矩阵操作

基本按照一下要求来写的

1. 矩阵沿行(或者列)求最大值,用标准 C++的递归写:

void Matrix_Max(float* data_in, int row, int colum, int* index, float* value, int flag);

其中，data_in 为输入数据矩阵，row 为矩阵行数(最大 16384)，colum 为矩阵列数(最大 8192)；

index 为矩阵最大值的位置，value 为矩阵最大值的返回值。

flag=0 时按行求最大值，flag=1 时按列求最大值。

给出了计算时的显存资源分析及计算时间,给出调用说明，程序加入注释。

2. 矩阵沿行(或者列)求和,用标准 C++的递归写:

实数求和：void Matrix_Sum(float* data_in, int row, int colum, float* value, int flag);

复数求和：void Matrix_Sum(cuComplex* data_in, int row, int colum, cuComplex * value, int flag);

flag=0 时按行求和，flag=1 时按列求和, flag=2 时为矩阵二维求和。

3. 矩阵乘法,用标准 C++写:

复数乘法：void Matrix_Multi(cuComplex* data_1, int row1, int colum1,cuComplex* data_2，int row2, int colum2, cuComplex * data_out, int flag);

其中，data_out= data_1* data_2;

row1 为 data_1 的行数，colum1 为 data_1 的列数。row2 为 data_2 的行数，colum2 为 data_2 的列数。

当 colum1 不等于 row2 时，出现错误，flag=1;

4 矩阵转置算法：

复数转置和实数转置:

/*实数矩阵转置

/*@param Matrix_in : 输入需要转置的矩阵内存数据一维连续存储

/*@param row : 输入矩阵行数

/*@param col : 输入矩阵列数

/*@param Matrix_out: 输出 转置后的矩阵

void Matrix_Transpose(const float *Matrix_in, int row, int col, float *Matrix_out);

/*复数矩阵转置

/*@param Matrix_in : 输入需要转置的矩阵内存数据一维连续存储

/*@param row : 输入矩阵行数

/*@param col : 输入矩阵列数

/*@param Matrix_out: 输出 转置后的矩阵

void Matrix_TransposeCom(const float2 *Matrix_in, int row, int col, float2 *Matrix_out);