NetCDF4 review

This memo is based on the incomplete reading of NetCDF4 alpha 3 release and NetCDF 4.0 alpha user's guide.

Summaries:
1) User's guide needs improvements. A separate section for NetCDF4 is suggested.
2) Parallel NetCDF code:
The current code reading seems to suggest that there is no misusing of HDF5 "collective calls only" APIs inside NetCDF4. Independent or Collective IO options are only allowed to be set for raw data IO. A bug is found for nc_get routines.
3) Several questions regarding the source code are addressed.

I. Documentation

1) User's guide is very confused. It spreads NetCDF4 material here and there. I strongly suggest that there is a separate section for NetCDF4 inside user's guide.
2) No words for the following topics about NetCDF4
   a. Potential Performance gain
   b. Compression filters
   c. Compound datatype,
   d. Group

With this document, general NetCDF3 users will find the inserted NetCDF4 contents redundant and potential NetCDF4 users will find the user's guide not helpful for them to make a decision why they need to choose NetCDF4.

I suggest putting part or even more contents in http://my.unidata.ucar.edu/content/software/netcdf/netcdf-4/index.html to the user's guide to give users a general idea whether they need to use NetCDF4 in their application.

II. Source codes

I cannot even configure NetCDF4 alpha 3 so far, so the following comments and questions are based on the incomplete code reading.

1) Group design and implementation

I don't quite grasp the group design and implementation inside NetCDF4.
I assume that restraints listed in http://my.unidata.ucar.edu/content/software/netcdf/netcdf-4/reqs.html#groups still valid for the current NetCDF4 implementation.

One group can have multiple children groups. There is a for-loop inside nc4grp.c to obtain group name, will this be enough to retrieve all sub-group names?
For (g = grp->children; g; g= g->next)

2) Unlimited dimension ID

Function: nc_inq_unlimdim will return the first unlimited dimension ID.

HDF can support multiple unlimited dimensions, What if users want to obtain more than one unlimited dimension ID.

3) Chunking-handling
Questions for comments inside nc4var.c for function nc_def_var. The comments are:

```
/* Chunking is required in any dataset with one or more unlimited
   dimension in HDF5. NetCDF-4 supports setting chunk parameters at
   variable creation with the following new function.

   Where chunksize is a pointer to an array of size ndims, with the
   chunksize in each dimension. If chunksizes is NULL, the user can
   select a chunking algorithm by setting chunkalg to NC_CHUNK_SEQ (to
   optimize for sequential access), NC_CHUNK_SUB (for chunk sizes set
   to favor equally subsetting in any dimension.

   When the (netcdf-3) function nc_def_var is used, a sequential
   chunking algorithm will be used. (Just as if the var had been created
   with NC_CHUNK_SEQ).

   The sequential chunking algorithm sets a chunksize of 1 all
   unlimited dimensions, and all other chunksizes to the size of that
   dimension, unless the resulting chunksize is greater than 250 KB,
   in which case subsequent dimensions will be set to 1 until the
   chunksize is less than 250 KB (one quarter of the default chunk
   cache size).

   The subsetting chunking algorithm sets the chunksize in each
   dimension to the nth root of (desired chunksize/product of n
   dimsizes).
*/
```

Q1) Does netCDF3 use chunking at all?

Q2) Why resulting chunksize is greater than 250 KB?

4) Parallel NetCDF4

From the current code reading, the implementation inside parallel NetCDF4 should not cause problems because of potential confliction of collective and independent calls. In parallel NetCDF4 data transfer property list (collective or independent) is ONLY set when doing raw data IO through HDF5.

