CS205-C/C++ Final Project Proposal

RESPONSOR: Han Zhu SID: 12012828 CONTRIBUTION: 45%

GROUP MEMBER 1: Tianao Wang SID: 12011014 CONTRIBUTION: 35%

GROUP MEMBER 2: Yu Qin SID: 11910305 CONTRIBUTION: 20%

1. Basic Contents

1.1 Introduction

In this project, our team members have implemented a matrix library with considerable performance through cooperation. In the process of implementation, we tried to use the various characteristics of C++ and solved various problems encountered. At the same time, we also came into contact with a variety of optimization strategies and means, and finally achieved the goal.

1.2 Basic Architecture of Our Library

In terms of the architecture of the library, we were going to follow the standard ROOT-> LIB - INCLUDE architecture. But in the actual process, we find that the target class of this project needs to be highly universal to provide support for self designed data type, so we use the template class, and the problem is that the derived template function can not generate the corresponding dynamic or static link library through pre-compilation. Therefore, we finally decided to design the architecture of the library as follows.

Its obvious that our whole library can be regarded as an include library, thus the `CPLUS/C_INCLUDE_PATH` environment variable is the only thing to modified when user need to invoke our library. This also avoids all kinds of headaches caused by links in regular compilation.

1.3 Header Details

In this part we will show partial code in the we've implemented in the header files listed above.

1.3.1 Header Details - Matrix Class Design

1. The class is based on the template with typename “value_t”

```
template<typename value_t> class matrix{ … }
```

2. Separation 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
```

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 “==“ and "=" operator

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

6. OpenCV matrix conversion

```
#ifdef OPENCV_ALL_HPP
    operator cv::Mat() const;
#endif
/* And the constructor mentioned before */
```

7. 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);
// ... ...
```

1.3.2 Header Details – mfunc.hpp Implementation

We finally implemented 31 matrix operation functions for our users. All the details of the APIs are listed below

1. Sparse data extraction:

```
exception_t get_sparse_data(const matrix<T>& A,T*& rst);
```

2. Matrix Standard four-rule operations:

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

The *float* version of MMULT is a specially optimized version for the float type matrix multiplication.

3. Scalar-Matrix related operations:

```
exception_t SMULT(const matrix<T>& A,T S,matrix<T>& rst);
//Matrix_elements / Scalar
exception_t SDIV(const matrix<T>& A,double S,matrix<T>& rst);
//Scalar / Matrix_elements
exception_t SDIV(double S,const matrix<T>& A,matrix<T>& rst);
exception_t VCMULT3D(const matrix<T>& A,const matrix<T>& B,matrix<T>& rst);
```

4. Matrix transpose:

```
exception_t MTRANS(const matrix<T>& A,matrix<T>& rst);
exception_t MTRANS(matrix<T>& A);
```

5. Reduction operations for matrix, including Row-base version, Col-base version and global version:

```
//Matrix Extreme Value calculation
exception_t MEXTR_ROW(const matrix<T>& A,index_t row_num,tag_t OP_TAG,T&
rst);
exception_t MEXTR_COL(const matrix<T>& A,index_t col_num,tag_t OP_TAG,T&
rst);
exception_t MEXTR(const matrix<T>& A,tag_t OP_TAG,T& rst);
//Matrix sum calculation
exception_t MSUM_ROW(const matrix<T>& A,index_t row_num,T& rst);
exception_t MSUM_COL(const matrix<T>& A,index_t col_num,T& rst);
exception_t MSUM(const matrix<T>& A,T& rst);
//Matrix Average number calculation
exception_t MAVG_ROW(const matrix<T>& A,index_t row_num,T& rst);
exception_t MAVG_COL(const matrix<T>& A,index_t col_num,T& rst);
exception_t MAVG(const matrix<T>& A,T& rst);
//Calculate the trace of the Matrix
exception_t MTRACE(const matrix<T>& A,T& rst);
```

