Portable Data Formats

Hierarchical Data Format - HDF5

17 March 2015 | Alexander Schnurpfeil
Outline

- Introduction
  - Motivation: Why bother with I/O efficiency
  - Structure of the HDF5 library
  - Terms and definitions

- Introduction to HDF5 - programming model and API
  - Creating/opening HDF5 files
  - Closing HDF5 files and other objects
  - HDF5 predefined datatypes
  - Creating dataspaces
  - Creating datasets
  - Writing/reading data
  - Row major / column major
  - Partial I/O
Outline

- Parallel HDF5
- Final example: Mandelbrot with HDF5
HDF5 - Introduction
Why do we need HDF5 and other I/O efficient libraries?

<table>
<thead>
<tr>
<th>ATOM</th>
<th>258</th>
<th>OG</th>
<th>SER</th>
<th>A</th>
<th>35</th>
<th>15.516</th>
<th>-5.241</th>
<th>-7.871</th>
<th>1.00</th>
<th>22.98</th>
<th>1HHEO</th>
<th>346</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATOM</td>
<td>259</td>
<td>N</td>
<td>PHE</td>
<td>A</td>
<td>36</td>
<td>11.426</td>
<td>-5.446</td>
<td>-9.595</td>
<td>1.00</td>
<td>24.24</td>
<td>1HHEO</td>
<td>347</td>
</tr>
<tr>
<td>ATOM</td>
<td>260</td>
<td>CA</td>
<td>PHE</td>
<td>A</td>
<td>36</td>
<td>9.991</td>
<td>-5.310</td>
<td>-9.822</td>
<td>1.00</td>
<td>30.22</td>
<td>1HHEO</td>
<td>348</td>
</tr>
<tr>
<td>ATOM</td>
<td>261</td>
<td>C</td>
<td>PHE</td>
<td>A</td>
<td>36</td>
<td>9.695</td>
<td>-5.307</td>
<td>-9.822</td>
<td>1.00</td>
<td>30.22</td>
<td>1HHEO</td>
<td>348</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ATOM</th>
<th>282</th>
<th>OG1</th>
<th>THR</th>
<th>A</th>
<th>38</th>
<th>3.758</th>
<th>5.866</th>
<th>-11.290</th>
<th>1.00</th>
<th>25.15</th>
<th>1HHEO</th>
<th>371</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATOM</td>
<td>283</td>
<td>CG2</td>
<td>THR</td>
<td>A</td>
<td>38</td>
<td>4.378</td>
<td>-3.154</td>
<td>-13.635</td>
<td>1.00</td>
<td>34.16</td>
<td>1HHEO</td>
<td>372</td>
</tr>
<tr>
<td>ATOM</td>
<td>284</td>
<td>N</td>
<td>THR</td>
<td>A</td>
<td>39</td>
<td>7.118</td>
<td>-3.174</td>
<td>-14.004</td>
<td>1.00</td>
<td>25.30</td>
<td>1HHEO</td>
<td>373</td>
</tr>
<tr>
<td>ATOM</td>
<td>285</td>
<td>CA</td>
<td>THR</td>
<td>A</td>
<td>39</td>
<td>7.377</td>
<td>-1.774</td>
<td>-15.471</td>
<td>1.00</td>
<td>20.71</td>
<td>1HHEO</td>
<td>374</td>
</tr>
<tr>
<td>ATOM</td>
<td>286</td>
<td>C</td>
<td>THR</td>
<td>A</td>
<td>39</td>
<td>7.820</td>
<td>-1.696</td>
<td>-16.134</td>
<td>1.00</td>
<td>21.18</td>
<td>1HHEO</td>
<td>375</td>
</tr>
<tr>
<td>ATOM</td>
<td>287</td>
<td>O</td>
<td>THR</td>
<td>A</td>
<td>39</td>
<td>7.578</td>
<td>-6.666</td>
<td>-13.035</td>
<td>1.00</td>
<td>20.37</td>
<td>1HHEO</td>
<td>376</td>
</tr>
<tr>
<td>ATOM</td>
<td>288</td>
<td>CB</td>
<td>THR</td>
<td>A</td>
<td>39</td>
<td>8.481</td>
<td>-1.324</td>
<td>-13.554</td>
<td>1.00</td>
<td>18.96</td>
<td>1HHEO</td>
<td>377</td>
</tr>
<tr>
<td>ATOM</td>
<td>289</td>
<td>OG1</td>
<td>THR</td>
<td>A</td>
<td>39</td>
<td>9.595</td>
<td>-2.164</td>
<td>-13.310</td>
<td>1.00</td>
<td>18.96</td>
<td>1HHEO</td>
<td>377</td>
</tr>
<tr>
<td>ATOM</td>
<td>290</td>
<td>CG2</td>
<td>THR</td>
<td>A</td>
<td>39</td>
<td>8.071</td>
<td>-1.511</td>
<td>-11.554</td>
<td>1.00</td>
<td>9.05</td>
<td>1HHEO</td>
<td>378</td>
</tr>
<tr>
<td>ATOM</td>
<td>291</td>
<td>N</td>
<td>LYS</td>
<td>A</td>
<td>40</td>
<td>8.460</td>
<td>-2.806</td>
<td>-15.928</td>
<td>1.00</td>
<td>16.83</td>
<td>1HHEO</td>
<td>379</td>
</tr>
<tr>
<td>ATOM</td>
<td>292</td>
<td>CA</td>
<td>LYS</td>
<td>A</td>
<td>40</td>
<td>8.967</td>
<td>-2.910</td>
<td>-17.316</td>
<td>1.00</td>
<td>13.00</td>
<td>1HHEO</td>
<td>380</td>
</tr>
</tbody>
</table>
What do we want to do with our data?

We want to …

- Write it,
- Read it,
- Describe it,
- Transfer it,
- Structure it,

… in an efficient way !!!
What is HDF5?

