中山大学计算机院本科生实验报告

（2023 学年秋季学期）

课程名称：高性能计算程序设计　　**批改人：**

实验	**通用矩阵乘法**	专业（方向）	**信息与计算科学**
学号	21311359	姓名	**何凯迪**
Email	hekd@mail2.sysu.edu.cn	完成日期	2023/8/28

1. 实验目的

随机生成 M*N 和 N*K 的两个矩阵 A,B,矩阵元素为单精度浮点数（float），对这两个矩阵做乘法得到矩阵 C。输出 A,B,C 三个矩阵以及矩阵计算的时间。采用多种语言实现矩阵乘法，通过编译器优化做对比。

2. 实验过程和核心代码

（一）先对 M、N、K 分别在 512~2048 范围内随机取值，并随机生成 A、B 矩阵。其中，使用 numpy 的 random.rand 函数来生成一个 MxN 的随机矩阵，所有元素的值都在 0 到 1 之间，服从均匀分布。这意味着矩阵中的每个元素都是一个 0 到 1 之间的随机浮点数，并使用 astype(np.float32)将生成的随机矩阵转换为单精度浮点数类型（float32）。同时创建一个零矩阵，其大小为 M*K，元素类型为单精度浮点数（float32）。计算矩阵 A*B 得到矩阵 C。代码如下：

```
import numpy as np
import time
# 随机生成M*N和N*K的矩阵 A 和 B，元素为单精度浮点数
M = np.random.randint(512, 2049)
N = np.random.randint(512, 2049)
K = np.random.randint(512, 2049)
A = np.random.rand(M, N).astype(np.float32)
B = np.random.rand(N, K).astype(np.float32)
# 初始化结果矩阵 C
C = np.zeros((M, K), dtype=np.float32)
# 执行矩阵乘法并测量时间
start_time = time.time()
for i in range(M):
    for j in range(K):
        for k in range(N):
            C[i, j] += A[i, k] * B[k, j]
end_time = time.time()
# 打印矩阵 A、B 和 C 以及执行时间
print('N=',N,' M=',M,' K=',K)
print("Matrix A:")
print(A)
print("\nMatrix B:")
print(B)
print("\nMatrix C (Result of A * B):")
print(C)
print("\nMatrix multiplication time:", end_time - start_time, "seconds")
```

运行代码，并由本次运行结果确定 M、N、K 的取值，便于后续对比实验。

（二）使用 C 语言同理编写代码，此时 M、N、K 的值固定为上一次 python 程序随机出的值（即 M=1513、N=1309、K=647）。

代码如下：

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// 定义单精度浮点数矩阵结构
typedef struct {
    int rows;
    int cols;
    float** data;
} Matrix;

// 创建一个随机矩阵
Matrix createRandomMatrix(int rows, int cols) {
    Matrix mat;
    mat.rows = rows;
    mat.cols = cols;
    mat.data = (float**)malloc(rows * sizeof(float*));

    for (int i = 0; i < rows; i++) {
        mat.data[i] = (float*)malloc(cols * sizeof(float));
        for (int j = 0; j < cols; j++) {
            mat.data[i][j] = (float)rand() / RAND_MAX; // 生成0到1之间的随机浮点数
        }
    }

    return mat;
}

// 矩阵乘法
Matrix matrixMultiply(const Matrix* A, const Matrix* B) {
    if (A->cols != B->rows) {
        fprintf(stderr, "Matrix dimensions are incompatible for multiplication\n");
        exit(1);
    }

    Matrix C;
    C.rows = A->rows;
    C.cols = B->cols;
    C.data = (float**)malloc(C.rows * sizeof(float*));

    for (int i = 0; i < C.rows; i++) {
        C.data[i] = (float*)calloc(C.cols, sizeof(float));
```

```
        for (int j = 0; j < C.cols; j++) {
            for (int k = 0; k < A->cols; k++) {
                C.data[i][j] += A->data[i][k] * B->data[k][j];
            }
        }
    }

    return C;
}

// 释放矩阵内存
void freeMatrix(Matrix* mat) {
    for (int i = 0; i < mat->rows; i++) {
        free(mat->data[i]);
    }
    free(mat->data);
}

