CS205 C/ C++ Programming - Project 4

Name：叶璨铭(Ye CanMing)

SID：12011404

Part 1 - Analysis

1-1 Review of last project

在上一次project中， 学习了c语言的基础语法，实现了Strassen算法，探究了simd的基本使用，深化了openmp的使用; 重新讨论了性能分析与基准测试。

然而，上一次project存在以下问题

- 对于c语言，存在一些指针、内存申请后没有得到释放，以致内存泄露；
- 工程逻辑复杂，有一些逻辑错误没有检查出来。比如说获得元素的宏写错了。

- 对于simd理解不够深刻，只是学习了于老师给的示例并且简单复现，没有实现加速；
- 对openblas的原理没有深入探究。
- 对于访存优化的原理没有深入探究。

1-1 Goals

本次project我的目标是：

- 首先，学会**C++的内存管理方法**，争取不泄露内存、不提前释放、不重复释放。
    - 从而解决上次project C语言编写过程中内存管理混乱的问题。
    - 学习opencv Mat类的基本使用接口。
    - 深入学习opencv实现思路。
        - 尽量理解opencv基本使用接口的实现方法
        - 只把这次project要用的opencv功能学来，与图像处理有关的由于时间有限暂时放弃。
        - 比较复杂的opencv实现方法，使用简单的替代。
        - 修改opencv不合理的地方，形成自己的数据结构。
    - 初步研究标准库，查询C++11的内容。
- 其次，实现**多通道泛型矩阵**， 实现**opencv当中的一些复杂概念**
- 再次，进一步**研究simd和矩阵运算的优化**
    - 研究neon和sse的相同点和不同点
    - 尝试把neon和sse放到矩阵乘法、矩阵逐元素操作等。
- 最后，在实现以上目标的同时，需要考虑泛型、宏定义等，使得工程上不至于混乱。

1-2 Some problems to consider

根据project4的要求，需要考虑

- 如何存储一个**多通道矩阵。**（学习opencv）
- 如何支持**泛型矩阵。**（学习opencv）
- 矩阵class的**C++内存管理。**（复习、学习c++）
- 矩阵乘法等**矩阵运算**的实现（学习opencv、simd、openmp）（重点是simd，尝试arm架构）
- **ROI**如何实现。（学习opencv）

Part 2 - Difficulties & Solutions

2-1 多通道泛型矩阵的存储方式与头信息

2-1-1 多通道泛型矩阵

- 矩阵

  mxn的矩阵是指一个m行n列的实数表，可以赋予线性变换、线性方程组的系数等数学概念，也可以表示图像。但是逻辑上的存储与顺序关系就是m行n列的实数表。

- 多通道

  多通道矩阵支持一个矩阵元素拥有逻辑上连续的多个通道值，mxnxcn的多通道矩阵是一个m行n列的，cn长度的向量的表。

- 泛型

  这里的泛型只是针对了基本类型，因为基本类型是对实数的计算机表示。泛型要求一个矩阵支持多种类型的表示。

多通道泛型矩阵支持以下操作

```
//cv_examples.cpp
cv::Mat a(2,3, CV_32FC3, cv::Scalar(1.0f, 2.0f, 3.0f)); //2x3的，用32bit表示浮点数的，三通道的矩阵，初始每个元素设置为向量<1,2,3>
```

- 多通道与Scalar[1]
  - opencv认为，**一个元素虽然有多个通道，但是对于矩阵的概念而言，仍然是一个元素**, 因此称之为标量。
  - 对于图像而言，Scalar很方便地**表示了一个像素。**
  - Scalar在*opencv*中是*cv::Vec*的子类，cv::Vec则是cv::Matx的子类，表示一个小的矩阵, 操作比较快。

2-1-2 MatStep

- 虽然逻辑上矩阵是数表，但是操作系统呈现给程序的内存没有数表。
  - 为了存储矩阵，我们需要把矩阵的元素映射到内存当中。
  - 内存可以看成是一个很大的一维数组，而数组是内存的一个roi。
- 因此，opencv使用了MatStep的概念[1]
  - $$addr(M_{i,j}) = M.data + M.step[0] * i + M.step[1] * j$$
  - 从openblas的角度来说，这种定义方法是"RowMajor"的。
  - 在内存中，矩阵的一行接着下一行；多维矩阵的话，矩阵的一个平面接着下一个平面。
  - step则是记录矩阵的从一行的某一列，到下一行的这列，地址所需的偏移量。
- MatStep的概念是通用的
  - Numpy, Win32等其他库也使用了类似的定义来表示矩阵[1]

2-1-3 MatFlag

- MatFlag实现泛型
  - 编译时泛型
    - 模板矩阵类的类型再编译时就确定了，这是因为传递给模板的尖括号的，无论是typename还是size_t（指示通道数），都要求是常数
    - 因此，模板矩阵类可以在编译时通过类型萃取（type_traits）等技术让其成员函数表现恰当的行为。
    - 特别地，使用高级的C++元编程技术可以接受或者拒绝一些特定的类型，实现 概念(concept)的基本思想（对类型有要求的泛型）。[2]
  - 运行时泛型
    - 然而，OpenCV没有采用编译时泛型。而是把类型作为一种特殊的常数整数进行定义，然后把类型信息存在矩阵当中。
    - 这样，矩阵的成员函数在运行时知晓this的类型，以表现恰当的行为。
  - 深度与通道
    - 深度（depth）是指float\int\uchar\double等类型
    - 而类型(type)则是指一个元素（标量）的类型，比如CV_32FC3.
- 运行时泛型的问题
  - 构造函数不能限制用户传入的类型是常数。
  - 这是因为，函数中写的const，只是表明函数不会修改这个值，而不是说原本的值是常数。
  - 这就导致了性能的降低和代码的不优雅。
  - OpenCV通过很多宏定义，尽可能地减少类型判断的开销。
- MatFlag记录矩阵的其他重要状态。

    - 对于连续的矩阵，可以对代码进行SIMD优化，如果矩阵的存储不连续（比如由于是ROI，还有可能是因为内存对齐），那么矩阵的运算、操作的逻辑是很不同的。
    - 因此，MatFlag还要能够在运行时记录矩阵是否连续、是否为子矩阵。
- **OpenCV的flags**
    - OpenCV使用一个int(32个bit去表示上面的这些信息)
    - 一个int，那么我要用到这些信息的时候，如何将我想要的信息分离呢?
    - 技术1.Mask取段
        - and的逻辑是输入都为1时才输出1
        - 换个角度去想，对于and运算，1是identity， 0是inverse。
        - 因此，对于这个flags，如果要取出其中channel数量的信息
        - 可以先&一个CHANNEL_MASK，这个CHANNEL_MASK在flags中存取channel的位是1，其他地方是0，排除了其他信息的干扰。
        - 然后，将&的结果移位，靠右，这样就得到了正确的数字。

```
Mat::Mat(Size _sz, int _type, void* _data, size_t _step)  _sz: 0x00000032141af610 {width=
    : flags(MAGIC_VAL + (_type & TYPE_MASK)), dims(2), rows(_sz.height), cols(_sz.width),
      data((uchar*)_data), datastart((uchar*)_data), dataend(0), datalimit(0),   _data: 0x
      allocator(0), u(0), size(&rows)
```

        (图片：OpenCV构造外源数据的矩阵。MAGIC_VAL是矩阵类型的签名，*type*把外源的*type*的多余信息去除，然后加到*flags*当中（不用移位是因为*types*存在最右边）。这样type其余地方是0，不会意外地修改类型签名或者其他信息)
    - 技术2.魔法移位取深度的大小
        - 在构造时，我们需要知道类型中深度的空间占用。如果是编译时泛型，我们只需要sizeof(Tp)即可。但是运行时泛型无法实现。
        - OpenCV的解法

```
/** Size of each channel item,
   0x28442211 = 0010 1000 0100 0100 0010 0010 0001 0001 ~ array of sizeof(arr_type_elem) */
#define CV_ELEM_SIZE1(type) ((0x28442211 >> CV_MAT_DEPTH(type)*4) & 15)

#define CV_ELEM_SIZE(type) (CV_MAT_CN(type)*CV_ELEM_SIZE1(type))
```

        - OpenCV方法的不足
            - 由于OpenCV只是支持8种深度，而且深度的size最大也只有1000， 所以一个魔法常数int恰好可以容纳。
            - 但是如果有更多的类型，OpenCV就不能支持了.
- 使用C++改写OpenCV的flags，**封装为MatFlag类**
    - constexpr的概念[2]
        - constexpr指示编译器，一个变量的值是可以编译时确定的，因此不会在运行时计算。
        - constexpr函数
            - constexpr函数不是说一定要在编译时求值，
            - constexpr的意思是如果参数是constexpr，那么得到的表达式也是constexpr
            - 函数必须足够简单才能在编译时求值，
                - 简单意味着不能有副作用，也就是纯函数。
                - 但是不意味着不能递归。
    - inline与constexpr的关系
        - 按照书上的说法[2]， inline函数是一种没那么强的constexpr函数。
    - 使用constexpr代替宏定义，定义SUSTech_Types

```
constexpr int SUSTech_CHANNEL_MAX = 1 << 12;
constexpr int SUSTech_CHANNEL_SHIFT = 4;
constexpr int SUSTech_CHANNEL_MASK = ((SUSTech_CHANNEL_MAX - 1) << SUSTech_CHANNEL_SHIFT);
constexpr int SUSTech_DEPTH_MAX = (1 << SUSTech_CHANNEL_SHIFT);
constexpr int SUSTech_MAT_DEPTH_MASK = SUSTech_DEPTH_MAX - 1;
enum depth_order { SUSTech_8U, SUSTech_32S, SUSTech_32F, SUSTech_64F };
constexpr std::array<int, 4> depth_step{ _Elems[0]: 1, _Elems[1]: 4, _Elems[2]: 4, _Elems[3]: 8 };
constexpr std::array<const char*, 4> depth_name{ _Elems[0]: "SUSTech_8U", _Elems[1]: "SUSTech_32S",
```

- 在运行时，可以访问这些常量。

- MatFlag封装矩阵的运行时类型信息和矩阵判断这些信息的方法。

```
struct MatFlag {
    enum MatrixClassSignature { NormalMatrixSignature = 0x42 };
    enum BooleanFlags { ContinuityFlag = 1 << 4, SubMatrixFlag = 1 << 0 };
    int8_t mSignature;  //表明是Mat还是SparseMat还是std::vector。这样InputArray就可以区分了
                        //(为什么不是这些类型的矩阵继承InputArray呢，是有原因的)。
    int8_t mFlags;      // flags包括continuity(左4位)和isSubMatrix(右4位)
    int16_t mType;      /*- depthOrder (float、double、int等等),
  这里设计占4位（opencv是三位），至多支持16种数据类型(深刻的问题：为用户实现高效的bool和float16矩阵)
umber of channels（depth和channel在一起合称type），这里占12位，支持2**12的通道。
为矩阵构造的时候就传了一个type进来，所以我没有把depth和channel拆开。*/
    bool isNormalMatrix() const;
    void flipContinuityFlag();
    bool isContinuous() const;
    void flipSubMatrixFlag();
    bool isSubMatrix() const;
    int depthOrder() const;
    int depthStep() const;
    std::string depthName() const;
    int channels() const;
    bool operator==(const MatFlag& rhs) const;
    bool operator!=(const MatFlag& rhs) const;
};
```

