Problem Set 5

Dongyu Lang
SID: 24174288

October, 17, 2017

1 Question 2

```
bits(2^0)
```

```
## [1] "00111111 11110000 00000000 00000000 00000000 00000000 00000000 00000000"
```

01111111111 in binary system is 1023 in decimal number; thus, $2^0 \times 1 = 1$

```
bits(2^1)
```

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

10000000000 in binary system is 1024 in decimal; thus, $2^1 \times 1 = 2$

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

```
## [1] "01000000 00010000 00000000 00000000 00000000 00000000 00000000 00000000"
```

10000000001 in binary system is 1025 in decimal; thus, $2^2 \times 1 = 4$

```
bits(10)
```

```
## [1] "01000000 00100100 00000000 00000000 00000000 00000000 00000000 00000000"
```

10000000010 in binary system is 1026 in decimal; 1.01 in binary system is 1.25 in decimal; thus, $2^3 \times 1.25 = 10$

For 2^{53} and $2^{53} + 2$

```
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"
```

They can be represented exactly. 10000110100 is the binary format of 1076, thus, $2^{53} \times 1 = 2^{53}$. For $2^{53}+2$, the last digit is 1, thus, $2^{53} \times (1+2^{-52}) = 2^{53}+2$.

For $2^{53} + 1$

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

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

It is the same as 2^{53}, because the minimum increment from 2^{53} is $2^{53} \times 2^{-52} = 2$; thus, the spacing of numbers of this magnitude is 2.

For 2^{54}, $2^{54} + 1$, $2^{54} + 2$, $2^{54} + 3$:

```
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"
```

We can notice that the binary format of 2^{54} is the same as those of $2^{54} + 1$ and $2^{54} + 2$. And the binary format of $2^{54} + 3$ is different from those three. Because the minimum increment from 2^{54} is $2^{54} \times 2^{-52} = 4$; thus, the spacing of numbers of this magnitude is 4.

For $2^{53} - 1$, 2^{53}, $2^{53} + 1$:

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

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

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

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

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

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

```
options(digits = 22)
2^53-1
```

```
## [1] 9007199254740991

2^53

## [1] 9007199254740992

2^53+1

## [1] 9007199254740992
```

The first two results shows that they follow the expression $(-1)^S \times 1.d \times 2^{e-1023}$. The last one is the same as 2^{53}, because the spacing is 2. And the calculation of these numbers also proves the result.

2 Question 3 (a)

I created a variable called int that contained all the integers from 1 to 10^8, and a variable called num that contained all the numerics from 1 to 10^8, and create a copy of both variables.

```
options(digits = 9)
## create intergers and a copy
int = seq(1:1e8)
int_copy = int
microbenchmark(int_copy[1] <- as.integer(3))

## Unit: nanoseconds
##                          expr min  lq       mean median    uq       max
##  int_copy[1] <- as.integer(3) 710 749 2692241.51    820 945.5 269091896
##  neval
##    100

## create numerics and a copy
num = as.numeric(int)
num_copy = num
microbenchmark(num_copy[1] <- 3)

## Unit: nanoseconds
##              expr min    lq       mean median    uq        max neval
##  num_copy[1] <- 3 569 582.5 26511804.6    645 759.5 2651111891   100
```

It is clear that it takes longer time to copy the numeric values than for integer values.

3 Question 3 (b)

```
## create intergers and take a subset
int_new = seq(1:1e7)
microbenchmark(int_half <- int_new[1:5e6])

## Unit: milliseconds
##                          expr       min         lq       mean     median
##  int_half <- int_new[1:5e+06] 38.200931 57.0956665 89.8801799 63.1128705
##          uq        max neval
##  122.762329 269.440762   100

## create numerics and take a subset
num_new = as.numeric(int_new)
microbenchmark(num_half <- num_new[1:5e6])

## Unit: milliseconds
##                          expr       min       lq       mean    median
##  num_half <- num_new[1:5e+06] 52.842301 63.25963 106.569573 74.851659
##          uq        max neval
##  138.215632 347.606799   100
```

From the results, it takes a bit longer to take a subset from numeric vector than integer vector.

4 Question 4(a)

The reason that it might be better to use p blocks than n blocks is that often the number of cores we have are less than the number of columns; thus, it is better to have one task per core, that is $m = n/p$ columns per core. In addition, for some small matrices and vectors, a single thread will outperform multiple threads. Moreover, the communication cost is higher in some situation for n blocks.

5 Question 4(b)

Approach A has a larger memory use than approach B, because at any single moment, when all p workers are doing their calculations, approach A uses p times more memory use because A has p times larger matrix calculation.

Approach A has a smaller communication cost than approach B, because the total number of communication of A is p, but the total number of communication of B is p^2.

6 Question 5

```r
options(digits = 22)
0.1
```

```
## [1] 0.10000000000000055511
```

```r
0.2
```

```
## [1] 0.20000000000000111022
```

```r
0.3
```

```
## [1] 0.29999999999999888978
```

```r
0.1+0.2
```

```
## [1] 0.30000000000000444089
```

The reason is that 0.1 can not be represented exactly. We can notice that the 22-digit 0.1 is slight higher than true 0.1, and 0.2 is just 2 times 22-digit 0.1; thus, adding 0.1 and 0.2 together will make the result to be higher than 22-digit 0.3.

7 Collaboration

In this problem set, I worked with Gaowei Chen.