beachmat: a relaxing C++ API for interacting with R matrix types

Plus some words about the 1.3 million data set

Aaron Lun

26 July 2017

Bioconductor Developer Day

Boston

The variety of matrix flavours

Lots of matrix types in R:

- good old `matrix`
- *Matrix* classes:
  - `dgeMatrix`, dense
  - `dgCMatrix`, column-sparse compressed
- Bioconductor classes:
  - `RleMatrix`, run-length encoding
  - `HDF5Matrix`, file-backed on HDF5
  - `DelayedMatrix`, delayed operations wrapper

Each class has its own advantages (memory/simplicity tradeoff).

Class instances are more-or-less interchangeable in R code, provided that `[,`, `dims`, `rbind`, etc. are supported.

What? C++?

R packages can contain compiled code, usually Fortran or C/C++.

Why? Because it can be faster than pure R.

- Avoid re-interpretation of code in loops
- Streamline memory management
- Access useful Fortran/C/C++ libraries (e.g., LAPACK)

Order of ease of use: Fortran $<$ C $<$ C++ (via *Rcpp*)

Dealing with matrix varieties in C++

Simplest use of matrices are for data access:

- rows = genes, columns = samples (cells in scRNA-seq)
- rows/columns = genomic regions (Hi-C)

S4 methods for access are not supported natively in compiled code.

Need to know exact implementation details of each representation:

- `dgCMatrix`: what are `i`, `p`, `x`?
- `HDF5Matrix`: need to interact with the HDF5 library
- `dspMatrix`: upper/lower triangular storage only

Becomes a pain to support many possible matrix types.

Introducing the *beachmat* API

Provides a common C++ interface for row/column data access to

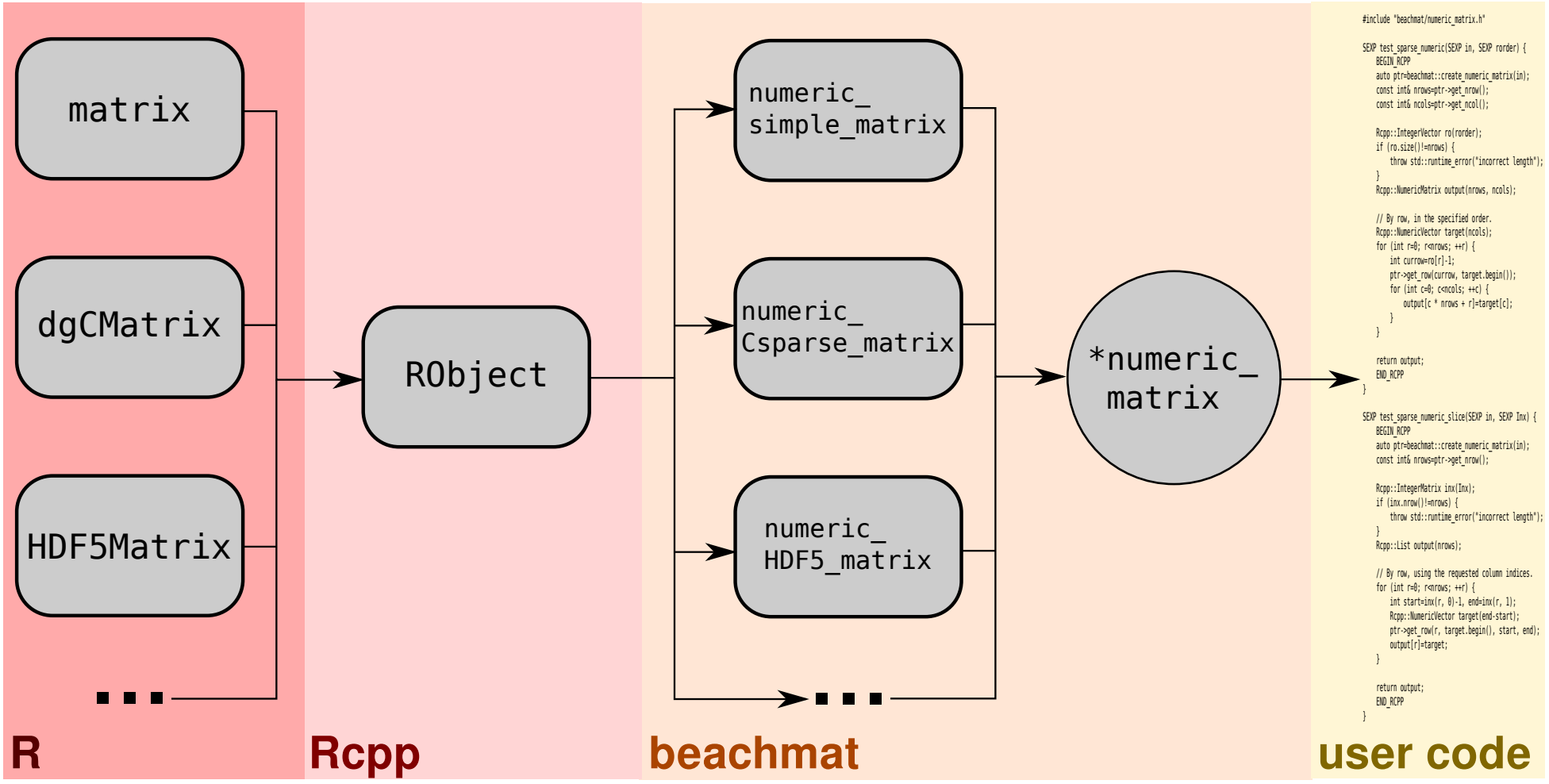
- `matrix` (obviously)
- *Matrix* classes, inc. `dgCMatrix`, `dgeMatrix`, `dspMatrix`
- `DelayedMatrix`, `HDF5Matrix`, `RleMatrix`

