STAT 243 PS 5

Junyuan Gao(SID:26484653)

October 18, 2017

1 Q2

(1)Every integers $1, 2, 3, ..., 2^{53}-2, 2^{53}-1$ can be stored exactly as $(-1)^s \times 1.d \times 2^{e-1023}$

For example:

$$1 = (-1)^2 \times 1.0 \times 2^{1023-1023}$$
$$2 = (-1)^2 \times 1.0 \times 2^{1024-1023}$$
$$3 = (-1)^2 \times 1.5 \times 2^{1024-1023}$$
$$5 = (-1)^2 \times 1.25 \times 2^{1025-1023}$$
$$.....$$
$$2^{53}-2 = (-1)^2 \times (2-2^{-51}) \times 2^{1075-1023}$$
$$2^{53}-1 = (-1)^2 \times (2-2^{-52}) \times 2^{1075-1023}$$

where $(2-2^{-52})$ and $(2-2^{-51})$ can be expressed as $1.d$ where d is represented as 52 bits.

(2) First, note that in the expression $(-1)^s \times 1.d \times 2^{e-1023}$, e is any integer in $[0, 2048]$, which implies that if $c < -52$, 2^c can't be exactly represented. Since

$$2^{53} = (-1)^2 \times 1.0 \times 2^{1076-1023}$$

and

$$2^{53}+2 = (-1)^2 \times (1+2^{-52}) \times 2^{1076-1023}$$

thus they can be represented exactly since $(1+2^{-52}) = 1.d$ where d is represented as 52 bits. However,

$$2^{53}+1 = (-1)^2 \times (1+2^{-53}) \times 2^{1076-1023}$$

Since 2^{-53} can't be exactly represented, $2^{53}+1$ can't be represented exactly. Therefore, the spacing of numbers of this magnitude is 2.

(3)

$$2^{54} = (-1)^2 \times 1.0 \times 2^{1077-1023}$$
$$2^{54} + 1 = (-1)^2 \times (1 + 2^{-54}) \times 2^{1077-1023}$$
$$2^{54} + 2 = (-1)^2 \times (1 + 2^{-53}) \times 2^{1077-1023}$$
$$2^{54} + 3 = (-1)^2 \times (1 + 1.5 \times 2^{-53}) \times 2^{1077-1023}$$
$$2^{54} + 4 = (-1)^2 \times (1 + 2^{-52}) \times 2^{1077-1023}$$

Similar to (2), $2^{54}+1, 2^{54}+2, 2^{54}+3$ can't be represented exactly since 2^{-53} and 2^{-54} can't be represented exactly, and $2^{54}, 2^{54}+4$ can be represented exactly because they are of the form $(-1)^s \times 1.d \times 2^{e-1023}$ where d is represented as 52 bits.
Thus, for numbers starting with 2^{54}, the spacing between integers that can be represented exactly is 4.

```
2^53-1
```

```
## [1] 9.007199e+15
```

```
2^53
```

```
## [1] 9.007199e+15
```

```
2^53+1
```

```
## [1] 9.007199e+15
```

2 Q3

2.1 3a

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

```
## Warning:  package 'data.table' was built under R version 3.4.2
```

```
library(microbenchmark)
```

```
## Warning:  package 'microbenchmark' was built under R version 3.4.2
```

```
x=rnorm(1e8)
#force(y=x)
microbenchmark(copy(x))
```

```
## Unit: milliseconds
##     expr      min       lq     mean   median       uq      max neval
##  copy(x) 272.9893 349.4846 351.1381 351.8073 353.9455 365.5236   100
```

```
xx= as.integer(round(x))
microbenchmark(copy(xx))

## Unit: milliseconds
##      expr      min       lq    mean   median       uq      max neval
##  copy(xx) 113.2278 130.8354 181.079 174.6091 221.0627 327.3734   100
```

Reason: Numerics stored in double precision takes maximum 8 bytes of storage and integers stored in single precision takes maximum 4 bytes of storage. Thus, copy a large integer vector is 2 times faster than copy a numeric vector of same size.

