STAT243 Problem set 5

Lei Zhang
ID:3033120716

October 19, 2017

1 Problem 2

The following examples shows how the integers are stored exactly in R.
$2 = (-1)^0 * 1.0 * 2^{1024-1023}$
$-3 = (-1)^1 * (2^1 + 2^0) = (-1)^1 * (1 + 2^{-1}) * 2^{1024-1023}$
$2^{53}-1 = 2^5 2 + 2^5 1 + ... + 2^1 + 2^0 = (-1)^0 * (1 + 2^{-1} + 2^{-2} + ... + 2^{-52}) * 2^{1075-1023}$

```
library(pryr)
bits(2)

## [1] "01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000"

bits(-3)

## [1] "11000000 00001000 00000000 00000000 00000000 00000000 00000000 00000000"

bits(2^53-1)

## [1] "01000011 00111111 11111111 11111111 11111111 11111111 11111111 11111111"
```

We then consider the situation when integers are larger than $2^{53} - 1$. The results below shows that $2^{53}, 2^{53} + 2$ can be represented exactly but $2^{53} - 1$ cannot.So we claim that the spacing of numbers of this magnitude is 2.

```
a <- 2^53-1
b <- 2^53
c <- 2^53 + 1
d <- 2^53 + 2

#find out how a,b,c,d are stored
identical(b-a,1)

## [1] TRUE
```

```
identical(c-b,1)
```

```
## [1] FALSE
```

```
identical(d-b,2)
```

```
## [1] TRUE
```

This is because
$2^{53} = (-1)^0 * (1.0) * 2^{1076-1023}$
$2^{53} + 2 = (-1)^0 * (1 + 2^{-52}) * 2^{1076-1023}$
They can be represented exactly.

```
bits(2^53)
```

```
## [1] "01000011 01000000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```
bits(2^53+2)
```

```
## [1] "01000011 01000000 00000000 00000000 00000000 00000000 00000000 00000001"
```

$2^{53} + 1 = (-1)^0 * (1 + 2^{-53}) * 2^{1076-1023}$

```
bits(2^53+1)
```

```
## [1] "01000011 01000000 00000000 00000000 00000000 00000000 00000000 00000000"
```

2^{-54} is less than 2^{-53} the minimum number that the computer can store.
So it can not be stored exactly.
For numbers staring with 2^{54}
$2^{54} = (-1)^0 * (1.0) * 2^{1077-1023}$
$2^{54} + 1 = (-1)^0 * (1 + 2^{-54}) * 2^{1077-1023}$
$2^{54} + 2 = (-1)^0 * (1 + 2^{-53}) * 2^{1077-1023}$
$2^{54} + 3 = (-1)^0 * (1 + 2^{-53} + 2^{-54}) * 2^{1077-1023}$
$2^{54} + 4 = (-1)^0 * (1 + 2^{-52}) * 2^{1077-1023}$

```
bits(2^54)
```

```
## [1] "01000011 01010000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```
bits(2^54+1)
```

```
## [1] "01000011 01010000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```
bits(2^54+2)
```

```
## [1] "01000011 01010000 00000000 00000000 00000000 00000000 00000000 00000000"
```

```
bits(2^54+3)

## [1] "01000011 01010000 00000000 00000000 00000000 00000000 00000000 00000001"

bits(2^54+4)

## [1] "01000011 01010000 00000000 00000000 00000000 00000000 00000000 00000001"
```

Since 2^{-53},2^{-54} are smaller than 2^{-53}, we can't precisely represent $2^{54}+1$,$2^{54}+2$, $2^{54}+3$. So the spacing of numbers of this magnitude is 4.

