Lab 1：矩阵乘法

问题描述

输入三个矩阵

$$A \in \mathbb{R}_{m\times k},\ B \in \mathbb{R}_{k\times n},\ C \in \mathbb{R}_{m\times n}$$

和 $\alpha, \beta \in \mathbb{R}$ ，请计算 $\alpha \cdot AB + \beta \cdot C$ ，并且计算结果保存在输入矩阵 C 。具体来说，请编写C语言程序完成矩阵乘法的接口函数。

C | 复制代码

```c
void student_gemm(int m, int n, int k, const double * A, const double * B, double * C, double alpha, double beta, int lda, int ldb, int ldc);
```

输入矩阵的格式规定是列主序（column major），即第一个元素 A[0, 0] 下一个元素为 A[1, 0] 而非 A[0, 1]。你可以透过 A[i + j * m] 方式索引一个 m x n 矩阵A的第 i 行、第 j 列的元素 Aij。

参数 lda, ldb, ldc 表示矩阵 A, B, C 的列偏移（Leading Dimension of A, B, C）。当你的输入矩阵是大矩阵中的子矩阵，如果继续使用 A[i + j * m] 方式索引将会出错，因此在列主序下读取元素 Aij 的通用方式为A[i + j * lda]。你或许需要思考一下调用上述接口时 lda, ldb, ldc 的初始值。

框架代碼

C | 复制代码

```c
#include <cstring>
#include <cstdlib>
#include <cstdio>
#include <ctime>

#define MEASURE(__ret_ptr, __func, ...)           \
    ((clock_gettime(CLOCK_MONOTONIC, &start), \
      *(__ret_ptr) = __func(__VA_ARGS__),         \
      clock_gettime(CLOCK_MONOTONIC, &end)),  \
     (end.tv_sec - start.tv_sec) + 1e-9 * (end.tv_nsec - start.tv_nsec

#define MEASURE_VOID(__func, ...)                 \
    ((clock_gettime(CLOCK_MONOTONIC, &start), \
      __func(__VA_ARGS__),                        \
      clock_gettime(CLOCK_MONOTONIC, &end)),  \
     (end.tv_sec - start.tv_sec) + 1e-9 * (end.tv_nsec - start.tv_nsec

#define MAX(x, y) (((x) > (y)) ? (x) : (y))
#define MIN(x, y) (((x) < (y)) ? (x) : (y))
#define ABS(x) (((x) >= 0.0) ? (x) : -(x))

void RandomFill(struct drand48_data * buf_p, double *d, size_t count)
{
    for (size_t i = 0; i < count; ++i)
    {
        drand48_r(buf_p, d + i);
        d[i] = 2 * d[i] - 1.0;
    }
}

void student_gemm(int m, int n, int k, const double * A, const double
B, double * C, double alpha, double beta, int lda, int ldb, int ldc)
{
    /* TODO */
}

void naive_gemm(int m, int n, int k, const double * A, const double *
B, double * C, double alpha, double beta, int lda, int ldb, int ldc)
{
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            C[i + j * ldc] *= beta;
            for (int p = 0; p < k; p++) {
                C[i + j * ldc] += alpha * A[i + p * lda] * B[p + j * l
b];
            }
        }
    }
    return;
```

```
48  }
49
50  void mm_test(int m, int n, int k)
51  {
52      int lda = m;
53      int ldb = k;
54      int ldc = m;
55
56      double *A = (double *)aligned_alloc(64, m * k * sizeof(double));
57      double *B = (double *)aligned_alloc(64, k * n * sizeof(double));
58      double *C = (double *)aligned_alloc(64, m * n * sizeof(double));
59      double *C_ans = (double *)aligned_alloc(64, m * n * sizeof(double)
60
61      struct drand48_data buffer;
62      srand48_r(time(NULL), &buffer);
63
64      RandomFill(&buffer, A, m * k);
65      RandomFill(&buffer, B, k * n);
66      RandomFill(&buffer, C, m * n);
67
68      memcpy(C_ans, C, sizeof(double) * m * n);
69
70      double alpha, beta;
71
72      drand48_r(&buffer, &alpha);
73      drand48_r(&buffer, &beta);
74      alpha = 2 * alpha - 1.0;
75      beta = 2 * beta - 1.0;
76
77      /* test performance */
78
79      const int TRIAL = 5;
80      struct timespec start, end;
81      double t_min = __DBL_MAX__;
82
83      for (int i = 0; i < TRIAL; i++) {
            double t = MEASURE_VOID(student_gemm, m, n, k, A, B, C, alpha,
84  eta, lda, ldb, ldc);
85
86          t_min = MIN(t, t_min);
87      }
88
89      printf("minimal time spent: %.4f ms\n", t_min * 1000);
90      fflush(stdout);
91
92      /* test correctness */
93
94      memcpy(C, C_ans, sizeof(double) * m * n);
95      student_gemm(m, n, k, A, B, C, alpha, beta, lda, ldb, ldc);
96      naive_gemm(m, n, k, A, B, C_ans, alpha, beta, lda, ldb, ldc);
97
```

```
98      double max_err = __DBL_MIN__;
99      for (int i = 0; i < m; i++) {
100         for (int j = 0; j < n; j++) {
101             int idx = i + j * ldc;
102             double err = ABS(C_ans[idx] - C[idx]);
103             max_err = MAX(err, max_err);
104         }
105     }
106     const double threshold = 1e-7;
107     const char * judge_s = (max_err < threshold) ? "correct" : "wrong"
108
109     printf("result: %s (err = %e)\n", judge_s, max_err);
110
111     free(A);
112     free(B);
113     free(C);
114     free(C_ans);
115 }
116
117 int main(int argc, const char * argv[])
118 {
119     if (argc != 4)
120     {
121         printf("Test usage: ./test m n k\n");
122         exit(-1);
123     }
124
125     int m = atoi(argv[1]);
126     int n = atoi(argv[2]);
127     int k = atoi(argv[3]);
128
129     printf("input: %d x %d x %d\n", m, n, k);
130     fflush(stdout);
131
132     mm_test(m, n, k);
133 }
```

助教批改原则

首先，你必须答案正确，才能获得基本分数。

本次实验的重点是性能优化，你的矩阵乘法函数计算速度越快越好。请注意，矩阵乘法的性能表现攸关你的成绩，这意味着抄袭框架代码中用来检查正确性的 naive_gemm 函数并没有意义。

另外，请勿修改框架代码，同学们唯一的任务是完成 student_gemm 函数。如果框架代码存在问题，请及时反馈，助教将会尽快修正文档。

提交方法

实验代码必须提交到校内服务器。进入校内网后（可能需要復旦大学校园VPN），使用命令行或SSH相关工具登入服务器。命令行SSH指令：

```
ssh [your-student-ID]@10.192.9.250
```

所有人的密码均为123。

请将代码与实验报告一起放在相同的文件夹底下，文件夹命名格式为 [student-id]_lab_1。你可以透过 scp 指令上传至自己帐户的目录/home/[your-student-id] 。关于具体的指令使用方法，请参考 scp 指令相关资料。

提交截止时间：2022.10.26 23:59