mex与mexcuda使用方法

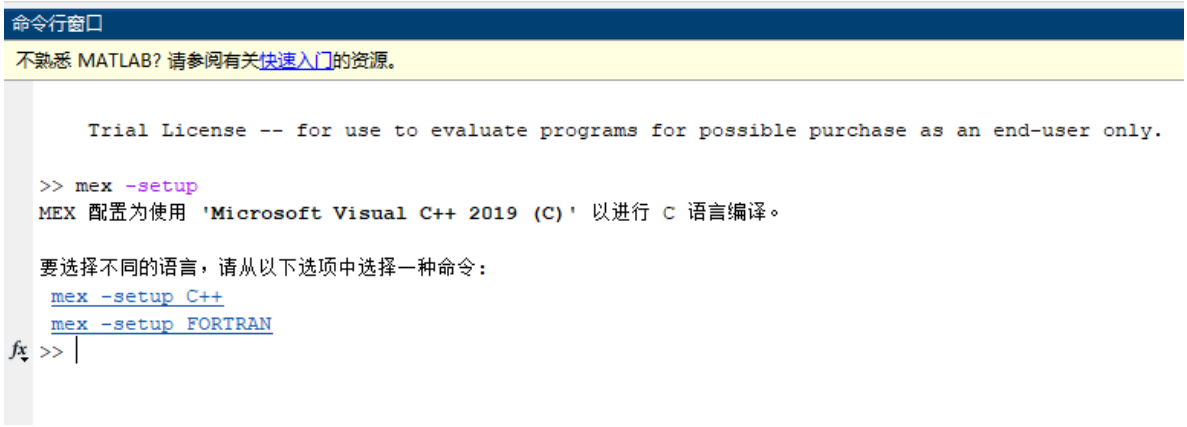
1.安装配置并测试环境

本次安装的各项工具版本分别为`MATLAB2022` `VS2019` `CUDA_11.3`

安装次序为 VS -> CUDA ->MATLAB2022，安装完之后重启电脑

不需要修改文件等繁杂配置过程

- `mex -setup` 自动配置VC++2019编译器



以上内容表示C++编译器以配置成功，如果不成功，重启电脑

2.以mex为例

- 代码 vector.cpp

  该函数的作用是对传入的数据进行取平方

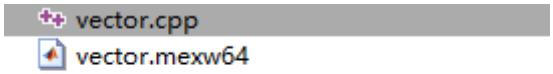
```
#include "mex.h"
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    int i, j, m, n;
    double *data1, *data2;
    /* Error checking */
    if (nrhs != nlhs)
        mexErrMsgTxt("The number of input and output arguments must be the
same.");
    for (i = 0; i < nrhs; i++)
    {
        /* Find the dimensions of the data */
        m = mxGetM(prhs[i]);
        n = mxGetN(prhs[i]);
        /* Create an mxArray for the output data */
        plhs[i] = mxCreateDoubleMatrix(m, n, mxREAL);
        /* Retrieve the input data */
        data1 = mxGetPr(prhs[i]);
        /* Create a pointer to the output data */
        data2 = mxGetPr(plhs[i]);
        /* Put data in the output array after squaring them */
        for (j = 0; j < m * n; j++)
```

```
        {
            data2[j] = data1[j] * data1[j];
        }
    }
}
```

- 编译

```
>> mex vector.cpp
使用 'Microsoft Visual C++ 2019' 编译。
MEX 已成功完成。
>>
```

```
>> mex vertor.cpp
```



产生文件verctor.mexw64

- 执行

```
>> a = [1,2,3,4,5,6,7,8,9]

a =

     1     2     3     4     5     6     7     8     9

>> b = vector(a)

b =

     1     4     9    16    25    36    49    64    81
```

文件名即为函数名，且支持多维度矩阵

```
>> a = [1,2,3,4,5,6,7,8,9;1,2,3,4,5,6,7,8,9]

a =

     1     2     3     4     5     6     7     8     9
     1     2     3     4     5     6     7     8     9

>> b = vector(a)

b =

     1     4     9    16    25    36    49    64    81
     1     4     9    16    25    36    49    64    81
```

