NiftiArray: Solving Your Big Nifti Needs

source at https://github.com/muschellij2/niftiarray_talk

John Muschelli, slides at http://johnmuschelli.com/niftiarray_talk

Start of a Package: Plotting a voxel from 800 Subjects in a Shiny App

How can you read in a NIfTI image in R?

Options for reading in NIfTI images

Array like

- `oro.nifti::readNIfTI/neurobase::readnii`
- `AnalyzeFMRI::f.read.volume()` - can't read gzipped
- `neuroim::loadVolume("example.nii")`

C++ pointer

- `ANTsCore::antsImageRead`
- `RNifti::readNifti`

From RNifti: https://github.com/jonclayden/RNifti#performance

```
library(microbenchmark)
microbenchmark(AnalyzeFMRI::f.read.volume("example.nii"),
               ANTsRCore::antsImageRead("example.nii"),
               neuroim::loadVolume("example.nii"),
               oro.nifti::readNIfTI("example.nii"),
               RNifti::readNifti("example.nii"),
               tractor.base::readImageFile("example.nii"), unit="ms")
# Unit: milliseconds
#                                       expr       min        lq      mean
#   AnalyzeFMRI::f.read.volume("example.nii") 26.312881 26.769244 29.685981
#     ANTsRCore::antsImageRead("example.nii")  1.150787  1.626918  2.149145
#          neuroim::loadVolume("example.nii") 34.596506 37.732245 54.065227
#         oro.nifti::readNIfTI("example.nii") 57.059828 61.953430 89.386706
#            RNifti::readNifti("example.nii")  0.986773  1.136562  1.683821
#  tractor.base::readImageFile("example.nii") 33.380407 34.096574 34.961812
#     median        uq        max neval
#  27.301693 28.214908 185.441664   100
#   2.210696  2.481675   3.376350   100
#  40.714061 45.425047 192.978225   100
#  65.064212 71.425561 220.709246   100
#   1.501528  1.856601   7.961566   100
#  34.617259 35.257633  42.108584   100
```

What do we want

1. Something fast
2. Has overloaded functions (e.g. `image + image = image`)
3. Can use most plotting functions on it (e.g. `ortho2`)
4. Is array like but has header
5. Keeps memory low
6. Has **random access**

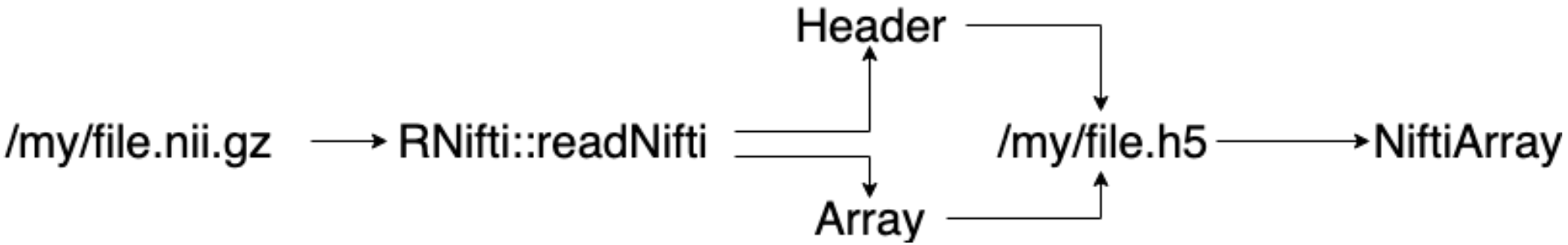
Current issues

- `oro.nifti::readNIfTI` - slow
- `ANTsCore::antsImageRead` - installation is
- `RNifti::readNifti` - from https://github.com/jonclayden/RNifti/issues/14 :

For your single-precision floating-point image with about 1bn elements, the “native” data size in the C struct is roughly 4 GiB, while the R array uses 8 GiB because it uses double precision. When you use readNifti(…, internal=TRUE) the resulting object has only the C struct, but with readNifti(…, internal=FALSE) the object has both the C struct and the R array, for a total of 12 GiB.

