University of California, Davis

ECS145 Term Project Report

E. Kristovich, C. Nagda, V. Nguyen, V. Nguyen, R. Park

March 20, 2019

Contents

1 Introduction

1.1 Purpose

The goal of this project is to mitigate slowdowns in an existing R package by implementing certain functions in C++.

1.2 Package

We have selected the `abind` package for its simplistic elegance in R, and we explore whether replacing some of the interpreted R with compiled C++ code will provide a significant speedup for arrays with many dimensions and elements. The package includes functions that allow for modifying and extracting data in n-dimensional arrays, generalizing the popular rbind and cbind functions. The `abind` package contains several functions: `abind()`, `acorn()`, `adrop()`, `afill()`, and `asub()`.

1.3 Motivation

We will implement the asub() and acorn() functions. asub() is commonly called as a sub-function. For example, afill() and acorn() call asub(). Both asub() and acorn() are frequently used functions with room for optimization because they are implemented with for loops in pure R. We believe that these two functions can benefit from speed ups by implementing them in C++, and are good targets given the scope of this project.

2 Implementation

2.1 Rcpp

We chose to use the Rcpp package to call our C++ functions. Rcpp essentially does the same thing as a .Call(), but it allows us to use C++ types from the Rcpp namespace that are much more similar to R types, making it easier to translate between R and C++.

2.2 asub()

For asub, we create an asubCpp.cpp file. As is standard, we start the cpp file with the following two lines to include Rcpp.

```
#include <Rcpp.h>
using namespace Rcpp;
```

For the function to be callable in R, we need to mark the function for export in a comment that is interpreted as an export by Rcpp before the function:

```
// [[Rcpp::export]]
std::string asubCpp(List idx, int numDims, NumericVector dims, bool drop, bool showDrop) {

 ...

}
```

The function itself, attached separately, replaces these lines of R code:

```
xic <- Quote(x[,drop=drop])
# Now duplicate the empty index argument the appropriate number of times
xic <- xic[c(1, 2, rep(3, length(d)), 4)]
if (is.null(drop)) {
    xic <- xic[-length(xic)]
} else {
    xic[[length(xic)]] <- drop
}
for (i in seq(along=dims))
    if (!is.null(idx[[i]]))
        xic[2+dims[i]] <- idx[i]
```

We attempted to repalce the most intensive parts of the R code: looping through the array and building a string to subset. We expect that calling a C++ function in place of the above R code will make the execution faster for arrays with upwards of 100 dimensions, as the for loop executes numDimensions times.

2.3 acorn()

For acorn(), we create an acorn1sr.cpp file. For the first few lines, it is similar with asubCpp.cpp.

```
#include <Rcpp.h>

using namespace Rcpp;
```

We also need to mark the function acorn1sr() to be exported in a comment which Rcpp will interpret as an export.

```
// [[Rcpp::export]]
List acorn1sr(int len_as, List as, NumericVector dim, int len_dim_x, List args, int len_args) {

...

}
```

The function acon1sr() will take length of list as, list as (this is a variable of acorn(), it it used to store the result matrix), list dim (the dimension of input matrix x), length of list dim, list args (the list of all the arguments of the acorn()), and the length of list args.

Our acorn1sr() will replace the first for loop inside function acorn.R

```
for (i in seq(4, length(dim(x)))) {
    if (length(args) >= i-3)
        a <- args[[i-3]]
    else
        a <- 1
    a <- sign(a)*min(abs(a), dim(x)[i])
    as <- c(as, list(if (a >= 0) seq(len=a) else seq(len=-a, to=dim(x)[i])))
}
```

As we expected, for loop in R code will slow down the performance. That is why we rewrote it in C++ to make it faster.

3 Results

3.1 microbenchmark

The microbenchmark package is an accurate benchmarking tool used in place of the popular *system.time*. It takes batches of sub-millisecond timings, calculates statistics such as mean and standard deviation, and is supported by *ggplot2*'s autoplot function to generate figures. For example, to compare two functions:

```
result <- microbenchmark(func1(), func2(), times=1000)
```

The *result* object can be printed or plotted to display the results. We use *microbenchmark* to perform experiments with 100 to 1000 trials, depending on the execution time of the experiment, and test a variety of cases. We compare our C++ implementation of functions versus the original *abind* functions.