- API, data model and file format for I/O management
- Tools suite for accessing data in HDF5 format
HDF5 - Features

- Supports parallel I/O
- Self describing data model which allows the management of complex data sets
- Portable file format
- Available on a variety of platforms
- Supports C, C++, Fortran 90 and Java
  - Pythonic interfaces also available
- Provides tools to operate on HDF5 files and data
Layers of the HDF5 Library

1. Application
2. Object API (C, F90, C++, Java)
3. Library Internals
   - Virtual File I/O
   - MPI I/O
   - Custom
   - stdio
   - split files

Storage device
HDF5 - File organization

- HDF5 file structure corresponds in many respects to a Unix/Linux file system (fs)

<table>
<thead>
<tr>
<th>HDF5</th>
<th>Unix/Linux fs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Directory</td>
</tr>
<tr>
<td>Data set</td>
<td>File</td>
</tr>
</tbody>
</table>

/DataSet1
/Group1/DataSet2
/Group1/Group3/DataSet5
/Group1/Group3/DataSet6
/Group2/DataSet3
/Group2/DataSet4

```
```

HDF5 File
**HDF5 - Terminology**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File</strong></td>
<td>Container for storing data</td>
</tr>
<tr>
<td><strong>Group</strong></td>
<td>Structure which may contain HDF5 objects, e.g. datasets, attributes, datasets</td>
</tr>
<tr>
<td><strong>Attribute</strong></td>
<td>Can be used to describe datasets and being attached to them</td>
</tr>
<tr>
<td><strong>Dataspace</strong></td>
<td>Describes the dimensionality of the data array and the shape of the data points respectively, i.e. it describes the shape of a dataset</td>
</tr>
<tr>
<td><strong>Dataset</strong></td>
<td>Multi-dimensional array of data elements</td>
</tr>
</tbody>
</table>
HDF5 library specific types - C

<table>
<thead>
<tr>
<th>HDF5 defined types</th>
</tr>
</thead>
<tbody>
<tr>
<td>hid_t</td>
</tr>
<tr>
<td>herr_t</td>
</tr>
<tr>
<td>hsize_t</td>
</tr>
<tr>
<td>hssize_t</td>
</tr>
<tr>
<td>hvl_t</td>
</tr>
</tbody>
</table>

- Defined types are integers of different size
- Own defined types ensure portability
- Include `hdf5.h` in your C application
HDF5 library specific types – F90

**HDF5 defined types**

```fortran
INTEGER(HID_T) Object identifier
INTEGER(HSIZE_T) Used for dimensions
INTEGER(HSSIZE_T) Used for coordinates and dimensions
```

- Defined types are integers of different size
- Own defined types ensure portability
- Include `use hdf5.h` in your Fortran application
- The HDF5 library interface needs to be initialized by calling `h5open_f` before it can be used in your code and closed (`h5close_f`) at the end.

```fortran
h5open_f(status) / h5close_f(status)
```

```fortran
INTEGER, INTENT(OUT) :: status
```

Returns 0 if successful and -1 if fails.
HDF5 – API
HDF5 API naming scheme (excerpt)

- **H5**
  - Library functions: general-purpose functions
- **H5D**
  - Dataset interface: dataset access and manipulation routines
- **H5F**
  - File interface: file access routines
- **H5P**
  - Property list interface: object property list manipulation routines
- **H5S**
  - Dataspace interface: dataspace definition and access routines
General Procedure

1. **H5Fcreate**, **H5Fopen**
2. **H5Screate_simple**
3. **H5Dcreate**, **H5Dopen**
4. **H5Dread**, **H5Dwrite**
5. **H5Dclose**
6. **H5Sclose**
7. **H5Fclose**

**create/close Dataset**

**create/close HDF5 File**
Creating an HDF5 file in C

```c
hid_t H5Fcreate( const char *name, unsigned access_flag,
                 hid_t creation_prp, hid_t access_prp )
```

- **const char * name**  IN: Name of the file
- **unsigned int access_flag** IN: File access flag
- **hid_t creation_prp** IN: File creation property list identifier, H5P_DEFAULT specifies default file creation properties
- **hid_t access_prp** IN: File access property list identifier, H5P_DEFAULT specifies default file access properties

- Avoid calling `H5Fcreate` with already opened files as it might damage your data
- `H5Fcreate` returns a file identifier on success, but a negative value on failure
Opening an existing HDF5 file in C

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hid_t H5Fopen</td>
<td>Primary function for accessing existing HDF5 files</td>
</tr>
<tr>
<td>const char * name</td>
<td>IN: Name of the file</td>
</tr>
<tr>
<td>unsigned int access_flag</td>
<td>IN: File access flag</td>
</tr>
<tr>
<td>hid_t access_prp</td>
<td>IN: File access property list identifier, H5P_DEFAULT specifies default file access properties</td>
</tr>
</tbody>
</table>

- H5Fopen returns a file identifier for a successfully opened file, otherwise a negative value is returned.
- Avoid multiple opens of the same file.
Access modes for HDF5 files in C codes

<table>
<thead>
<tr>
<th>HDF5 file access modes in C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H5F_ACC_TRUNC</td>
<td>Create a new file, overwrites an existing file</td>
</tr>
<tr>
<td>H5F_ACC_EXCL</td>
<td>Create a new file, H5Fcreate fails if file already exists</td>
</tr>
<tr>
<td>H5F_ACC_RDWR</td>
<td>Open file in read-write mode, irrelevant for H5Fcreate</td>
</tr>
<tr>
<td>H5F_ACC_RDONLY</td>
<td>Open file in read-only mode, irrelevant for H5Fcreate</td>
</tr>
</tbody>
</table>

- More specific settings are controlled through file creation property list (`fcpl`) and file access property lists (`fapl`) which defaults to H5P_DEFAULT
- `fcpl` controls file metadata
- `fapl` controls different methods of performing I/O on files
Creating an HDF5 file in F90