Solution: NiftiArray

- Stores image and header in HDF5 (Hierarchical Data Format) format
- Based on `HDF5Array` from Bioconductor (https://bioconductor.org/packages/release/bioc/html/HDF5Array.html)



Takes longer than RNifti (the first time)

```
fname = fslr::mni_fname()
system.time({
  rnifti = RNifti::readNifti(fname)
})
```

```
##    user  system elapsed
##   0.140   0.021   0.296
```

```
system.time({
  narray = NiftiArray::writeNiftiArray(fname)
  narray@seed@filepath
  narray
})
```

```
##    user  system elapsed
##   4.721   0.133   5.450
```

Sizes

```
format(object.size(rnifti), units = "Kb")
```

```
## [1] "28208.3 Kb"
```

```
format(object.size(narray), units = "Kb")
```

```
## [1] "8.7 Kb"
```

Header Information

```
print(nifti_header(narray))
```

```
## NIfTI-1 header
##     sizeof_hdr: 348
##       dim_info: 0
##            dim: 3  182  218  182  1  1  1  1
##      intent_p1: 0
##      intent_p2: 0
##      intent_p3: 0
##    intent_code: 0
##       datatype: 8
##         bitpix: 32
##    slice_start: 0
##         pixdim: -1  1  1  1  0  0  0  0
##     vox_offset: 352
##      scl_slope: 0
##      scl_inter: 0
##      slice_end: 0
##     slice_code: 0
##     xyzt_units: 10
##        cal_max: 8000
##        cal_min: 3000
```

Accessing voxels

```
narray[18, 5, 14]
```

```
## [1] 572
```

```
sum(narray > 9990)
```

```
## [1] 20
```

```
narray[ narray > 9990]
```

```
##  [1] 9999 9999 9992 9997 9991 9993 9997 9998 9999 9999 9994 9997 9998 9999
## [15] 9998 9991 9993 9991 9996 9994
```

```
DelayedArray::extract_array(narray, index = list(NULL ,NULL, 1))
```

```
## , , 1
##
##        [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
##   [1,]    0    0    0    0   30   31   29   29   29    29    29    28
```

What are the benefits?

- Overall most benefits come from saving H5 files on "big" computer
- Random access - the `DelayedArray` package
- Operations on `DelayedArray` objects, the DelayedMatrixStats package https://bioconductor.org/packages/release/bioc/html/DelayedMatrixStats.html

What are the benefits? Reshaping to a Matrix

```
d = dim(narray)
d
```

```
## [1] 182 218 182
```

```
dim(narray) = c(d, 1)
new_dim = c(prod(d), 1)
newarray = writeNiftiArray(narray)
mat = ReshapedNiftiArray(newarray@seed@filepath, dim = new_dim)
dim(mat)
```

```
## [1] 7221032       1
```

```
head(mat)
```

```
## <6 x 1> DelayedMatrix object of type "integer":
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
```

Why? Voxel-wise operations in low-memory setting

```
big_mat = DelayedArray::acbind(mat, mat, mat)
dim(big_mat)
```

```
## [1] 7221032       3
```

```
format(object.size(big_mat), units = "Kb")
```

```
## [1] "26.7 Kb"
```

```
rmed = DelayedMatrixStats::rowMedians(big_mat)
```

Why? 4D Example

```
nii_fname = system.file("extdata",
                        "example_4d.nii.gz", package = "RNifti")
from = NiftiArray::NiftiArray(nii_fname)
dim(from)
```

```
## [1] 96 96 60  3
```

```
A <- ReshapedNiftiArray(filepath = from@seed@filepath,
                        name = from@seed@name,
                        dim = c(prod(dim(from)[1:3]), dim(from)[4]))
dim(A)
```

```
## [1] 552960      3
```

```
med = DelayedMatrixStats::rowMedians(A)
```

Back to an image

```
arr = array(med, dim = dim(from)[1:3])
med_array = writeNiftiArray(arr, header = nifti_header(from))
as(med_array, "niftiImage")
```

```
## Image array of mode "double" (4.2 Mb)
## - 96 x 96 x 60 voxels
## - 2.5 x 2.5 x 2.5 mm per voxel
```

Issues

- Need to write H5 file first (do in parallel on cluster)
- Memory tradeoff with speed due to chunking
- Most of this needs things in the same space

Use cases

- Something on Shiny for speed or memory
    - shinyapps.io has very restrictive memory settings for imaging
- Analyzing 1000s of subjects together
- Pass object from R to Matlab: https://www.mathworks.com/help/matlab/ref/hdf5read.html

Extensions

- Anything that works with DelayedArrays should work here
- Check out Bioconductor
- Example: `mbkmeans` - mini-batch k-means https://bioconductor.org/packages/release/bioc/html/mbkmeans.html