3.2 asub()

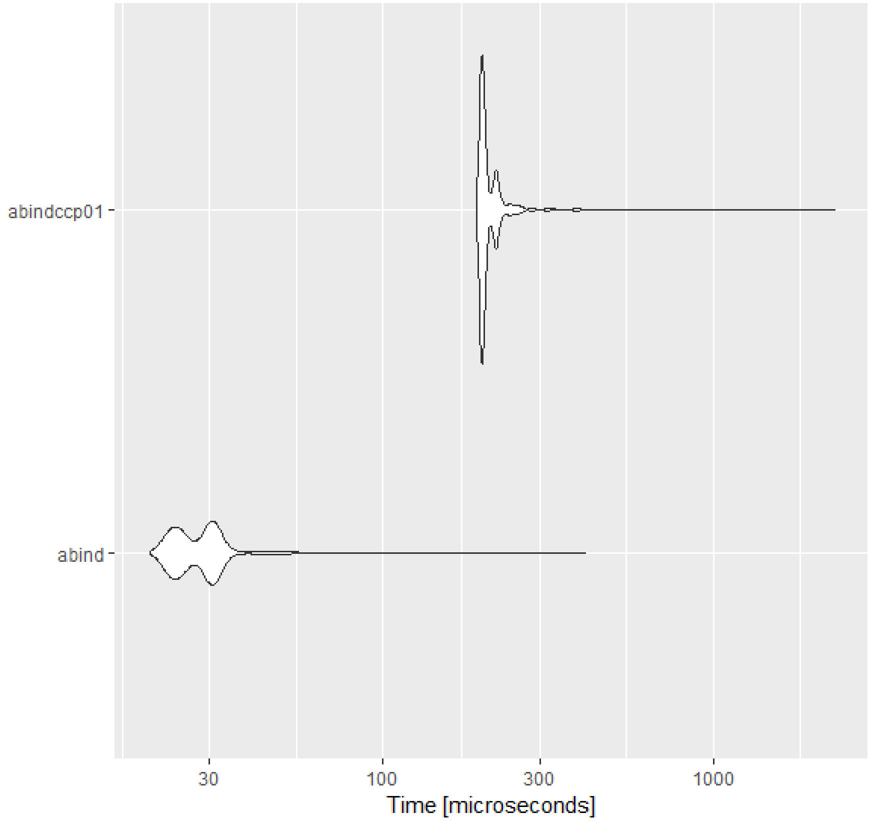
We performed three different experiments comparing the performance of asub(). *abindcpp01* refers to the modified package, where as *abind* refers to the original package and acts as the control.

3.2.1 Case 1: Small Array

For our first experiment, we tested asub() on a trivial array with 1000 trials:

```
x <- array(1:24,dim=c(2,3,4))
asub(x, 1, 1)
```

```
Unit: microseconds
       expr     min      lq      mean  median      uq      max neval
      abind  18.732  22.837  28.58523  28.226  30.535  125.218  1000
 abindccp01 192.958 197.834 217.73057 203.222 222.466 2671.898  1000
```