Simple to use in the C++ code in your package:

1. Link to *beachmat* (now on BioC)
2. Incoming `SEXP`'s used to make a `*_matrix` pointer
3. Call `get_row`, `get_col` on the pointer

... and now your C++ code supports all matrix types.

Or in other words



- *beachmat* is built on top of a *Rcpp* base
- also, "users" here are really other package developers

What *beachmat* does and does not do

What it does do:

- access data from rows/columns
- write data to rows/columns

Probably the most common operations in package code.

What it doesn't do:

- combine matrices *a la* `rbind`, `cbind`
- matrix operations like multiplication
- matrix factorizations

Hard to beat BLAS and LAPACK for the last two.

Analyzing 1.3 million cells with Bioconductor

1.3 million neuron data set from 10X:

- about 28000 genes; 145 GB as a simple `matrix`
- 7% non-zero; est. 20 GB as a `dgCMatrix`[1]
- provided as a 3GB compressed HDF5 file[2]

Analysis strategy:

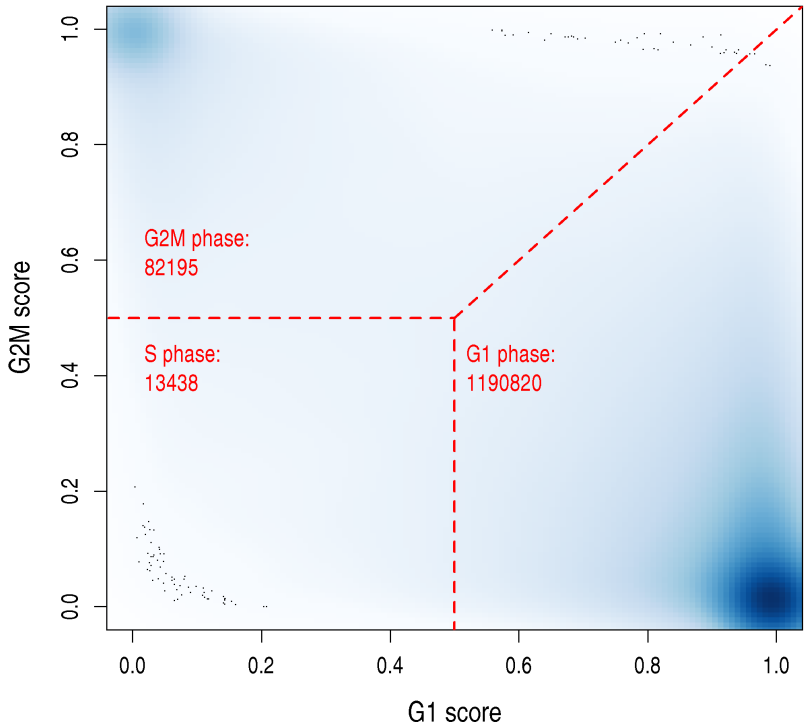
1. Use *TENxGenomics* package (Martin Morgan) to read file
2. Convert `TENxMatrix` into `HDF5Matrix`
3. Use *scran* with C++ functions based on *beachmat* to analyze

Running on a desktop with 8GB ram, Core i5 processor.