// 打印矩阵的内容，只输出部分内容，用省略号代替其余部分
void printMatrix(const Matrix* mat, int maxRows, int maxCols) {
    int numRows = mat->rows;
    int numCols = mat->cols;

    for (int i = 0; i < numRows && i < maxRows; i++) {
        for (int j = 0; j < numCols && j < maxCols; j++) {
            printf("%f ", mat->data[i][j]);
        }

        if (numCols > maxCols) {
            printf("..."); // 用省略号代替超出部分的列
        }

        printf("\n");
    }

    if (numRows > maxRows) {
        printf("...\n"); // 用省略号代替超出部分的行
    }
}

int main() {
    srand(time(NULL)); // 初始化随机数生成器

    int M = 1513;
    int N = 1309;
    int K = 647;

    printf("N=%d, M=%d, K=%d\n", N, M, K);
```

```
    // 创建随机矩阵 A 和 B
    Matrix A = createRandomMatrix(M, N);
    Matrix B = createRandomMatrix(N, K);

    // 初始化结果矩阵 C
    Matrix C;
    C.rows = M;
    C.cols = K;
    C.data = (float**)malloc(M * sizeof(float*));
    for (int i = 0; i < M; i++) {
        C.data[i] = (float*)calloc(K, sizeof(float));
    }

    // 执行矩阵乘法并测量时间
    clock_t start_time = clock();
    C = matrixMultiply(&A, &B);
    clock_t end_time = clock();

    // 打印执行时间（以秒为单位）
    double execution_time = (double)(end_time - start_time) / CLOCKS_PE
R_SEC;
    printf("Matrix multiplication time: %f seconds\n", execution_time);

    // 打印矩阵 A、B 和 C 的内容
    printf("\nMatrix A:\n");
    printMatrix(&A,2,2);
    printf("\nMatrix B:\n");
    printMatrix(&B,2,2);
    printf("\nMatrix C (Result of A * B):\n");
    printMatrix(&C,2,2);

    // 释放矩阵内存
    freeMatrix(&A);
    freeMatrix(&B);
    freeMatrix(&C);

