Using parallel NetCDF on Cray XC30 Systems

Klaus Ketelsen

1 Introduction

Most of the programs in environmental research use NetCDF as common data format.Without parallel NetCDF there are mainly two versions of doing I/O in a parallel environment.

1. Collecting and all data on PE0. PE0 writes the NetCDF file of the global model.
2. Every PE writes its subset of the global model into an individual NetCDF file. Postprocessing is required to combine the local files to a final global NetCDF file.

Moving to a high number of PEs, the first approach is not capable anymore, because

1. it creates a hotspot which prevents good parallelization.
2. it requires memory for least one global 3-D array. This may not be available on a single MPI process in case of large models.

The second approach show a good IO performance, but becomes very unhandy because of a high number of files ans the required postprocessing

An optimal solution would be the use of parallel NetCDF, where every PE writes its local portion of the model into one global output file. The following chapters suggest a setup for parallel NetCDF to achieve good I/O performance on Cray XC30 systems.

The current implementation of parallel NetCDF uses three library layers

1. NetCDF
2. HDF5
3. MPIIO

The use of the three layers is transparent to the user. Only NetCDF calls have to be issued from the application program.

2 Modules

To use parallel NetCDF, the following modules have to be loaded:

```
module load cray-hdf5-parallel
module load cray-netcdf-hdf5parallel
```

3 Environment Variables

The following environment variables enable the display of MPIIO setup and statistics to stdout.

```
export MPICH_MPIIO_HINTS_DISPLAY=1
export MPICH_MPIIO_STATS=1
```

With the environment variable MPICH_MPIIO_HINTS, the MPIIO can be tuned. For parallel NetCDF I/O, the striping factor should be set to 8 on HLRN system. Other tuning options are not required

```
export MPICH_MPIIO_HINTS=*.nc:striping_factor=8
```

*.nc has to match the names of the parallel NetCDF file

At this point of time, there is a bug in the batch system. The input parser reports an error while interpreting the striping_factor=8 setting of the MPICH_MPIIO_HINTS variable. The following workaround using an additional E8 environment variable enables the correct setting.

```
E8="=8"
export MPICH_MPIIO_HINTS=*.nc:striping_factor$E8
```

4 Opening NetCDF File

With the introduction of parallel NetCDF, the parameter list of NF90_CREATE and NF90_OPEN has changed. For MPIIO, the arguments *comm* and *info* were introduced. These two arguments are mandatory for parallel IO.

1. New File

   ```
   status = nf90_create(FILE_NAME, IOR(NF90_NETCDF4, NF90_MPIIO), ncid, &
             comm = MPI_COMM_WORLD, info = MPI_INFO_NULL)
   ```

   At least *cmode* NF90_NETCDF4 and NF90_MPIIO have to be set to enable parallel IO. Additional *cmode* flags are possible.

2. Existing File

   ```
   status = nf90_open(file_name, IOR(NF90_WRITE, NF90_MPIIO), ncid, &
             comm = MPI_COMM_WORLD, info = MPI_INFO_NULL)
   ```

   At least *cmode* NF90_MPIIO has to be set to enable parallel IO. Because nf90_open opens an existing file, the format NF90_NETCDF4 is already specified. NF90_WRITE is an example for an additional flag.

Because of MPIIO as the lowest layer of the IO, a MPI communicator has to be supplied in both calls. Only PEs which are part of this communicator can participate in the IO. NF90_CREATE and NF90_Open are collective calls.

5 Define Phase

The define phase of attributes and variables is similar to sequential NetCDF. The dimensions of the variables have to be declares with the global size of the model, although every PE later writes only the local part. All calls must be collective. Example (2-D domain decomposition):

```
status = nf90_def_dim(ncid, "x", NX_GLO, dimid(1))
status = nf90_def_dim(ncid, "y", NY_GLO, dimid(2))
status = nf90_def_dim(ncid, "z", NZ,     dimid(3))
status = nf90_def_dim(ncid, "t", NR_time_steps, dimid(4))
```

Two additional calls are required for parallel setup:

1. Fill Mode

   Per default, all NetCDF variables are preset with an initial values during creation phase. This causes a twofold output of the variables while writing a NetCDF file. There is a possibility to disable the prefilling. In this case the user is responsible for writing complete records, i.e. **every** element of the variable has to be written.

   The following call disables prefilling on variables base

```
status = NF90_DEF_VAR_FILL (ncid, varid(i), 1, 0 )
```

There is a NF90_SET_FILL call to disable the prefilling for the whole NetCDF file, but this call seems to have no effect in the current NETCDF/HDF5 implementation.

2. collective IO

   For better Performance, distributed arrays should be written collective. The following call enables collective IO on variable base.

   ```
   status = NF90_VAR_PAR_ACCESS (ncid, varid(i),NF90_COLLECTIVE)
   ```

   Variables set to NF90_INDEPENDENT or NF90_COLLECTIVE can be mixed within one NetCDF file.

Both routines have to be called before NF90_ENDDEF.

6 Writing data

At the begin of an NetCDF file, there are typically a couple 1D arrays, for example time axis of the coordinates in X, Y or Z. These variables should be set to NF90_INDEPENDENT and only written from PE0. Some models are updating the time variable during every output step. This update should also only be done on PE0.

The distributed variables are written collectively on every PE. The following example shows a typical setup for the output of one 3-D variable.

```
start = (/ index_x_local, index_y_local, 1, time /)
count = (/ nx_local, ny_local, nz, 1 /)
status = nf90_put_var(ncid, varid(i), data_3d, start = start, count = count)
```

The output *array data_3d* contains the local data and is dimensioned by local sizes, whereas start and count indicate the position of the data in the global 3-D array. This example describes a setup for 2-D domain decomposition in X and Y.

Typical for the NetCDF output in climate all ocean models is the declaration an unlimited time dimension. For reasons which are not obvious the I/O performance drops significantly if an unlimited time dimension is declared for a NetCDF variable. As a workaround it is suggested to declare the time dimension with a fixed value.
The disadvantage of this method is that the number of time steps has to be known in advance. The size of a NetCDF file is defined by the fixed dimensions specified in the declaration phase. Therefore supplying too high number of time steps results in an output file which is too big.

7 Performance

Setting up a NetCDF I/O with the methods described above an overall I/O performance of approximately 1 GB/sec can be measured. Please note that due to internal buffering a significant part of the I/O time can be observed during the NF90_CLOSE call.
It is important, that the elements in a collective output variables are exactly written once over all MPI processes. If not so, performance drops significantly!