[1] though this doesn't work due to overflow

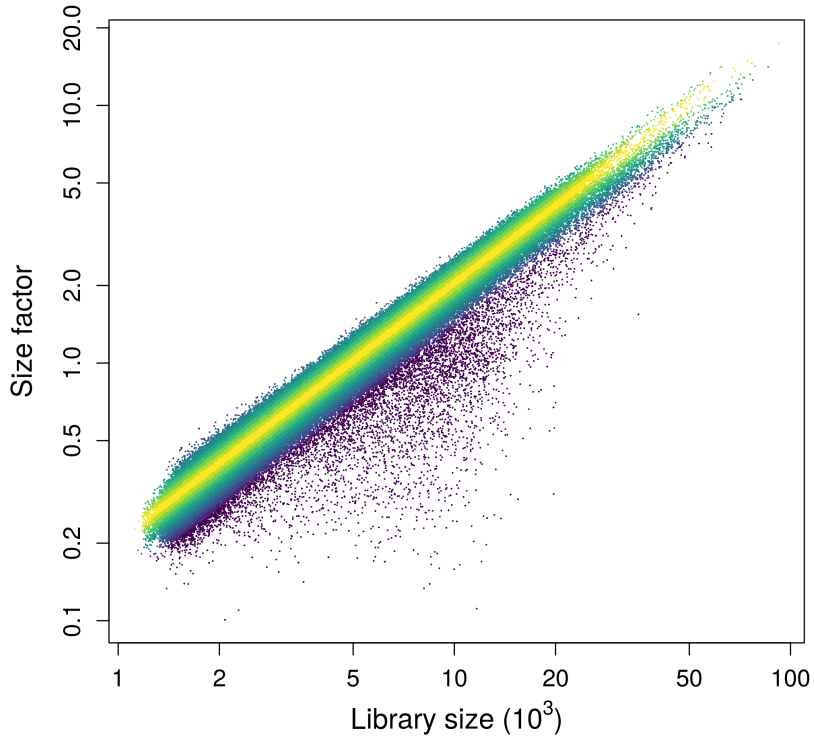
[2] stored in CSC format

Cell cycle assignment with `cyclone`



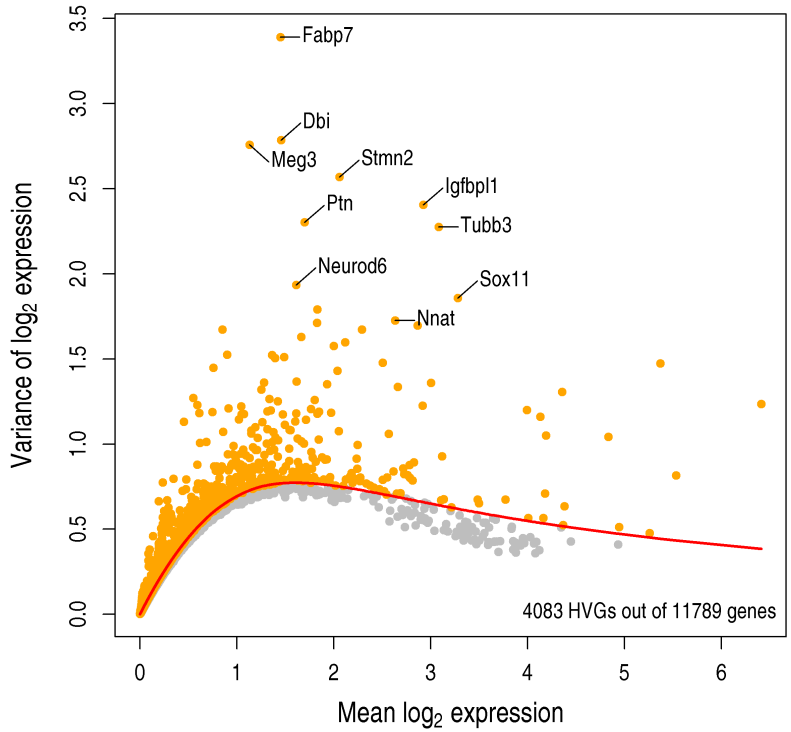
- 24 hours on 3 cores (fully parallelizable)
- most neurons in G1, which is probably sensible