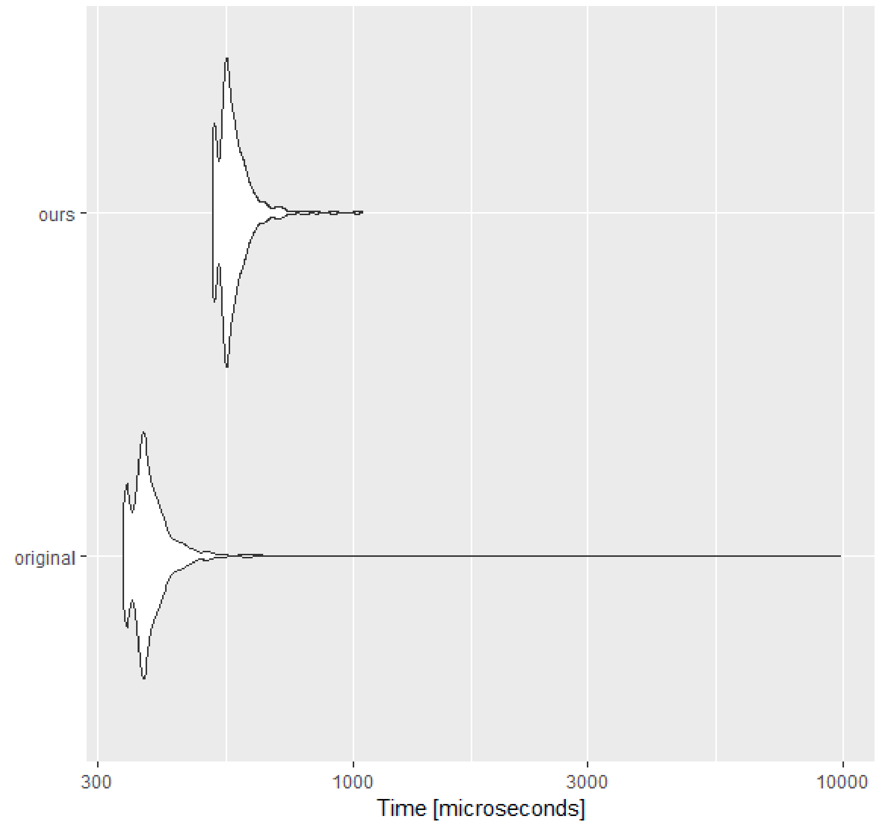


3.2.2 Case 2: Medium-sized Array

Medium-sized 6-dimension array of size 1MB, 1000 trials:

```
x <- array(1:(2**10), rep(2**3, times=6))
asub(x, 1, 1)
```

```
Unit: microseconds
     expr     min        lq      mean    median        uq      max neval
 original 337.676 362.0530 393.1370 375.5235 396.3075 9797.465  1000
     ours 514.725 541.0255 569.2024 555.0100 582.8500 1038.942  1000
```

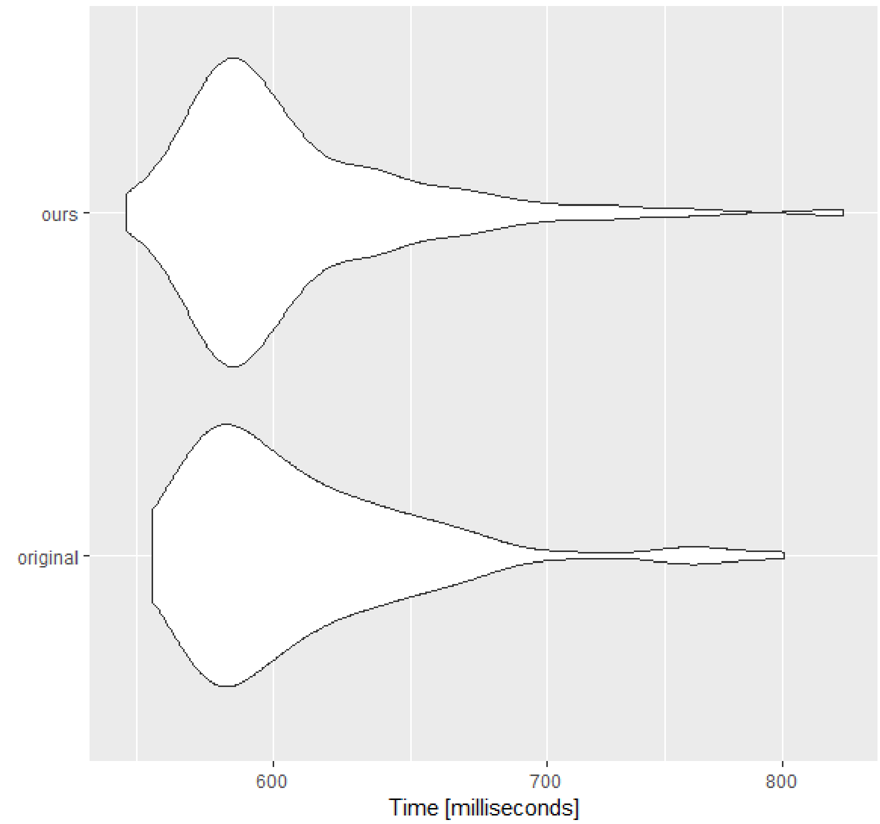


3.2.3 Case 3: High-Dimensionality Large Array

Large 25-dimension array of size 128MB, 100 trials:

```
x <- array(1:(2**25), rep(2, times=25))
asub(x, as.list(rep(1, 25)), 1:25)
```

```
Unit: milliseconds
     expr      min       lq     mean   median       uq      max neval
 original 560.0189 580.7659 611.0943 596.3183 629.2422 800.4877   100
     ours 551.9873 582.6090 608.4764 592.7386 624.3100 827.2007   100
```



3.3 acorn()