- 定义好了这些成员函数之后，我们就可以轻易地在矩阵成员函数需要的时候，调用MatFlag的方法获取相应的信息。

2-1-4 本矩阵的基本信息（header）表示设计

```
private:  // opencv用的是public。
    static constexpr int dims = 2;
    MatFlag mFlag;
    size_t mRows, mCols;
    uint8_t* mData {nullptr};
    uint8_t* mDataEnd {nullptr};
```

2-2 多通道泛型矩阵的内存管理

2-2-1Mat只是数据的头信息——复制构造

- OpenCV中[3]
  - “*Mat*基本上是一个具有两个数据部分的类：矩阵标头（包含诸如矩阵大小、用于存储的方法、存储的矩阵地址等信息）和指向包含像素值的矩阵的指针（根据选择的存储方法获取任何维度）。”
  - 需要有大量的图像处理函数，这些函数接受Mat等类型作为参数。
  - 由于计算的性能需要保证，OpenCV要尽可能避免传递参数的时候复制
  - 因此尽管参数加个引用就能解决，为了不让用户犯错，OpenCV化复制为共享
  - 每个Mat对象都有自己不同的标头，但是可以共享相同的数据地址。
  - 因此，“复制运算符**将仅复制标头**和指向大型矩阵的指针，而不是数据本身”

2-2-2 持有源数据的引用——析构与引用计数

- 为了实现上述概念，Mat不仅要持有源数据的引用，
  - 还要持有与该数据对应的，在所有共享了这块数据的所共享的一个计数器。
  - 这就是OpenCV的UMatData*。 包括了元数据的引用和对引用的计数。
- 我的实现。
  - 为了避免OpenCV中MatAllocator等复杂概念，简化的版本如下：
  - ```
    size_t mOriginalDataStepCol, mOriginalDataStepRow;  /
                                                        /
                                                        /
    //这里的左右顺序是有讲究的。
    SUSTechAtomicInt* mOriginalDataRefCount{nullptr};  //
    uint8_t* mOriginalDataStart{nullptr};//uint8_t* 是指s
    uint8_t* mOriginalDataEnd{nullptr};//这里start和end语
    ```
  - 使用Original来命名，以表现出原有数据的独特性。对于OriginalData这块内存，可以有很多个Mat指向（拥有地址的拷贝）
- 复制与析构
  - 当矩阵复制时，不仅需要增加OriginalDataRefCount的计数，如果是赋值的话，还要把自己原本持有的内存释放掉，以避免内存泄露。
  - 而当矩阵析构时，如果自己是最后一个引用该数据的矩阵，则负责释放内存，以避免内存泄露和重复释放。
  - 可以看到，释放和引用减数不是析构函数的特权，赋值函数也要用到。
  - 因此，可以把这个逻辑单独分出一个release函数和destroy函数来执行。
- 析构的并发问题
  - 如果有两个同源矩阵并发析构，那么这两个矩阵会不会double free?
  - 要解决这个问题，OpenCV使用了编译器的一些特性，使得减数的过程是加锁的。
  - 我们可以使用C++11的原子类型来存储引用计数，由于gcc不支持atomic_int32_t， 所以我们定义
  - ```
    #include <atomic>
    using SUSTechAtomicInt = std::atomic<std::int32_t>;
    ```

2-2-3 右值引用与移动构造

2-2-3-1 什么是右值引用和移动构造[2]

- 右值是不能作为左值的值。左值是有身份的但是不可以移动的值。
- 如果是常量左值，不可以被修改。其余的，我们可以说可以被修改而且修改逻辑对的值是左值。
- 右值引用是对临时对象的引用，右值引用绑定到右值.
- 移动，是把内存的一部分移动到另一部分的过程。
- 但是内存本质上是不支持移动的，因此移动构造函数实际上是复制之后，清除原有内存的状态，即为窃取。
- std::move转换为右值，是显式地指示调用移动构造函数
- 清除对方状态，是因为对方即将析构，在对方析构之前，把对方的状态转移过来。

2-2-3-2 为什么移动构造更高效?[2]

- 考虑资源句柄的概念。比如vector，是一个指针的资源句柄。
  - 当一个vector即将消亡时（比如从函数返回），调用移动构造函数
  - 这个时候，函数栈上的vector拥有一个在动态内存申请的数组，
  - 我们只需要把这个栈上的指针窃取过来，我们就间接拥有了这个动态申请的数组
  - 而不需要复制一遍到另一块动态申请的新区域，然后让栈上vector把其原来拥有的区域释放掉。
- 之所以要窃取对方的状态，而不是复制对方的状态（这里状态说的就是指针，对于资源的一把钥匙）,在于让对方析构的时候，不去释放其申请的内存，防止重复释放。
- 完全类似的，swap两个vector的时候
  - 需要局部变量temp保存A的值，A保存B的值，B设置为原来A的值。
  - 实际上，我们只需要交换AB的指针，而不是进行三次复制
  - 但是复制构造函数和复制赋值语句不知道我们的目的。
  - 因此，将AB转换为右值，执行右值构造与右值赋值，就可以实现高效的交换。

2-2-3-3 Mat的移动构造

- 搞懂了移动的构造概念之后，我们就可以明白如何写Mat的移动构造函数
-

```
Mat::Mat(Mat&& m) SUSTech_NOEXCEPT : mFlag(m.mFlag),
                                      mRows(m.mRows),
                                      mCols(m.mCols),
                                      mData(m.mData),
                                      mDataEnd(m.mDataEnd),
                                      mOriginalDataRefCount(m.mOriginalDataRefCount),
                                      mOriginalDataStepCol(m.mOriginalDataStepCol),
                                      mOriginalDataStepRow(m.mOriginalDataStepRow),
                                      mOriginalDataStart(m.mOriginalDataStart),
                                      mOriginalDataEnd(m.mOriginalDataEnd) {
    //清零操作。
    /.../ //如果对不关键的成员清零，可能导致析构器不知道mFlag，从而不知道怎么析构(虽然这不需要释放内存)。
    //关键的清零只有这三个，给析构函数提供信息告诉它不用释放数据就可以了。
    m.mRows = 0;
    m.mCols = 0;
    m.mOriginalDataRefCount = nullptr;
}
```

- 在移动构造时，我们窃取了对方的引用计数，因此，对方释放的时候，由于他的引用计数不存在，也就不能对计数减1.
- OpenCV的实现也是类似的。
- 然而，由于我们的类的复制语义发生了改变，所以Mat的复制开销不大，所以这里的移动构造不能节省开销。
- 尽管如此，我们Mat的移动构造和复制构造的意思还是不一样的。
  - 比如，

```
int main(){
    SUSTech::Mat matA;
    SUSTech::Mat matB { rows: 3, cols: 3, type: SUSTech::SUSTech_32FC3, s: { .mScalar._Elems[0]: 1,
    SUSTech::Mat matC = std::move(matB);
```

  - 这一段不会引起double free。matC从matB移动构造，尽管matB没有被显式的析构。
  - 如果在这段代码之后我们还使用matB，就会发现matB变成了空矩阵。
  - main结束时，matB不会释放，释放权交由matC。

2-3 多通道泛型矩阵的基本接口——切片（ROI）与推导赋值

2-3-1需求

我们希望实现OpenCV中这样的写法：[1]

```
cv::Mat M( rows: 4, cols: 4, type: CV_32FC1, s: cv::Scalar( v0: 1));
std::cout << M <<std::endl;
M = { v0: 2};
Mat M1 = M.col( x: 1);
M1 = { v0: 3};
```

```
Mat roi(M, roi: cv::Rect( _x: 0, _y: 1, _width: 2, _height: 3));
// the original image will be modified
std::cout << roi <<std::endl;
roi = cv::Scalar( v0: 2);
std::cout << M <<std::endl;
```

我们发现，对于矩阵某个元素的读写（如果不考虑效率，只是给用户使用的话），实际上更一般地是对矩阵的感兴趣区域的整体读写。这与此前project中的逻辑有所不同，之前取元素是取出一个引用，修改左值引用即可修改元素，也可以查看元素。

2-3-2 宏定义作为代码生成的手段

- 考虑
  ```
  template<typename Tp, size_t channels>
  Mat& operator = (const std::array<Tp, channels>& s); //全员赋值
  ```
  的实现，其中array表示一个标量，由于一个标量可能有多个通道，所以使用array传参（OpenCV的Scalar仅支持四个元素，四个通道)。
-
  - 这里，array的类型是逻辑上的，比如，Mat的类型是int，但是array类型是double，值为{1.5,2.0, 3.0},
  - 那么三通道矩阵Mat就需要将每一个值设置为{1,2,3}
  - 两通道矩阵只要设置为{1,2}
  - 四通道矩阵需要把每一个元素设置为{1,2,3,0}
- 注意到，由于我们是运行时泛型，我们无法得知矩阵Mat的类型。
  - 因此，我们对array的类型转换，乃至对Mat类型的认知，都需要动态判断确定。**Switch是不可避免的。**
- 为了**避免代码的冗长**，比如我们最高支持16个类型，我们就得写16个代码。
- 我们尽可能保证每个switch形式上是一致的，只是对switch的case进行了一些**推导。**
- 这里使用decltype和auto，以及一些特殊形式，实现了逻辑的统一。

```
define MatOperatorAssignmentSwitchGenerate(CASE)
  {
      const auto convertedScalar = convert<decltype(depth_to_type<CASE>::value), double, MaxChannels>(_s)
      if(mFlag.isContinuous()) {
          for(uint8_t* elementIt = mData; (elementIt < mDataEnd); elementIt += mOriginalDataStepCol) {
              auto concretePtr = reinterpret_cast<decltype(depth_to_type<CASE>::value)*>(elementIt);
              for(int i = 0; i < assignmentChannels; ++i) {
                  concretePtr[i] = convertedScalar[i];
              }
          }
      } else {
          for(size_t i = 0; i < mRows; ++i) {
              uint8_t* rowIt = mData + i * mOriginalDataStepRow;
              for(size_t j = 0; j < mCols; ++j) {
                  uint8_t* elementIt = rowIt + j * mOriginalDataStepCol;
                  auto concretePtr = reinterpret_cast<decltype(depth_to_type<CASE>::value)*>(elementIt);
                  for(int k = 0; k < assignmentChannels; ++k) {
                      concretePtr[k] = convertedScalar[k];
                  }
              }
          }
switch((SUSTech::depth_order)depth) {  // switch是无法避免的，switch是高效的。
    case SUSTech_8U:
        MatOperatorAssignmentSwitchGenerate(SUSTech_8U) case SUSTech_32S
            : MatOperatorAssignmentSwitchGenerate(SUSTech_32S) case SUSTech_32F
            : MatOperatorAssignmentSwitchGenerate(SUSTech_32F) case SUSTech_64F
            : MatOperatorAssignmentSwitchGenerate(SUSTech_64F) default
            : throw std::runtime_error( _Message: "new type cast not supported!");
}
```