6. Subdivision algebraic operation of matrix:

```
// Conjunction for Matrix self-edit version involved
exception_t MCONJ(const matrix<std::complex<T>>& A,matrix<T>& rst);
```

```
exception_t MCONJ(matrix<std::complex<T>>& A);
// Calculate the eigenvalue(vector) at the same time
exception_t MEGVLCT(const matrix<T>& A,matrix<T>& rstValue,matrix<T>&
rstVector,double EPS,long MAX_Count);
// Matrix convolution calculation
exception_t MCONV(const matrix<T>& CONV_CORE,const matrix<T>& A,matrix<T>&
rst);
// Matrix determinant calculation
exception_t MDET(const matrix<T>& A,T& rst);
// Algebraic cofactor calculation
exception_t MALCO(const matrix<T>& A,matrix<T>& complement,int i,int j);
// Adjoint matrix calculation
exception_t MADJ(const matrix<T>& A,matrix<T>& adj,int size);
// Calculate the inverse of a specific matrix
exception_t MINVER(const matrix<T>& A,matrix<T>& rst);
```

2. Highlights

Highlights in our project can be viewed from three perspectives:

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

2.1 Optimization Level Details

2.1.1 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**), thus for methods that are difficult to parallelize, we can only use a small number of threads to improve their performance to a certain extent

2.1.2 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. The test code and results are shown below:

```
#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;
```

2.1.3 Core optimization – MMULT<float>

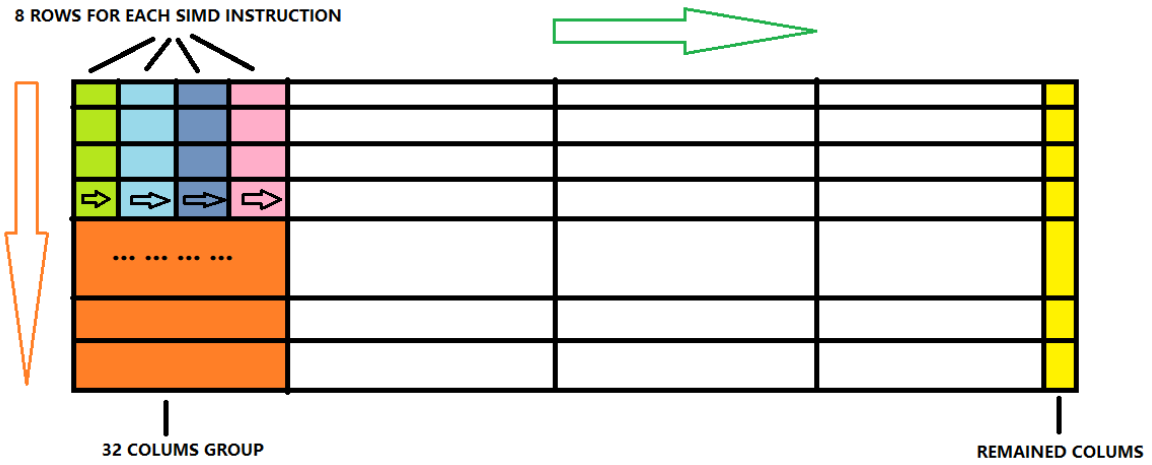
When the data type is determined, there are many more places to optimize. In multiplication of matrices of type float, we did the following:

- 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. And the test code is almost the same as that in MMULT<T>

```
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 compared with the unoptimized one. The detailed calculation process in the result matrix is shown in the figure below:



First level (Black pointer): Using an SIMD instruction (FMA etc.) to calculate the product-sum in a single unit and store it to the result matrix.

Second level (Orange pointer): After using loop expansion to calculate a 32-column unit, repeat the first level process for all rows in the result matrix.

Third level (green pointer): After finishing the Second level calculation in the first 32-column group, repeat the progress for all group horizontally.

Forth level (The yellow part): This part of the result is calculated separately by traditional matrix multiplication.

3. Implementation of Matrix Algebraic Operations

To implement the function that find the inverse of a matrix, we need to calculate the determinant of the matrix and the adjoint of the matrix. To calculate the determinant and adjoint of the matrix, we build a function that can calculate algebraic complements the matrix.

By using Jacobi, our `MEGVLCT` function that caulculate the eigenvalue and eigenvector of a real symmetric matrix at same time that store the eigenvalue into a matrix sizeof `N*1` and the eigenvector into a matrix sizeof `N*N`.

3.1Original Code

calculate the algebraic complement of the matrix