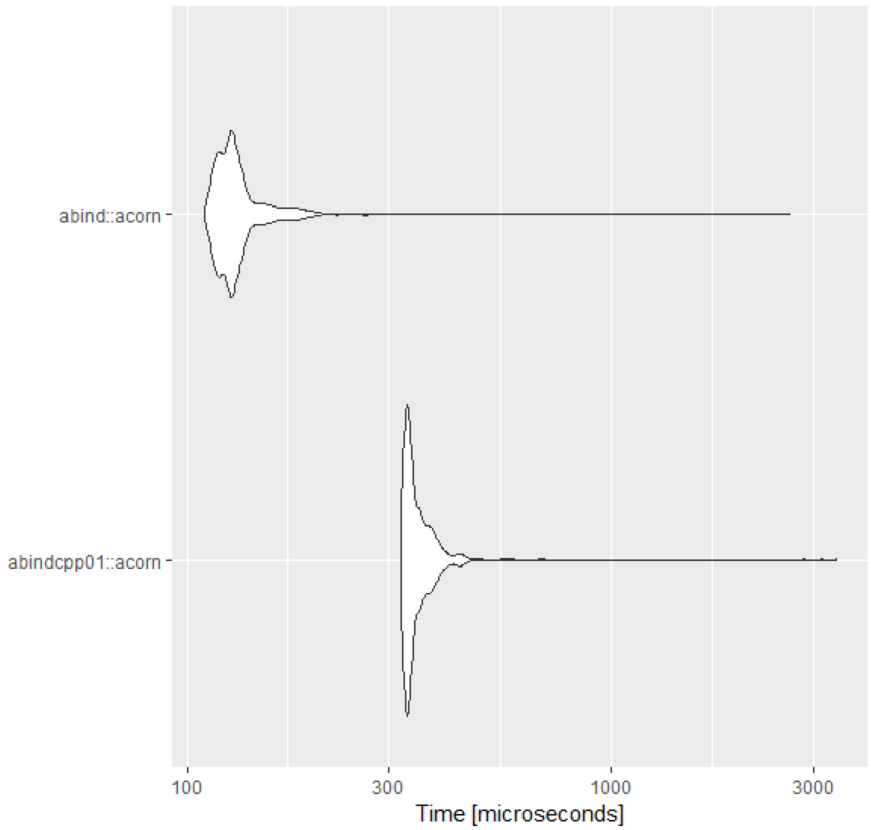
We refer to our implementation as *abindcpp01:acorn* and the original version/control as *abind:acorn*.

3.3.1 Case 1: Small-sized Array

Small-sized array, 1000 trials:

```
x <- array(1:24,dim=c(4,3,2))
acorn(x, -3)
```

```
Unit: microseconds
               expr     min       lq     mean   median        uq      max neval
 abindcpp01::acorn 317.918 326.386 360.3345 336.9065 368.8520 2697.299  1000
      abind::acorn 108.539 118.803 138.1707 127.1420 137.2775 2611.084  1000
```

