STAT243 PS5

Zicheng Huang

10/18/2017

```
library(data.table)
```

2.

We first show that the integers $1, 2, 3, \cdots, 2^{53}-2, 2^{53}-1$ can be stored exactly in the $(-1)^S \times 1.d \times 2^{e-1023}$ format. Since d is represented using 52 bits, the precision is up to 2^{-52}, which means that any d that involves 2^{-x} where $x > 52$ cannot be represented in the above format.

$$
\begin{aligned}
1 &= 2^0 = (-1)^0 \times (2^0) \times 2^0 \\
2 &= 2^1 = (-1)^0 \times (2^0) \times 2^1 \\
3 &= 2^1 + 1 = 2^1 + 2^0 = (-1)^0 \times (2^0 + 2^{-1}) \times 2^1 \\
4 &= 2^2 = (-1)^0 \times (2^0) \times 2^2 \\
5 &= 2^2 + 1 = 2^2 + 2^0 = (-1)^0 \times (2^0 + 2^{-2}) \times 2^2 \\
&\vdots \\
2^{53} - 2 &= (2^1 - 2^{-51}) \times 2^{52} = \underbrace{(-1)^0}_{(-1)^S} \times \underbrace{(2^0 + 2^{-1} + 2^{-2} + \cdots + 2^{-51})}_{1.d} \times \underbrace{2^{52}}_{2^{e-1023}} \\
2^{53} - 1 &= (2^1 - 2^{-52}) \times 2^{52} = \underbrace{(-1)^0}_{(-1)^S} \times \underbrace{(2^0 + 2^{-1} + 2^{-2} + \cdots + 2^{-52})}_{1.d} \times \underbrace{2^{52}}_{2^{e-1023}}
\end{aligned}
$$

In this way, we have shown that the integers $1, 2, 3, \cdots, 2^{53}-2, 2^{53}-1$ can be stored exactly in the $(-1)^S \times 1.d \times 2^{e-1023}$ format.

Then we show that 2^{53} and $2^{53}+2$ can be represented exactly as follows:

$$
\begin{aligned}
2^{53} &= (2^0) \times 2^{53} = \underbrace{(-1)^0}_{(-1)^S} \times \underbrace{(2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + \cdots + 0 \cdot 2^{-52})}_{1.d} \times \underbrace{2^{53}}_{2^{e-1023}} \\
2^{53} + 2 &= (2^0 + 2^{-52}) \times 2^{53} = \underbrace{(-1)^0}_{(-1)^S} \times \underbrace{(2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + \cdots + 0 \cdot 2^{-51} + 1 \cdot 2^{-52})}_{1.d} \times \underbrace{2^{53}}_{2^{e-1023}}
\end{aligned}
$$

Below we show that $2^{53}+1$ cannot be represented exactly:

$$
2^{53} + 1 = (2^0 + 2^{-53}) \times 2^{53} = \underbrace{(-1)^0}_{(-1)^S} \times (2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + \cdots + 0 \cdot 2^{-52} + 1 \cdot 2^{-53}) \times \underbrace{2^{53}}_{2^{e-1023}}
$$

This above expression shows that $2^{53}+1$ involves the precision up to 2^{-53}, which exceeds the precision of d represented by 52 bits. Thus, the next integer after 2^{52} than can be represented exactly is $2^{52}+2$, so the spacing of numbers of this magnitude is 2. This is consistent with the fact that, with a machine epsilon equals to 2^{-52}, the absolute spacing starting at 2^{53} is $2^{53} \times 2^{-52} = 2$.

Now we need to show that for numbers starting with 2^{54} that the spacing between integers that can be represented exactly is 4.

$$2^{54} = (2^0) \times 2^{54} = \underbrace{(-1)^0}_{(-1)^S} \times \underbrace{(2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + \cdots + 0 \cdot 2^{-52})}_{1.d} \times \underbrace{2^{54}}_{2^{e-1023}}$$

$$2^{54}+1 = (2^0 + 2^{-54}) \times 2^{54} = \underbrace{(-1)^0}_{(-1)^S} \times (2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + \cdots + 0 \cdot 2^{-52} + 1 \cdot 2^{-54}) \times \underbrace{2^{54}}_{2^{e-1023}}$$

$$2^{54}+2 = (2^0 + 2^{-53}) \times 2^{54} = \underbrace{(-1)^0}_{(-1)^S} \times (2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + \cdots + 0 \cdot 2^{-52} + 1 \cdot 2^{-53}) \times \underbrace{2^{54}}_{2^{e-1023}}$$

$$2^{54}+3 = (2^0 + 2^{-53} + 2^{-54}) \times 2^{54} = \underbrace{(-1)^0}_{(-1)^S} \times (2^0 + 0 \cdot 2^{-1} + \cdots + 0 \cdot 2^{-52} + 1 \cdot 2^{-53} + 1 \cdot 2^{-54}) \times \underbrace{2^{54}}_{2^{e-1023}}$$

$$2^{54}+4 = (2^0 + 2^{-52}) \times 2^{54} = \underbrace{(-1)^0}_{(-1)^S} \times \underbrace{(2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + \cdots + 0 \cdot 2^{-51} + 1 \cdot 2^{-52})}_{1.d} \times \underbrace{2^{54}}_{2^{e-1023}}$$

As we can see from the results above, $2^{54}+1, 2^{54}+2, 2^{54}+3$ cannot be represented exactly since they involves precision up to 2^{-53} and 2^{-54} which exceeds the precision, 2^{-52}, of d. Therefore, the next integer after 2^{54} that can be represented exactly is $2^{54}+4$. This is consistent with the fact that, with a machine epsilon equals to 2^{-52}, the absolute spacing starting at 2^{54} is $2^{54} \times 2^{-52} = 2^2 = 4$.