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>h5fcreate_f</code></td>
<td>Creates a new HDF5 file. &lt;br&gt;<strong>Arguments:</strong>&lt;br&gt;name, access_flags, file_id, hdferr, creation_prp, access_prp.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the file.</td>
</tr>
<tr>
<td>access_flags</td>
<td>File access flag.</td>
</tr>
<tr>
<td>file_id</td>
<td>File identifier.</td>
</tr>
<tr>
<td>hdferr</td>
<td>Error code, 0 on success, -1 on failure.</td>
</tr>
<tr>
<td>creation_prp</td>
<td>File creation property list, H5P_DEFAULT_F if not specified.</td>
</tr>
<tr>
<td>access_prp</td>
<td>File access property list, H5P_DEFAULT_F if not specified.</td>
</tr>
</tbody>
</table>

- Avoid calling `h5fcreate_f` with already opened files as it might damage your data.
# Opening an existing HDF5 file in F90

The `h5fopen_f` function is the primary function for accessing existing HDF5 files.

- **Avoid multiple opens of the same file**

```fortran
h5fopen_f(name, access_flags, file_id, hdferr, access_prp)
```

- **Name of the file**: `CHARACTER(LEN=*)`, INTENT(IN)
- **File access flag**: `INTEGER`, INTENT(IN)
- **File identifier**: `INTEGER(HID_T), INTENT(OUT)`
- **Error code**: `INTEGER`, INTENT(OUT)
- **File access property list**: `INTEGER(HID_T), OPTIONAL, INTENT(IN)`, H5P_DEFAULT_F if not specified

- **Primary function for accessing existing HDF5 files**
- **Avoid multiple opens of the same file**
## Access modes for HDF5 files in F90 codes

<table>
<thead>
<tr>
<th>Fortran</th>
<th>HDF5 file access modes in Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H5F_ACC_TRUNC_F Create a new file, overwrites an existing file</td>
</tr>
<tr>
<td></td>
<td>H5F_ACC_EXCL_F Create a new file, H5Fcreate fails if file already exists</td>
</tr>
<tr>
<td></td>
<td>H5F_ACC_RDWR_F Open file in read-write mode, irrelevant for h5fcreate_f</td>
</tr>
<tr>
<td></td>
<td>H5F_ACC_RDONLY_F Open file in read-only mode, irrelevant for h5fcreate_f</td>
</tr>
</tbody>
</table>

- More specific settings are controlled through file creation property list (`fcpl`) and file access property lists (`fapl`) which defaults to H5P_DEFAULT_F
  - `fcpl` controls file metadata
  - `fapl` controls different methods of performing I/O on files
# Closing an HDF5 file

C

```c
herr_t H5Fclose( hid_t file_id )
```

**hid_t**  
**file_id**  
IN: Identifier of a file to terminate access to

Retrurns a non-negative value if successful; otherwise a negative value

Fortran

```fortran
h5fclose_f(file_id, hdferr)
```

**INTEGER(HID_T), INTENT(IN)**  
:: **file_id**  
File identifier

**INTEGER, INTENT(OUT)**  
:: **hdferr**  
Error code, 0 on success and -1 on failure
How to compile and link on JUROPA

- Use the `h5dump` command-line tool to view HDF5 files
  - `h5dump your_hdf5_file.h5`
  - `h5dump -h` for usage message

### Compile & Link

**C**
```
mpicc program.c -I$HDF5_INCLUDE -L$HDF5_LIB -lhdf5 -o program
```

**Fortran**
```
mpif90 program.f90 -I$HDF5_INCLUDE -L$HDF5_LIB -lhdf5_fortran -o program
```

module load hdf5/1.8.13 (on the login nodes and the compute nodes as well)
Exercise

1.1 Create/close an HDF5 file

• Write a program in C or F90 which creates and closes an HDF5 file.
Names for files: "empty-hdf5-file-c.h5" (C), "empty-hdf5-file-f.h5" (F90)

1.2 Use h5dump to view the content of the priorly created file. What do you see?
Creating an HDF5 file in C

```c
#include "hdf5.h"
int main() {
  char filename[] = "empty-hdf5-file-c.h5";
  // Create HDF5 file
  hid_t file = H5Fcreate(filename, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
  // Close object
  H5Fclose(file);
  return 0;
}
```

HDF5 "empty-hdf5-file-c.h5" {
GROUP "/" {
}
}
Creating an HDF5 file in F90

Fortran - Example 1

PROGRAM CREATEFILE
  USE HDF5 ! Has always to be included for HDF5 usage in fortran
  IMPLICIT NONE
  CHARACTER(LEN=20), PARAMETER :: filename = "empty-hdf5-file-f.h5"
  INTEGER(HID_T) :: file_id
  INTEGER :: status

  ! Initialize fortran interface
  CALL h5open_f(status)

  ! Create an empty HDF5 file
  CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id, status)

  ! Close previously opened file
  CALL h5fclose_f(file_id, status)

  ! Shut down fortran interface.
  CALL h5close_f(status)
END PROGRAM CREATEFILE

HDF5 "empty-hdf5-file-f.h5" {
  GROUP "/" {
  }
}

Group "/"
HDF5 pre-defined datatypes (excerpt)

<table>
<thead>
<tr>
<th>C type</th>
<th>HDF5 file type (pre-defined)</th>
<th>HDF5 memory type (native)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>H5T_STD_I32BE, H5T_STD_I32LE</td>
<td>H5T_NATIVE_INT</td>
</tr>
<tr>
<td>float</td>
<td>H5T_IEEE_F32BE, H5T_IEEE_F32LE</td>
<td>H5T_NATIVE_FLOAT</td>
</tr>
<tr>
<td>double</td>
<td>H5T_IEEE_F64BE, H5T_IEEE_F64LE</td>
<td>H5T_NATIVE_DOUBLE</td>
</tr>
<tr>
<td>integer</td>
<td>H5T_STD_I32(8,16)BE</td>
<td>H5T_NATIVE_INTEGER</td>
</tr>
<tr>
<td>real</td>
<td>H5T_IEEE_F32BE, H5T_IEEE_F32LE</td>
<td>H5T_NATIVE_REAL</td>
</tr>
</tbody>
</table>