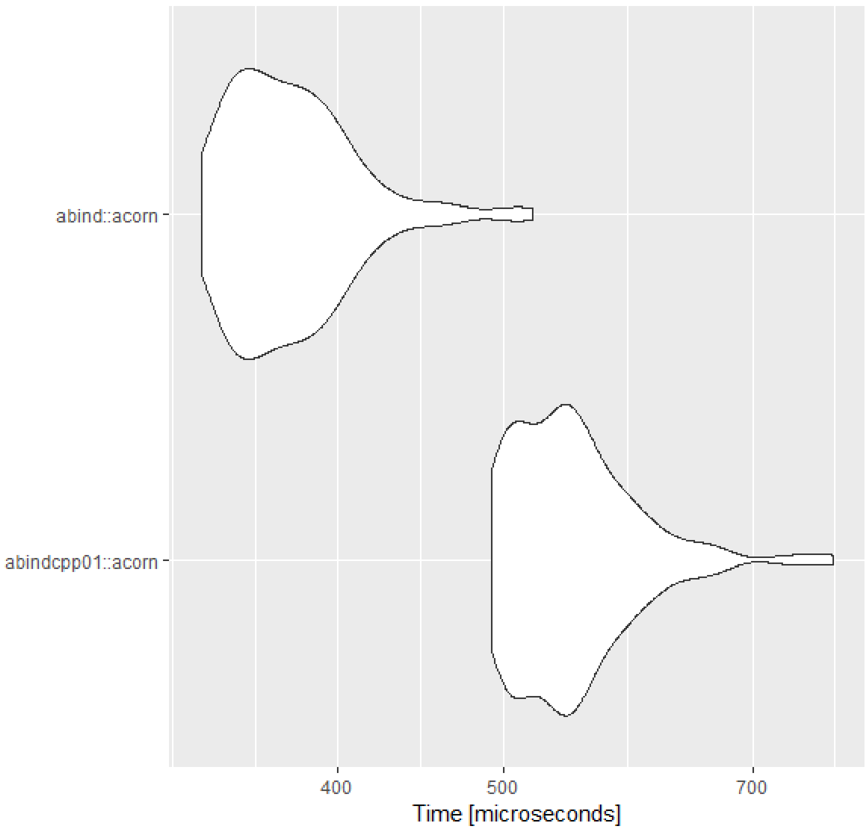


3.3.2 Case 2: Large Array, Simple Query

Large 25-dimension array of size 128MB, a basic asub() call, 100 trials:

```
x <- array(1:(2**25), rep(2, times=25))
acorn(x, 1, 2, -3)
```

```
Unit: microseconds
              expr     min        lq      mean    median        uq      max neval
 abindcpp01::acorn 489.322 498.4315 523.7388 508.8230 541.2820  647.126   100
      abind::acorn 329.979 341.1400 390.8394 349.9925 373.7275 3385.223   100
```

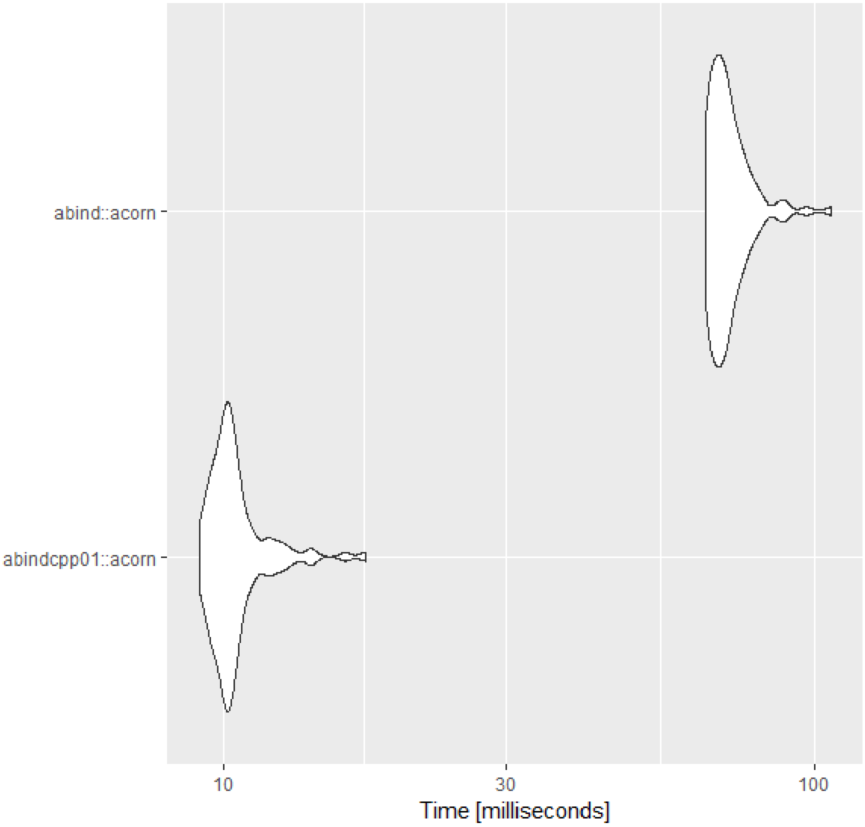


3.3.3 Case 3: Large Array, Complex Query

Large 25-dimension array of size 128MB, a more complex asub() call, 100 trials:

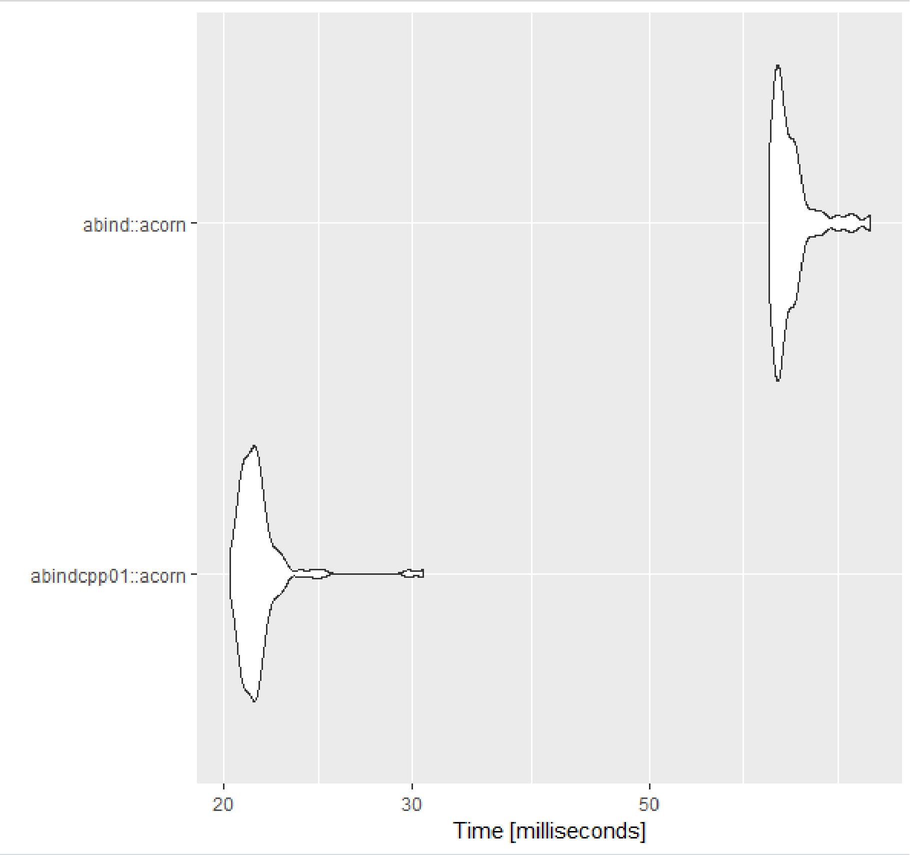
```
x <- array(1:(2**25), rep(2, times=25))
acorn(x, -3, 1, 2, 4, 5, 7, 9, -3, 1, 2, 4, 5, 7, 9, -3, 1, 2, 4,
5, 7, 9,-3, 1, 2, 4, 5, 7, 9,-3, 1, 2, 4, 5, 7, 9,-3, 1, 2, 4, 5, 7, 9)
```

```
Unit: milliseconds
              expr       min        lq     mean   median       uq      max neval
 abindcpp01::acorn  9.520858  9.811962 10.35923 10.08164 10.38878 16.52608   100
      abind::acorn 67.889450 69.604512 72.73413 70.66809 72.15222 96.52130   100
```


abind::acorn
abindcpp01::acorn
10
30
100
Time [milliseconds]

3.4 Case 4: acorn() using the default asub()

Because acorn() calls asub(), we tested our modified version of acorn() using the unmodified version of asub(). The results are of similar shape.



This is the only case where our implementation runs faster than the original.

4 Analysis

4.1 asub()

The asub function subsets arrays at indices in the specific dimensions. The original R code loops through each dimension. It constructs a string which is evaluated to subset the array. Therefore, we assumed that by looping through dimensions and constructing the string in C++, we would significantly increase performance. However, we failed to account for the time of making a C++ call. In addition to the overhead from using Rcpp the call requires passing potentially large lists and arrays to the C++ function. In the future, writing the entire function in C++ may avoid this issue. In addition, the dimensionality of an array is limited by the amount of memory R can allocate for a vector. We found that we could not construct an array with more than 25 dimensions. Thus, moving our for loop to C++ likely had little performance gain as the number of iterations would be relatively small and building a string in C++ is not much faster than building a string in R.

Despite these issues, we do see some small performance gain for high dimensionality arrays. From our benchmarking data, we determine that when the dimension is large, we overcome time lost in the Rcpp call for an overall improvement.

4.2 acorn()

Acorn returns a small corner of an array, taking some specified slices from each dimension. It handles one, two, three, and four dimension arrays as special cases outside of any loop. After four dimensions the R code loops through each dimension and constructs a list of elements. Therefore, we assumed that by looping through dimensions and constructing the list in C++, we would significantly increase performance. This is a similar strategy to that using in asub above. However, we achieved much better results. When using a large array and a complex query with a high number of specified slices as arguments we saw close to a 10 times performance increase. We believe this is because it was more computationally intensive to construct the matrix returned by acorn than the string in asub. However, our version of the acorn function is still slower for small matrices (as seen in the graphs). This can be explained by the overhead of making the Rcpp call. Also, in the case where the dimension of a matrix x is less than 4, it will be handled by R code, making no affect to performance when using acorn's R implementation.
After more testing, we could determine a threshold for the complexity of the call. Below the threshold, the original acorn would be used, and above the threshold out modified C++ acorn would be used. Thus, the performance of acorn would remain consistently high.