2 Problem 3

2.1 Problem a

The following results show that even if it takes longer to create a numeric vector, it doesn't imply it takes significantly longer to copy a numeric vector. Based on the result of rbenchmark and system.time, we see that it is a bit faster to copy a integer vector than a numeric vector. Note that if we directly copy a vector using syntax like "a¡-integervec", no actual copy will be made.

```
p <- 1e7

copy <- function(x){
  vectorcopy<-vector()
  vectorcopy<-c(vectorcopy,x)
}

integervec <- 1:p
numericvec <- rnorm(p)

library(rbenchmark)
benchmark(copy(integervec),copy(numericvec),replications=50)
##test replications elapsed relative user.self sys.self user.child sys.child
##1 copy(integervec)           50    5.82    1.000      4.38     1.41         NA        NA
##2 copy(numericvec)           50    7.91    1.359      4.99     2.89         NA        NA




system.time(integervec <- 1:p)
##user system elapsed
  0.02   0.03    0.04

system.time(numericvec <- rnorm(p))
##user system elapsed
```

```
  3.76   0.00    3.78

system.time(vector1 <- c(vector1,integervec))
##user system elapsed
  0.08   0.03    0.11

system.time(vector2 <- c(vector2,numericvec))
##user system elapsed
  0.06   0.06    0.12
```

2.2 Problem b

The following results show that it is not necessarily faster to take a subset of size roughly n/2 from an integer vector of size n than from a numerica vector of size n. The time takes is about the same.

```
#This tests the time difference when taking the first half of each vector.

takehalf <- function(x){
  vectorhalf <- x[1:round(p/2)]
}
benchmark(takehalf(integervec),takehalf(numericvec),replications=30)
## test replications elapsed relative user.self sys.self user.child sys.child
##1 takehalf(integervec)           30    4.84    1.000      3.67     1.14          NA
##2 takehalf(numericvec)           30    4.96    1.025      3.39     1.52          NA


takeodd <- function(x){
  vectorhalf <- x[c(TRUE,FALSE)]
}
benchmark(takeodd(integervec),takeodd(numericvec),replications=30)
##                 test replications elapsed relative user.self sys.self user.child sys.chi
##1 takeodd(integervec)           30   28.15    1.000     27.27     0.81          NA        1
##2 takeodd(numericvec)           30   28.62    1.017     27.27     1.26          NA        1

system.time(integervec[1:round(p/2)])
##user system elapsed
  0.15   0.03    0.19

system.time(numericvec[1:round(p/2)])
##user system elapsed
  0.13   0.06    0.19

#This tests the time difference when taking the vector elements with odd indices.
system.time(integervec[c(TRUE, FALSE)])
```

```
##user system elapsed
  0.98   0.02    1.01

system.time(numericvec[c(TRUE, FALSE)])
##user system elapsed
  0.92   0.04    0.99
```

3 Problem 4

3.1 Problem a

I think there are several reasons for this.

First, if we break up Y into n columns, it's possible that the overhead of threading outweights the gains from distributing the computations.

Second, it can avoid the possible problem of adding or subtracting numbers that are very different in magnitude, i.e. the partial sum may be much larger than the new term if we add the $X * Y_i$ one by one. Breaking up Y into p blocks is kind of like doing the summation in a tree like fashion.

3.2 Problem b

Based on the result of calculation below, I think that Approach A is more communication efficient and Appraoch B is more memory efficient

Amount of memeory used when all p workers are doing their calculations:
Approach A:$p * (n * n + m * n + n * m) = (1 + 2/p)n^2$
Approach B:$p * (m * n + m * n + m * m) = (1/p^2 + 2/p) * n^2$

Communication cost:
Total number of numbers passed to the worker:
Approach A:$p * (n * n + n * (n/p)) = (p + 1) * n^2$
Approach B:$p^2 * (n * m + n * m) = 2 * p * n^2$
Total number of numbers passed to the master:
Approach A:$p * n * (n/p) = n^2$
Approach B:$p^2 * (n/p)^2 = n^2$
Total communication cost:
Approach A:$(p + 1) * n^2 + n^2 = (p + 2)n^2$
Approach B:$2 * p * n^2 + n^2 = (2p + 1) * n^2$

4 Problem 5

Since 0.3 can not be represented exactly in R, I think it is simply a coincidence that 0.2+0.3==0.5 is TRUE while 0.3==0.2+0.1 is FALSE. It is probably because the difference between 0.2+0.3 and 0.5 in terms of their storage format in

R is less than the minimum value that R can distinguish, while the difference between 0.2+0.1 and 0.3 is a bit larger and can be detected by R.