- Native datatype might differ from platform to platform
- HDF5 file type depends on compiler switches and underlying platform
- Native datatypes are not in an HDF file but the pre-defined ones which are referred by native datatypes appear in the HDF5 files.
Dataspace

- The dataspace is part of the metadata of the underlying dataset
- Metadata are:
  - Dataspase
  - Datatype
  - Attributes
  - Storage info
- The dataspace describes the size and shape of the dataset

### Simple dataspace

| rank: int |
| current_size: hsize_t[rank] |
| maximum_size: hsize_t[rank] |

rank = 2, dimensions = 2x5
Creating a dataspace in C

```c
hid_t H5Screate_simple( int rank, const hsize_t * current_dims, const hsize_t * maximum_dims )
```

- `maxmum_dims` may be `NULL`. Then `maximum_dims` and `current_dims` are the same.
- `H5S_UNLIMITED` can be used to set dimensions to “infinite” size.
- Returns a dataspace identifier if successful; otherwise returns a negative value.
Creating a dataspace in F90

```fortran
h5screate_simple_f(rank, dims, space_id, hdferr, maxdims)
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER, INTENT(IN)</td>
<td>:: rank</td>
<td>Number of dataspace dimensions</td>
</tr>
<tr>
<td>INTEGER(HSIZE_T), INTENT(IN)</td>
<td>:: dims(*)</td>
<td>Array with current dimension sizes</td>
</tr>
<tr>
<td>INTEGER(HID_T), INTENT(OUT)</td>
<td>:: space_id</td>
<td>Dataspace identifier</td>
</tr>
<tr>
<td>INTEGER, INTENT(OUT)</td>
<td>:: hdferr</td>
<td>Error code 0 on success and -1 on failure</td>
</tr>
<tr>
<td>INTEGER(HSIZE_T), OPTIONAL, INTENT(IN)</td>
<td>:: maxdims(*)</td>
<td>Array with the maximum dimension sizes</td>
</tr>
</tbody>
</table>
Dataset (metadata + dataset)

Data

Metadata

Dataspace
- rank = 3
- dim[0] = 2
- dim[1] = 4
- dim[2] = 3

Datatype
- Integer

Attributes
- Time = 2.1
- Temp = 122

Storage
- Contiguous
Creating a dataset in C

- Creates a new dataset and links it into the file
- `H5Dcreate2` is a macro mapped to `H5Dcreate2` in HDF5 1.8.0 and newer versions
- `H5P_DEFAULT` for `lcpl_id`, `dcpl_id`, `dapl_id`
- Returns a dataset identifier if successful; otherwise returns a negative value
Creating a dataset in F90

```fortran
h5dcreate_f(loc_id, name, type_id, space_id, dset_id, hdferr, dcpl_id, lcpl_id, dapl_id)
```

- **loc_id**: File or group identifier
- **name**: Name of the dataset
- **type_id**: Datatype identifier
- **space_id**: Dataspace identifier
- **dset_id**: Dataset identifier
- **hdferr**: Error code 0 on success and -1 on failure
- **dcpl_id**: Dataset creation property list
- **lcpl_id**: Link creation property list
- **dapl_id**: Dataset access property list

- Creates a new dataset and links it into the file
Recipe: Creating an empty dataset

1. Get identifier for dataset location
2. Specify datatype (integer, composite etc.)
3. Define dataspace
4. Specify property lists (or H5P_DEFAULT, H5P_DEFAULT_F)
5. Create dataset
6. Close all opened objects
Exercise

2.1 Create an empty dataset
• Write a program in C or F90 which creates an HDF5 file named h5file-c.h5 (C), h5file-fortran (F90) which includes an empty dataset for integers. The dataset should be a two-dimensional array of size 2x4. The name of the dataset should be "dset"

2.2 Use h5dump to view the content of the priorly created file. What do you see?
Creating an empty dataset in C

<table>
<thead>
<tr>
<th>C - Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>// File</td>
</tr>
<tr>
<td>file_id = H5Fcreate(filename, H5F_ACC_TRUNC,H5P_DEFAULT,H5P_DEFAULT);</td>
</tr>
<tr>
<td>// Dataspace</td>
</tr>
<tr>
<td>int rank = 2;</td>
</tr>
<tr>
<td>hsize_t dims[rank];</td>
</tr>
<tr>
<td>dims[0] = 2;</td>
</tr>
<tr>
<td>dims[1] = 4;</td>
</tr>
<tr>
<td>dataset_id = H5Screate_simple(rank, dims, NULL);</td>
</tr>
<tr>
<td>// Dataset</td>
</tr>
<tr>
<td>dataset_id = H5Dcreate2(file_id,&quot;/dset&quot;,H5T_STD_I32BE, dataspace_id, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);</td>
</tr>
<tr>
<td>// Close objects</td>
</tr>
</tbody>
</table>

Creating an empty dataset in F90

<table>
<thead>
<tr>
<th>Fortran - Example 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER(HSIZE_T), DIMENSION(2) :: dims =(/2, 4/)</td>
</tr>
<tr>
<td>! ... and further variables</td>
</tr>
<tr>
<td>! Init fortran interface</td>
</tr>
<tr>
<td>CALL h5open_f(status)</td>
</tr>
<tr>
<td>! File</td>
</tr>
<tr>
<td>CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id, status)</td>
</tr>
<tr>
<td>! Dataspace</td>
</tr>
<tr>
<td>CALL h5screate_simple_f(rank, dims, dataspace_id, status)</td>
</tr>
<tr>
<td>! Dataset</td>
</tr>
<tr>
<td>CALL h5dcreate_f(file_id, dsetname, H5T_NATIVE_INTEGER, dataspace_id, &amp;</td>
</tr>
<tr>
<td>&amp; dataset_id, status)</td>
</tr>
<tr>
<td>! Close objects</td>
</tr>
</tbody>
</table>
Resulting files

C
HDF5 "h5file-c.h5" {
GROUP "/" {
   DATASET "dset" {
      DATATYPE H5T_STD_I32BE
      DATASPACE SIMPLE { ( 2, 4 ) / ( 2, 4 ) }
      DATA {
         (0,0): 0, 0, 0, 0,
         (1,0): 0, 0, 0, 0
      }
   }
}
}

Fortran
HDF5 "h5file-fortran.h5" {
GROUP "/" {
   DATASET "dset" {
      DATATYPE H5T_STD_I32LE
      DATASPACE SIMPLE { ( 4, 2 ) / ( 4, 2 ) }
      DATA {
         (0,0): 0, 0,
         (1,0): 0, 0,
         (2,0): 0, 0,
         (3,0): 0, 0
      }
   }
}
}
Write to disk

Memory

HDF5 "filename" {
GROUP "/" {
   DATASET "dset" {
      DATATYPE H5T_STD_I32BE
      DATASPACE SIMPLE { ( 2, 4 ) / ( 2, 4 ) }
      DATA {
         0 0 0 0
         0 0 0 0
      }
   }
}
}

File
## Writing to a dataset in C

**herr_t H5Dwrite( hid_t dataset_id, hid_t mem_type_id, hid_t mem_space_id, hid_t file_space_id, hid_t xfer_plist_id, const void * buf )**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hid_t</td>
<td>dataset_id IN: Identifier of the dataset to write to</td>
</tr>
<tr>
<td>hid_t</td>
<td>mem_type_id IN: Identifier of the memory datatype</td>
</tr>
<tr>
<td>hid_t</td>
<td>mem_space_id IN: Identifier of the memory dataspace</td>
</tr>
<tr>
<td>hid_t</td>
<td>file_space_id IN: Identifier of the dataset's dataspace in the file</td>
</tr>
<tr>
<td>hid_t</td>
<td>xfer_plist_id IN: Identifier of a transfer property list for this I/O operation</td>
</tr>
<tr>
<td>const void *</td>
<td>buf IN: Buffer with data to be written to the file</td>
</tr>
</tbody>
</table>

- Writes raw data from a buffer to a dataset
- **H5S_ALL** can be used for *mem_space_id* as well as *file_space_id*
- **H5P_DEFAULT** can be used for *xfer_plist_id*
## Writing to a dataset in F90

The Fortran function `h5dwrite_f` is used to write data to an HDF5 dataset. The function signature is:

```
INTEGER(HID_T), INTENT(IN) :: dset_id
INTEGER(HID_T), INTENT(IN) :: mem_type_id
TYPE, INTENT(IN) :: buf
DIMENSION(*), INTEGER(HSIZE_T), INTENT(IN) :: dims
INTEGER, INTENT(OUT) :: hdferr
INTEGER(HID_T), OPTIONAL, INTENT(IN) :: mem_space_id
INTEGER(HID_T), OPTIONAL, INTENT(IN) :: file_space_id
INTEGER(HID_T), OPTIONAL, INTENT(IN) :: xfer_prp
```

The arguments have the following meanings:

- `dset_id`: Dataset identifier
- `mem_type_id`: Memory datatype identifier
- `buf`: Data buffer; may be a scalar or an array
- `dims`: Array to hold corresponding dimension sizes of data buffer `buf`
- `hdferr`: Error code 0 on success and -1 on failure
- `mem_space_id`: Memory dataspace identifier; default value is `H5S_ALL_F`
- `file_space_id`: File dataspace identifier; default value is `H5S_ALL_F`
- `xfer_prp`: Transfer property list identifier; default value is `H5P_DEFAULT_F`

The function call is:

```
h5dwrite_f(dset_id, mem_type_id, buf, dims, hdferr, mem_space_id, file_space_id, xfer_prp)
```
Writing to an existing dataset – open a dataset in C

- **H5Dopen is a marco mapped to H5Dopen2 (in older versions may be H5Dopen1)**
Writing to an existing dataset– open a dataset in F90

```
h5dopen_f(loc_id, name, dset_id, hdferr, dapl_id)
INTEGER(HID_T), INTENT(IN) :: loc_id File or group identifier
CHARACTER(LEN=*) , INTENT(IN) :: name Name of the dataset
INTEGER(HID_T), INTENT(OUT) :: dset_id Dataset identifier
INTEGER, INTENT(OUT) :: hdferr Error code 0 on success and -1 on failure
INTEGER(HID_T), OPTIONAL, INTENT(IN) :: dapl_id Dataset access property list; default to H5P_DEFAULT_F
```
Excursion: row-major / column-major order

\[ M[i,j] = \begin{bmatrix} 22 & 33 \\ 44 & 55 \\ 66 & 77 \end{bmatrix} \]

<table>
<thead>
<tr>
<th>Adresse</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value C</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
</tr>
<tr>
<td>Value Fortran</td>
<td>22</td>
<td>44</td>
<td>66</td>
<td>33</td>
<td>55</td>
<td>77</td>
</tr>
</tbody>
</table>

- Be cautious when passing arrays between different languages
- Traversing sequential elements is much more efficient
Excursion: row-major / column-major order

- HDF5 stores arrays along the fastest changing dimension (i.e. in C order)
- HDF5 Fortran wrapper transposes dimension to adhere to the C order convention

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Dataset stored by C program

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>6</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Same dataset stored by Fortran program

Both versions result in the same layout in the HDF5 file:
Recipe: read from / write to a dataset

Your application specifies:

- Dataset
- Dataset’s datatype in memory
- Dataset’s dataspace in memory
- Dataset’s dataspace in file
- Dataset transfer property list
- Data buffer

**Datatype**
- Integer

**Dataspace**
- rank = 2
- dim[0] = 2
- dim[1] = 5

**I/O**
- serial/parallel
- caching
- compressed, etc.
Reading a dataset in C

- **herr_t H5Dread( hid_t dataset_id, hid_t mem_type_id, hid_t mem_space_id, hid_t file_space_id, hid_t xfer_plist_id, void * buf )**

  - **hid_t dataset_id**: IN: Identifier of the dataset read from
  - **hid_t mem_type_id**: IN: Identifier of the memory datatype
  - **hid_t mem_space_id**: IN: Identifier of the memory dataspace
  - **hid_t file_space_id**: IN: Identifier of the dataset's dataspace in the file
  - **hid_t xfer_plist_id**: IN: Identifier of a transfer property list for this I/O operation
  - **void * buf**: OUT: Buffer to receive data read from file

- Reads raw data from a dataset into a buffer
- **H5S_ALL** can be used for `mem_space_id` as well as `file_space_id`
- **H5P_DEFAULT** can be used for `xfer_plist_id`
## Reading a dataset in F90