    return 0;
}
```

（三）调换循环顺序，从外到内，将 i、j、k 变为 i、k、j，使其便于连续存取。调换部分代码如下：

```
for (int i = 0; i < C.rows; i++) {
        C.data[i] = (float*)calloc(C.cols, sizeof(float));
        for (int k = 0; k < A->cols; k++) {
            for (int j = 0; j < C.cols; j++) {
                C.data[i][j] += A->data[i][k] * B->data[k][j];
            }
        }
    }
```

（四）编译优化

输入下述指令分别进行编译：

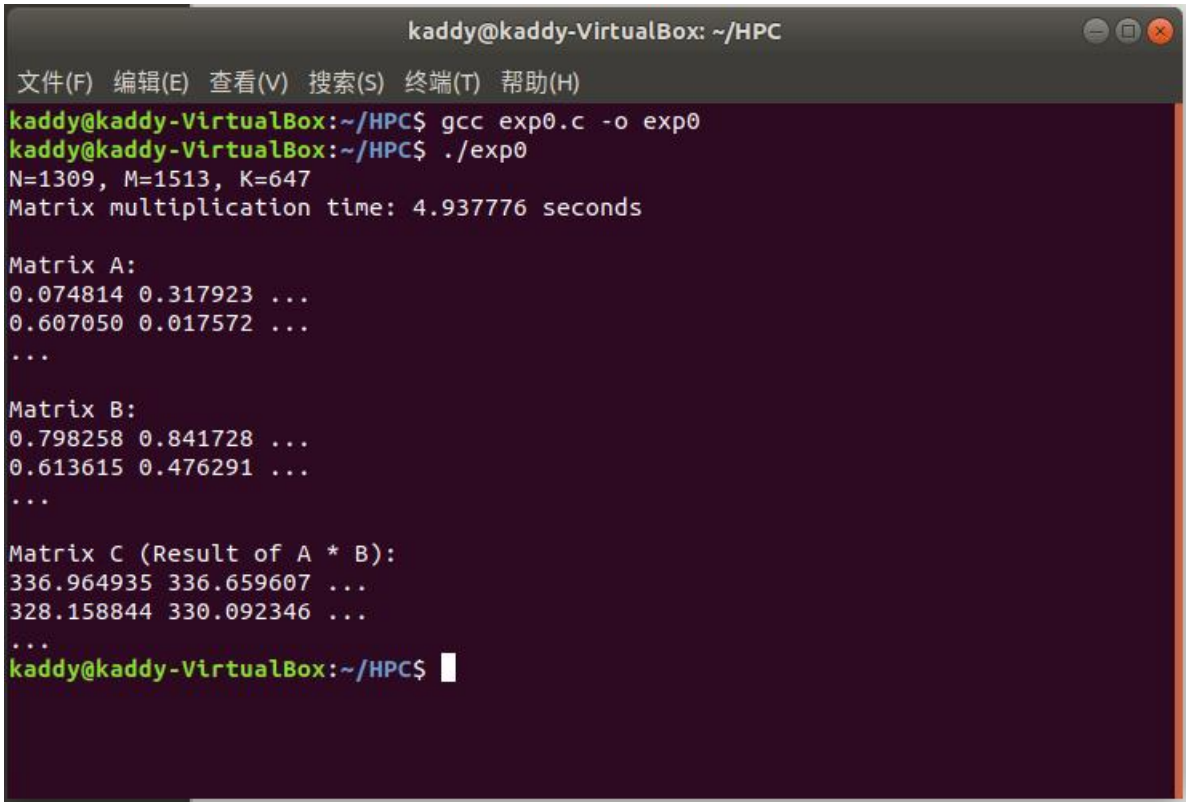
```
gcc exp0.c -O2 -o exp0
gcc exp0.c -O3 -o exp0
```

3. 实验结果

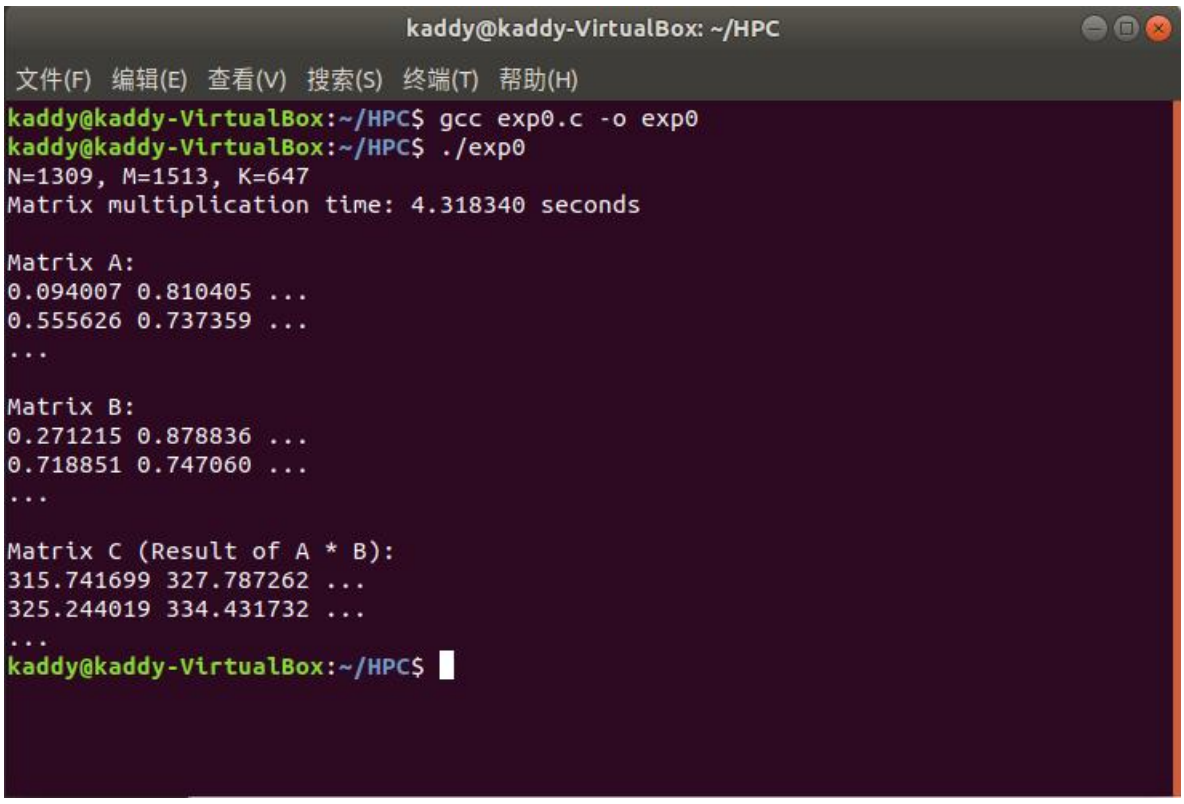
Python 实现的通用矩阵乘法结果如下：