A flag parallel_access has been used to set the property list. Function nc_var_par_access is used to set the flag parallel_access. Parallel_access is stored inside the struct NC_VAR_INFO_T.

```
typedef struct NC_VAR_INFO
{
   char name[NC_MAX_NAME + 1];
   int ndims;
   int dimids[NC_MAX_DIMS];
   int varid;
   int natts;
   struct NC_VAR_INFO *next;
   struct NC_VAR_INFO *prev;
   int dirty;
   int created;
   nc_type xtype;
   NC_ATT_INFO_T *att;
   void *fill_value;
   int chunkalg;
   int chunksizes[NC_MAX_DIMS];
   int parallel_access;
} NC_VAR_INFO_T;
```

It is not explicitly initialized inside NetCDF4. Given NC_INDEPENDENT =1 and NC_COLLECTIVE = 0 and on most platforms, a declared variable is to set to be 0 without explicitly initialization. So collective-IO is used by default for those NetCDF applications without using nc_var_par_access. Furthermore, NetCDF4 will keep the setting of data transfer property list set by the last nc_var_par_access according to the current implementation.

Bug:

Data transfer property list has only been set for nc_put_var(H5Dwrite). No data transfer property list is set for nc_get_var(H5Dread). This is a bug. That means, even applications use nc_var_par_access before nc_get_var to set either collective or independent, H5Dread will always use the default xfer_plistid(H5P_DEFAULT), that means Independent IO for reading. See the code at function pg_vara inside nc4hdf.c.

```
/* Write or read some data, an arrayfull at a time. */
int
pg_vara(NC_PG_T pg, int ncid, int varid, const size_t *startp,
      const size_t *countp, nc_type mem_nc_type, void *data)
{
   NC_GRP_INFO_T *grp, *g;
   NC_HDF5_FILE_INFO_T *h5;
   NC_VAR_INFO_T *var;
   NC_DIM_INFO_T *dim;

   hid_t datasetid = 0, file_spaceid = 0, mem_spaceid = 0, ds_typeid =
0;
```

```
hid_t mem_typeid = 0, xfer_plistid = H5P_DEFAULT;

......

......

/* Read/write this hyperslab into memory. */
if (pg == GET)
{
   LOG((5, "About to H5Dread some data..."));
   if (H5Dread(datasetid, mem_typeid, mem_spaceid,
           file_spaceid, xfer_plistid, bufr) < 0)
    BAIL(NC_EHDFERR);

   /* Eventually the block below will go away. Right now it's
    needed to support conversions between int/float, and range
    checking converted data in the netcdf way. These features are
    being added to HDF5 at the HDF5 World Hall of Coding right
    now, by a staff of thousands of programming gnomes. */
   if (need_to_convert)
   {
    if ((retval = convert_type(bufr, data, var->xtype, mem_nc_type,
                         len, &range_error, var->fill_value,
                         (h5->cmode & NC_STRICT_NC3))))
       BAIL(retval);

    /* For strict netcdf-3 rules, ignore erange errors between UBYTE
     * and BYTE types. */
    if ((h5->cmode & NC_STRICT_NC3) &&
        (var->xtype == NC_UBYTE || var->xtype == NC_BYTE) &&
        (mem_nc_type == NC_UBYTE || mem_nc_type == NC_BYTE) &&
        range_error)
       range_error = 0;
   }

   /* Now we need to fake up any further data that was asked for,
    using the fill values instead. First skip past the data we
    just read, if any. */
   if (!scalar)
   {
    void *filldata;
    int real_data_size = 0, fake_data_size = 0;
    hid_t clistid = 0;

    /* Get the fill value from the HDF5 variable. */
    if (!(fillvalue = malloc((size_t)nc4typelen(var->xtype))))
       BAIL(NC_ENOMEM);
    if ((clistid = H5Dget_create_plist(datasetid)) < 0)
       BAIL(NC_EHDFERR);
    if (H5Pget_fill_value(clistid, mem_typeid, fillvalue) < 0)
       BAIL(NC_EHDFERR);

    /* Copy the fill value into the rest of the data buffer. */
    filldata = (char *)data + real_data_size;
    for (i=0; i<fake_data_size; i++)
    {
       memcpy(filldata, fillvalue, (size_t)nc4typelen(var->xtype));
```

```
            filldata = (char *)filldata + 1;
            /*((char *)filldata)++;*/
         }
        }
    }
```

Others: dimensional data I/O is always using independent IO, that may need to be considered.

```
if (H5Dwrite(dim_info_id, type, mem_space, file_space,
H5P_DEFAULT, dim_info) < 0)
```