5 Exporting the Package

To export the package, we enter R's interactive mode and use devtools::build() then devtools::check(). The latter is analogous to R CMD check. devtools::build() recompiles all C++ code, and builds a tar.gz file for the package. If the check() commands succeeds, we can then use R CMD INSTALL abindcpp01$_1.0.tar.gztoinstallthepackagefromscratch.Wecopiedthetestsfromtheoriginalabindpackage, socheck()te$

6 Conclusion

In conclusion, we found that we could make improvements to the abind, especially for high dimension arrays, for the asub and acorn functions. Another function, afill, calls asub ans should thus also improve. Performance increased drastically in the case of acorn because the intensive list operation was shifted to a C++ function using Rcpp. Our C++ functions decreased performance in both functions for arrays with small dimensions. In the future, a threshold could be found to determine when a dimension is high enough to warrant use of an Rcpp call. This would increase the performance of the abind package overall.

7 Contribution

Reimplement asub(): E. Kristovich, C. Nagda
Reimplement acorn(): V. Nguyen, V. Nguyen
Benchmarking: R. Park
Report: all members

References

[1] CRAN - Package abind,
https://cran.r-project.org/web/packages/abind/index.html

8 Exporting Package Output

```
emkristo@ad3.ucdavis.edu@pc17:~/ECS145$ R CMD INSTALL abindcpp01_1.0.tar.gz
* installing to library ‘/home/emkristo/R/x86_64-pc-linux-gnu-library/3.4’
* installing *source* package ‘abindcpp01’ ...
** libs
g++  -I/usr/share/R/include -DNDEBUG  -I"/home/emkristo/R/x86_64-pc-linux-gnu-library/3.4/Rcpp/inc
g++  -I/usr/share/R/include -DNDEBUG  -I"/home/emkristo/R/x86_64-pc-linux-gnu-library/3.4/Rcpp/inc
g++  -I/usr/share/R/include -DNDEBUG  -I"/home/emkristo/R/x86_64-pc-linux-gnu-library/3.4/Rcpp/inc
g++  -I/usr/share/R/include -DNDEBUG  -I"/home/emkristo/R/x86_64-pc-linux-gnu-library/3.4/Rcpp/inc
g++  -I/usr/share/R/include -DNDEBUG  -I"/home/emkristo/R/x86_64-pc-linux-gnu-library/3.4/Rcpp/inc
g++  -I/usr/share/R/include -DNDEBUG  -I"/home/emkristo/R/x86_64-pc-linux-gnu-library/3.4/Rcpp/inc
g++ -shared -L/usr/lib/R/lib -Wl,-Bsymbolic-functions -Wl,-z,relro -o abindcpp01.so RcppExports.o
installing to /home/emkristo/R/x86_64-pc-linux-gnu-library/3.4/abindcpp01/libs
** R
** inst
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (abindcpp01)
```

```
  > compileAttributes()
> compileAttributes()
> load_all()
Loading abindcpp01
Re-compiling abindcpp01
  installing *source* package ‘abindcpp01’ ...
   ** libs
   g++  -I/usr/share/R/include -DNDEBUG  -I"/home/emkristo/R/x86_64-pc-linux-gnu-library/3.4/Rcpp/
   make: Warning: File 'RcppExports.cpp' has modification time 5.5 s in the future
   g++  -I/usr/share/R/include -DNDEBUG  -I"/home/emkristo/R/x86_64-pc-linux-gnu-library/3.4/Rcpp/
   g++ -shared -L/usr/lib/R/lib -Wl,-Bsymbolic-functions -Wl,-z,relro -o abindcpp01.so RcppExports
   make: warning:  Clock skew detected.  Your build may be incomplete.
   installing to /tmp/RtmpTvm9wP/devtools_install_45bc3c2be50c/abindcpp01/libs
  DONE (abindcpp01)
> build()
  checking for file ‘/home/emkristo/ECS145/abindcpp01/DESCRIPTION’ ...
  preparing ‘abindcpp01’: (4s)
  checking DESCRIPTION meta-information ...
  cleaning src
  checking for LF line-endings in source and make files and shell scripts
  checking for empty or unneeded directories
  building ‘abindcpp01_1.0.tar.gz’
   Warning: invalid uid value replaced by that for user 'nobody'
   Warning: invalid gid value replaced by that for user 'nobody'

[1] "/home/emkristo/ECS145/abindcpp01_1.0.tar.gz"
> check()
 Building  abindcpp01
Setting env vars:
 CFLAGS    : -Wall -pedantic -fdiagnostics-color=always
 CXXFLAGS  : -Wall -pedantic -fdiagnostics-color=always
 CXX11FLAGS: -Wall -pedantic -fdiagnostics-color=always

  checking for file ‘/home/emkristo/ECS145/abindcpp01/DESCRIPTION’ ...
  preparing ‘abindcpp01’: (2.8s)
  checking DESCRIPTION meta-information ...