```
PS D:\python文件\HPC> & D:/PY/python.exe d:/python文件/HPC/test.py
N= 1513  M= 1309  K= 647
Matrix A:
[[0.47904006 0.63165337 0.42766872 ... 0.7812961  0.91277164 0.39673203]
 [0.7996164  0.7705143  0.5782888  ... 0.20705439 0.66393423 0.7243632 ]
 [0.81833434 0.37122896 0.39885643 ... 0.7286834  0.9899494  0.424748  ]
 ...
 [0.02737558 0.4499484  0.95744175 ... 0.2920888  0.6950227  0.71485335]
 [0.56744885 0.8401313  0.6001886  ... 0.5873423  0.36035064 0.3384805 ]
 [0.3560158  0.51604116 0.32023346 ... 0.5175942  0.6661902  0.9904802 ]]

Matrix B:
[[0.66163266 0.55099136 0.07128776 ... 0.15889713 0.5515195  0.6466148 ]
 [0.7635212  0.4810009  0.11374903 ... 0.18077748 0.56454885 0.85257405]
 [0.628886   0.3020747  0.6642693  ... 0.5213948  0.04199375 0.4051045 ]
 ...
 [0.37783203 0.8131346  0.45874146 ... 0.86111224 0.14795953 0.73824894]
 [0.3056947  0.81209177 0.08781526 ... 0.5903913  0.9941543  0.29012227]
 [0.7540702  0.9284576  0.38205898 ... 0.1701566  0.34669667 0.5991845 ]]

Matrix C (Result of A * B):
[[390.62708 386.95767 396.29718 ... 398.1305  381.62042 382.95538]
 [376.87857 368.1209  382.37366 ... 380.9498  376.99414 375.6605 ]
 [371.2219  380.74323 382.5674  ... 385.10406 370.40326 376.65448]
 ...
 [383.15366 386.06732 383.12018 ... 389.52914 382.752   373.4138 ]
 [380.85086 381.23407 389.6975  ... 392.62753 376.65256 377.01218]
 [376.30997 373.8885  381.825   ... 379.31076 375.5608  375.89386]]

Matrix multiplication time: 524.3367810249329 seconds
```

C 实现的通用矩阵乘法结果如下（未优化）：



C 实现的通用矩阵乘法结果如下（调换顺序）：



编译优化结果：

```
kaddy@kaddy-VirtualBox:~/HPC$ gcc exp0.c -O2 -o exp0
kaddy@kaddy-VirtualBox:~/HPC$ ./exp0
N=1309, M=1513, K=647
Matrix multiplication time: 0.893970 seconds

Matrix A:
0.244063 0.276131 ...
0.171151 0.034668 ...
...

Matrix B:
0.360279 0.172986 ...
0.874797 0.693779 ...
...

Matrix C (Result of A * B):
330.888794 334.002899 ...
330.390137 337.261841 ...
...
```

```
kaddy@kaddy-VirtualBox:~/HPC$ gcc exp0.c -O3 -o exp0
kaddy@kaddy-VirtualBox:~/HPC$ ./exp0
N=1309, M=1513, K=647
Matrix multiplication time: 0.246265 seconds

Matrix A:
0.440080 0.395052 ...
0.087957 0.230235 ...
...

Matrix B:
0.611827 0.753701 ...
0.506854 0.872469 ...
...

Matrix C (Result of A * B):
328.489838 327.412903 ...
316.083923 316.996033 ...
...
```

版本	实现	运行时间 (s)	相对加速比 (相对前一版本)	绝对加速比 (相对版本 1)	浮点性能 (GFLOPS)	达到峰值性能的百分比 (409.6GFLOPS)
1	Python	524.3367810249329	1	1	0.0048876773302	0.001
2	C	4.937776	106.18885527	106	0.5190168606271	0.127
3	+调整循环顺序	4.318340	1.143443082296	121	0.5934662388788	0.145
4	+编译优化 例如： gcc –O2 gcc –O3	-O2：0.893970 -O3：0.246265	-O2（相对 3）：4.8305200398 -O3（相对 3）：17.5353379490	-O2：586 -O3：2129	2.866750559862 10.40663106004	-O2：0.700 -O3：2.541

4. 实验感想

//可以写写过程中遇到的问题，你是怎么解决的。以及可以写你对此次实验的一些理解……

本次实验相较于我之前接触的实验较为新奇，令我饶有兴趣。实验过程中，我遇到最大的问题是不知道自己电脑处理器的峰值浮点性能是多少，于是我首先上京东淘宝到处找技术客服，可惜他们都不知道。然后我又找到 AMD 官网，发现竟然也没有相关数据。最后还是在各种帖子博客中得知 AMD 系列的浮点计算单元几乎都为 8，最后用 8 代入计算才求得峰值浮点性能。（看着自己代码越跑越快还是蛮有成就感的！）