2-4 多通道泛型矩阵——算术运算与新优化策略

2-5-1 语言层次的优化

2-5-1-1 惰性求值、熔合运算与特殊矩阵

- 惰性求值是函数式编程的概念，包括两个子概念[^]
- 最小化求值
  - 我目前的理解是，考虑 (condition1()&&conditon2())
  - 如果condition1是false，那么condition2就不求了，何况condition2可能很耗时。
  - 但是如果condition2求值的过程有副作用，也就是说不是函数式的，那么为了保证语义会避免短路。（当然我不是说c/c++里面的&&，是确实会短路的）
- 延迟求值
  - 与及早求值是反义词。
  - 当用户实际需要的时候，才进行计算。
  - 这样的好处是可以生成一个无限的流，从流中可以不断的获取东西。比如说斐波那契函数流。
- 熔合运算[2]
  - 熔合运算的意思是说，通过把小函数合并为大函数(大小指的是参数的多少)，实现
    - 最小化相同数据的多次循环访问。
    - 比如A*B
- OpenCV的MatExpr[4]

2-5-1-2 复制消除与移动构造

复制消除与移动构造，可以最小化临时变量数和矩阵拷贝数。

也是考虑上面说的A+B+C, A+B得到了一个右值，如果有移动构造函数，那么A+B函数中的新临时变量可以（看做）用于+C。

2-5-2 初探SIMD

2-5-2-1 Intel指令集基础概念

2-5-2-2 ARM指令集基础概念

2-5-2-3 OpenCV统一指令集

2-5-3 矩阵乘法

2-5-3-1 最优访存、行主序与列主序

for的写法有A(3,3) = 6种，经过论文研究，行主序和列主序分别有一种最优访存的for写法。[5]

2-5-3-2 核心函数

可以写一个4x4的核心函数，arm的指令好像比intel的高级，多了很多概念，用在这个核心函数里面（比如说只取大寄存器中的一个float取和另一整个寄存器去乘）[6]

Part 3 - Result & Verification

- 本地benchmark跑分

```
-------------------------------------------------------------------------------
Benchmark                                     Time             CPU   Iterations
-------------------------------------------------------------------------------
readMatrix/32/iterations:5                189920 ns        0.000 ns            5
readMatrix/32/iterations:5                246260 ns        0.000 ns            5
readMatrix/64/iterations:5                653320 ns        0.000 ns            5
  L1 Instruction 32 KiB (x8)
  L2 Unified 512 KiB (x8)
  L3 Unified 4096 KiB (x2)
readMatrix/128/iterations:5              2669180 ns      3125000 ns            5
readMatrix/256/iterations:5             12286900 ns     12500000 ns            5
readMatrix/512/iterations:5             46771740 ns     46875000 ns            5
readMatrix/1024/iterations:5           141966620 ns    140625000 ns            5
readMatrix/2048/iterations:5           450911220 ns    446875000 ns            5
```

```
writeMatrix/32/iterations:5               1251160 ns        0.000 ns            5
writeMatrix/32/iterations:5               1049680 ns        0.000 ns            5
writeMatrix/64/iterations:5               3851820 ns      3125000 ns            5
writeMatrix/128/iterations:5             14521120 ns     15625000 ns            5
writeMatrix/256/iterations:5             59255600 ns     59375000 ns            5
writeMatrix/512/iterations:5            238448840 ns    237500000 ns            5
writeMatrix/1024/iterations:5           900459680 ns    900000000 ns            5
writeMatrix/2048/iterations:5          3620701540 ns   3618750000 ns            5
```

```
testMultiplication/32/iterations:5        216480 ns        0.000 ns            5
testMultiplication/32/iterations:5         68520 ns        0.000 ns            5
testMultiplication/64/iterations:5        445620 ns        0.000 ns            5
testMultiplication/128/iterations:5      2840160 ns      3125000 ns            5
testMultiplication/256/iterations:5     12084000 ns      6250000 ns            5
testMultiplication/512/iterations:5     69088400 ns     59375000 ns            5
testMultiplication/1024/iterations:5   502101540 ns    450000000 ns            5
testMultiplication/2048/iterations:5  3769290960 ns   3537500000 ns            5
```

- arm服务器跑分

```
-------------------------------------------------------------------------------
Benchmark                                     Time             CPU   Iterations
-------------------------------------------------------------------------------
readMatrix/32/iterations:5                75107 ns        74900 ns            5
readMatrix/32/iterations:5                72201 ns        72162 ns            5
readMatrix/64/iterations:5               297167 ns       293560 ns            5
readMatrix/128/iterations:5             1176873 ns      1174642 ns            5
readMatrix/256/iterations:5             4651218 ns      4643354 ns            5
readMatrix/512/iterations:5            18764248 ns     18728976 ns            5
readMatrix/1024/iterations:5           75039543 ns     74955548 ns            5
readMatrix/2048/iterations:5          297805529 ns    297531174 ns            5
writeMatrix/32/iterations:5               759908 ns       757990 ns            5
writeMatrix/32/iterations:5               751258 ns       750332 ns            5
writeMatrix/64/iterations:5              2884100 ns      2880100 ns            5
writeMatrix/128/iterations:5            11257908 ns     11249180 ns            5
writeMatrix/256/iterations:5            44794149 ns     44523916 ns            5
writeMatrix/512/iterations:5           178030993 ns    177892122 ns            5
writeMatrix/1024/iterations:5          710485444 ns    709769030 ns            5
writeMatrix/2048/iterations:5         2839255838 ns   2836242206 ns            5
testMultiplication/32/iterations:5          89801 ns        89670 ns            5
testMultiplication/32/iterations:5          89229 ns        89098 ns            5
testMultiplication/64/iterations:5         606011 ns       604990 ns            5
testMultiplication/128/iterations:5       4275912 ns      4269892 ns            5
testMultiplication/256/iterations:5      32751536 ns     32718598 ns            5
testMultiplication/512/iterations:5     262743437 ns    262459436 ns            5
testMultiplication/1024/iterations:5   2090483902 ns   2088316272 ns            5
testMultiplication/2048/iterations:5  19618034575 ns  19585809704 ns            5
```

可见，ARM服务器上由于CPU核心（2核）少于我的电脑上的（16核），所以慢了8倍。（使用OpenMP加速）

Part 4 - Code

- mat.hpp

```
//
// Created by 叶璨铭 on 2021/11/23.
//
#pragma once
#include "sustech_simd_intrinsics.hpp"
#include "sustech_def.hpp"
#include "types.hpp"
#include <atomic>
#include <cstdint>
#include <ostream>
#include <vector>
namespace SUSTech {
    class MatExpr;//TODO
    class Mat;
    class BadArgumentSize:public std::runtime_error{
    public:
        BadArgumentSize(const std::string& message);
    };
    class Mat {
    public:
        //创造构造器
        Mat();  //默认构造器的意思是就创建了一个header，需要用create函数或者被别的处理函数（如matRead）来具体分配。
        Mat(size_t rows, size_t cols, int type);
        Mat(size_t rows, size_t cols, int type, const Scalar& s);//前三个构造函数同属一个重载序列。
    private:
```

```
        Mat(size_t rows, size_t cols, int type, void* data);    //第四个让矩阵的来源于已经分配好的内存，提供给matRead函数使用。
        //注意，这里的设计让Mat就像一个shared_ptr一样，会保护data。如果最后一个Mat析构，data会被矩阵释放。(opencv不会，机制较为复杂)
        //opencv是因为友元函数多才不封装，看不下去了。这个函数很容易用错，很多东西会转换成void*.
    public:
        //无需知道size，从文件推断。
        static Mat matRead(const std::string& file_name);
        //根据当前Mat的信息，从输入流读入恰当数量的元素填入当前矩阵。
        friend std::istream&  operator>>(std::istream &in, Mat& mat);
    private: void create();
    public:  void create(size_t rows, size_t cols, int type, const Scalar& s = {});//空矩阵也可以调用，所以在public当中. 行为是去除以往申请的矩阵，重新开始。
        //复制构造器(opencv语义，不是c++语义)
        Mat(const Mat& m);
        //移动构造器
        Mat(Mat&& m) SUSTech_NOEXCEPT; //虽然我这个矩阵复制开销不大，移动构造器仍是必须的， 否则函数栈上返回会出问题、复杂表达式求值无法执行拷贝消除。
        //ROI构造器
        Mat(Mat  m, const Rect& roiRect);
        Mat(Mat  m, const int& channel);
        //析构函数
         ~Mat();//除了析构函数，还有很多地方需要释放。因此指针释放委托给release，Mat本身的释放（内存空间的清空）委托给系统的析构。
    private:void release(); //释放的语义是对与当前矩阵所用的信息而言，放弃当前矩阵对originalData的共享权。
    private:void destroy(); //如果释放发现this是最后拥有originalData的矩阵，销毁originalData。
    public:
        //复制赋值(opencv语义，不是c++语义)
        Mat& operator=(const Mat& m);
        //移动赋值
        Mat& operator=(Mat&& m) SUSTech_NOEXCEPT;
        //真正内部使用的指针求位
//#define GetElementPtr(rowIndex, colIndex, channelIndex, CASE)
        //单元素取值，由于channel的性质，暂时不能修改。//No: (或许要实现一个channel类才好，或者视作三维矩阵才行)
        //operator()，这里把operator()与operator[]区分开，operator()的目的只是简单读取。
        VarScalar operator()(size_t rowIndex, size_t colIndex) const;
        VarScalar operator()(const Point& location) const;
        // operator[]是为了切片（roi），甚至为了修改roi（切片），虽然用=Mat不能修改，但是允许并且鼓励用=Scalar
        //注意，下面的const只是表面的。
        Mat operator[](const Rect& roiRect) const; //虽然不是返回mat的引用，但是实际上就是通过roi引用了原矩阵的一部分。
        Mat operator[](const Point& location) const;
        Mat row(size_t rowIndex) const;
        Mat col(size_t colIndex) const;
        Mat channel(int channelIndex) const;
        Mat operator[](size_t rowsThenColumnsIndex) const;
        //调试函数
        friend std::ostream& operator<<(std::ostream& os, const Mat& mat);
        //全元素操作
        template<typename Tp, size_t channels>
        Mat& operator = (const std::array<Tp, channels>& s); //全员赋值
```

```
        Mat& operator = (const Scalar& s);

        bool empty() const;
        // 类型转换

        //数学运算(非MatExpr版本), 与opencv语义不同
        Mat& operator+=(const Mat& matB);
        Mat operator+(const Mat& matB);
        Mat& operator-=(const Mat& matB);
        Mat operator-(const Mat& matB);
    private:
        friend Mat&
fast_matrix_element_by_element_linear_combinations_assignment(Mat& ths, const
Mat& matrixA,
                                                              const Mat&
matrixB, float alpha,
                                                              float
beta);
    public:
        Mat matMul(const Mat& matB);
    private:  // opencv用的是public。
        static constexpr int dims = 2;                                  //本
次project只实现二维矩阵，但是对扩展开放。
        MatFlag mFlag;
        size_t mRows, mCols;
        uint8_t* mData {nullptr};
        uint8_t* mDataEnd {nullptr};
        size_t mOriginalDataStepCol, mOriginalDataStepRow;  // step0是从一行到另一
行（第一维），所需要的指针字节偏移数。step1是从一列到另一列（第二维）的偏移数
                                                            //  （本次project是二维
矩阵，所以step1也就是一个元素的字节大小。）
                                                            //  （注意，这里说的元素
包括了channel）
        //这里的左右顺序是有讲究的。
        SUSTechAtomicInt* mOriginalDataRefCount{nullptr};  // allocated and
deallocated together with mOriginalDataStart.
        uint8_t* mOriginalDataStart{nullptr};//uint8_t* 是指step按照字节来计算。
const是指Mat操作data应该根据mData指针来进行，而不是mOriginalDataStart 。
        uint8_t* mOriginalDataEnd{nullptr};//这里start和end语义与opencv不同。
    };
//    class SingleMatExpr{
//        Mat
//    };
    class MatExpr{//TODO
    public:
        Mat& A,B,C;
        Scalar a, b, c;
    };
}  // namespace SUSTech
```