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

## [1] 9007199254740991

# exactly represented
2^53

## [1] 9007199254740992

# not exactly represented
2^53+1

## [1] 9007199254740992
```

3.(a)

```
options(digits = 5)
# create a large vector of integers
int <- as.integer(round(rnorm(100000000)))

# create a numeric vector of same length
num <- rnorm(100000000)

# time of making a copy of the large vector of integers
system.time(intCopy <- copy(int))

##    user  system elapsed
##    0.06    0.05    0.10

# time of making a copy fo the numeric vector
system.time(numCopy <- copy(num))

##    user  system elapsed
##    0.09    0.12    0.21
```

As we can see from the result above, it is faster to copy a large vector of integers than a numeric vector of the same length in R. More specifically, time to make a copy of a large vector of integers is approximately half the time to make a copy of a numeric vector of the same length. This is because integers are stored in single precision floating point taking up 4 bytes whereas numeric values are stored in double precision floating point taking up 8 bytes.

3.(b)

```
# size of the subset
k <- length(int)/2

# time needed to take a subset of size k from an integer vector
system.time(sample(int, size = k, replace = FALSE))

##    user  system elapsed
##    8.69    0.21    8.89

# time needed to take a subset of size k from a numeric vector
system.time(sample(num, size = k, replace = FALSE))

##    user  system elapsed
##    8.66    0.28    8.94
```

As we can see from the result above, it takes approximately the same amount of time to get a subset of size $k \approx \frac{n}{2}$ from an integer vector of size n as getting a subset from a numeric vector of size n.

4.(a)

First of all, splitting into n tasks to do parallelization and combining the n total results from each task would be time-consuming and inefficient if n is large. Splitting into $m = \frac{n}{p}$ blocks of columns, instead, would certainly reduce the communication cost. Moreover, if we split Y into n individual columns, some columns of Y might take up more time than the others when doing matrix multiplication with X where the parallelization would wait until the most time-consuming task is finished, which in some cases might not be a lot faster than splitting into m tasks counting the fact about communication cost.

4.(b)

(i). Approach B is better for minimizing memory use.

At any single moment in time, each worker in Approach A is dealing with matrix multiplication involving an $n \times n$ matrix and an $n \times m$ matrix. The resulting matrix would have dimension of $n \times m$. As a result, the amount of memory used for each worker would be $n \times n + n \times m + n \times m = n^2 + n \times \frac{n}{p} + n \times \frac{n}{p} = n^2 + \frac{n^2}{p} + \frac{n^2}{p}$. Since there are p workers in total, then at any single moment in time, Approach A uses up $(n^2 + \frac{n^2}{p} + \frac{n^2}{p})p = n^2p + n^2 + n^2$ amount of memory.

In comparison, each worker in Approach B is dealing with matrix multiplication involving an $m \times n$ matrix and an $n \times m$ matrix. The resulting matrix would have dimension of $m \times m$. As a result, the amount of memory used for each worker would be $m \times n + n \times m + m \times m = \frac{n}{p} \times n + n \times \frac{n}{p} + \frac{n}{p} \times \frac{n}{p} = \frac{n^2}{p} + \frac{n^2}{p} + \frac{n^2}{p^2}$. Since there are p workers in total, then at any single moment in time, Approach B uses up $(\frac{n^2}{p} + \frac{n^2}{p} + \frac{n^2}{p^2})p = n^2 + n^2 + \frac{n^2}{p}$ amount of memory, which is smaller than that of Approach A.

(ii). Approach A is better for minimizing communication cost.

For Approach A, we pass in an $n \times n$ matrix and an $n \times m$ matrix to each of the p workers for one time. Then the passing-in communication cost would be $(n \times n + n \times m)p = (n^2 + n \times \frac{n}{p})p = n^2p + n^2$. After the p workers have finished their jobs, each of them needs to pass out an $n \times m$ matrix back to the master for one time. The resulting passing-out communication cost would be $(n \times m)p = (n \times \frac{n}{p})p = n^2$. Therefore, the total communication cost for Approach A is $n^2p + n^2 + n^2$.

For Approach B, we pass in an $m \times n$ matrix and an $n \times m$ matrix to each of the p workers for p times. Then the passing-in communication cost would be $[(m \times n + n \times m)p]p = [(\frac{n}{p} \times n + n \times \frac{n}{p})p]p = n^2p + n^2p$. After the p workers have finished their jobs, each of them needs to pass out an $m \times m$ matrix back to the master for p times. The resulting passing-out communication cost would be $[(m \times m)p]p = [(\frac{n}{p} \times \frac{n}{p})p]p = n^2$. Therefore, the total communication cost for Approach B is $n^2p + n^2p + n^2$.

Comparing the communication cost for the two approaches, we can see that the communication cost for Approach A is smaller than that of Approach B in that p is always greater than or equal to one.

5.

Since 0.5 is a power of 2, then 0.5 can be represented exactly in binary form. When doing the calculation of $0.2 + 0.3$, 0.2 is the decimal number that is closest to 0.2, which is not 0.2 itself, and 0.3 is the decimal number that is closest to 0.3, which is also not 0.3 itself, in that 0.2 and 0.3 cannot be represented exactly. The sum of these two numbers would be the decimal that is closest to 0.5, which would be 0.5 itself since it can be represented exactly. So I think that addition and subtraction that involves number that is some power of 2 will be true. $0.1 + 0.2 == 0.3$ does not involve number as power of 2 and is false in that $0.1 + 0.2$ returns the number that is closest to 0.3, not 0.3 itself.