

|  |  |
| --- | --- |
| **Date:**  **16 June 2016** | **M4.2 ACME Migration to FFIO Stack** |

**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.

i

**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.

ii

**ACME Migration to FFIO Stack**

M. Scot Breitenfeld1, Neil Fortner2

1 [brtnfld@hdfgroup.org](mailto:brtnfld@hdfgroup.org), 2[nfortne2@hdfgroup.org](mailto:nfortne2@hdfgroup.org)

## 

## Nomenclature

|  |  |
| --- | --- |
| BB | Burst Buffer |
| CN | Compute Node |
| DAOS | Distributed Application Object Storage |
| EFF | Exascale FastForward |
| ION | I/O Node |

## 

## I. NetCDF-4 Overview

[NetCDF](http://www.unidata.ucar.edu/software/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. This project uses the EFF version of netCDF, meaning a netCDF version built using the [EFF HDF5](https://svn.hdfgroup.uiuc.edu/hdf5/features/hdf5_ff/) implementation, found at

<https://hdfgit.hdfgroup.org/scm/ffwd2/netcdf-c.git>

## II. ACME/PIO Overview

The end goal of implementing an EFF version of netCDF is to demonstrate an application which uses netCDF. The netCDF application identified for this quarter is software associated with the [Accelerated Climate Modeling for Energy](http://climatemodeling.science.energy.gov/projects/accelerated-climate-modeling-energy) (ACME) program. Their software uses the package [Parallel I/O](http://ncar.github.io/ParallelIO/) (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 would need EFF versions), and complete list of those functions is given in Appendix A.

The PIO performance testing program [*pioperformance.F90*](https://hdfgit.hdfgroup.org/projects/FFWD2/repos/parallelio/browse/tests/performance/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 will implement the program *pioperformance.F90* within the EFF stack framework via an EFF version of PIO. *Pioperformance.F90* uses the following PIO functions,

1

* 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 EFF netCDF APIs and therefore have an EFF equivalent API. All new EFF PIO C APIs are indicated by appending a “*\_ff”* to the C function names. The Fortran EFF PIO APIs are implemented by overloading the current Fortran PIO APIs. For example, pio\_write\_darray would be,

/Users/brtnfld/Desktop/Screen Shot 2016-06-09 at 4.43.10 PM.png

where the last four arguments are optional EFF parameters. If these optional parameters are not present, then PIO well automatically default to the non-EFF netCDF APIs.

The EFF 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 uses 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 the *subset* method, each IO process is associated with a unique subset of compute 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.

Additionally, since PIO has the capability of using a subset of processes for IO, *EFF\_init* (an EFF HDF5 API used to start the EFF stack) was extended to handle a MPI sub-communicator group. Hence, in the current EFF PIO implementation, only those processes involved in IO will initialize the EFF stack. This initialization of the EFF stack happens automatically when the IO MPI sub-communicator is created in PIO and it is finalized when this same sub-communicator is freed in PIO.

### II.a. Updates to the EFF NetCDF schema

Additional NetCDF EFF APIs not implemented last quarter, but needed by PIO were:

* nc\_sync
* nc\_put\_vara\_double

2

* nc\_get\_vara\_double

Firstly, in the initial implementation of EFF NetCDF, the only EFF stack variables exposed to the calling program were the transaction and version number. Additionally, the EFF stack variables were globally accessible by the inclusion of *nc\_h5ff.h* in the application*.* In the current implementation, the global EFF stack variables are passed as arguments to the NetCDF and PIO APIs. The current NetCDF implementation allows access to all the EFF stack variables,

* Read context identifier,
* Version number,
* Event stack identifier,
* Transaction identifier

from PIO. Furthermore, the Fortran PIO APIs are overloaded by having the EFF stack variables being optional arguments, eliminating the need to have additional Fortran EFF PIO APIs.

Secondly, in the initial implementation, the NetCDF APIs managed the transactions internally, from creation to closing, with no control of the stack given to the calling program. In the current implementation, stack control is now shifted to within PIO. This allows for a PIO API to call multiple NetCDF APIs and to use the same or different transactions depending on which set of NetCDF calls are made within the PIO API. The transaction number is initialized in the application code and is then automatically incremented as needed within the EFF PIO APIs. Hence, transaction management is handle in the PIO library, but it is still accessible to the application.

Exposing the event stack identifier allows the application to use asynchronous I/O. To do this, the application creates an event stack and passes it to an operation that can be asynchronous. At the point in the future when the application needs that operation to be finished, it can use *H5ESwait* or *H5ESwait\_all* to block until it is completed. Even if no event stack is passed to some metadata operations, NetCDF will use an internal event stack to issue asynchronous operations that can run concurrently, improving performance. In this case, NetCDF will wait on all operations before returning.