```fortran
h5dread_f(dset_id, mem_type_id, buf, dims, hdferr, mem_space_id, file_space_id, xfer_prp)
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER(HID_T), INTENT(IN)</td>
<td>:: dset_id</td>
<td>Dataset identifier</td>
</tr>
<tr>
<td>INTEGER(HID_T), INTENT(IN)</td>
<td>:: mem_type_id</td>
<td>Memory datatype identifier</td>
</tr>
<tr>
<td>TYPE, INTENT(INOUT)</td>
<td>:: buf</td>
<td>Data buffer; may be a scalar or an array</td>
</tr>
<tr>
<td>DIMENSION(*)</td>
<td>:: dims</td>
<td>Array to hold corresponding dimension sizes of data buffer buf</td>
</tr>
<tr>
<td>INTEGER, INTENT(OUT)</td>
<td>:: hdferr</td>
<td>Error code 0 on success and -1 on failure</td>
</tr>
<tr>
<td>INTEGER(HID_T), OPTIONAL, INTENT(IN)</td>
<td>:: mem_space_id</td>
<td>Memory dataspace identifier; default value is H5S_ALL_F</td>
</tr>
<tr>
<td>INTEGER(HID_T), OPTIONAL, INTENT(IN)</td>
<td>:: file_space_id</td>
<td>File dataspace identifier; default value is H5S_ALL_F</td>
</tr>
<tr>
<td>INTEGER(HID_T), OPTIONAL, INTENT(IN)</td>
<td>:: xfer_prp</td>
<td>Transfer property list identifier; default value is H5P_DEFAULT_F</td>
</tr>
</tbody>
</table>
Recipe: read from / write to a dataset

The following steps are necessary:

1. Get dataset identifier
2. Specify memory datatype
3. Specify memory dataspace
4. Specify file dataspace
5. Perform desired operation on dataset
6. Close dataset
7. Close all other objects (dataspace, datatype, etc.)
Exercises

3.1 Write to an existing dataset
• Write a program in C or F90 which opens the HDF5 file from exercise 2 and fills the given dataset "dset" with integers.
For example:

```
0 1 2 3
1 2 3 4
```

3.2 Use h5dump to view the content of the previously created file. What do you see?
Writing to an existing dataset in C

C - Example 3

```c
// Open file
file_id = H5Fopen(filename, H5F_ACC_RDWR, H5P_DEFAULT);
// Open dataset
dataset_id = H5Dopen2(file_id, "dset", H5P_DEFAULT);
// Write dataset
H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, H5P_DEFAULT, data);
H5Dclose(dataset_id);
H5Fclose(file_id);
// Close objects
```

HDF5 "h5file-c.h5" {
GROUP "/" {
   DATASET "dset" {
      DATASPACE  SIMPLE { ( 2, 4 ) / ( 2, 4 ) }
      DATA {
      (0,0): 0, 0, 0, 0,
      (1,0): 0, 0, 0, 0
      }
   }
}
```

Memory

```
0  1  2  3
1  2  3  4
```

Empty dataset from example 2
Writing to existing dataset in F90

Fortran - Example 3

CALL h5open_f(status)
CALL h5fopen_f(filename, H5F_ACC_RDWR_F, file_id, status)
CALL h5dopen_f(file_id, dataset, dataset_id, status)
dims(1) = 2, dims(2) = 4
CALL h5dwrite_f(dataset_id, H5T_NATIVE_INTEGER, data, dims, status)
Partial I/O - hyperslabs

(a) Hyperslab from a 2D array to the corner of a smaller 2D array

(b) Regular series of blocks from a 2D array to a contiguous sequence at a certain offset in a 1D array
Partial I/O - hyperslabs

(c) A sequence of points from a 2D array to a sequence of points in a 3D array.

(d) Union of hyperslabs in file to union of hyperslabs in memory.
Partial I/O - hyperslabs

- Hyperslabs are portions of datasets
  - contiguous collection of points in a dataspace
  - regular pattern of points in a dataspace
  - blocks in a dataspace

- Hyperslabs are described by four parameters:
  - **start** (or offset): starting location
  - **stride**: separation blocks to be selected
  - **count**: number of blocks to be selected
  - **block**: size of block to be selected from dataspace

  Dimension of these four parameters corresponds to dimension of the underlying dataspace
Hyperslab example

- \( \text{start}[0] = 0 \)
- \( \text{start}[1] = 1 \)
- \( \text{count}[0] = 2 \)
- \( \text{count}[1] = 4 \)
- \( \text{stride}[0] = 4 \)
- \( \text{stride}[1] = 3 \)
- \( \text{block}[0] = 3 \)
- \( \text{block}[1] = 2 \)

Diagram shows a grid with highlighted regions corresponding to the specified indices and strides.
## Creating hyperslabs - C

### Function Description

The function `H5Sselect_hyperslab` is used to select a hyperslab region to add to the current selected region or to replace the existing selection with the parameters from this call.

### Function Signature