  cleaning src
  checking for LF line-endings in source and make files and shell scripts
  checking for empty or unneeded directories
  building ‘abindcpp01_1.0.tar.gz’
   Warning: invalid uid value replaced by that for user 'nobody'
   Warning: invalid gid value replaced by that for user 'nobody'

 Checking  abindcpp01
Setting env vars:
 _R_CHECK_CRAN_INCOMING_USE_ASPELL_: TRUE
 _R_CHECK_CRAN_INCOMING_REMOTE_    : FALSE
 _R_CHECK_CRAN_INCOMING_           : FALSE
 _R_CHECK_FORCE_SUGGESTS_          : FALSE
 R CMD check
  using log directory ‘/tmp/RtmpTvm9wP/abindcpp01.Rcheck’ (1.9s)
  using R version 3.4.4 (2018-03-15)
  using platform: x86_64-pc-linux-gnu (64-bit)
  using session charset: UTF-8
  using options ‘--no-manual --as-cran’
  checking for file ‘abindcpp01/DESCRIPTION’
  checking extension type ... Package
  this is package ‘abindcpp01’ version ‘1.0’
  checking package namespace information
  checking package dependencies (1.3s)
  checking if this is a source package
  checking if there is a namespace
  checking for executable files ...
  checking for hidden files and directories
  checking for portable file names
  checking for sufficient/correct file permissions
  checking whether package ‘abindcpp01’ can be installed (18.2s)
  checking installed package size ...
  checking package directory ...
  checking DESCRIPTION meta-information ...
N  checking top-level files
   Non-standard file/directory found at top level:
     ‘core’
  checking for left-over files
  checking index information
  checking package subdirectories ...
  checking R files for non-ASCII characters ...
  checking R files for syntax errors ...
  checking whether the package can be loaded ...
  checking whether the package can be loaded with stated dependencies ...
  checking whether the package can be unloaded cleanly ...
  checking whether the namespace can be loaded with stated dependencies ...
  checking whether the namespace can be unloaded cleanly ...
  checking loading without being on the library search path ...
  checking dependencies in R code ...
  checking S3 generic/method consistency (780ms)
  checking replacement functions ...
  checking foreign function calls ...
  checking R code for possible problems (2.6s)
  checking Rd files ...
  checking Rd metadata ...
  checking Rd line widths ...
  checking Rd cross-references ...
W  checking for missing documentation entries ...
   Undocumented code objects:
     ‘acorn1sr’ ‘afillcpp’ ‘asubCpp’
   All user-level objects in a package should have documentation entries.
   See chapter ‘Writing R documentation files’ in the ‘Writing R
   Extensions’ manual.
  checking for code/documentation mismatches (711ms)
  checking Rd \usage sections (884ms)
  checking Rd contents ...
  checking for unstated dependencies in examples ...
  checking line endings in C/C++/Fortran sources/headers
  checking compiled code ...
  checking examples (732ms)
W  checking for unstated dependencies in ‘tests’ ...
   'library' or 'require' call not declared from: ‘abind’
  checking tests ...
> > library(abind) (435ms)o ‘abind.Rout.save’ ...
> > library(abind) (755ms)o ‘adrop.Rout.save’ ...
< >  (1.2s) ‘afill.Rout’ to ‘afill.Rout.save’ ...
> [1] "x" "y" (1.8s)ut’ to ‘asub.Rout.save’ ...
> > library(abind)Rout’ to ‘dnns.Rout.save’ ...
```

```
   See
     ‘/tmp/RtmpTvm9wP/abindcpp01.Rcheck/00check.log’
   for details.


 R CMD check results  abindcpp01 1.0
Duration: 34s

 checking for missing documentation entries ... WARNING
  Undocumented code objects:
    ‘acorn1sr’ ‘afillcpp’ ‘asubCpp’
  All user-level objects in a package should have documentation entries.
  See chapter ‘Writing R documentation files’ in the ‘Writing R
  Extensions’ manual.

 checking for unstated dependencies in ‘tests’ ... WARNING
  'library' or 'require' call not declared from: ‘abind’

 checking top-level files ... NOTE
  Non-standard file/directory found at top level:
    ‘core’

0 errors  | 2 warnings  | 1 note
```