Other improvements to NetCDF include:

* Support for unlimited dimensions (supported only with collective access, and only for the slowest changing dimension)
* "Links" from variables to their dimensions, allowing the variables to be queried about their dimensions
* Support for multiple server processes and nodes
* Fixed several bugs, including some that affect mainline NetCDF4 (these have been or will soon be forwarded to NetCDF4 developers)

3

## III Demonstration Example

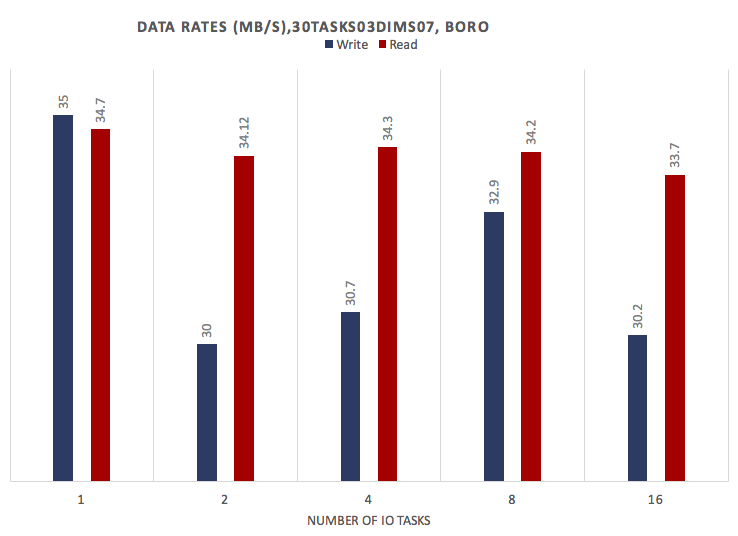
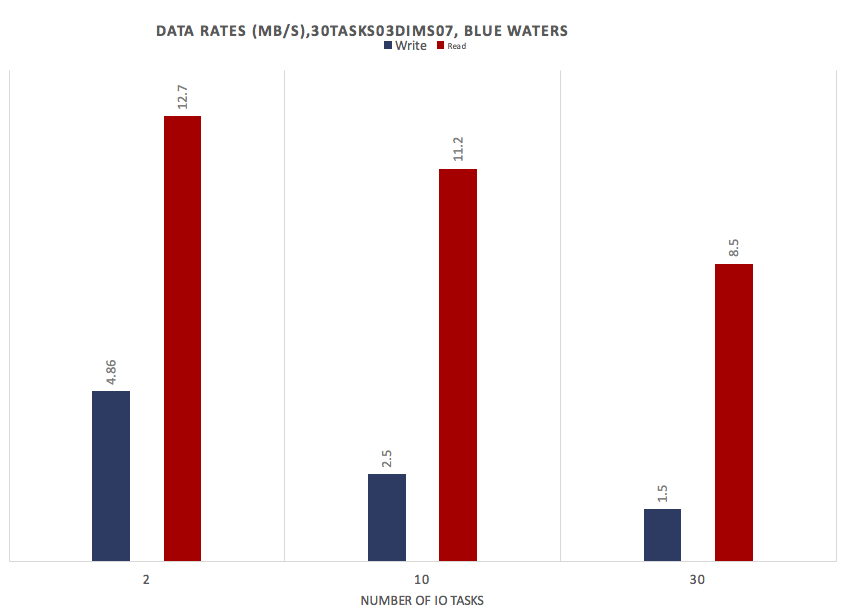
The test program uses two input files; the first file contains namelist settings for the testing parameters and the second file contains the decomposition information from a PIO program (e.g. CESM, ACME). Decomposition files are available at,

<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 Initial benchmarks for PIO on boro

Several testing runs were made on Intel’s *boro* cluster in order to verify the EFF PIO implementation, and a sampling of those test’s data rates are presented in Figures 1-4. Furthermore, for qualitative purposes, the performance of a standard implementation of PIO is also presented for 30 and 1024 processes runs on [Blue Waters](https://bluewaters.ncsa.illinois.edu/) at NCSA at the University of Illinois at Urbana-Champaign.

