HDF5

Christina Lee

September 20, 2017

Category: Programming

1 HDF5 in Julia

So last summer, my program was producing three dimensional data, and I needed a way to export and save that data from my C++ program. Simple ASCII files, my default method, no longer covered my needs. Of course, I wasn't the first person to encounter this problem, so I discovered the HDF5 standard.

Instead of storing data in a human readable format like ASCII, the Hierarchical Data Format, HDF, stores data in binary format. This preserves the shape of the data in the computer and keeps it at its minimum size. WOHOO!!

Sadly, the syntax for HDF5 in C++ and Fortran is just as bad as FFTW or OpenBLAS. But Happily, just like FFTW and OpenBLAS, HDF5 has wonderful syntax in Julia, Python, and MATLAB, amoung others.

So how does it work?

We don't just print a single variable. Each HDF5 file is like its own file system. In my home directory, I have my documents folder, my programming folder, my pictures, configuration files,... and inside each folder I can have subfolders or files.

The same is true for an HDF5 file. We have the root, and then we have groups and subgroups. A group is like a folder. Then we can have datasets. Datasets are objects that hold data (files).

1.1 Installing the Package

While running `Pkg.add("HDF5");` will hopefully add the HDF5 library as well, additional steps may be required. I remember before having a horrible time with it when using C++ a year ago, but not what I did. If at all possible, just use a package manager, and not try and install it from source! See the HDF5.jl or HDFGroup pages for details.

```
In [1]: using HDF5;
```

2 Hello World

First lets open a file and then write some data to it.

We can open a file in three ways:

Symbol	Meaning
"w"	Write. Will overwrite anything already there.
"r"	Ready-only.
"r+"	Read-write. Preserving existing contents.

If we open with this syntax, we have to always remember to close it with `close()`

```
In [2]: fid=h5open("test.h5","w")

        fid["string"]="Hello World"

        close(fid)
```

2.0.1 Navigating a File

Now lets see if we were successful by reading. Instead of reading the dataset, we are going to checkout the structure of the file first.

`names(fid)` tells us what is inside the location `fid`.

`dump(fid)` is much more in depth, exploring everything below `fid`. If we had a bunch of subdirectories, it would go down each one to see what was there.

Both these functions help you find your way around a file.

```
In [3]: fid=h5open("test.h5","r")

        println("names \n",names(fid))

        println("\n dump")
        println(dump(fid))

        close(fid)
```

```
names
String["string"]

 dump
HDF5.HDF5File
  id: Int32 16777216
  filename: String "test.h5"
nothing
```

2.0.2 Reading Data

Now when we are reading data, we need to know the difference between dataset and the data the dataset constains.

Look at the below example

```
In [4]: fid=h5open("test.h5","r")

        dset=fid["string"]
        println("the dataset: \t", typeof(dset))

        data=read(dset)
        println("the string: \t", typeof(data),"\t",data)

        data2=read(fid,"string")
        println("read another way: \t", typeof(data2),"\t",data2)

        close(fid)
```

```
the dataset:          HDF5.HDF5Dataset
the string:          String        Hello World
read another way:          String        Hello World
```

A dataset is like the filename "fairytale.txt", so we then need to read the file to get "Once upon a time ...".

2.0.3 Groups

So I've talked about groups, but we haven't done anything with them yet. So lets make some groups!

Here we use `g_create` to create two groups, one inside the other. For the subgroup, it's parent is `g`, so we have to create it at location `g`. Just like in a filesystem, it's name/ path is nested within its parent's path.

```
In [5]: fid=h5open("test.h5","w")

        g=g_create(fid,"mygroup");
        h=g_create(g,"mysubgroup");

        println(dump(fid))

        println("\n path of h:  ",name(h))

        close(fid)
```

```
HDF5.HDF5File
  id: Int32 16777216
  filename: String "test.h5"
nothing

 path of h:  /mygroup/mysubgroup
```

2.0.4 Attributes

Say in a file I want to include the information that I ran the simulation with 100 sites, at 1 Kelvin, for 100,000 timesteps. Instead of creating new datasets for each of these individual numbers, I can create attributes and tie them to either a group or a dataset.

```
In [6]: fid=h5open("test.h5","w")

        fid["data"]=randn(3,3);

        attrs(fid["data"])["Temp"]="1";
        attrs(fid["data"])["N Sites"]="100";

        close(fid)

        fid=h5open("test.h5","r")

        dset=fid["data"];

        println("typeof attrs: \t", typeof(attrs(dset)))
```

```
        println("Temp: \t",read( attrs(dset),"Temp"  ))
        println("N Sites: \t",read(  attrs(dset),"N Sites"  ))

        close(fid)
```

```
typeof attrs:          HDF5.HDF5Attributes
Temp:          1
N Sites:          100
```

2.0.5 Final Tips

Before diving in to learn how to use this, think about whether you need it or not. How large and complex is your data? Is it worth the time to learn? While the syntax might be relatively simple in Julia, ASCII files are still much easier to deal with.

If you are going to play around or use this format, I recomend getting an HDF viewer, like https://www.hdfgroup.org/pro While you can have much more control via code, sometimes it is just that much simpler to check everything is working via a gui.

For more information, checkout the Package page at https://github.com/JuliaLang/HDF5.jl or the HDFGroup page at https://www.hdfgroup.org/

I just showed some of the functionality in test cases. As much control as you want, you can get; you might just have to work a bit for it.

In [ ]: