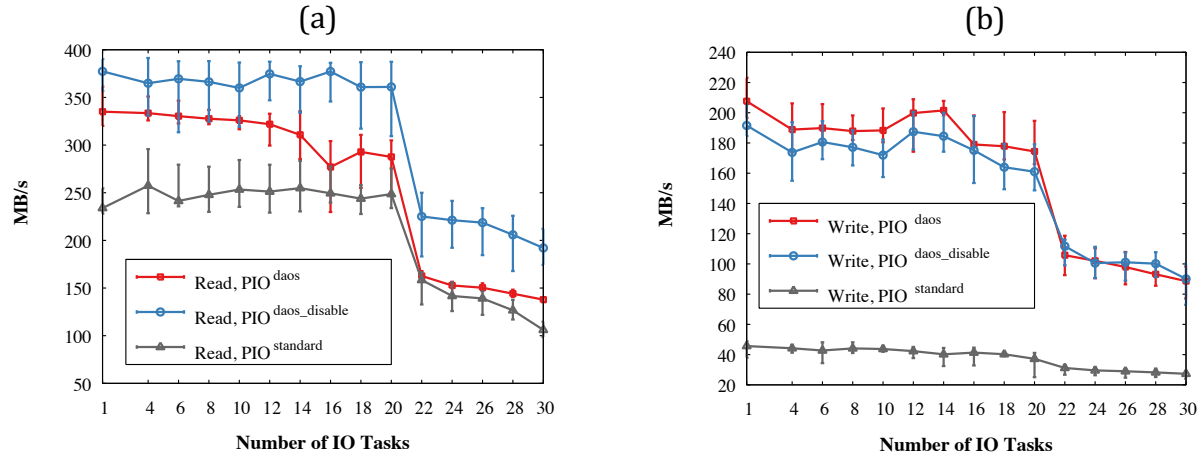

Figure 3. Read (a) and write (b) performance of 30 processes as a function of IO tasks, decomposition file *piodecomp30tasks03dims07.dat*.

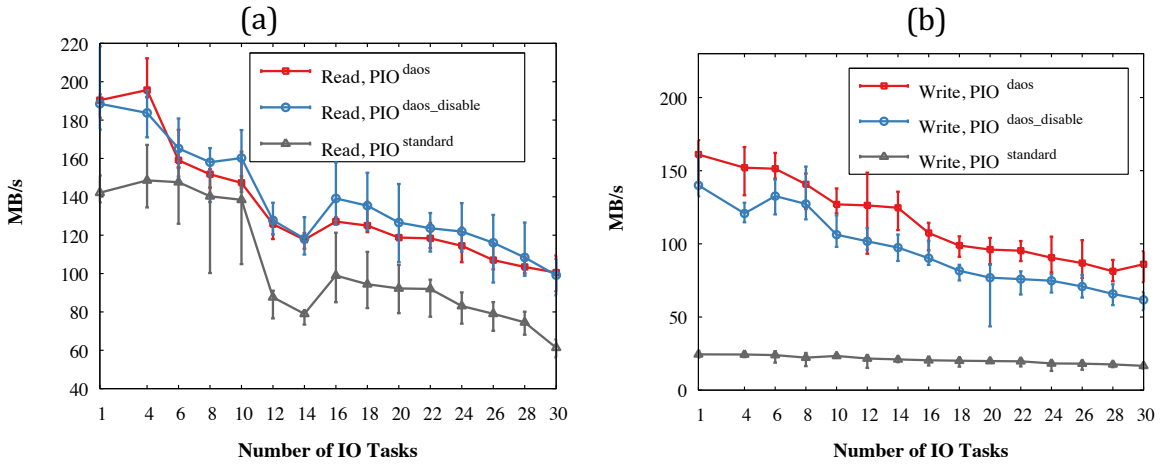

Figure 4 Read (a) and write (b) performance of 30 processes as a function of IO tasks, piodecomp30tasks03dims15.dat decomposition file.

The second benchmark was the 1024 processes and used the decomposition file, *piodecomp1024tasks03dims05.dat*. The results show a similar behavior as that found for the 30 processes case, but the DOAS implementation is on average faster than the Lustre implementation for reading, Figure 5a, and matches the Lustre performance for writing, Figure 5b.

