Stat243: Problem Set 5

Linqing Wei

October 17, 2017

Problem 2

$$
\begin{aligned}
1 &= 1.0 * 2^{1023-1023} \\
2 &= 1.0 * 2^{1024-1023} \\
3 &= 1.5 * 2^{1024-1023} \\
4 &= 1.0 * 2^{1025-1023} \\
2^{53} - 2 &= 1.0 * (2^{1076-1023}) - 1.0 * (2^{1024-1023}) \\
2^{53} &= 1.0 * (2^{1076-1023}) \\
2^{53} + 1 &= 1.0 * (2^{1076-1023}) + 1.0 * (2^{1023-1023}) \\
2^{53} + 2 &= 1.0 * (2^{1076-1023}) + 1.0 * (2^{1024-1023})
\end{aligned}
$$

```
> library(pryr)
> 2^53

[1] 9.007199e+15

> 2^53 + 1

[1] 9.007199e+15

> 2^53 + 2

[1] 9.007199e+15

> bits(2^53) == bits(2^53 + 1 )

[1] TRUE

> bits(2^53) == bits(2^53 + 2)

[1] FALSE
```

Since the bits of 2^{53} are equal to the bits of $2^{53}+1$, it means $2^{53}+1$ cannot be represented exactly. Since the bits of 2^{53} are not equal to the bits of $2^{53}+2$, it means $2^{53}+2$ is unique, CAN be represented exactly.

```
> 2^53

[1] 9.007199e+15

> 2^53 -1

[1] 9.007199e+15

> 2^53 + 1

[1] 9.007199e+15

> bits(2^53) == bits(2^53 - 1)

[1] FALSE

> bits (2^53) == bits(2^53 +1)

[1] TRUE
```

2^{53} can be represented exactly, and $2^{53}-1$ can be represented exacly since it's less than 2^{53}. The bits of 2^{53} are equal to $2^{53}+1$, meaning that the bits of $2^{53}+1$ are not unique, CANNOT be represented exactly. Therefore, the spacing is 2 of numbers starting with 2^{53}

```
> 2^54

[1] 1.80144e+16

> 2^54 - 2

[1] 1.80144e+16

> 2^54 + 2

[1] 1.80144e+16

> bits(2^54) == bits (2^54-2)

[1] FALSE

> bits(2^54) == bits (2^54+2)

[1] TRUE
```

When analyzing the spacing of 2^{54}, lets assume 2^{54} is fixed and can be represented exactly, Since the bits of 2^{54} is not equal to $2^{54}-2$, meaning that the bits of $2^{54}-2$ is unique, CAN be represented exactly. However, the bits of 2^{54} is equal to $2^{54}+2$, meaning that the bits of $2^{54}+2$ is not unique, CANNOT be represented exactly. Therefore, based on the representatipn of $2^{54}-2$ and $2^{54}+2$, spacing of numbers starting with 2^{54} is 4.

Problem 3

3a

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

This data.table install has not detected OpenMP support. It will work but slower in single t

> a = as.integer(rnorm(1e8))
> b = as.numeric(rnorm(1e8))
> system.time(c <- copy(a)) #Make a deep copy of a

   user  system elapsed
  0.114   0.109   0.223

> system.time(d <- copy(b)) #Make a deep copy of b

   user  system elapsed
  0.268   0.239   0.507
```

It is faster to copy a large vector of integers than to copy a numeric vector, since each interger takes 4 bytes while each numeric value(double) takes 8 bytes.

3b

```
> I = as.integer(rnorm(1e6))
> N = as.numeric(rnorm(1e6))
> microbenchmark(a[c(1:length(I)/2)])

Unit: milliseconds
                expr      min       lq     mean   median       uq      max
 a[c(1:length(I)/2)] 22.45112 32.75253 38.06377 34.07807 36.23189 170.5435
 neval
   100

> microbenchmark(b[c(1:length(N)/2)])

Unit: milliseconds
                expr      min       lq     mean   median       uq      max
 b[c(1:length(N)/2)] 22.88178 34.41606 39.24622 35.61536 37.20703 173.7124
 neval
   100
```

It is a bit faster to take a subset of size k = n/2 from an integer vector than from a numeric vector, since each interger takes 4 bytes while each numeric value(double) takes 8 bytes. However, the running time does not show much a difference.

Problem 4

4a

The naive algorithm of running matrix column by column takes at least O(n*n) times of running. The parallel computing algorithm takes O(np) times for partial matrix multiplication. Then summing up the partial results would take roughly O(log n) running times, which is more efficient.

4b

Approach A is better for minimizing communication. Each time submatrix of Y interacts with the whole matrix X. The total communication times are p (or j, if you think about it in terms of number of tasks). However, each time, the whole matrix of x is passed into the function such that the memory usage is large for approach A, with at least p*objectsize(x) amout of memory.

Approach B is better for minimizing memory usage. Each time, only a subset of X and a subset of Y are passed into the function. However, the communication cost is high since the total number of partitioned blocks is large. It would take at least p*p of communication times.

Extra Credit

R uses IEEE754 scientic notation. For example, 0.1 has an index of -4, 0.2 has an index of -3 and 0.3 has an index of -2.

When summing up 0.1 and 0.2, R will change the index of 0.1 from -4 to -3 (rising the power),which leads to a change in floating points.

Originally:

$$0.1 = 1.10011001100110011001100110011001100110011001100110011010.$$

After change,

$$0.1 = 0.11001100110011001100110011001100110011001100110011001101.$$

After adjusting the floating point,

$$0.1 + 0.2 = 1.00110011001100110011001100110011001100110011001100110011001100111,$$

which is different from the original representation of 0.3.

However, 0.2 + 0.3 == 0.5 is true because when doing addition using IEEE754 scientific notation, we unify the index (power) of 0.2 and 0.3 as -2.

After performing the similar addition as in "0.1 + 0.2" case, and readjust the floating point position, the final result of 0.5 is actually equal to the real representation of 0.5.