

Q3

Here we use the openblas method `cblas_dgemm`. We store all the matrix data into a consecutive dynamically allocated memory for the calculation.

Below is the comparison of the two optimizations. The middle one is the one we use for Q2. We could tell that the openblas has the shortest running time. It improved by 99%, which is so impressive. For the second improvement in Q2, it improved by nearly 50%.

```
[zhang.yam@cc0229 ~]$ module load openblas/0.3.6
[zhang.yam@cc0229 ~]$ gcc -lopenblas Q3.c -o Q3 && ./Q3
starting dense matrix multiply
a result 4.44488e+07
The total time for matrix multiplication with dense matrices = 155.401289 ms
starting sparse matrix multiply
A result -1.11183e+07
The total time for matrix multiplication with sparse matrices = 159.811027 ms
The sparsity of the a and b matrices = 0.750977
[zhang.yam@cc0229 ~]$ gcc -fopenmp Q2.c -o Q2 && ./Q2
starting dense matrix multiply
a result 4.44488e+08
The total time for matrix multiplication with dense matrices = 6334.194620 ms
starting sparse matrix multiply
A result -1.11183e+08
The total time for matrix multiplication with sparse matrices = 6154.083572 ms
The sparsity of the a and b matrices = 0.750977
[zhang.yam@cc0229 ~]$ gcc matmul.c -o matmul && ./matmul
starting dense matrix multiply
a result 4.44488e+07
The total time for matrix multiplication with dense matrices = 12284.792842 ms
starting sparse matrix multiply
A result 0
The total time for matrix multiplication with sparse matrices = 13550.813273 ms
The sparsity of the a and b matrices = 0.750977
```