- sustech_def.hpp

```
//
// Created by 叶璨铭 on 2021/11/23.
//
#pragma once
```

```
#include <array>
#include <cstdint>
#include <stdexcept>
#include <vector>
#include <string>
// opencv的interface.h的功能也放到这个里面了。
//注意宏定义不能加分号
#if defined __GNUC__
#define __SUSTC_Deprecated
#define SUSTC_Deprecated__ __attribute__ ((deprecated))
#elif defined(_MSC_VER)
#define __SUSTC_Deprecated __declspec(deprecated)
#define SUSTC_Deprecated__
#else
#define __SUSTC_Deprecated
#define SUSTC_Deprecated__
#endif

#define SUSTech_FINAL final
#define SUSTech_NOEXCEPT noexcept
#ifndef SUSTech_ALWAYS_INLINE
#if defined(__GNUC__) && (__GNUC__ > 3 || (__GNUC__ == 3 && __GNUC_MINOR__
>= 1))
#define SUSTech_ALWAYS_INLINE inline /* __attribute__((always_inline)) */
#elif defined(_MSC_VER)
#define SUSTech_ALWAYS_INLINE  __forceinline  /*inline*/
#else
#define SUSTech_ALWAYS_INLINE inline
#endif
#endif
#define SUSTech_Assert(expectedCondition, exception) \
    do {                                             \
        if(!(expectedCondition))                     \
            throw exception;                         \
    } while(false)

#include <atomic>
using SUSTechAtomicInt = std::atomic<std::int32_t>;
namespace SUSTech {

    //类型的定义：比opencv类型要少。
    // 1.最常用的四个都有了。
    // 2.需要重新计算二进制量，检验我是否真的看懂了 opencv的类型计算
    constexpr int SUSTech_CHANNEL_MAX = 1 << 12;
    constexpr int SUSTech_CHANNEL_SHIFT = 4;
    constexpr int SUSTech_CHANNEL_MASK = ((SUSTech_CHANNEL_MAX - 1) <<
SUSTech_CHANNEL_SHIFT);
    constexpr int SUSTech_DEPTH_MAX = (1 << SUSTech_CHANNEL_SHIFT);
    constexpr int SUSTech_MAT_DEPTH_MASK = SUSTech_DEPTH_MAX - 1;
    enum depth_order { SUSTech_8U, SUSTech_32S, SUSTech_32F, SUSTech_64F };
    constexpr std::array<int, 4> depth_step{ 1, 4, 4, 8 };  //避免opencv的魔法
数字移位方法。
    constexpr std::array<const char*, 4> depth_name{ "SUSTech_8U",
"SUSTech_32S", "SUSTech_32F", "SUSTech_64F"};
    constexpr double convertTo64F(double d) {
        return d;
    }
    constexpr float convertTo32F(double d) {
```

```
        return static_cast<float>(d);
    }
    constexpr int convertTo32S(double d) {
        return static_cast<int>(d);
    }
    constexpr uint8_t convertTo8U(double d) {
        return static_cast<uint8_t>(d);
    }
    template <int depth>
    struct depth_to_type;
    template <>
    struct depth_to_type<SUSTech_8U> {
        static double convert(double d) {
            return static_cast<uint8_t>(d);
        }
        static uint8_t value;
    };
    template <>
    struct depth_to_type<SUSTech_32S> {
        static double convert(double d) {
            return static_cast<int>(d);
        }
        static int32_t value;
    };
    template <>
    struct depth_to_type<SUSTech_32F> {
        static double convert(double d) {
            return static_cast<float>(d);
        }
        static float value;
    };
    template <>
    struct depth_to_type<SUSTech_64F> {
        static double convert(double d) {
            return d;
        }
        static double value;
    };  //学习了一波标准库，大概是这样写的吧
        //    constexpr auto convert(double d, int depth)-
>decltype(depth_to_type<depthOrder>::value){
        //    //decltype推断的前提是类型确定。这里编译器认为depth是变量，不能被发现是
什么
        //        switch((SUSTech::depth_order)depthOrder){
        //            case SUSTech_8U:
        //                return convertTo8U(d);
        //            case SUSTech_32S:
        //                return convertTo32S(d);
        //            case SUSTech_32F:
        //                return convertTo32F(d);
        //            case SUSTech_64F:
        //                return convertTo64F(d);
        //            default:
        //                throw std::runtime_error("new type cast not
supported!");
        //        }
        //    }
    SUSTech_ALWAYS_INLINE double doubleValueOf(const void* block, int depth)
{  //不能是constexpr，因为throw exception
```

```
        switch((SUSTech::depth_order)depth) {
            case SUSTech_8U: {
                auto concretePtr = reinterpret_cast<const
decltype(depth_to_type<SUSTech_8U>::value)*>(block);
                return static_cast<double>(*concretePtr);
            }
            case SUSTech_32S: {
                auto concretePtr = reinterpret_cast<const
decltype(depth_to_type<SUSTech_32S>::value)*>(block);
                return static_cast<double>(*concretePtr);
            }
            case SUSTech_32F: {
                auto concretePtr = reinterpret_cast<const
decltype(depth_to_type<SUSTech_32F>::value)*>(block);
                return static_cast<double>(*concretePtr);
            }
            case SUSTech_64F: {
                auto concretePtr = reinterpret_cast<const
decltype(depth_to_type<SUSTech_64F>::value)*>(block);
                return static_cast<double>(*concretePtr);
            }
            default:
                throw std::runtime_error("new type cast not supported!");
        }
        return 0;
    }

    constexpr int SUSTech_MAT_DEPTH(int type) {
        return (type & SUSTech_MAT_DEPTH_MASK);
    }
    constexpr int SUSTech_MAKE_TYPE(int depth, int channel) {
        return SUSTech_MAT_DEPTH(depth) + (((channel)-1) <<
SUSTech_CHANNEL_SHIFT);  //减1是因为0通道不存在。
    }
    constexpr int SUSTech_8UC(int channel) {
        return SUSTech_MAKE_TYPE(SUSTech_8U, channel);
    }
    constexpr int SUSTech_32SC(int channel) {
        return SUSTech_MAKE_TYPE(SUSTech_32S, channel);
    }
    constexpr int SUSTech_32FC(int channel) {
        return SUSTech_MAKE_TYPE(SUSTech_32F, channel);
    }
    constexpr int SUSTech_64FC(int channel) {
        return SUSTech_MAKE_TYPE(SUSTech_64F, channel);
    }
    enum type {
        SUSTech_8UC1 = SUSTech_8UC(1),
        SUSTech_8UC2 = SUSTech_8UC(2),
        SUSTech_8UC3 = SUSTech_8UC(3),
        SUSTech_8UC4 = SUSTech_8UC(4),

        SUSTech_32SC1 = SUSTech_32SC(1),
        SUSTech_32SC2 = SUSTech_32SC(2),
        SUSTech_32SC3 = SUSTech_32SC(3),
        SUSTech_32SC4 = SUSTech_32SC(4),

        SUSTech_32FC1 = SUSTech_32FC(1),
```

```
        SUSTech_32FC2 = SUSTech_32FC(2),
        SUSTech_32FC3 = SUSTech_32FC(3),
        SUSTech_32FC4 = SUSTech_32FC(4),

        SUSTech_64FC1 = SUSTech_64FC(1),
        SUSTech_64FC2 = SUSTech_64FC(2),
        SUSTech_64FC3 = SUSTech_64FC(3),
        SUSTech_64FC4 = SUSTech_64FC(4)
    };
    /**
     * 有了个结构体，不仅容易分离Mat的逻辑、更好的修改flag的定义，还可以避免用masks不断位操作浪费时间。（前提是取结构体的成员快于移位）
     * 坏处：使用mask的时候，如果用int8，int16，需要进行类型转换为int32，再移位。比如这里的mType不是一个好设计。
     * 但是我认为不比flags直接位操作差太多。
     */
    struct MatFlag {
        enum MatrixClassSignature { NormalMatrixSignature = 0x42 };
        enum BooleanFlags { ContinuityFlag = 1 << 4, SubMatrixFlag = 1 << 0
};
        int8_t mSignature;  //表明是Mat还是SparseMat还是std::vector。这样InputArray就可以区分了
                            //(为什么不是这些类型的矩阵继承InputArray呢，是有原因的)。
        int8_t mFlags;      // flags包括continuity(左4位)和isSubMatrix(右4位)
        int16_t mType;      /*- depthOrder (float、double、int等等),
     这里设计占4位（opencv是三位），至多支持16种数据类型(深刻的问题：为用户实现高效的bool和float16矩阵)
- number of channels（depth和channel在一起合称type），这里占12位，支持2**12的通道。因为矩阵构造的时候就传了一个type进来，所以我没有把depth和channel拆开。*/
        bool isNormalMatrix() const;
        void flipContinuityFlag();
        bool isContinuous() const;
        void flipSubMatrixFlag();
        bool isSubMatrix() const;
        int depthOrder() const;
        int depthStep() const;
        std::string depthName() const;
        int channels() const;
        bool operator==(const MatFlag& rhs) const;
        bool operator!=(const MatFlag& rhs) const;
    };
    SUSTech_ALWAYS_INLINE bool MatFlag::isNormalMatrix() const {
        return mSignature == NormalMatrixSignature;
    }
    SUSTech_ALWAYS_INLINE bool MatFlag::isContinuous() const {
        return (mFlags & ContinuityFlag) != 0;
    }
    SUSTech_ALWAYS_INLINE void MatFlag::flipContinuityFlag() {
        mFlags ^= ContinuityFlag;
    }

    SUSTech_ALWAYS_INLINE bool MatFlag::isSubMatrix() const {
        return (mFlags & SubMatrixFlag) != 0;
    }
    SUSTech_ALWAYS_INLINE void MatFlag::flipSubMatrixFlag() {
        mFlags ^= SubMatrixFlag;
    }
```

```
    SUSTech_ALWAYS_INLINE int MatFlag::depthOrder() const {
        return mType & SUSTech_MAT_DEPTH_MASK;
    }
    SUSTech_ALWAYS_INLINE int MatFlag::channels() const {
        return ((mType & SUSTech_CHANNEL_MASK) >> SUSTech_CHANNEL_SHIFT) +
1;
    }
    SUSTech_ALWAYS_INLINE int MatFlag::depthStep() const {
        return depth_step[depthOrder()];
    }
    SUSTech_ALWAYS_INLINE
    std::string MatFlag::depthName() const {
        return {depth_name[depthOrder()]};
    }
    SUSTech_ALWAYS_INLINE
    bool MatFlag::operator==(const MatFlag& rhs) const {
        return mSignature == rhs.mSignature && mFlags == rhs.mFlags && mType
== rhs.mType;
    }
    SUSTech_ALWAYS_INLINE
    bool MatFlag::operator!=(const MatFlag& rhs) const {
        return !(rhs == *this);
    }

    //    using GenericPtr = union {
    //        uint8_t* uc8;  // unsigned char
    //        int32_t* i32;  // int
    //        float* f32;    // float
    //        double* f64;   // double
    //    };
}  // namespace SUSTech
```

