Basic Contents 1

highlights 2

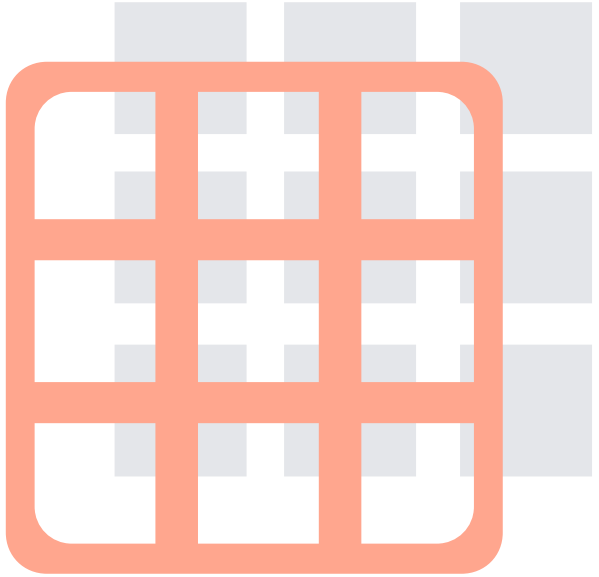
difficulties 3

2022

CPP Final Proposal

MATRIX

By Han Zhu, Tiantian Wang, Yu Qin



Basic Architecture of Our Library

MAT (Root)

- exception_handler.hpp — Exception-handle relative funtions
- exceptions.hpp — Definitions of exception code
- matrix.hpp — Main part of the lib, including implementation of class
- mfunc.hpp — Implementation of matrix related operation functions
- triplet.hpp — Structure for sparse matrix storage

Header Details – Matrix Class Design

1. The class is based on the template with typename “value_t”
   template<typename value_t> class matrix{ … }
2. Seperation of data storage
   value_t* dense_data;
   std::vector<triplet<value_t>> sparse_data;
3. We implemented 5 different constructors
   matrix();
   matrix(index_t row_num, index_t col_num);
   static const tag_t SPARSE_TAG = 0x10000001;
   static const tag_t DENSE_TAG = 0x10000002;
   matrix(const value_t* data, index_t row_num, index_t col_num,const tag_t MAT_TAG);
   matrix(const matrix<value_t>& A);
   #ifdef OPENCV_ALL_HPP
   matrix(const cv::Mat& CVMat);
   #endif

Header Details – Matrix Class Design

4. We provide several reconstruction funtions to users

```
void resize(index_t row,index_t col)
void resize_and_clear(index_t row,index_t col)
void slice_row(index_t s_row,index_t e_row)
void slice_col(index_t s_col,index_t e_col)
```

5. We overloaded “<<“ and “==“ operator

```
bool operator == (const matrix<value_t>& A) const
friend std::ostream& operator << (std::ostream& os,const matrix<value_t>& A)
```

6. OpenCV matrix conversion

```
#ifdef OPENCV_ALL_HPP
    operator cv::Mat() const;
#endif
```

And the constructor mentioned before

6. Lots of friend functions for direct data access to boost calculation (implemented in mfunc.hpp)

```
friend exception_t MMULT(const matrix<float>& A,const matrix<float>& B,matrix<float>& rst);
friend exception_t MMULT(const matrix<T1>& A,const matrix<T2>& B,matrix<T3>& rst);
```

… …

Header Details – mfunc.hpp implemention

We finaaly implemented 31 matrix operation funtions for our users

1. exception_t get_sparse_data(const matrix<T>& A,T*& rst)
2. exception_t MMULT(const matrix<T1>& A,const matrix<T2>& B,matrix<T3>& rst)
3. exception_t MMULT(const matrix<float>& A,const matrix<float>& B,matrix<float>& rst)
4. exception_t DMULT(const matrix<T1>& A,const matrix<T2>& B,matrix<T3>& rst)
5. exception_t MADD(const matrix<T1>& A,const matrix<T2>& B,matrix<T3>& rst)
6. exception_t MSUB(const matrix<T1>& A,const matrix<T2>& B,matrix<T3>& rst)
7. exception_t SMULT(const matrix<T>& A,T S,matrix<T>& rst)
8. exception_t SDIV(const matrix<T>& A,double S,matrix<T>& rst)
9. exception_t SDIV(double S,const matrix<T>& A,matrix<T>& rst)
10. exception_t VCMULT3D(const matrix<T>& A,const matrix<T>& B,matrix<T>& rst)
11. exception_t MTRANS(const matrix<T>& A,matrix<T>& rst)
12. exception_t MTRANS(matrix<T>& A)
13. exception_t MCONJ(const matrix<std::complex<T>>& A,matrix<T>& rst)
14. exception_t MCONJ(matrix<std::complex<T>>& A)
15. exception_t MEXTR_ROW(const matrix<T>& A,index_t row_num,tag_t OP_TAG,T& rst)
16. exception_t MEXTR_COL(const matrix<T>& A,index_t col_num,tag_t OP_TAG,T& rst)
17. exception_t MEXTR(const matrix<T>& A,tag_t OP_TAG,T& rst)