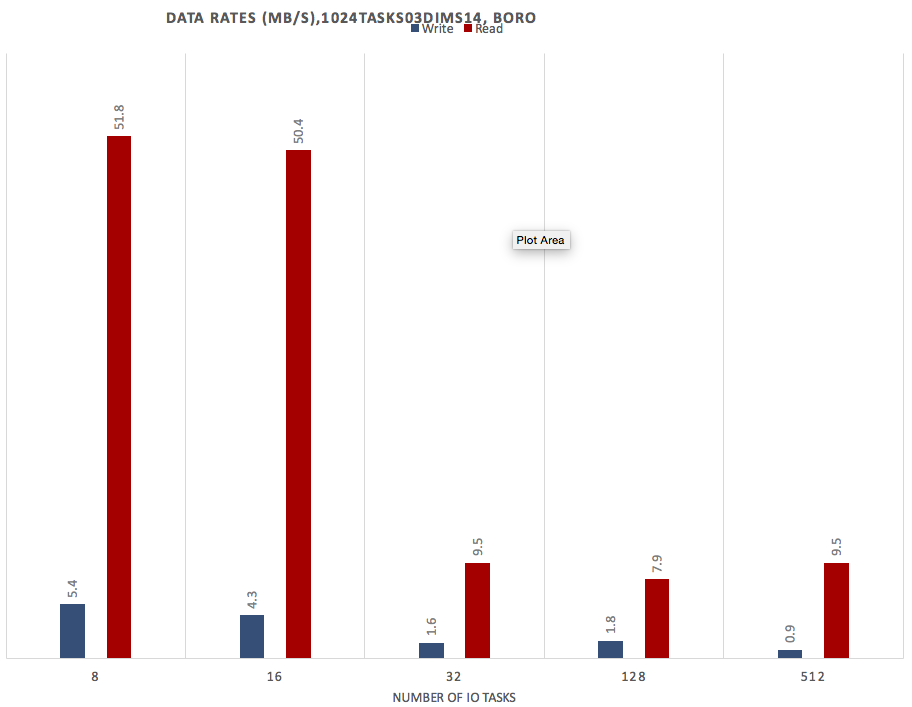
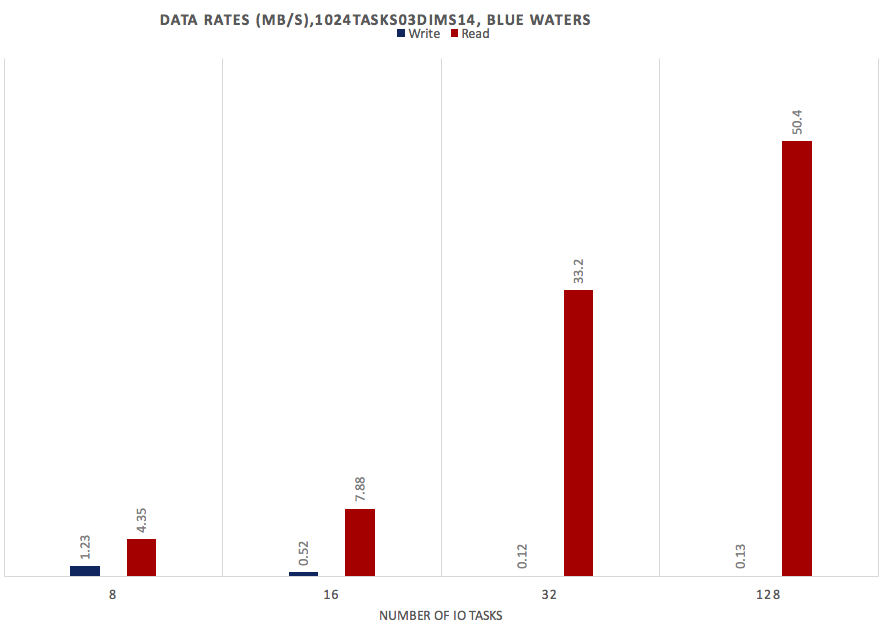
 

(a)

(b)

Figure 1 – 30 process data rates for various numbers of IO tasks for boro (a) and Blue Waters at NCSA (b).

4

(b)

(a)

Figure 2 – 1024 process data rates for various numbers of IO tasks for boro (a) and Blue Waters at NCSA (b).