```
template<typename T>
exception_t MALCO(const matrix<T>& A,matrix<T>& complement,int i,int j){
    if(A.get_colnum()!=A.get_rownum()||A.get_colnum()!=complement.get_colnum()+1
    ||complement.get_colnum()!=complement.get_rownum())return
MATRIX_SIZE_INVALID;
    if(!A.is_valid()) return MATRIX_DATA_INVALID;
    for(int p=0; p<A.get_rownum()-1; p++)
    {
        for(int q=0; q<A.get_colnum()-1; q++)
        {
            complement.set_val(p,q,A.get_val(p+(p>=i),q+(q>=j)));
        }
    }
    return OP_SUCCESS;
}
```

calculate the adjoint matrix of the matrix

```
template<typename T>
exception_t MADJ(const matrix<T>& A,matrix<T>& adj,int size){
    if(A.get_colnum()!=A.get_rownum()||A.get_colnum()!=size||
    adj.get_colnum()!=adj.get_rownum()||adj.get_colnum()!=A.get_colnum())
    return MATRIX_SIZE_INVALID;
    if(!A.is_valid())return MATRIX_DATA_INVALID;
    for (msize_t i = 0; i < adj.get_rownum(); i++)
    {
        for (msize_t j = 0; j < adj.get_colnum(); j++)
        {
            matrix<T> m=matrix<T>(A.get_colnum()-1,A.get_colnum()-1);

            exception_t e=MALCO(A,m,i,j);
            if(e!=OP_SUCCESS)return e;

            if((i+j)%2==0){
                adj.set_val(j,i,det(m,size-1));
            }
            else if((i+j)%2==1){
```

```
                adj.set_val(j,i,-det(m,size-1));
            }
        }
    }
    return OP_SUCCESS;
}
```

calculate the inverse of a matrix

```
template<typename T>
exception_t MINVER(const matrix<T>& A,matrix<T>& rst){
    if(A.get_colnum() != A.get_rownum()){
        return MATRIX_NOT_INVERTIBLE;
    }
    if(!A.is_valid()){
        return MATRIX_DATA_INVALID;
    }
    T det=0;
    exception_t e=MDET(A,det);
    if(e!=OP_SUCCESS){
        return e;
    }
    if(det==0){
        return MATRIX_NOT_DETERMINANT;
    }
    matrix<T> adj=matrix<T>(A.get_colnum(),A.get_colnum());
    exception_t e0 = MADJ(A,adj,A.get_colnum());
    if(e0!=OP_SUCCESS){
        return e0;
    }
    exception_t e1=SMULT(adj,1/det,rst);
    if(e1!=OP_SUCCESS){
        return e1;
    }
    return OP_SUCCESS;
}
```

the eigenvalue and the eigenvector of matrix A

```
/*
    find the eigenvalue and the eigenvector of matrix A and store it to rstVector
and rstValue
*/
template<typename T>
exception_t MEGVLCT(const matrix<T>& A,matrix<T>& rstValue,
matrix<T>& rstVector,double EPS,long MAX_Count){

 if(A.get_colnum()!=A.get_rownum()||rstVector.get_colnum()!=rstVector.get_rownum
()
    ||rstVector.get_rownum()!=A.get_rownum()||
    rstValue.get_colnum()!=1||rstValue.get_rownum()!=A.get_rownum())
    return MATRIX_SIZE_INVALID;
    if(!A.is_valid())return MATRIX_DATA_INVALID;
    int M=A.get_rownum();
```

```
    matrix<T> ACOPY = A;
    //initialize the eigenvector matrix
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < M; j++)
        {
            if(i==j) rstVector.set_val(i,j,1);
            else rstVector.set_val(i,j,0);
        }
    }
    long cnt=0;
    while (1)
    {
        //if cnt>=MAX_Count, then finish the calculation
        if(cnt >= MAX_Count){
            break;
        }
        cnt++;
        //find the max value in the matrix that not in major line
        T max_value=ACOPY.get_val(0,1);
        int nRow=0,nCol=1;
        for (int i = 0; i < M; i++)
        {
            for (int j = 0; j < M; j++){
                T abs=fabs(ACOPY.get_val(i,j));
                if((i!=j) && (abs > max_value)){
                    max_value=abs;
                    nRow=i;
                    nCol=j;
                }
            }
        }
        //if the max value is less than EPS, then break
        if(max_value < EPS){
            break;
        }
        T pp=ACOPY.get_val(nRow,nRow);
        T qq=ACOPY.get_val(nCol,nCol);
        T pq=ACOPY.get_val(nRow,nCol);
        T theta=atan2(2*pq,pp-qq)*0.5;
        T c=cos(theta);
        T s=sin(theta);
        T s2=sin(2*theta);
        T c2=cos(2*theta);
        ACOPY.set_val(nRow,nRow,c*c*pp+2*s*c*pq+s*s*qq);
        ACOPY.set_val(nCol,nCol,c*c*qq-2*s*c*pq+s*s*pp);
        ACOPY.set_val(nRow,nCol,c2*pq+s2*(qq-pp)*0.5);
        ACOPY.set_val(nCol,nRow,c2*pq+s2*(qq-pp)*0.5);
        for (int i = 0; i < M; i++)
        {
            if((i!=nRow) && (i!=nCol)){
                T tempr=ACOPY.get_val(i,nRow);
                T tempc=ACOPY.get_val(i,nCol);
                ACOPY.set_val(i,nRow,c*tempr+s*tempc);
                ACOPY.set_val(i,nCol,-s*tempr+c*tempc);
```

