CS205 Assignment 3 Report

SID: 11910718 Name: 陆彦青

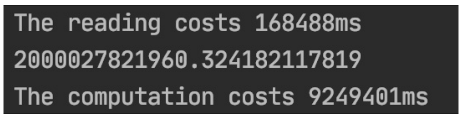
Part 1. Introduction & Features

1. This is a function which can compute the dot product of two vectors. The type of vector elements is float.
2. The time used for computing vectors with 200M elements are averagely 800ms.
3. Compared with the first version, the latest is nearly **10000** times faster.
4. **Multi-thread** is used to boost the computation.
5. The function `cblas_sdot` in **OpenBLAS** is imported as a comparison.

Part 2. Improvement

Version #1:

In this version, the high-precision addition and multiplication I created in assignment 2 are used. Moreover, the test datas are stores in a txt file. Thus both the reading and computing speed are very slow.

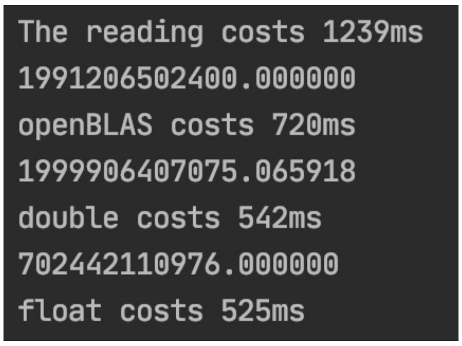


```
The reading costs 168488ms
2000027821960.324182117819
The computation costs 9249401ms
```

The result above does not equal the following results because they used different test datas.

Version #2:

The main change of this version is to use the fundamental operators '+' and '*' instead of user-defined functions. In addition, the test datas are altered to save as binary files which increases the reading. I also compare the difference of speed between `double` and `float` (the type that stores the result).

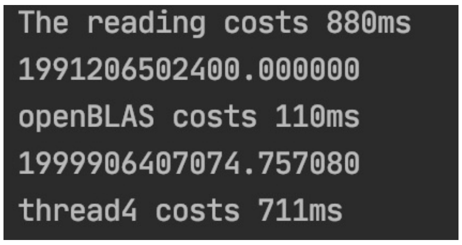


```
The reading costs 1239ms
1991206502400.000000
openBLAS costs 720ms
1999906407075.065918
double costs 542ms
702442110976.000000
float costs 525ms
```

The difference of speed is little but `float` does not return the correct result. As for `double`, the speed is fast but the precision seems insufficient.

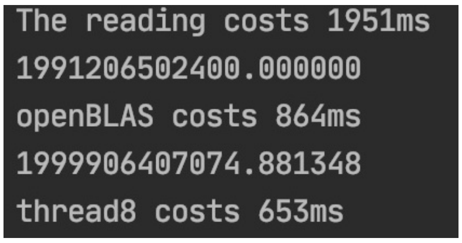
I also try to use **multi-thread** but find the speed even slower for both 4 threads and 8 threads. The switch of threads may cost too much time compared with the time used to compute.

4 threads



```
The reading costs 880ms
1991206502400.000000
openBLAS costs 110ms
1999906407074.757080
thread4 costs 711ms
```

8 threads



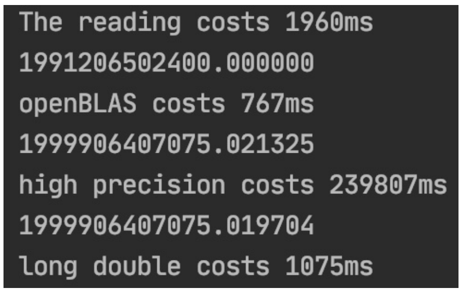
```
The reading costs 1951ms
1991206502400.000000
openBLAS costs 864ms
1999906407074.881348
thread8 costs 653ms
```

Version #3:

In this version, I use the **high-precision addition** again and create 8 threads to improve the efficiency. Although the time is still long but I find the difference of result between `double` and `high-precision` is less than the difference above. It may mean that the result of `cblas_sdot` is not completely accurate.

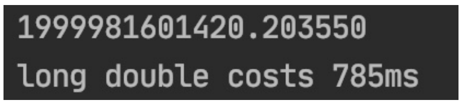
Then I use **long double** to store the result and join it in the comparison. Its result is closest to the `high-precision` and meanwhile the speed is not too low. Thus, I choose the last method ultimately.

Comparison of 3 methods



```
The reading costs 1960ms
1991206502400.000000
openBLAS costs 767ms
1999906407075.021325
high precision costs 239807ms
1999906407075.019704
long double costs 1075ms
```

Another set of test data



```
1999981601420.203550
long double costs 785ms
```

Part 3. Difficulties & Solutions

1. Generate random vectors having 200m elements and save them. I use `random_device` to generate the vector and `fopen,fwrite` to save it in a binary file.
2. Balance the efficiency and precision of the algorithm. The first version only focus on the precision so the efficiency becomes too much low. In the next versions, I make a lot of comparison and finally choose the most balanced method.
3. Import the function `cblas_sdot`. In order to use the function in OpenBLAS library, I edit `CmakeList.txt` to set the path of include and link directories.

Part 4. Source Code

All the source codes are included in a GitHub repository: `https://github.com/Beauchamp-West/2020Fall_CS205`