Normalization by deconvolution



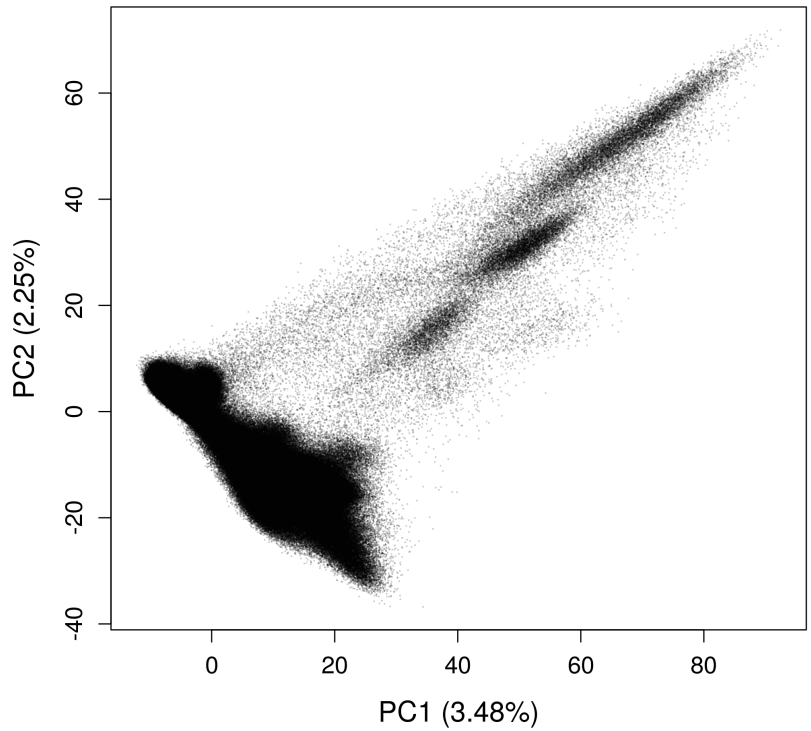
- 5 hours on 1 core, using chunks of 2000-3000 cells
- + 3 hours to write to normalized expression values
- mostly correlated with library size (+ some outliers)

Detection of highly variable genes



- 8 hours
- blocked on library of origin for each cell
- lots of brain-related stuff

Dimensionality reduction with PCA



Used a subsetting strategy (5 hours):

- randomly sampled 10000 cells and did PCA
- projected all other cells to the first two PCs

Doesn't actually use *beachmat* here.

Acknowledgements

Hervé Pagès (*HDF5Array*, *DelayedArray*)
Mike Smith (*Rhdf5lib*, Windows)
Everyone in the single-cell big data working group:

- Martin Morgan
- Davide Risso, Peter Hickey, Andrew McDavid
- Raphael Gottardo, Mike Jiang
- John Readey
- and others...

Accessing simple `matrix` objects

Stored column-major in a one-dimensional array

- entry (r, c) is element $r + cN_c$ for 0-based indices
- Easy and fast to access and write
- Stored wholly in memory, $N_r N_c$ elements

beachmat requires 20-50% more time than native *Rcpp*

- at the same speed for read-only column acccess
- probably okay, as data access usually isn't the bottleneck.