- sustech_simd_intrinsics.hpp

```
#pragma once
//#if __ARM_NEON_FP==1
//#include <arm64intr.h>
//#include <arm_neon.h>
//#endif
//#if (defined (_M_IX86) || defined (_M_X64)) && !defined(_CHPE_ONLY_) &&
(!defined(_M_ARM64EC) || !defined(_DISABLE_SOFTINTRIN_))
//        #include <immintrin.h>
//        #include <ammintrin.h>
//#endif
#include <intrin.h>
//TODO: add support for matmul core function using simd. add support for
vector encapsulating different simd and different depth
```

- types.hpp

```
//
// Created by 叶璨铭 on 2021/11/23.
//
#pragma once
#include "sustech_def.hpp"
#include <array>
```

```
#include <ostream>
#include <typeinfo>
namespace SUSTech {
    // opencv写了很多其实编译器可以帮我们写。
    template <typename Tp>
    class Point_ SUSTech_FINAL {  //写成aggregate就可以不用写拷贝、移动了。
    public:
        //        using Size<Tp> = Point_<Tp>;//元编程不能完成的任务。
        Tp x, y;
        template <typename Tp_>
        friend std::ostream& operator<<(std::ostream& os, const Point_<Tp_>&
point);
    };
    using Point2i = Point_<int>;
    using Point2st = Point_<size_t>;
    using Point = Point2st;

    template <typename Tp>
    class Rect_ SUSTech_FINAL {
    public:
        Tp x;       //!< x coordinate of the top-left corner
        Tp y;       //!< y coordinate of the top-left corner
        Tp width;   //!< width of the rectangle
        Tp height;  //!< height of the rectangle
        //! area (width*height) of the rectangle
        Tp area() const;
        //! true if empty
        bool empty() const;
        //! conversion to another data type
        template <typename Tp2>
        explicit operator Rect_<Tp2>() const;
        template <typename Tp2_>
        friend std::ostream& operator<<(std::ostream& os, const Rect_<Tp2_>&
rect);
    };
    using Recti = Rect_<int>;
    using Rectst = Rect_<size_t>;
    using Rect = Rectst;

    template <typename Tp>
    class Scalar_ SUSTech_FINAL {
    public:
        std::array<Tp, 4> mScalar;
        Tp& s0() const;
        Tp& s1() const;
        Tp& s2() const;
        Tp& s3() const;
        explicit operator std::array<Tp, 4>() const;
        template <typename Tp_>
        friend std::ostream& operator<<(std::ostream& os, const
Scalar_<Tp_>& scalar);
    };
    using Scalar = Scalar_<double>;

    template <typename Tp>
    class VarScalar_ {
        std::vector<Tp> mVector; //不能继承标准库的成员（因为不保证是virtual的），
所以要给他加功能，只能自己适配。
```

```
    public:
        //复制构造和移动构造,以及赋值，都是默认我想要的功能，就是把mVector复制、移动。
        VarScalar_();
        explicit VarScalar_(size_t initialCapacity);
        void push_back(const Tp& val);
        void push_back(Tp&& val);
        Tp& operator[](size_t rank);
        bool operator==(const VarScalar_& rhs) const;
        bool operator!=(const VarScalar_& rhs) const;
        template <typename Tp_>
        friend std::ostream& operator<<(std::ostream& os, const
VarScalar_<Tp_>& scalar);
    };
    using VarScalar = VarScalar_<double>;

    ////////////////////////////////////Point_
//////////////////////////////////
    template <typename Tp>
    std::ostream& operator<<(std::ostream& os, const Point_<Tp>& point) {
        os << "Point_<" << typeid(Tp).name() << ">{" << point.x << ", " <<
point.y << "}";
        return os;
    }
    ////////////////////////////////////Rect_
//////////////////////////////////
    //类型转换
    //要写两个template，是因为类本身只有一个模板参数，而不是两个。
    template <typename Tp>
    template <typename Tp2>
    SUSTech_ALWAYS_INLINE Rect_<Tp>::operator Rect_<Tp2>() const {
        return Rect_<Tp2>(static_cast<Tp2>(x), static_cast<Tp2>(y),
static_cast<Tp2>(width), static_cast<Tp2>(height));
    }
    template <typename Tp>
    SUSTech_ALWAYS_INLINE Tp Rect_<Tp>::area() const {
        return width * height;  // int，uchar的话有可能溢出的。
    }
    template <typename Tp>
    SUSTech_ALWAYS_INLINE bool Rect_<Tp>::empty() const {
        return width <= 0 || height <= 0;
    }
    template <typename Tp>
    SUSTech_ALWAYS_INLINE std::ostream& operator<<(std::ostream& os, const
Rect_<Tp>& rect) {
        os << "Rect_<" << typeid(Tp).name() << ">{"
           << "x: " << rect.x << " y: " << rect.y << " width: " <<
rect.width << " height: " << rect.height << "}";
        return os;
    }
    ////////////////////////////////////Scalar_
//////////////////////////////////
    template <typename Tp>
    SUSTech_ALWAYS_INLINE Tp& Scalar_<Tp>::s0() const {
        return mScalar[0];
    }
    template <typename Tp>
    SUSTech_ALWAYS_INLINE Tp& Scalar_<Tp>::s1() const {
        return mScalar[1];
```

```
    }
    template <typename Tp>
    SUSTech_ALWAYS_INLINE Tp& Scalar_<Tp>::s2() const {
        return mScalar[2];
    }
    template <typename Tp>
    SUSTech_ALWAYS_INLINE Tp& Scalar_<Tp>::s3() const {
        return mScalar[3];
    }
    template <typename Tp>
    SUSTech_ALWAYS_INLINE std::ostream& operator<<(std::ostream& os, const
Scalar_<Tp>& scalar) {
        os << "Scalar_<" << typeid(Tp).name() << ">{";
        for(int i = 0; i < 3; ++i) {
            if(scalar.mScalar[i] == 0)
                break;
            os << scalar.mScalar[i] << ", ";
        }
        if(scalar.mScalar[3] == 0)
            os << scalar.mScalar[3];
        return os;
    }
    template <typename Tp>
    SUSTech_ALWAYS_INLINE Scalar_<Tp>::operator std::array<Tp, 4>() const {
        return mScalar;
    }
    template <typename Cvt, typename Tp, size_t Channels>
    SUSTech_ALWAYS_INLINE std::array<Cvt, Channels> convert(const
std::array<Tp, Channels>& src) {
        std::array<Cvt, Channels> newArray;
        for(int i = 0; i < Channels; ++i) {
            newArray[i] = static_cast<Cvt>(src[i]);
        }
        return newArray;  // move constructor.
    }
    ////////////////////////////////////VarScalar_
////////////////////////////////
    //不好，不应该修改标准库。
    //    using VarScalar = std::vector<double>;
    //    std::ostream& operator<<(std::ostream& os, const VarScalar& vec){
    //        for(const auto& e:vec)
    //            os << e;
    //        return os;
    //    }
    template <typename Tp>
    bool VarScalar_<Tp>::operator==(const VarScalar_& rhs) const {
        return mVector == rhs.mVector;
    }
    template <typename Tp>
    bool VarScalar_<Tp>::operator!=(const VarScalar_& rhs) const {
        return rhs != *this;
    }
    template <typename Tp>
    Tp& VarScalar_<Tp>::operator[](const size_t rank) {
        return mVector[rank];
    }
    template <typename Tp>
```

```
    std::ostream& operator<<(std::ostream& os, const VarScalar_<Tp>& scalar)
{
        os<<"(";
        int i;
        for(i = 0; i < scalar.mVector.size()-1; ++i) {
            os<<scalar.mVector[i]<<", ";
        }
        return os<<scalar.mVector[i]<<")";
    }
    template <typename Tp>
    VarScalar_<Tp>::VarScalar_(size_t initialCapacity)
:mVector(initialCapacity){}
    template <typename Tp>
    VarScalar_<Tp>::VarScalar_() :VarScalar_(0){}
    template <typename Tp>
    void VarScalar_<Tp>::push_back(const Tp& val) {
        mVector.push_back(val);
    }
    template <typename Tp>
    void VarScalar_<Tp>::push_back(Tp&& val) {
        mVector.push_back(val);
    }

    SUSTech_ALWAYS_INLINE size_t file_len(FILE* file) {
        fseek(file, 0L, SEEK_END);
        size_t result = ftell(file);
        fseek(file, 0L, SEEK_SET);
        return result;
    }
}  // namespace SUSTech
```