Header Details – mfunc.hpp implemention

18. exception_t MSUM_ROW(const matrix<T>& A,index_t row_num,T& rst)
19. exception_t MSUM_COL(const matrix<T>& A,index_t col_num,T& rst)
20. exception_t MSUM(const matrix<T>& A,T& rst)
21. exception_t MAVG_ROW(const matrix<T>& A,index_t row_num,T& rst)
22. exception_t MAVG_COL(const matrix<T>& A,index_t col_num,T& rst)
23. exception_t MAVG(const matrix<T>& A,T& rst)
24. exception_t MTRACE(const matrix<T>& A,T& rst)
25. exception_t MEGVL(const matrix<T>& A,T& rst)
26. exception_t MEGVCT(const matrix<T>& A,T*& rst)
27. exception_t MCONV(const matrix<T>& CONV_CORE,const matrix<T>& A,matrix<T>& rst)
28. exception_t MDET(const matrix<T>& A,T& rst)
29. exception_t MALCO(const matrix<T>& A,matrix<T>& complement,int i,int j)
30. exception_t MADJ(const matrix<T>& A,matrix<T>& adj,int size)
31. exception_t MINVER(const matrix<T>& A,matrix<T>& rst)

Highlights ②

Difficulties ③

1. Design level: The whole matrix class is implement on the basis of template, thus versatility and ease of use have been greatly improved. However, the trade-off is that lack of optimization on specified data type is insufficient which can’t be ignored.

2. Coding level: New features have been used in our design, including “auto”, “friend”, anonymous functions etc. “typedef” has been widely used to unify variable types, which has improved the scalability for subsequent modify.

3. Optimization level: Massive parallelization via OpenMP was involved to our code. We have optimized the matrix multiplication carefully and richly.

Highlights ②

Optimization Details

Edge optimization – Matrix class

OpenMP is used to assist the acceleration in some extent for the functions in matrix class

- Resize(_and_clear) #2 threads limited
- Slice(row | col) #2 threads limited
- Operator “==” #thread num unlimited
- Constructor(All) #2 threads limited

The simultaneous reading and writing of a large number of threads to the same or continuous memory space will seriously affect the IO performance due to the cache consistency principle (**False Sharing 虚假内存共享**).

Highlights ②

Optimization Details

Core optimization – MMULT<T>

We have implemented a total of two versions of the mmult function. The first is implemented based on templates. To optimize the general multiplication, we first transpose the multiplied matrix, which can greatly improve the cache hit rate and greatly reduce the calculation time of multiplication. Then we use OpenMP to parallelize and avoid the problem of False sharing. In the complex number benchmark, our MMULT improved by 18 times compared with the naked multiplication.

```
#define M 2333
#define K 2377
#define N 2122
```

```
adkoishi@ADKOISHI:~/source/CPP_AS/ProjectII$ g++ multtest.cpp -o main -fopenmp -mavx2 -mfma -std=c++14 -l
adkoishi@ADKOISHI:~/source/CPP_AS/ProjectII$ ./main
2.17707
36.9869
0.0077143
```

```
srand((unsigned)time(NULL));
for(int i = 0 ; i < M ; i ++)
    for(int j = 0 ; j < K ; j ++)
        data[i*K+j] = complex<int>(rand()%10,rand()%10);
for(int i = 0 ; i < K ; i ++)
    for(int j = 0 ; j < N ; j ++)
        data3[i*N+j] = complex<int>(rand()%10,rand()%10);
matrix<complex<int>>M1(data,M,K,matrix<int>::SPARSE_TAG);
matrix<complex<int>>M2(data3,K,N,matrix<int>::DENSE_TAG);
matrix<complex<int>>M3;
double t1 = omp_get_wtime();
MMULT(M1,M2,M3);
cout<<omp_get_wtime()-t1<<endl;
```

```
t1 = omp_get_wtime();
#pragma omp parallel for
for(int i = 0 ; i < M ; i ++)
    for(int j = 0 ; j < N ; j ++)
        if(data4[i*N+j] != M3.get_val(i,j))
            cout<<"WA"<<endl;
cout<<omp_get_wtime()-t1<<endl;
```

```
t1 = omp_get_wtime();
for(int i = 0 ; i < M ; i ++){
    for(int j = 0 ; j < N ; j ++){
        for(int k = 0 ; k < K ; k ++)
            data4[i*N+j]+=data[i*K+k]*data3[k*N+j];
    }
}
cout<<omp_get_wtime()-t1<<endl;
```

Highlights ②

Optimization Details

Core optimization – MMULT<float>

When the data type is determined, there are many more places to optimize. In multiplication of matrices of type float, we did the following:

- OpenMP is used to parallelize the calculation process by row
- Rearrange the loop order to greatly improve the cache efficiency
- Loop expansion is carried out, so that the effect of the previous step is more obvious, and the efficiency of the pipeline can be improved at the same time
- We use AVX2 and FMA instruction sets to achieve SIMD effect in computation.

In this round of test, we set the values of M, K and N to 4367,4577 and 4824 respectively.

```
adkoishi@ADKOISHI:~/source/CPP_AS/ProjectII$ g++ multtest.cpp -o main -fopenmp -mavx2
adkoishi@ADKOISHI:~/source/CPP_AS/ProjectII$ ./main
4.79646
187.766
0.0094028
```

And the test result performed a 40 times acceleration

Difficulties we met in this project

Problems below has been confronted and approached:

- Loop expansion leads to intractable boundary problems in MMULT
- Pointer or pointer to pointer or reference in parameter
- Incorrect use of key word “const” produced masses of errors during compiling
- Compile flags varies on different platform
- "Negative optimization" caused by wrong application of multithreading
- Dynamic memory management based on “new” is quite difficult to handle

… …

DIFFICULTIES

Difficulties ③