```
            }
        }
        for (int j = 0; j < M; j++){
            if((j!=nRow) && (j!=nCol)){
                T tempr=ACOPY.get_val(nRow,j);
                T tempc=ACOPY.get_val(nCol,j);
                ACOPY.set_val(nRow,j,c*tempr+s*tempc);
                ACOPY.set_val(nCol,j,-s*tempr+c*tempc);
            }
        }
        for(int i =0; i < M ; i++){
            T tempr=rstVector.get_val(i,nRow);
            T tempc=rstVector.get_val(i,nCol);
            rstVector.set_val(i,nRow,c*tempr+s*tempc);
            rstVector.set_val(i,nCol,-s*tempr+c*tempc);
        }
    }
    for (int i = 0; i < M; i++)
    {
        rstValue.set_val(i,0,ACOPY.get_val(i,i));
    }
    return OP_SUCCESS;
}
```

3.2 Test Result and Validation

The test for the type double :

```
(base) wangtianaoo@wangtianaodeMacBook-Pro ProjectII % g++ -std=c++14 main.cpp -o main && ./main

----------------------------------------the original data in double----------------------------------------
The original matrix is:
-0.800347 2.99057 -0.308458
0.670639 2.29451 1.43383
2.60496 1.20505 3.4769

-----------------test the sum and average of matrix--------------------
test the sum of row 2:7.28691
test the average of row 2:2.42897
test the sum of column 2:4.60227
test the average of column 2:1.53409
test the sum of matrix:13.5677
test the average of matrix:1.50752
------------------------test the determinant--------------------------
The determinant is 0.789071

---------------test inverse matrix and adjoint matrix-----------------
The inverse matrix is:
7.92062 -13.6485 6.33117
1.77847 -2.50827 1.19216
-6.55069 11.0951 -4.86902

The adjoint matrix is:
6.24993 -10.7696 4.99574
1.40334 -1.9792 0.9407
-5.16896 8.75479 -3.842

--------------------test eigenvector and eigenvalue--------------------
The M1 is changed as a real symmetric matrix:
-0.800347 0.670639 2.60496
0.670639 2.29451 1.20505
2.60496 1.20505 3.4769

The eigenvalue is:
5.33225
1.67297
-2.03417

The eigenvector is:
0.39401 -0.156101 -0.905753
0.412543 0.91066 0.0225132
0.821319 -0.382533 0.423207
```

The test for the type int :

```
----------------------------------------the original data in interger-----------------------------------
The original matrix is:
6 8 2
7 -1 1
3 7 2

------------------test the sum and average of matrix-------------------
test the sum of row 2:12
test the average of row 2:4
test the sum of column 2:5
test the average of column 2:1
test the sum of matrix:35
test the average of matrix:3
-------------------------test the determinant---------------------------
The determinant is -38

----------------test inverse matrix and adjoint matrix-----------------
The inverse matrix is(almost inverse element is smaller than 1):
0 0 0
0 0 0
0 0 0

The adjoint matrix is:
-9 -2 10
-11 6 8
52 -18 -62
```

By test by matrix tool in website, we can see that the result is correct.

4. Difficulties

In the implementation of the parts mentioned above, we've met several problems. But after hours of striking , these cease to be stumbling blocks in our project

- 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
- There exists some precision missing in some function operation because of using the template.