Project 5 Report

0.链接

https://github.com/Haosonn/CS205-Project5-Matlib-CPP

1.概述

本次Project需求是 用C++完成一个矩阵Class。

2.完成内容

- 内存管理

  为进行内存管理，设置了三个类

```
//存储数据
template <typename T>
class MemBlock {
public:
    T* data; //数据指针
    size_t stride //用于ROI;
    size_t size //数据尺寸大小;
};
//矩阵的智能指针
template <typename T>
class MatPtr {
public:
    MemBlock<T> memBlock;
    int * refCount //记录被引用次数;
};
//矩阵本体
template <typename T>
class Matrix {
public:
    size_t rows;
    size_t cols;
    size_t bias //用于ROI;
    MatPtr<T> matPtr;
}

//内存管理关键代码在MatPtr的创建和析构
template <typename T>
void MatPtr<T>::release() {
    if (this->refCount != nullptr) {
        (*this->refCount)--;
        if (*this->refCount == 0) {
            delete this->refCount;
            this->memBlock.release();
        }
        this->refCount = nullptr;
    }
```

```
}
//当且仅当指针指向内容没有其他指针指向后删除数据
//拷贝函数，赋值号重载等等都令refCount++
```

- ROI（避免硬拷贝）

  所有矩阵都含有bias参数，用于计算矩阵第一个元素在MemBlock的位置

```
//以获取第i行第j列元素为例 （下标从0开始）
template <typename T>
T Matrix<T>::get(size_t i, size_t j) const {
    if (this->matPtr.getData() == nullptr) {
        std::cerr << "Callstack: Matrix<T>::get(size_t i, size_t j)" <<
std::endl;
        throw std::string("Exception in Matrix<T>::get(size_t i, size_t j),
matrix is empty");
    }
    if (i >= this->rows || j >= this->cols || i < 0 || j < 0) {
        std::cerr << "Callstack: Matrix<T>::get(size_t i, size_t j)" <<
std::endl;
        throw std::string("Exception in Matrix<T>::get(size_t i, size_t j),
index out of range");
    }
    T * data = this->matPtr.getData() + this->bias;
    size_t stride = this->matPtr.memBlock.stride;
    return data[i * stride + j];
}
//获取子矩阵函数，仅对子矩阵的存储信息作一些处理，不对数据来源改动
template<typename T>
Matrix<T> Matrix<T>::subMatrix(size_t beginRow, size_t endRow, size_t
beginCol, size_t endCol) {
    if (beginRow > endRow || beginCol > endCol || endRow > this->rows ||
endCol > this->cols) {
        std::cerr << "Callstack: Matrix<T>::subMatrix(size_t beginRow, size_t
endRow, size_t beginCol, size_t endCol)" << std::endl;
        throw std::string("Exception in Matrix<T>::subMatrix(size_t beginRow,
size_t endRow, size_t beginCol, size_t endCol): Invalid range");
    }
    Matrix<T> res(*this);
    res.rows = endRow - beginRow + 1;
    res.cols = endCol - beginCol + 1;
    res.bias += beginRow * this->matPtr.memBlock.stride + beginCol;
    return res;
}
```

- 实现一些矩阵常用方法，加减乘（加入AVX优化）除，行列式，矩阵逆，矩阵转置，行最简型等等，具体在Matrix.h文件有注释。
- 加入Img类，用于存多channel的Matrix

```
template <typename T>
class Img {
public:
    Matrix<T>* img;
    size_t channels;
    size_t rows;
    size_t cols;
}
```

3.遇到的问题和解决方法

- 模板类的头文件和实现文件的分离

  被这点折磨很久，最终在实现文件加上int，double等等类型的实例化解决。

- 支持不同矩阵类型的运算。

  解决办法，新增一个模板Copy constructor，强行对矩阵每个元素按底层基本类型逻辑强制转换。

  这样一来，在两种不同类型的矩阵进行运算时，程序将一个矩阵强制转换成另外一个。

```
template <typename T>
class Matrix {
    template<class U>
    Matrix(const Matrix<U>& m) : rows(m.rows), cols(m.cols), bias(0),
matPtr(m.rows, m.cols) {
        T * dataFrom = this->matPtr.getData();
        U * dataTo = m.matPtr.getData();
        for(size_t i = 0; i < this->rows; i++){
            for(size_t j = 0; j < this->cols; j++){
                dataFrom[i * this->cols + j] = (T)dataTo[i *
m.matPtr.memBlock.stride + j];
            }
        }
    }
}
```