3.以mexcuda为例

- 代码 cudaTest.cu

```
#include "cuda_runtime.h"
#include "mex.h"
/* Kernel to square elements of the array on the GPU */
__global__ void square_elements(float *in, float *out, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N)
        out[idx] = in[idx] * in[idx];
}
/* Gateway function */
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    int i, j, m, n;
    double *data1, *data2;
    float *data1f, *data2f;
    float *data1f_gpu, *data2f_gpu;
    mxClassID category;
    if (nrhs != nlhs)
        mexErrMsgTxt("The number of input and output arguments must be the
same.");
    for (i = 0; i < nrhs; i++)
    {
        /* Find the dimensions of the data */
        m = mxGetM(prhs[i]);
        n = mxGetN(prhs[i]);
        /* Create an mxArray for the output data */
        plhs[i] = mxCreateDoubleMatrix(m, n, mxREAL);
        /* Create an input and output data array on the GPU*/
        cudaMalloc((void **)&data1f_gpu, sizeof(float) * m * n);
        cudaMalloc((void **)&data2f_gpu, sizeof(float) * m * n);
        /* Retrieve the input data */
        data1 = mxGetPr(prhs[i]);
        /* Check if the input array is single or double precision */
        category = mxGetClassID(prhs[i]);
        if (category == mxSINGLE_CLASS)
        {
            /* The input array is single precision, it can be sent directly to
the card */
            cudaMemcpy(data1f_gpu, data1, sizeof(float) * m * n,
cudaMemcpyHostToDevice);
        }
        if (category == mxDOUBLE_CLASS)
        {
            /* The input array is in double precision, it needs to be converted
to floats before being sent to the card */
            data1f = (float *)mxMalloc(sizeof(float) * m * n);
            for (j = 0; j < m * n; j++)
            {
                data1f[j] = (float)data1[j];
            }
            cudaMemcpy(data1f_gpu, data1f, sizeof(float) * n * m,
cudaMemcpyHostToDevice);
        }
```

```
        data2f = (float *)mxMalloc(sizeof(float) * m * n);
        /* Compute execution configuration using 128 threads per block */
        dim3 dimBlock(128);
        dim3 dimGrid((m * n) / dimBlock.x);
        if ((n * m) % 128 != 0)
            dimGrid.x += 1;
        /* Call function on GPU */
        square_elements<<<dimGrid, dimBlock>>>(data1f_gpu, data2f_gpu, n * m);
        /* Copy result back to host */
        cudaMemcpy(data2f, data2f_gpu, sizeof(float) * n * m,
cudaMemcpyDeviceToHost);
        /* Create a pointer to the output data */
        data2 = mxGetPr(plhs[i]);
        /* Convert from single to double before returning */
        for (j = 0; j < m * n; j++)
        {
            data2[j] = (double)data2f[j];
        }
        /* Clean-up memory on device and host */
        mxFree(data1f);
        mxFree(data2f);
        cudaFree(data1f_gpu);
        cudaFree(data2f_gpu);
    }
}
```

- 编译

```
>> mexcuda cudaTest.cu
使用 'NVIDIA CUDA Compiler' 编译。
MEX 已成功完成。
```

`>> mexcuda cudaTest.cu`

即可完成编译

- 执行

```
>> b = cudaTest(a)

b =

     1     4     9    16    25    36    49    64    81
     1     4     9    16    25    36    49    64    81
```

`>> b = cudaTest(a)`

程序顺利执行

4.基本函数

MATLAB是以矩阵为核心的，所有的数据类型都可以用mxArray来描述，并且其接口函数的所有输入输出参数都是采用mxArray的形式来实现的，这里简单介绍几个常用的接口操作函数。

- (1) `double mxGetSalar(const mxArray *array_ptr)`

功能：获得某个阵列的实数部分的第一个数据

参数说明：*array_ptr 输入阵列指针

返回值：函数返回一个双精度类型的数据，相当于C语言中的标量。通过此函数，用户可以获得某个阵列的实数部分的第一个数据

- (2) `double * mxGetPr(const mxArray * array_ptr)`

功能：获得阵列的实数部分的数据指针

参数说明：*array_ptr 输入阵列指针

返回值：函数返回一个双精度类型指向实数部分数据的指针。通过此函数，用户可以获得由array_ptr指向的阵列的实数部分数据的指针

- (3) `int mxGetM(const mxArray * array_ptr)`

功能：获得阵列的行数

参数说明：*array_ptr 输入阵列指针

返回值：返回一个整数。通过此函数，用户可以获得输入阵列的的行数

- (4) `int mxGetN(const mxArray * array_ptr)`

功能：获得阵列的列数

参数说明：*array_ptr 输入阵列指针

返回值：返回一个整数。通过此函数，用户可以获得输入阵列的的列数

- (5) `mxArray * mxCreatDoubleMatrix(int m , int  n , mxREAL)`

功能：创建一个二维的未赋值的双精度浮点类型的阵列

参数说明：m 表示行数，n 表示列数，mxREAL表示是实数阵列

返回值：返回一个指向新创建阵列的指针

5.代码解释：

- （1）在编写C-MEX文件，一定要包含 #include”mex.h” 语句。
- （2）接口子程序：

```
void mexFunction(
    int nlhs,
    mxArray *plhs[],
    int nrhs,
    const mxArray *prhs[] )
```

mexFunction

它是MATLAB为C代码提供的接口函数，其中

`mexFunction`：接口子程序的名字

`nrhs`：整型，表示输入元素的个数

`nlhs`：整型，表示输出元素的个数

`prhs[]`：mxArray 结构体类型指针数组，这个数组的数组元素按顺序(prhs[0],prhs[1],……)指向所有的输入参数

`plhs[]`：mxArray 结构体类型指针数组，这个数组的数组元素按顺序(plhs[0],plhs[1],……)指向所有的输出参数

- （3）剩下要考虑的就是MATLAB与C代码模块的数据交互问题了， prhs[]是一个数组指针，功能相当C语言中的scanf()函数，通过它可以获得在MATLAB命令窗口中的输入参数；同理，plhs[]功能相当于C语言中的printf()函数，通过它可以在MATLAB命令窗口输出结果。