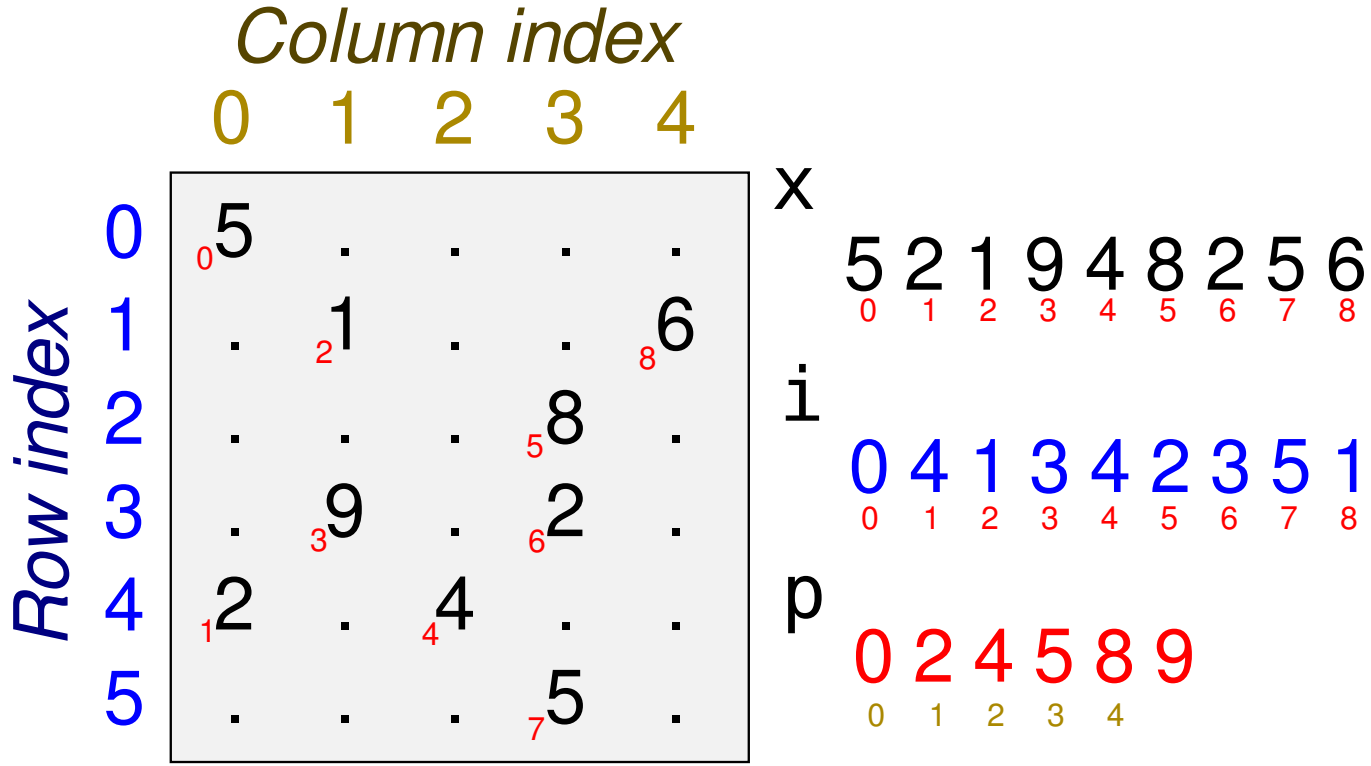
An important aside

Each access copies the row/column data from the matrix to a new `Rcpp::Vector`, much like when you subset the matrix in R.

This is necessary for consistency across matrix types, as the data in the underlying representation may not be contiguous.

Accessing sparse `dgCMatrix` objects

Stored in column-sparse compressed format:



- Reduced memory for sparse data ($<$ 66% non-zero)
- Generally slower access than simple matrices

Accessing sparse `dgCMatrix` objects (cont)

Column access with *beachmat* slower than *RcppArmadillo*

- Due to need for copying
- still pretty fast (20% more time than equivalent `matrix`)

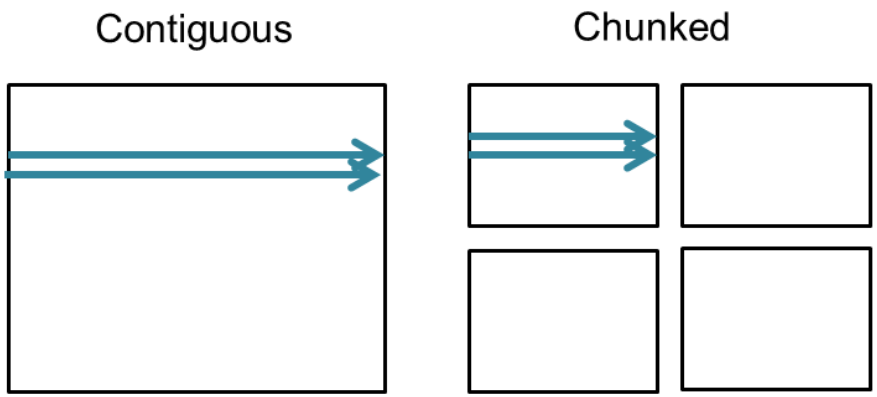
Row access with *beachmat* caches the indices

- avoid repeated binary searches for consecutive access
- about 4× faster than a naïve implementation
- orders of magnitude faster than *RcppArmadillo* (?)

Accessing file-backed `HDF5Matrix` objects

Class developed by Hervé Pagès in the *HDF5Array* package:

- Data stored on disk as a HDF5 file, with a stub object in R
- very memory efficient: 3 kbp size, regardless of dimensions
- much slower (20-40×) as disk reads required for access
- need to consider file layout for optimal performance



beachmat uses the HDF5 C++ API (*Rhdf5lib*, Mike Smith)
↪ optimize chunk cache for consecutive row/column access

Other stuff

Also supports `RleMatrix`, `*spMatrix`, `*geMatrix` classes

- `DelayedMatrix` instances are implicitly realized
- easily extensible to more classes

Limited support for matrix output:

- `matrix`, `dgCMatrix`, `HDF5Matrix`
- can dynamically choose output type based on input type.