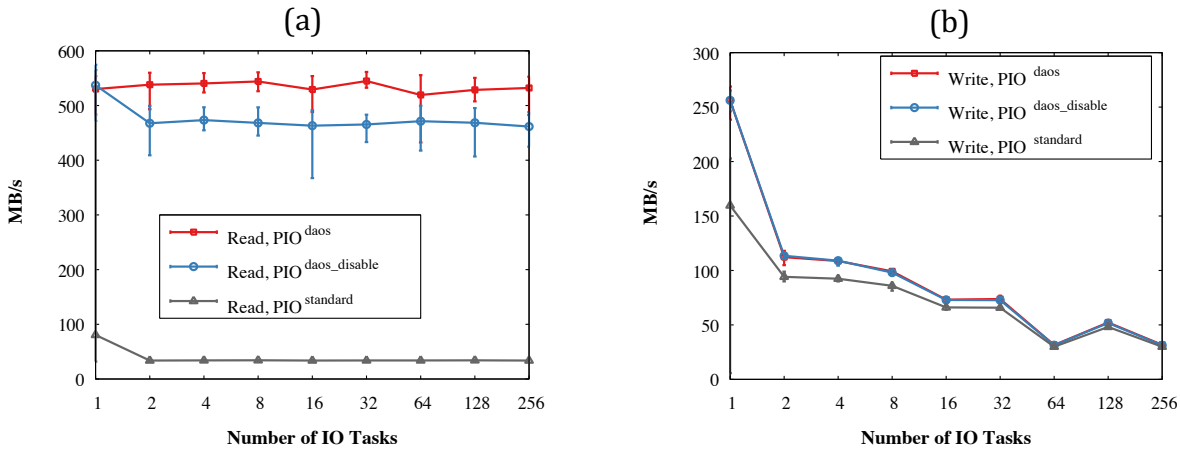

Figure 5. Read (a) and write (b) performance of 1024 processes as a function of IO tasks.