- matrix.cpp

```
//
// Created by 叶璨铭 on 2021/11/23.
//
#include "mat.hpp"
#include <cstdint>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <utility>
#include <omp.h>
namespace SUSTech {
#define GetElementPtr_l3_frd(rowIndex, colIndex, channelIndex, CASE, ths)
(reinterpret_cast<decltype(depth_to_type<CASE>::value)*>((ths).mData+
(rowIndex)*(ths).mOriginalDataStepRow+(colIndex)*(ths).mOriginalDataStepCol+
(channelIndex)*(ths).mFlag.depthStep()))
#define GetElementPtr_l1_frd(depthSteps,CASE, ths)
(reinterpret_cast<decltype(depth_to_type<CASE>::value)*>((ths).mData+
(depthSteps)*(ths).mFlag.depthStep()))
    BadArgumentSize::BadArgumentSize(const std::string& message) :
runtime_error(message) {}
    // Mat
    Mat::Mat() : Mat(0, 0, 0) {}
```

```
    Mat::Mat(size_t rows, size_t cols, int type) : mFlag(MatFlag{
MatFlag::NormalMatrixSignature, MatFlag::ContinuityFlag,
static_cast<int16_t>(type) }), mRows(rows),
          mCols(cols), mData(nullptr), mDataEnd(nullptr),
mOriginalDataRefCount(nullptr),
          mOriginalDataStepCol(mFlag.depthStep() * mFlag.channels()),
mOriginalDataStepRow(mCols * mOriginalDataStepCol),
          mOriginalDataStart(nullptr), mOriginalDataEnd(nullptr) {
        if(!empty())
            create();
        //不会赋值为0.
    }
    Mat::Mat(size_t rows, size_t cols, int type, const Scalar& s)
        : mFlag(MatFlag{ MatFlag::NormalMatrixSignature,
MatFlag::ContinuityFlag, static_cast<int16_t>(type) }), mRows(rows),
          mCols(cols), mData(nullptr), mDataEnd(nullptr),
mOriginalDataRefCount(nullptr),
          mOriginalDataStepCol(mFlag.depthStep() * mFlag.channels()),
mOriginalDataStepRow(mCols * mOriginalDataStepCol),
          mOriginalDataStart(nullptr), mOriginalDataEnd(nullptr) {
        if(!empty())
            create();
        *this = static_cast<std::array<double, 4>>(s);
    }

    void Mat::create(size_t rows, size_t cols, int type, const Scalar& s) {
        if(!empty())
            release();  //将旧的矩阵释放了。 并且保证以往数据都是0.
        //构造新的矩阵头信息。
        mFlag = MatFlag{ MatFlag::NormalMatrixSignature,
MatFlag::ContinuityFlag, static_cast<int16_t>(type) };
        mRows = rows;
        mCols = cols;
        mOriginalDataStepCol = mFlag.depthStep() * mFlag.channels();
        mOriginalDataStepRow = mCols * mOriginalDataStepCol;
        create();
    }

    void Mat::create() {
        mOriginalDataRefCount = new SUSTechAtomicInt{ 1 };
        mOriginalDataStart = static_cast<uint8_t*>(malloc(mRows *
mOriginalDataStepRow));
        mOriginalDataEnd = mOriginalDataStart + mRows *
mOriginalDataStepRow;
        mData = mOriginalDataStart;   //作为创始者。
        mDataEnd = mOriginalDataEnd;  //作为创始者。
    }
    Mat::Mat(size_t rows, size_t cols, int type, void* data)
        : mFlag(MatFlag{ MatFlag::NormalMatrixSignature,
MatFlag::ContinuityFlag, static_cast<int16_t>(type) }), mRows(rows),
          mCols(cols), mData(static_cast<uint8_t*>(data)),
mOriginalDataRefCount(new SUSTechAtomicInt{ 1 }),
          mOriginalDataStepCol(mFlag.depthStep() * mFlag.channels()),
mOriginalDataStepRow(mCols * mOriginalDataStepCol),
          mOriginalDataStart(mData), mOriginalDataEnd(mOriginalDataStart +
mRows * mOriginalDataStepRow) {
        mDataEnd = mOriginalDataEnd;
        //小问题，如果opencv不保护外源数据，那么imread是怎么实现的呢？
```

```
    }

    Mat::Mat(const Mat& m)
        : mFlag(m.mFlag), mRows(m.mRows), mCols(m.mCols), mData(m.mData),
mDataEnd(m.mDataEnd),
          mOriginalDataRefCount(m.mOriginalDataRefCount),
mOriginalDataStepCol(m.mOriginalDataStepCol),
          mOriginalDataStepRow(m.mOriginalDataStepRow),
mOriginalDataStart(m.mOriginalDataStart),
          mOriginalDataEnd(m.mOriginalDataEnd) {
        (*mOriginalDataRefCount)++;  //矩阵头信息逐成员拷贝，除了引用计数递增。
    }
    Mat::Mat(Mat&& m) SUSTech_NOEXCEPT : mFlag(m.mFlag),
                                         mRows(m.mRows),
                                         mCols(m.mCols),
                                         mData(m.mData),
                                         mDataEnd(m.mDataEnd),

mOriginalDataRefCount(m.mOriginalDataRefCount),

mOriginalDataStepCol(m.mOriginalDataStepCol),

mOriginalDataStepRow(m.mOriginalDataStepRow),

mOriginalDataStart(m.mOriginalDataStart),

mOriginalDataEnd(m.mOriginalDataEnd) {
        //清零操作。
        /*m.mFlag = {0,0,0};
        m.mData = nullptr;
        m.mOriginalDataStepCol = 0;
        m.mOriginalDataStepRow = 0;
        m.mOriginalDataStart = nullptr;
        m.mOriginalDataEnd = nullptr;*/ //如果对不关键的成员清零，可能导致析构器不
知道mFlag，从而不知道怎么析构(虽然这不需要释放内存)。
        //关键的清零只有这三个，给析构函数提供信息告诉它不用释放数据就可以了。
        m.mRows = 0;
        m.mCols = 0;
        m.mOriginalDataRefCount = nullptr;
    }
    Mat::Mat(Mat m, const Rect& roiRect) : Mat(std::move(m)) {
        SUSTech_Assert(roiRect.x + roiRect.width <= mCols && roiRect.y +
roiRect.height <= mRows,
                       BadArgumentSize("Bad roi region. "));
        //还有别的特殊情况，矩形有0或者负数。因为是size_t，所以保证没有负数。0的时候，矩
阵状态可以保证置空。
        if(!mFlag.isSubMatrix())
            mFlag.flipSubMatrixFlag();
        mRows = roiRect.height;  //矩形的长度是行数
        mCols = roiRect.width;   //矩形的宽度是列数
        if(roiRect.height > 1 && mFlag.isContinuous())
            mFlag.flipContinuityFlag();  //这个连续性对于赋值、切片、取元素乃至加减
乘除至关重要。 如果是一行roi，一定是连续的。
        mData += roiRect.x * mOriginalDataStepCol;  // x是列标，和矩阵是反的。
        mData += roiRect.y * mOriginalDataStepRow;
        mDataEnd = mData + (mRows - 1) * mOriginalDataStepRow +
            mCols *
```

```
                mOriginalDataStepCol;  //这个-1很关键。rows减一，以便于定位到最后一行。但是cols不减一，表示合法位置的下一个，以吻合左闭右开语义。
        (*mOriginalDataRefCount)++;  //引用计数递增。子矩阵也是原矩阵的一份子，需要负起释放资源的责任。
    }
    Mat::Mat(Mat m, const int& channel) : Mat(std::move(m)) {
        SUSTech_Assert(0 <= channel && channel < mFlag.channels(), BadArgumentSize("Bad channel index."));
        if(!mFlag.isSubMatrix())
            mFlag.flipSubMatrixFlag();
        if(mFlag.isContinuous())
            mFlag.flipContinuityFlag();
        mData += channel * mFlag.depthStep();
        mDataEnd += channel * mFlag.depthStep();
        mFlag.mType = (int16_t)SUSTech_MAKE_TYPE(mFlag.depthOrder(), 1);
        // OriginalData的step和指针都不发生改变。但是计数加一。
        (*mOriginalDataRefCount)++;
    }

#pragma clang diagnostic push
#pragma ide diagnostic ignored "misc-unconventional-assign-operator"
    Mat& Mat::operator=(const Scalar& s) {
        return this->operator=(static_cast<std::array<double, 4>>(s));
    }
#pragma clang diagnostic pop
    //如果实际channel是3，但是传入100，就不赋值后面的。如果实际channel是100，传入了3，那么不改变后面的值(而不是设置为0)
    template <typename Tp, size_t MaxChannels>
    Mat& Mat::operator=(const std::array<Tp, MaxChannels>& _s) {
        if(this->empty() || _s.empty())
            return *this;
        const int assignmentChannels = std::min(static_cast<int>(MaxChannels), mFlag.channels());
        //把s转换为我想要的类型.
        //        depth_to_type a{depth_to_type<mFlag.depthOrder()>::value}::convert(_s[0]); //失败的，因为模板不能传入非常数。
        //                const std::array<uint8_t, MaxChannels> newS = _s; //array没有提供转换函数，我的rect才有.
        //自己写一个模板函数，避免对Scalar多次类型转换。
        const auto depth = mFlag.depthOrder();
#define MatOperatorAssignmentSwitchGenerate(CASE)                                   \
    {                                                                               \
        const auto convertedScalar = convert<decltype(depth_to_type<CASE>::value), double, MaxChannels>(_s); \
        if(mFlag.isContinuous()) {                                                  \
            for(uint8_t* elementIt = mData; (elementIt < mDataEnd); elementIt += mOriginalDataStepCol) {     \
                auto concretePtr = reinterpret_cast<decltype(depth_to_type<CASE>::value)*>(elementIt);       \
                for(int i = 0; i < assignmentChannels; ++i) {                       \
                    concretePtr[i] = convertedScalar[i];                            \
```

```
                }                                                       \
            }                                                           \
        } else {                                                        \
            for(size_t i = 0; i < mRows; ++i) {                         \
                uint8_t* rowIt = mData + i * mOriginalDataStepRow;      \
                for(size_t j = 0; j < mCols; ++j) {                     \
                    uint8_t* elementIt = rowIt + j * mOriginalDataStepCol;  \
                    auto concretePtr = reinterpret_cast<decltype(depth_to_type<CASE>::value)*>(elementIt);   \
                    for(int k = 0; k < assignmentChannels; ++k) {       \
                        concretePtr[k] = convertedScalar[k];            \
                    }                                                   \
                }                                                       \
            }                                                           \
        }                                                               \
        break;                                                          \
    }
        switch((SUSTech::depth_order)depth) {  // switch是无法避免的，switch是高效的。
            case SUSTech_8U:
                MatOperatorAssignmentSwitchGenerate(SUSTech_8U) case SUSTech_32S
                    : MatOperatorAssignmentSwitchGenerate(SUSTech_32S) case SUSTech_32F
                    : MatOperatorAssignmentSwitchGenerate(SUSTech_32F) case SUSTech_64F
                    : MatOperatorAssignmentSwitchGenerate(SUSTech_64F) default
                    : throw std::runtime_error("new type cast not supported!");
        }
        //                elementIt[i] = s[i];
        //                memcpy(elementIt[i*mFlag.depthOrder()], reinterpret_cast<size_t>newS[i], );
        return *this;
    }
    //    template Mat& Mat::operator=(const std::array<double, 4>& _s);
    //    template Mat& Mat::operator=(const std::array<float, 3>& _s);

    SUSTech_ALWAYS_INLINE bool Mat::empty() const {
        return mRows == 0 || mCols == 0;
    }

    //析构函数
```

```
    Mat::~Mat() {
        //引用减数
        release();
        //销毁自己的存在（系统自动）
    }
    void Mat::release() {
        if(empty() || mOriginalDataRefCount == nullptr)  // opencv就是通过没有计数，使得释放的时候不释放外源数据。
            return;
        if(((*mOriginalDataRefCount)--) == 1)
            destroy();
        mRows = mCols = 0;
        mData = nullptr;
        mDataEnd = nullptr;
        mOriginalDataRefCount = nullptr;  //为了保险，把自己拥有的地址清空。
        mOriginalDataStart = nullptr;
        mOriginalDataEnd = nullptr;
    }
    void Mat::destroy() {
        free(mOriginalDataStart);
        delete mOriginalDataRefCount;
    }
    //复制赋值(opencv语义，不是c++语义)
    Mat& Mat::operator=(const Mat& m) {
        if(this == &m)
            return *this;
        release();
        mFlag = m.mFlag;
        mRows = m.mRows;
        mCols = m.mCols;
        mData = m.mData;
        mDataEnd = m.mDataEnd;
        mOriginalDataRefCount = m.mOriginalDataRefCount;
        mOriginalDataStepCol = m.mOriginalDataStepCol;
        mOriginalDataStepRow = m.mOriginalDataStepRow;
        mOriginalDataStart = m.mOriginalDataStart;
        mOriginalDataEnd = m.mOriginalDataEnd;

        (*mOriginalDataRefCount)++;
        return *this;
    }
    //移动赋值
    Mat& Mat::operator=(Mat&& m) SUSTech_NOEXCEPT {
        if(this == &m)
            return *this;
        release();
        mFlag = m.mFlag;
        mRows = m.mRows;
        mCols = m.mCols;
        mData = m.mData;
        mDataEnd = m.mDataEnd;
        mOriginalDataRefCount = m.mOriginalDataRefCount;
        mOriginalDataStepCol = m.mOriginalDataStepCol;
        mOriginalDataStepRow = m.mOriginalDataStepRow;
        mOriginalDataStart = m.mOriginalDataStart;
        mOriginalDataEnd = m.mOriginalDataEnd;

        m.mRows = 0;
```

```
        m.mCols = 0;
        m.mOriginalDataRefCount = nullptr;
        return *this;
    }
    SUSTech_ALWAYS_INLINE VarScalar Mat::operator()(const Point& location)
const {
        return this->operator()(location.x, location.y);
    }
    SUSTech_ALWAYS_INLINE VarScalar Mat::operator()(size_t rowIndex, size_t
colIndex) const {
        SUSTech_Assert(rowIndex < mRows && colIndex < mCols,
BadArgumentSize{ "element access out of range. " });
        VarScalar scalar(mFlag.channels());
        const uint8_t* elementLocation = mData + (rowIndex *
mOriginalDataStepRow) + (colIndex * mOriginalDataStepCol);
        for(int i = 0; i < mFlag.channels(); i++) {
            const uint8_t* channelLocation = elementLocation + (i *
mFlag.depthStep());
            //            scalar.push_back(doubleValueOf(channelLocation,
mFlag.depthOrder()));
            scalar[i] = doubleValueOf(channelLocation, mFlag.depthOrder());
        }
        return scalar;
    }
    SUSTech_ALWAYS_INLINE Mat Mat::operator[](const Rect& roiRect) const {
        return { *this, roiRect };
    }
    SUSTech_ALWAYS_INLINE Mat Mat::operator[](const Point& location) const {
        return this->operator[]({ location.y, location.x, 1, 1 });
    }
    SUSTech_ALWAYS_INLINE Mat Mat::row(size_t rowIndex) const {
        return this->operator[]({ 0, rowIndex, mCols, 1 });  //
row(rowsThenColumnsIndex)
    }
    SUSTech_ALWAYS_INLINE Mat Mat::col(size_t colIndex) const {
        return this->operator[]({ colIndex, 0, 1, mRows });  //
col(rowsThenColumnsIndex)
    }
    //    SUSTech_ALWAYS_INLINE
    Mat Mat::channel(int channelIndex) const {
        return { *this, channelIndex };