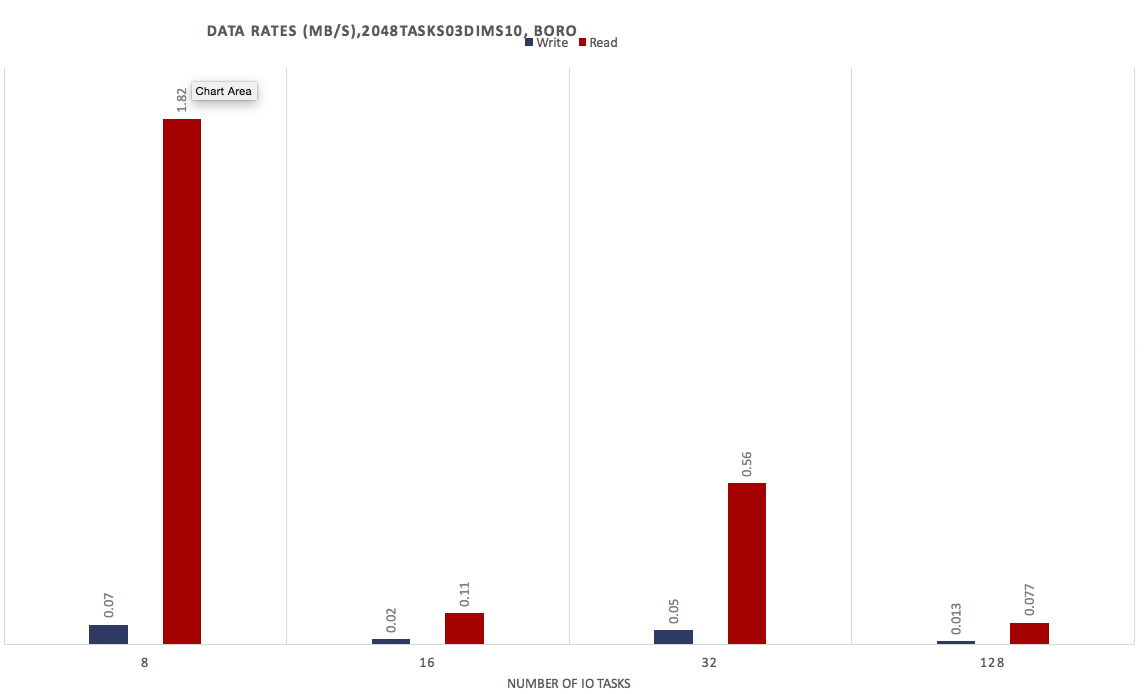


Figure 3 – 2048 process data rates for various numbers of IO tasks for boro.

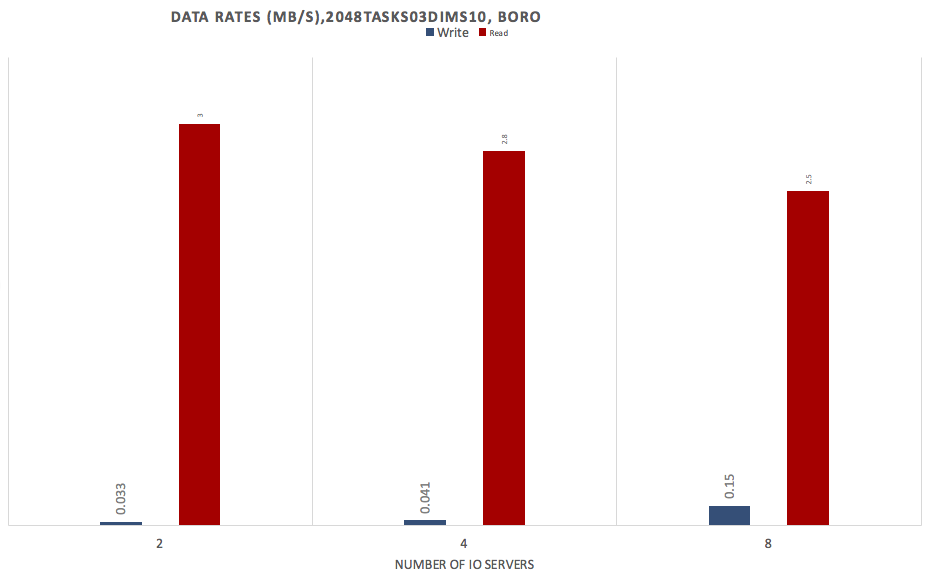


Figure 4 –Effect of the number of EFF servers on the data rate for 8 PIO IO tasks.

5

### III.b 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 EFF netCDF via PIO APIs. It then reads the data back using EFF PIO, checks for correctness, and outputs the data rate in reading and writing the data.

### III.c Summary of the Demonstration on June 16th, 2016

**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 | 8 | 3 | 1 |

**CASE II** – This demonstration is for 30 processes, creates an **unlimited** dataset, and uses the following parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| Decomposition File | Number of IO tasks | Number of variables | Number of frames |
| piodecomp30tasks03dims07.dat | 16 | 1 | 3 |

The steps for starting the servers were as follows,

(1) Clean-up any files created by previous runs:

**cleanup-all.sh; ssh boro-9 /scratch/ESSIO/opt/sbin/cleanup-all.sh;**

(2) Start two servers

**mpirun -n 2 --hostfile host\_srv ./h5ff\_server**

6

Once the servers are started, then the PIO application can be run,

**mpirun -n 30 --hostfile hosts -iface ib0 pioperf pioperf.nl.30.caseII**

where the “*hosts*” file is the list of nodes on boro to use, option -*iface* instructs MPI to use InfiniBand, and the input argument for *pioperf* is the namelist control file. In the demonstration, both cases I and II completed successfully and the correctness of each program was verified.

## 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

7

* 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

8