Matlab Single and Double Precision Analysis

Serwah Sabetghadam

Abstract

In this report we analyze the single or double precision in Matlab matrix multiplications. Based on the memory requirements and hardware specifications, we determine which size of matrix we can use.

1 Matlab Single and Double Analysis

1.1 What matrix size fits into memory?

For any matrix size we need double of its size to fit into memory. The reason is that we have the matrix itself and also $b = b * a$ or $b = a^2$ where a is the matrix and b is the result of multiplications. In Table 1, we show the memory needed in each case. In any case the memory should not be used up and we need about 5GB for the Java program itself to run.

For single matrixes we need the memory of size: a double size matrix + a single size matrix, as you see in the 4th row, 19GB is from the double matrix and 19GB is the two single ones needed. The reason is that we cannot directly create a single matrix in Matlab. We should first create a double matrix and then apply *single* function on that matrix. Of course we can delete the double matrix after creating the single matrix, but this does not change the need.

Table 1: Memory needed to create the matrix. Those matrix sizes that need more than 80GB are not practical, since the memory is not always 100GB available. The real free memory is about 90GB. If we touch the border of available memory it needs to swap and will be very slow.

matrix size	precision	memory	real need (program requirement)	total	practical
50000	double	19GB	38GB + 10GB	48GB	yes
60000	double	27.5GB	55GB+ 10GB	65GB	yes
70000	double	38GB	75GB + 10GB	85GB	no
50000	single	9.5GB	19GB+19GB+ 10GB	48GB	yes
60000	single	14GB	27.5GB + 28GB+ 10GB	66GB	yes
70000	single	19GB	38GB+ 38GB+ 10GB	90GB	no

1.2 Matrix Multiplication

In this section we compare the time taken for double and single precision matrix multiplication for different matrix sizes. With double precision, the matrix of size 50,000 takes 45min for each multiplications. This takes 3.125 days for 100

steps for just one topic, more than 150 days for all topics. Then I calculated the matrix multiplication with power operator and also single precision which is much faster. Meanwhile, we compared the time needed to multiply in loops, or using power operator of Matlab. The results are shown in Table 3.

Table 2: Matlab matrix time needed for one iteration

matrix size	precision	one iteration
50,000	single	18 m
60,000	single	32 m
70,000	single	59 m
50,000	double	47.5 m
60,000	double	4h 37m

Table 3: Matlab matrix multiplications with different matrix sizes and operators

matrix size	steps	precision	**multiply** for one topic	**power** for one topic	50 topics (with **loop**)	50 topics (with **power**)
50,000	50	**single**	15h	2h 24m	31d 5h	**5d**
60,000	50	**single**	1d 1h	4h 16m	52d 2h	**9d**
70,000	50	single	2d 2h	6h 37m	104d 4h	14d
50,000	50	**double**	1d 153h	5h 48m	83d 7h	**12d 2h**
60,000	50	double	9d 5m	20h 30m	450d 4h	42d 17h
50,000	100	single	1d 6h	2h 55m	62d 10h	6d
60,000	100	single	2d 2h	4h 31m	104d 4h	9d 11h
70,000	100	single	4d 4h	7h 23m	208d 8h	15d 7h
50,000	100	double	3d 3h	6h 40m	166d 14h	13d 18h
60,000	100	double	18d 10m	22h 18m	900d 8h	50d

As shown in the table, multiplicaions with single precision is much faster compared to what we had for double precision.

Considerations There is an issue with using power operator, that we will not have the matrix multiplication result in each step, to compute the performance ***stepwise***.

One idea could be to write the power in a loop like:

```
n = number of iterations;
m = matrix;
a = activation_vec
for i = 1 to log(n) do
        z = m*m;
        res = a.z;
        m = z;
end
```

In this case in binary indexes (2,4,8,16) we can have the steps results. However we miss the results of steps in between. Therefore, time needed to do a 64 step in the graph is 6 times to the time of one iteration (Table 2).

2 Are single and double multiplication result the same?

In this section we did the verification of matrix multiplication of single and double precision values. First, let's have a look at max/min double and precision values in Matlab.

Largest and Smallest Double-Precision Values The MATLAB functions realmax and realmin return the maximum and minimum values that you can represent with the double data type: The range for double is:

```
-1.79769e+308 to -2.22507e-308 and
 2.22507e-308 to  1.79769e+308
```

Numbers larger than realmax or smaller than -realmax are assigned the values of positive and negative infinity, respectively:

```
realmax + .0001e+308
ans =
   Inf

-realmax - .0001e+308
ans =
  -Inf
```

Largest and Smallest Single-Precision Values The MATLAB functions realmax and realmin, when called with the argument 'single', return the maximum and minimum values that you can represent with the single data type:

```
str = 'The range for single is:\n\t%g to %g and\n\t %g to  %g';
sprintf(str, -realmax('single'), -realmin('single'), ...
    realmin('single'), realmax('single'))

ans =
The range for single is:
-3.40282e+38 to -1.17549e-38 and
 1.17549e-38 to  3.40282e+38
```

In the single precision interval (1.17549e-38 to 3.40282e+38), the results of multiplication with single or double precision are completely the same. We tested with Matlab *rand* function. For the numbers bigger than this interval it goes to Infinity value and make *Inf* values in the matrix. For numbers less than the min value, it rounds to 0.

3 Test in Astera

To analyse what happens in practice, we performed the real matrix multiplications in Astera. *These matrixes are not normalized and they are as generated before.* We compared the result of using Matlab file with double precision matrixes with the result of single precision matrixes. In small number of iterations,

the results (the final calculated precision) are the same. When it reaches higher multiplications, it may cross the margins. Once it happened in the 88th step. Since in the step 87 that all the numbers were greater than $1.0e{+}38$. In the next iteration, some cells will have Inf value, since it goes beyond the single max value. Another time it happened in the 110th step - depending on the topicID that we start from we have different sub-graphs and matrixes. We cannot say a step number that all matrixes from different topics will cross the upper margin. This happens since we generate energy, and this value explodes in larger steps.

We did not cross the lowest margin), since the generated value is always getting larger and larger.

Considerations There is a point with the new experiments that we are thinking of now. That in this case the matrixes have normalized value. This means that the sum of the rows remains always 1. This scenario will not happen with crossing the margin max values. But it may happen with crossing the min value of single precision.

4 Conclusion

- **Matrix size**: For 60,000 and 70,000 size single and double matrixes fit into memory. Anyway 50,000 will be faster.
- **Time needed**: With single precision time of power multiplications are thinkable. For double or loop iterations the time is too long.
- **Snapshots to the iterations**: we will have the results only in 2nd, 4th, 8th, 2^nth steps.
- **Number of steps**: This is a remained concern. We do not know the minimum step that this margin cross happens, specially if we do the multiplications with power operator.