    }
    //    SUSTech_ALWAYS_INLINE
    Mat Mat::operator[](size_t rowsThenColumnsIndex) const {
        if(mRows > 1)
            return row(rowsThenColumnsIndex);
        else
            return col(rowsThenColumnsIndex);
    }

    //调试函数
    std::ostream& operator<<(std::ostream& os, const Mat& mat) {
        os<<"Mat<"<<"depth="<<mat.mFlag.depthName()<<", "<<"channel="
<<mat.mFlag.channels()<<", "<<mat.mRows<<"x"<<mat.mCols<<">{"<<std::endl;
        for(int i = 0; i < mat.mRows; ++i) {
            //            for(int j = 0; ; ++j) {
            //                os << mat(i, j);
```

```
            //                 if(j < mat.mCols) break;//优雅地消除最后一个空格，但是这个对熔合运算不友好。
            //                 os << " ";
            //             }
            int j = 0;
            for(;j < mat.mCols-1; ++j) {
                os << mat(i, j)<< "\t";
            }
            os << mat(i, j) <<std::endl;
        }
        return os<<"};";
    }
    //对于roi，也可以从文件读入数据。所以这个函数的作用是改变已有mat的数据，而不是生成新的矩阵。
    std::istream& operator>>(std::istream& in, Mat& mat) {
        if(mat.empty())
            return in;
//        if(mat.mData==nullptr || mat.mOriginalDataRefCount==nullptr) //不是某个矩阵的roi，是刚创建的矩阵，没有申请内存
//            mat.create();
#define MatOperatorInsertionSwitchGenerate(CASE)                                    \
    {                                                                               \
        if(mat.mFlag.isContinuous()) {                                              \
            for(uint8_t* elementIt = mat.mData; elementIt < mat.mDataEnd; elementIt += mat.mOriginalDataStepCol) { \
                auto concretePtr = reinterpret_cast<decltype(depth_to_type<CASE>::value)*>(elementIt);    \
                for(int j = 0; j < mat.mFlag.channels(); ++j) {                     \
                    in >> *(concretePtr+j);                                         \
                }                                                                   \
            }                                                                       \
        } else {                                                                    \
            for(size_t i = 0; i < mat.mRows; ++i) {                                 \
                uint8_t* rowIt = mat.mData + i * mat.mOriginalDataStepRow;          \
                for(size_t j = 0; j < mat.mCols; ++j) {                             \
                    uint8_t* elementIt = rowIt + j * mat.mOriginalDataStepCol;      \
                    auto concretePtr = reinterpret_cast<decltype(depth_to_type<CASE>::value)*>(elementIt);  \
                    for(int k = 0; k < mat.mFlag.channels(); ++k) {                 \
                        in >> concretePtr[k];                                       \
                    }                                                               \
```

```
                }
                                        \
            }
                                        \
        }
                                        \
        break;
                                       \
    }
        switch((SUSTech::depth_order)mat.mFlag.depthOrder()) {
            case SUSTech_8U:
                MatOperatorInsertionSwitchGenerate(SUSTech_8U) case
SUSTech_32S
                    : MatOperatorInsertionSwitchGenerate(SUSTech_32S) case
SUSTech_32F
                    : MatOperatorInsertionSwitchGenerate(SUSTech_32F) case
SUSTech_64F
                    : MatOperatorInsertionSwitchGenerate(SUSTech_64F)
default
                    : throw std::runtime_error("new type cast not
supported!");
        }
        return in;
    }
     Mat Mat::matRead(const std::string& file_name) {
        FILE* file = fopen(file_name.c_str(), "r");
        SUSTech_Assert(file != nullptr, std::ios_base::failure{ "Fatal
Error: File not found.\nmatrix reading terminated." });
        size_t file_length = file_len(file);
        size_t max_matrix_length = (file_length + 1) / 2;  //根据文件大小，解不
等式得到
        float* data = (float*)calloc(max_matrix_length, sizeof(float));
        int k = 0;
        size_t rowCount = 0, columnCount = 0;
        size_t columnCountMax = 0;  //千万不能设置为INT32_MIN! ! !
        size_t buffer_max_size = max_matrix_length * 32;
        //一行的信息，极端情况下一行有2048个数，每个数很长很长，假设字面量最长为32
        char* buffer = (char*)calloc(buffer_max_size, sizeof(char));
        for(rowCount = 0; fgets(buffer, buffer_max_size, file) != nullptr;
++rowCount) {
            columnCount = 0;
            char* token = strtok(buffer, " ");  //不能正则的。而且还是个指针，只能
一步步走。
            while(token != NULL) {
                columnCount++;
                sscanf(token, "%f", &data[k++]);
                token = strtok(NULL, " ");
            }
            if(columnCount > columnCountMax)
                columnCountMax = columnCount;
        }
        fclose(file);
        data = static_cast<float*>(realloc(data, rowCount * columnCountMax *
sizeof(float)));
        return { rowCount, columnCount, SUSTech_32FC1, data };
    }

    //前提是同型。
```

```
    Mat& fast_matrix_element_by_element_linear_combinations_assignment(Mat&
ths, const Mat& matrixA, const Mat& matrixB, float _alpha,
                                                                       float
_beta) {
#define alpha_beta(CASE) \
        const auto alpha = static_cast<decltype(depth_to_type<CASE>::value)>
(_alpha);\
        const auto beta = static_cast<decltype(depth_to_type<CASE>::value)>
(_beta)
#define
slow_matrix_element_by_element_linear_combinations_assignment_CASE(CASE) \
        for(int64_t i = 0; i < ths.mRows; ++i) {\
            for(size_t j = 0; j < ths.mCols; ++j) {\
                for(int k = 0; k < ths.mFlag.channels(); ++k) {\
                    (*GetElementPtr_l3_frd(i, j, k, CASE, ths)) =
(*GetElementPtr_l3_frd(i, j, k, CASE, matrixA))*alpha +
(*GetElementPtr_l3_frd(i, j, k, CASE, matrixB))*beta; \
                }\
            }\
        }
#define
fast_matrix_element_by_element_linear_combinations_assignment_CASE(CASE) \
    for(int64_t i = 0; i < ths.mRows*ths.mCols*ths.mFlag.channels(); ++i) {\
        (*GetElementPtr_l1_frd(i, CASE, ths)) = (*GetElementPtr_l1_frd(i,
CASE, matrixA))*alpha + (*GetElementPtr_l1_frd(i, CASE, matrixB))*beta;\
    }
        switch ((SUSTech::depth_order)ths.mFlag.depthOrder()) {
            case SUSTech_8U: {
                alpha_beta(SUSTech_8U);
                if(ths.mFlag.isContinuous()) {
#pragma omp parallel for

 fast_matrix_element_by_element_linear_combinations_assignment_CASE(SUSTech_
8U)
                } else {
#pragma omp parallel for

 slow_matrix_element_by_element_linear_combinations_assignment_CASE(SUSTech_
8U)
                }
                break;
            }
            case SUSTech_32S: {
                alpha_beta(SUSTech_32S);
                if(ths.mFlag.isContinuous()) {
#pragma omp parallel for

 fast_matrix_element_by_element_linear_combinations_assignment_CASE(SUSTech_
32S)
                } else {
#pragma omp parallel for

 slow_matrix_element_by_element_linear_combinations_assignment_CASE(SUSTech_
32S)
                }
                break;
            }
            case SUSTech_32F: {
```

```
                alpha_beta(SUSTech_32F);
                if(ths.mFlag.isContinuous()) {
#pragma omp parallel for

 fast_matrix_element_by_element_linear_combinations_assignment_CASE(SUSTech_
32F)
                } else {
#pragma omp parallel for

 slow_matrix_element_by_element_linear_combinations_assignment_CASE(SUSTech_
32F)
                }
                break;
            }
            case SUSTech_64F: {
                alpha_beta(SUSTech_64F);
                if(ths.mFlag.isContinuous()) {
#pragma omp parallel for

 fast_matrix_element_by_element_linear_combinations_assignment_CASE(SUSTech_
64F)
                } else {
#pragma omp parallel for

 slow_matrix_element_by_element_linear_combinations_assignment_CASE(SUSTech_
64F)
                }
                break;
            }
        }
        return ths;
    }
    Mat& Mat::operator+=(const Mat& matB) {
        SUSTech_Assert(matB.mFlag==this->mFlag && matB.mRows==this->mRows &&
matB.mCols==this->mCols, BadArgumentSize("Incompatible matrix addition."));
        return
fast_matrix_element_by_element_linear_combinations_assignment(*this, *this,
matB, 1.f, 1.f);
    }
    Mat& Mat::operator-=(const Mat& matB) {
        SUSTech_Assert(matB.mFlag==this->mFlag && matB.mRows==this->mRows &&
matB.mCols==this->mCols, BadArgumentSize("Incompatible matrix
subtraction."));
        return
fast_matrix_element_by_element_linear_combinations_assignment(*this, *this,
matB, 1.f, -1.f);
    }
    Mat Mat::operator+(const Mat& matB) {
        SUSTech_Assert(matB.mFlag==this->mFlag && matB.mRows==this->mRows &&
matB.mCols==this->mCols, BadArgumentSize("Incompatible matrix addition."));
        Mat matC{mRows, mCols, mFlag.mType, Scalar{0} };
        return
fast_matrix_element_by_element_linear_combinations_assignment(matC, *this,
matB, 1.f, 1.f);
    }

    Mat Mat::operator-(const Mat& matB) {
```

```
        SUSTech_Assert(matB.mFlag==this->mFlag && matB.mRows==this->mRows &&
matB.mCols==this->mCols, BadArgumentSize("Incompatible matrix
subtraction."));
        Mat matC{mRows, mCols, mFlag.mType, Scalar{0} };
        return
fast_matrix_element_by_element_linear_combinations_assignment(matC, *this,
matB, 1.f, -1.f);
    }
    Mat Mat::matMul(const Mat& matB) {
        SUSTech_Assert(matB.mFlag==this->mFlag && this->mCols == matB.mRows
, BadArgumentSize("Incompatible matrix multiplication."));
        Mat result {mRows, matB.mCols, mFlag.mType, Scalar{0}};
        switch ((SUSTech::depth_order)mFlag.depthOrder()) {
            case SUSTech_8U: {
#pragma omp parallel for
                for(int64_t i = 0; i < this->mRows; ++i) {
                    std::vector<decltype(depth_to_type<SUSTech_8U>::value)>
temps(mFlag.channels());
                    for(size_t k = 0; k < this->mCols; ++k) {
#pragma omp simd
                        for(int l = 0; l < mFlag.channels(); ++l) {
                            temps[l] = *GetElementPtr_l3_frd(i, k,
l,SUSTech_8U, (*this));
                        }
                        for(size_t j = 0; j < matB.mCols; ++j) {
#pragma omp simd
                            for(int l = 0; l < mFlag.channels(); ++l) {
                                (*GetElementPtr_l3_frd( i, j, l, SUSTech_8U,
result)) += temps[l] * (*GetElementPtr_l3_frd( k, j, l, SUSTech_8U, matB));
                            }
                        }
                    }
                }
                break;
            }
            case SUSTech_32S: {
#pragma omp parallel for
                for(int64_t i = 0; i < this->mRows; ++i) {
                    std::vector<decltype(depth_to_type<SUSTech_32S>::value)>
temps(mFlag.channels());
                    for(size_t k = 0; k < this->mCols; ++k) {
#pragma omp simd
                        for(int l = 0; l < mFlag.channels(); ++l) {
                            temps[l] = *GetElementPtr_l3_frd(i, k,
l,SUSTech_32S, (*this));
                        }
                        for(size_t j = 0; j < matB.mCols; ++j) {
#pragma omp simd
                            for(int l = 0; l < mFlag.channels(); ++l) {
                                (*GetElementPtr_l3_frd( i, j, l,
SUSTech_32S, result)) += temps[l] * (*GetElementPtr_l3_frd( k, j, l,
SUSTech_32S, matB));
                            }
                        }
                    }
                }
                break;
            }
```

```
            case SUSTech_32F: {
#pragma omp parallel for
                for(int64_t i = 0; i < this->mRows; ++i) {
                    std::vector<decltype(depth_to_type<SUSTech_32F>::value)>
temps(mFlag.channels());
                    for(size_t k = 0; k < this->mCols; ++k) {
#pragma omp simd
                        for(int l = 0; l < mFlag.channels(); ++l) {
                            temps[l] = *GetElementPtr_l3_frd(i, k,
l,SUSTech_32F, (*this));
                        }
                        for(size_t j = 0; j < matB.mCols; ++j) {
#pragma omp simd
                            for(int l = 0; l < mFlag.channels(); ++l) {
                                (*GetElementPtr_l3_frd( i, j, l,
SUSTech_32F, result)) += temps[l] * (*GetElementPtr_l3_frd( k, j, l,
SUSTech_32F, matB));
                            }
                        }
                    }
                }
                break;
            }
            case SUSTech_64F: {
#pragma omp parallel for
                for(int64_t i = 0; i < this->mRows; ++i) {
                    std::vector<decltype(depth_to_type<SUSTech_64F>::value)>
temps(mFlag.channels());
                    for(size_t k = 0; k < this->mCols; ++k) {
#pragma omp simd
                        for(int l = 0; l < mFlag.channels(); ++l) {
                            temps[l] = *GetElementPtr_l3_frd(i, k,
l,SUSTech_64F, (*this));
                        }
                        for(size_t j = 0; j < matB.mCols; ++j) {
#pragma omp simd
                            for(int l = 0; l < mFlag.channels(); ++l) {
                                (*GetElementPtr_l3_frd( i, j, l,
SUSTech_64F, result)) += temps[l] * (*GetElementPtr_l3_frd( k, j, l,
SUSTech_64F, matB));
                            }
                        }
                    }
                }
                break;
            }
        }
        return result;
    }

}  // namespace SUSTech
```