  这样能使doubleMat + intMat运算出我们想要的结果，但是当表达式为intMat + doubleMat时，由于+号是int类型的矩阵的运算符，doubleMat 转成intMat类型丢失精度。因此，遇到这样的情况，写上Matrix(intMat) + doubleMat即可解决问题。

- 在模板的前提下，进行AVX指令集优化矩阵乘法

  解决办法：在stackoverflow上提问，得到方案是用一个AVXTraits的模板类存avx指令集类型”变量“, 再用AVXType 调用这个”变量“

```
namespace avx {
    template <typename T>
    struct AVXTraits {static size_t size;};
    template <> struct AVXTraits<float> {
        using type = __m256;
        static size_t size;
    };
    template <> struct AVXTraits<double> {
        using type = __m256d;
        static size_t size;
    };
```

```
    template <> struct AVXTraits<int> {
        using type = __m256i;
        static size_t size;
    };
    //short 和 uchar 类型不支持avx，这里写上是为了过编译
    template <> struct AVXTraits<short> {
        using type = __m256i;
        static size_t size;
    };
    template <> struct AVXTraits<unsigned char> {
        using type = __m256i;
        static size_t size;
    };
    size_t AVXTraits<float>::size = 8;
    size_t AVXTraits<double>::size = 4;
    size_t AVXTraits<int>::size = 8;
    size_t AVXTraits<short>::size = 0;
    size_t AVXTraits<unsigned char>::size = 0;

    template<class T>
    using AVXType = typename AVXTraits<T>::type;

    inline __m256 mm256_loadu(const float* a) {
        return _mm256_loadu_ps(a);
    }
    inline __m256d mm256_loadu(const double* a) {
        return _mm256_loadu_pd(a);
    }
    inline __m256i mm256_loadu(const int* a) {
        return _mm256_loadu_si256((__m256i const*)a);
    }
    inline __m256i mm256_loadu(unsigned char* a) {
        return _mm256_setzero_si256();
    }
    inline __m256i mm256_loadu(short* a) {
        return _mm256_setzero_si256();
    }

    inline __m256 mm256_mul(const __m256& m1, const __m256& m2) {
        return _mm256_mul_ps(m1, m2);
    }
    inline __m256d mm256_mul(const __m256d& m1, const __m256d& m2) {
        return _mm256_mul_pd(m1, m2);
    }
    inline __m256i mm256_mul(const __m256i& m1, const __m256i& m2) {
        return _mm256_mullo_epi32(m1, m2);
    }
}
```