```
hid_t H5Sselect_hyperslab(hid_t space_id, H5S_seloper_t op, const hsize_t *start, const hsize_t *stride, const hsize_t *count, const hsize_t *block)
```

### Parameters

- **space_id**: Identifier of dataspace selection to modify (IN)
- **op**: Operation to perform on current selection. Here: H5S_SELECT_SET (IN)
- **start**: Offset of start of hyperslab (IN)
- **count**: Number of blocks included in hyperslab (IN)
- **stride**: Hyperslab stride (IN)
- **block**: Size of block in hyperslab (IN)

### Returns

The function returns a non-negative value if successful; otherwise, it returns a negative value.

### Examples

- **H5Sselect_hyperslab** selects a hyperslab region to add to the current selected region.
- **H5_S_SELECT_SET** replaces the existing selection with the parameters from this call.

### Notes

In C programming, you can use the `H5Sselect_hyperslab` function to create hyperslabs within your datasets. This function is particularly useful in large-scale data analysis and simulation where data is stored in multi-dimensional arrays and needs to be accessed or modified in specific regions.
## Creating hyperslabs – F90

**h5sselect_hyperslab_f***(space_id, operator, start, count, hdferr, stride, block)***

<table>
<thead>
<tr>
<th>Fortran</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER(HID_T) :: space_id</td>
<td>Dataspace identifier</td>
</tr>
<tr>
<td>INTEGER :: op</td>
<td>Flag, valid values are: H5S_SELECT_SET_F, H5S_SELECT_OR_F</td>
</tr>
<tr>
<td>INTEGER(HSIZE_T), DIMENSION(*) :: start</td>
<td>Offset of start of hyperslab</td>
</tr>
<tr>
<td>INTEGER(HSIZE_T), DIMENSION(*) :: count</td>
<td>Number of blocks to select from dataspace</td>
</tr>
<tr>
<td>INTEGER, INTENT(OUT) :: hdferr</td>
<td>Error code, 0 on success and -1 on failure</td>
</tr>
<tr>
<td>INTEGER(HSIZE_T), DIMENSION(*), OPTIONAL :: stride</td>
<td>Array of how many elements to move in each direction</td>
</tr>
<tr>
<td>INTEGER(HSIZE_T), DIMENSION(*), OPTIONAL :: block</td>
<td>Size of the element block</td>
</tr>
</tbody>
</table>

- **h5sselect_hyperslab_f** selects a hyperslab region to add to the current selected region
- **H5S_SELECT_SET_F** replaces the existing selection with the parameters from this call
Exercise

4.1 Using hyperslabs

**C:** copy "/lustre/jwork/hpclab/train000/exercises2015/hdf5/exercises/c/ex04-partial-io.c" to your local directory. Add the missing parts to the code.

**F90:** copy "/lustre/jwork/hpclab/train000/exercises2015/hdf5/exercises/f90/ex04-partial-io.f90" to your local directory. Add the missing parts to the code.

"ex04-partial-io.c" ("ex04-partial-io.f90") creates an two dimensional dataset as given on the left. After you have completed the code the dataset in the resulting HDF5 file should look like the dataset given here on the right hand side:
PHDF5 – Parallel HDF5
PHDF5 - Factoids

- Support MPI programming
- PHDF5 files compatible with serial HDF5 files
  - Shareable between different serial or parallel platforms
- Single file image to all processes
  - One file per process design is undesirable
- A standard parallel I/O interface must be portable to different platforms.
PHDF5 implementation layers

PHDF5 is built on top of the standard MPI-IO API

- User Application
- HDF5 library
- Parallel I/O layer
- Parallel file systems (GPFS, Lustre, …)
Important to know

- Most functions of the PHDF5 API are collectives
  - i.e. all processes of the communicator must participate
- PHDF5 opens a parallel file with a communicator
  - Returns a file-handle
  - Future access to the file via the file-handle
  - Different files can be opened via different communicators
- Function names equal to corresponding function names in the serial version of the HDF5 library
What Does PHDF5 Support?

- After a file is opened by the processes of a communicator
  - All parts of file are accessible by all processes
  - All objects in the file are accessible by all processes
  - Multiple processes may write to the same data array
  - Each process may write to an individual data array
- C and F90 language interfaces
Programming model

- HDF5 uses access template object (property list) to control the file access mechanism
- General model to access HDF5 file in parallel:
  - Setup MPI-IO (access property list)
  - Open File
  - Access Data
  - Close File
Setup MPI-IO access template in C

```c
hid_t H5Pcreate(hid_t cls_id)
```

- **hid_t cls_id** IN: The class of the property list to create. Here: H5P_FILE_ACCESS
- Returns a property list identifier (plist_id) if successful: otherwise -1