- matmul_benchmark.cpp

```
#include <iostream>
#include <sstream>
#include <fstream>
```

```cpp
#include <benchmark/benchmark.h>
#include "mat.hpp"
constexpr int MATRIX_DEPTH = SUSTech::SUSTech_32F;
constexpr int CHANNELS = 1;
constexpr int MATRIX_TYPE = SUSTech::SUSTech_MAKE_TYPE(MATRIX_DEPTH,
CHANNELS);
static void readMatrix(benchmark::State& state){
    size_t size = state.range(0);
    char path_a_in[256];
    sprintf(path_a_in,"../resources/input_data/mat-A-%d.txt", size);
    std::ifstream fA{path_a_in};
    SUSTech::Mat matA {size, size/CHANNELS, MATRIX_TYPE};
    for (auto _: state) {
        fA>>matA;
    }
}
BENCHMARK(readMatrix)->Arg(32)->RangeMultiplier(2)->Range(32, 2048)-
>Iterations(5)->Complexity(benchmark::oN);
static void writeMatrix(benchmark::State& state){
    size_t size = state.range(0);
    char path_a_in[256];
    char path_a_out[256];
    sprintf(path_a_in,"../resources/input_data/mat-A-%d.txt", size);
    sprintf(path_a_out,"../resources/output_data/mat-A-%d.txt", size);
    std::ifstream fA{path_a_in};
    std::ofstream fAo{path_a_out};
    SUSTech::Mat matA {size, size/CHANNELS, MATRIX_TYPE};          fA>>matA;
    for (auto _: state) {
        fAo<<matA;
    }
}
BENCHMARK(writeMatrix)->Arg(32)->RangeMultiplier(2)->Range(32, 2048)-
>Iterations(5)->Complexity(benchmark::oN);

//测试乘法所需时间
static void testMultiplication(benchmark::State& state){
    //    数据准备
    size_t size = state.range(0);
    char path_a_in[256];
    char path_b_in[256];
    char path_c_out[256];
    sprintf(path_a_in,"../resources/input_data/mat-A-%d.txt", size);
    sprintf(path_b_in,"../resources/input_data/mat-A-%d.txt", size);
    sprintf(path_c_out,"../resources/output_data/mat-C-%d.txt", size);
    std::ifstream fA{path_a_in};
    SUSTech::Mat matA {size, size/CHANNELS, MATRIX_TYPE}; fA>>matA;
    std::ifstream fB{path_b_in};
    SUSTech::Mat matB {size, size/CHANNELS, MATRIX_TYPE}; fB>>matB;
    std::ofstream fC{path_c_out};
    SUSTech::Mat matC {};
    //    开始测试
    for (auto _: state) {
        matC = matA.matMul(matB);
    }
    fC<<matC;
}
BENCHMARK(testMultiplication)->Arg(32)->RangeMultiplier(2)->Range(32, 2048)-
>Iterations(5)->Complexity(benchmark::oN);
```

```
BENCHMARK_MAIN();
```

参考文献

1. OpenCV官方文档 https://docs.opencv.org/4.x/

2. 本贾尼·斯特劳斯特鲁普, 斯特劳斯特鲁普, 王刚,等. C++程序设计语言[M]. 机械工业出版社, 2016.

3. OpenCV: Mat - The Basic Image Container

4. opencv - What is the purpose of cv::MatExpr? - Stack Overflow

5. https://github.com/flame/how-to-optimize-gemm/wiki#the-gotoblasblis-approach-to-optimizing-matrix-matrix-multiplication---step-by-step

6. Optimizing C Code with Neon Intrinsics (arm.com)