4.一些测试 (main.cpp)

```
#include <iostream>
#include <tuple>
#include "Matrix.h"
using namespace std;

void conversionTest() {
    cout << "Conversion test" << endl;
    Matrix<int> intMat = Matrix<int>::randomMatrix(3, 3);
    cout << "intMat:" << endl;
    intMat.printMatrix();
    Matrix<double> doubleMat = Matrix<double>::randomMatrix(3, 3);
    cout << "doubleMat:" << endl;
    doubleMat.printMatrix();
    cout << "intMat + doubleMat:" << endl;
    (intMat + doubleMat).printMatrix();
    cout << "doubleMat + intMat:" << endl;
    (doubleMat + intMat).printMatrix();
}

void inverseTest() {
    cout << "Inverse test" << endl;
    Matrix<double> doubleMatrix = Matrix<double>::randomMatrix(5, 5);
    doubleMatrix.printMatrix();
    Matrix<double> subDoubleMatrix = doubleMatrix.subMatrix(1, 2, 1, 2);
    subDoubleMatrix.printMatrix();
    Matrix<double> doubleInvMatrix = subDoubleMatrix.inverse();
    subDoubleMatrix.printMatrix();
    (subDoubleMatrix * doubleInvMatrix).printMatrix();
}

void PLUtest() {
    cout << "PLU test" << endl;
    Matrix<double> doubleMatrix = Matrix<double>::randomMatrix(5, 5);
    doubleMatrix.printMatrix();
    Matrix<double> subDoubleMatrix = doubleMatrix.subMatrix(1, 2, 1, 2);
    subDoubleMatrix.printMatrix();
    tuple<Matrix<double>, Matrix<double>, Matrix<double>> PLU =
subDoubleMatrix.PLUFactorization();
    get<0>(PLU).printMatrix();
    get<1>(PLU).printMatrix();
    get<2>(PLU).printMatrix();
    (get<0>(PLU) * get<1>(PLU) * get<2>(PLU)).printMatrix();
}

void ImgTest() {
    cout << "Image test" << endl;
    Img<double> img(3, 5, 5);
    img.setChannel(0, Matrix<double>::randomMatrix(5, 5));
    img.setChannel(1, Matrix<double>::randomMatrix(5, 5));
    img.setChannel(2, Matrix<double>::randomMatrix(5, 5));
    img.printImg();
}
```

```cpp
void mulTest() {
    cout << "Mul test" << endl;
    Matrix<double> doubleMatrix = Matrix<double>::randomMatrix(5, 5);
    cout << "doubleMatrix:" << endl;
    doubleMatrix.printMatrix();
    cout << "doubleMatrix * doubleMatrix:" << endl;
    (doubleMatrix * doubleMatrix).printMatrix();
}

int main() {
    conversionTest();
    inverseTest();
    PLUtest();
    ImgTest();
    mulTest();
}
```

```
-----------------------------------
Conversion test
intMat:
4 6 3
0 6 2
7 5 6
doubleMat:
6.4 0.5 4.5
8.1 2.7 6.1
9.1 9.5 4.2
intMat + doubleMat:
10 6 7
8 8 8
16 14 10
doubleMat + intMat:
10.4 6.5 7.5
8.1 8.7 8.1
16.1 14.5 10.2
Inverse test
2.7 3.6 9.1 0.4 0.2
5.3 9.2 8.2 2.1 1.6
1.8 9.5 4.7 2.6 7.1
3.8 6.9 1.2 6.7 9.9
3.5 9.4 0.3 1.1 2.2
9.2 8.2
9.5 4.7
9.2 8.2 1 0
9.5 4.7 0 1
9.2 8.2 1 0
9.5 4.7 0 1
9.2 8.2
9.5 4.7
1 0
0 1
PLU test
3.3 7.3 6.4 4.1 1.1
5.3 6.8 4.7 4.4 6.2
5.7 3.7 5.9 2.3 4.1
```

```
2.9 7.8 1.6 3.5 9
4.2 8.8 0.6 4 4.2
6.8 4.7
3.7 5.9
1 0
0 1
1 0
0.544118 1
6.8 4.7
1.19209e-07 3.34265
6.8 4.7
3.7 5.9
Image test
(6.4, 2.3, 9.7) (4.8, 3.7, 1.2) (4.6, 3.8, 8.6) (0.5, 1.8, 9) (9, 8.2, 6.1)
(2.9, 2.9, 3.6) (7, 4.1, 5.5) (5, 3.3, 6.7) (0.6, 1.5, 5.5) (0.1, 3.9, 7.4)
(9.3, 5.8, 3.1) (4.8, 0.4, 5.2) (2.9, 3, 5) (2.3, 7.7, 5) (8.4, 0.6, 4.1)
(5.4, 7.3, 2.4) (5.6, 8.6, 6.6) (4, 2.1, 3) (6.6, 4.5, 0.7) (7.6, 2.4, 9.1)
(3.1, 7.2, 0.7) (0.8, 7, 3.7) (4.4, 2.9, 5.7) (3.9, 7.7, 8.7) (2.6, 7.3, 5.3)
Mul test
doubleMatrix:
8.3 4.5 0.9 0.9 5.8
2.1 8.8 2.2 4.6 0.6
3 1.3 6.8 0 9.1
6.2 5.5 1 5.9 2.4
3.7 4.8 8.3 9.5 4.1
doubleMatrix * doubleMatrix:
108.08 110.91 72.53 88.58 84.97
73.25 117.93 45.79 75.21 50.98
81.7 77.46 127.33 95.13 117.37
111.47 121.57 50.3 88.49 72.36
139.76 141.61 113.86 120.41 139.48
```