```c
herr_t H5Pset_fapl_mpio( hid_t plist_id, MPI_Comm comm, MPI_Info info )
```

- **hid_t plist_id** IN: File access property list identifier
- **MPI_Comm comm** IN: MPI-2 communicator
- **MPI_Info info** IN: MPI-2 info object
- Returns a non-negative value if successful. Otherwise returns a negative value

- **H5Pset_fapl_mpio** is the standard HDF5 file driver for parallel file systems. This driver uses the MPI standard for both communication and file I/O.
- Each process of the MPI communicator creates an access template and sets it up with MPI parallel access information
Setup MPI-IO access template in F90

- `h5pcreate_f(classtype, plist_id, hdferr)`
  - `INTEGER, INTENT(IN) :: classtype` The type of the property list.
  - `INTEGER(HID_T), INTENT(OUT) :: plist_id` Property list identifier
  - `INTEGER, INTENT(OUT) :: hdferr` Error code, 0 on success and -1 on failure

- `h5pset_fapl_mpio_f(plist_id, comm, info, hdferr)`
  - `INTEGER(HID_T), INTENT(IN) :: plist_id` Property list identifier
  - `INTEGER, INTENT(OUT) :: comm` MPI communicator to be used for file open
  - `INTEGER, INTENT(IN) :: info` MPI info object to be used for file open
  - `INTEGER, INTENT(OUT) :: hdferr` Error code, 0 on success and -1 on failure

- `H5Pset_fapl_mpio_f` is the standard HDF5 file driver for parallel file systems. This driver uses the MPI standard for both communication and file I/O.

- Each process of the MPI communicator creates an access template and sets it up with MPI parallel access information
# Dataset transfer property - C

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hid_t H5Pcreate(hid_t cls_id)</code></td>
<td><code>hid_t</code></td>
<td><code>cls_id</code></td>
</tr>
<tr>
<td><code>herr_t H5Pset_dxpl_mpio(hid_t plist_id, H5FD_mpio_xfer_t xfer_mode)</code></td>
<td><code>hid_t</code></td>
<td><code>plist_id</code></td>
</tr>
<tr>
<td><code>hid_t</code></td>
<td><code>xfer_mode</code></td>
<td>IN: Transfer mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns a non-negative value if successful. Otherwise returns a negative value</td>
</tr>
</tbody>
</table>

- **Sets data transfer mode**
  - **H5FD_MPIIO_INDEPENDENT**: Use independent I/O access (default)
  - **H5FD_MPIIO_COLLECTIVE**: Use collective I/O access
- **Access dataset with the defined transfer property**
### Dataset transfer property – F90

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>h5pcreate_f(classtype, plist_id, hdferr)</code></td>
<td>The type of the property list. Here: H5P_DATASET_XFER_F</td>
</tr>
<tr>
<td><code>h5pset_dxpl_mpio_f(plist_id, data_xfer_mode, hdferr)</code></td>
<td>Sets data transfer mode&lt;br&gt;- H5FD_MPIO_INDEPENDENT_F: Use independent I/O access (default)&lt;br&gt;- H5FD_MPIO_COLLECTIVE_F: Use collective I/O access&lt;br&gt;Access dataset with the defined transfer property</td>
</tr>
</tbody>
</table>

- Sets data transfer mode
  - H5FD_MPIO_INDEPENDENT_F: Use independent I/O access (default)
  - H5FD_MPIO_COLLECTIVE_F: Use collective I/O access
Writing dataset by rows - C

$P_0$

$P_1$

$P_2$

$P_3$

File
Writing dataset by rows - C

Memory

\[
\begin{align*}
\text{count}[1] \\
P_1
\end{align*}
\]

File

\[
\begin{align*}
\text{offset}[0] &= 2 \\
\text{offset}[1] &= 0 \\
\text{count}[0] &= 2 \\
\text{count}[1] &= 4
\end{align*}
\]
Writing dataset by columns – F90

- $P_0$
- $P_1$
- $P_2$
- $P_3$

File
Writing dataset by columns – F90

Memory

File

P₁

count(1) = 4
count(2) = 2
offset(1) = 0
offset(2) = 2
Writing regularly spaced columns - C
Writing regularly spaced columns - C

Memory

File

count[0] = 1
count[1] = 3
offset[0] = 0
offset[1] = 1
stride[0] = 1
stride[1] = 2
block[0] = 4
block[1] = 1
Writing regularly spaced rows – F90

Memory

File

P₀

P₁
Writing regularly spaced columns – F90

Memory

File

<table>
<thead>
<tr>
<th>P₁</th>
<th>dimsm(1)</th>
<th>dimsm(2)</th>
</tr>
</thead>
</table>

| count(1) = 3  | offset(1) = 1  | block(1) = 1  |
| count(2) = 1  | offset(2) = 0  | block(2) = 4  |
| stride(1) = 2 | stride(2) = 1  |              |
|              | block(2)      |              |
5.1 Parallel write

**C:** copy "/lustre/jwork/hpclab/train000/exercises2015/hdf5/exercises/c/ex05-phdf5-parallel-write.c" to your local directory. Add the missing parts to the code.

**F90:** copy "/lustre/jwork/hpclab/train000/exercises2015/hdf5/exercises/f90/ex05-phdf5-parallel-write.f90" to your local directory. Add the missing parts to the code.

"ex05-phdf5-parallel-write.c" ("ex05-phdf5-parallel-write.f90") add 10 to each rank and the result is written out in parallel to an HDF5 file. In depends of the number of tasks, the dataset should look like this:

<table>
<thead>
<tr>
<th>MPI tasks = 1</th>
<th>MPI tasks = 2</th>
<th>MPI tasks = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,10,10,10,10,</td>
<td>10,10,10,10,10,</td>
<td>10,10,10,10,10,</td>
</tr>
<tr>
<td>10,10,10,10,10,</td>
<td>10,10,10,10,10,</td>
<td>10,10,10,10,10,</td>
</tr>
<tr>
<td>10,10,10,10,10,</td>
<td>10,10,10,10,10,</td>
<td>10,10,10,10,10,</td>
</tr>
<tr>
<td>10,10,10,10,10,</td>
<td>10,10,10,10,10,</td>
<td>10,10,10,10,10,</td>
</tr>
<tr>
<td>10,10,10,10,11,</td>
<td>11,11,11,11,11,</td>
<td>12,12,12,12,12,</td>
</tr>
<tr>
<td>10,10,10,11,11,</td>
<td>11,11,11,11,11,</td>
<td>13,13,13,13,13,</td>
</tr>
<tr>
<td>10,10,11,11,11,</td>
<td>11,11,11,11,11,</td>
<td>13,13,13,13,13,</td>
</tr>
<tr>
<td>10,11,11,11,11,</td>
<td>11,11,11,11,11,</td>
<td>13,13,13,13,13,</td>
</tr>
<tr>
<td>10,11,11,11,11,</td>
<td>11,11,11,11,11,</td>
<td>13,13,13,13,13,</td>
</tr>
<tr>
<td>10,10,10,10,10,</td>
<td>11,11,11,11,11,</td>
<td>13,13,13,13,13,</td>
</tr>
<tr>
<td>10,10,10,10,10,</td>
<td>11,11,11,11,11,</td>
<td>13,13,13,13,13,</td>
</tr>
</tbody>
</table>