2.2 3b

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

y=rnorm(1e6)
yy= as.integer(round(y))
microbenchmark(sample(y, size=1/2*length(y)))

## Unit: milliseconds
##                                expr      min       lq     mean   median
##  sample(y, size = 1/2 * length(y)) 25.19933 31.43512 36.36862 34.19329
##        uq      max neval
##  37.57103 161.2884   100

microbenchmark(sample(yy, size=1/2*length(yy)))

## Unit: milliseconds
##                                  expr      min       lq     mean   median
##  sample(yy, size = 1/2 * length(yy)) 21.52927 25.71864 28.65928 27.84105
##       uq      max neval
##  31.1657 44.90878   100
```

When taking a subset of $k \approx n/2$ from an integer vector of size n, it will be a little faster than taking a size k subset of a numeric vector at same size.

3 Q4

3.1 4a

(1)Due to different sparse level of matrices, some matrix may have some columns dense but other columns very sparse. If breaking up into n computations, the computing time on dense columns will be much longer than sparse columns and

may not be significant slower than the computation time when breaking into p computations. Therefore, breaking up into n computations may not perform faster than breaking up into p computations.

(2)The process that breaking up the matrix into n blocks will take longer time than breaking up into p blocks.

(3)Storing information of n blocks take more memory than storing info of p blocks.

Therefore, it might be better to break up Y into p blockss rather than into n individual column-wise computations since breaking into n blocks is much harder and may not have significant improvement than breaking into n blocks.

3.2 4b

Approach B is better for minimizing memory use and Approach A is better for minimizing communication.
Explanation:
(1)The memory used at single moment consist of memory used in storing matrices for computations and the result.
The amount of memory used at any single moment in time of approach A is approximately

$$(n \times n + n \times \frac{n}{p} + n \times \frac{n}{p}) \times p = n^2 p + 2n^2$$

where the amount of memory used at any single moment in time of approach B is approximately

$$(\frac{n}{p} \times n + n \times \frac{n}{p} + m \times m) \times p = 2n^2 + \frac{n^2}{p}$$

Since $n^2 p > \frac{n^2}{p}$, Apprach B is better for minimizing memory use.

(2)Total number of numbers that need to be passed to the workers and the numbers passed from the workers back to the master of Approach A is approximately

$$\#\text{Pass To Worker} = (n \times n + n \times \frac{n}{p}) \times p = n^2 p + n^2$$

$$\#\text{Send Back} = (n \times \frac{n}{p}) p = n^2$$

and that of Approach B is approximately

$$\#\text{Pass To Worker} = [(\frac{n}{p} \times n + n \times \frac{n}{p}) p] \times p = 2n^2 p$$

$$\#\text{Send Back} = [(m \times m) p] \times p = n^2$$

Since communication cost = #Pass To Worker + #Send Back, so we get communication cost of A is $n^2p+n^2+n^2 = n^2p+2n^2$ and that of B is $2n^2p+n^2$ Since $n^2p > n^2$, Apprach A is better for minimizing communication cost.

4 Q5

Explanation: As we know, any number that equals to power of 2 can be accurately expressed in R(e.g. 0.5, 1, 2, 4...), so any addition with that kind of number involved in, e.g. 0.2+0.3=0.5, will be accurately expressed, thus 0.3+0.2==0.5 and 0.01+0.49==0.5 are true in R. We can show this in 2 cases:

(1)Addition like 0.2+0.3==0.5: Since R first find the expression "0.2" as the nearest number to 0.2, and 0.5 can be accurately expressed, so the summand "0.3" must be the nearest number to 0.3, so the evaluation will return True.

(2)Addition like 0.1+0.5==0.6: Since R first find the nearest number to 0.1, and 0.5 can be accurately expressed, then the R expression of 0.6(the nearest number to 0.6) is same to "the nearest number to 0.1 + accurate 0.5", which ensures the return value to be True.

However, when evaluating 0.3==0.2+0.1, "0.1" is expressed as nearest number to 0.1 in R and "0.2" is expressed as nearest number to 0.2 in R, but "0.1"+"0.2" may not equal to the nearest number to 0.3, which is "0.3". Thus, it returns False.