References

[1] http://ncar.github.io/ParallelIO/decomp.html

[2] https://groups.google.com/forum/#!topic/parallelio/vtvOXP-sjZE

Appendix A

List of NetCDF APIs called by PIO,

- nc_function
- nc_var_par_access
- nc_put_vara_double
- nc_put_vara_int
- nc_put_vara_float
- nc_get_vara_double
- nc_get_vara_int
- nc_get_vara_float
- nc_open
- nc_open_par
- nc_create_par
- nc_create
- nc_close
- nc_delete
- nc_sync
- nc_get_var1_schar
- nc_get_vars_ulonglong
- nc_get_varm_uchar
- nc_get_varm_schar
- nc_get_vars_short
- nc_get_var_double
- nc_get_var_int
- nc_get_var_ushort
- nc_get_vara_text
- nc_get_var1_float
- nc_get_var1_short
- nc_get_vars_int
- nc_get_var_text
- nc_get_varm_double
- nc_get_vars_schar
- nc_get_vara_ushort
- nc_get_var1_ushort
- nc_get_var_float
- nc_get_vars_uchar
- nc_get_var
- nc_get_var1_longlong
- nc_get_vars_ushort
- nc_get_var_long
- nc_get_var1_double
- nc_get_vara_uint
- nc_get_vars_longlong
- nc_get_var_longlong
- nc_get_vara_short
- nc_get_vara_long
- nc_get_var1_int
- nc_get_var1_ulonglong
- nc_get_var_uchar
- nc_get_vara_uchar
- nc_get_vars_float
- nc_get_vars_long
- nc_get_var1
- nc_get_var_uint
- nc_get_vara
- nc_get_vara_schar
- nc_get_var1_uint
- nc_get_vars_uint
- nc_get_varm_text
- nc_get_var1_text
- nc_get_varm_int
- nc_get_varm_uint
- nc_get_varm
- nc_get_vars_double
- nc_get_vara_longlong
- nc_get_var_ulonglong
- nc_get_vara_ulonglong
- nc_get_var_short
- nc_get_varm_float
- nc_get_var1_long
- nc_get_varm_long
- nc_get_varm_ushort
- nc_get_varm_longlong
- nc_get_vars_text
- nc_get_var1_uchar
- nc_get_vars
- nc_get_varm_short
- nc_get_varm_ulonglong
- nc_get_var_schar
- nc_inq
- nc_inq_dimname
- nc_put_att_short
- nc_rename_dim
- nc_get_att_double
- nc_set_fill
- nc_def_var
- nc_def_var_deflate
- nc_put_att_double
- nc_inq_dim
- nc_get_att_uchar
- nc_inq_var_fill
- nc_inq_attid
- nc_inq_vartype
- nc_put_att_schar
- nc_inq_vardimid
- nc_get_att_ushort
- nc_inq_varid
- nc_inq_attlen
- nc_inq_atttype
- nc_rename_var
- nc_inq_natts
- nc_put_att_ulonglong
- nc_inq_var
- nc_rename_att
- nc_put_att_ushort
- nc_inq_dimid
- nc_put_att_text
- nc_get_att_uint
- nc_inq_format
- nc_get_att_long
- nc_inq_attname
- nc_inq_att
- nc_put_att_long
- nc_inq_unlimdim
- nc_get_att_float
- nc_inq_ndims
- nc_put_att_int
- nc_inq_nvars
- nc_enddef
- nc_put_att_uchar
- nc_put_att_longlong
- nc_inq_varnatts
- nc_get_att_ubyte
- nc_get_att_text
- nc_del_att
- nc_inq_dimlen
- nc_get_att_schar
- nc_get_att_ulonglong
- nc_inq_varndims
- nc_inq_varname
- nc_def_dim
- nc_put_att_uint
- nc_get_att_short
- nc_redef
- nc_put_att_ubyte
- nc_get_att_int
- nc_get_att_longlong
- nc_put_att_float
- nc_inq_var_deflate
- nc_inq_var_szip

- nc_def_var_fletcher32
- nc_inq_var_fletcher32
- nc_def_var_chunking
- nc_inq_var_chunking
- nc_def_var_fill
- nc_def_var_endian
- nc_inq_var_endian
- nc_set_chunk_cache
- nc_get_chunk_cache
- nc_set_var_chunk_cache
- nc_get_var_chunk_cache
- nc_put_vars_uchar
- nc_put_vars_ushort
- nc_put_vars_ulonglong
- nc_put_varm
- nc_put_vars_uint
- nc_put_varm_uchar
- nc_put_var_ushort
- nc_put_var1_longlong
- nc_put_vara_uchar
- nc_put_varm_short
- nc_put_var1_long
- nc_put_vars_long
- nc_put_var_short
- nc_put_var1_ushort
- nc_put_vara_text
- nc_put_varm_text
- nc_put_varm_ushort
- nc_put_var_ulonglong
- nc_put_var_int
- nc_put_var_longlong
- nc_put_var_schar
- nc_put_var_uint
- nc_put_var
- nc_put_vara_ushort
- nc_put_vars_short
- nc_put_vara_uint
- nc_put_vara_schar
- nc_put_varm_ulonglong
- nc_put_var1_uchar
- nc_put_varm_int
- nc_put_vars_schar
- nc_put_var1
- nc_put_var1_float
- nc_put_varm_float
- nc_put_var1_text
- nc_put_vars_text
- nc_put_varm_long
- nc_put_vars_double
- nc_put_vara_longlong
- nc_put_var_double
- nc_put_var_float
- nc_put_var1_ulonglong
- nc_put_varm_uint
- nc_put_var1_uint
- nc_put_var1_int
- nc_put_vars_float
- nc_put_vara_short
- nc_put_var1_schar
- nc_put_vara_ulonglong
- nc_put_varm_double
- nc_put_vara
- nc_put_vara_long
- nc_put_var1_double
- nc_put_varm_schar
- nc_put_var_text
- nc_put_vars_int
- nc_put_var1_short
- nc_put_vars_longlong
- nc_put_vars
- nc_put_var_uchar
- nc_put_var_long
- nc_put_varm_longlong