实验 1

班级：
姓名：
学号：

实验内容：
1 课程实验平台环境的安装，基本实验工具的使用；
2 从高级语言的角度展示和解释位运算、浮点数运算的精度、cache 对程序性能的影响。

实验目标：
1 完成课程实验平台环境的搭建与设置；掌握常用实验工具的基本使用方法；
2 掌握 C 语言中位操作语句的使用；了解浮点数表示精度在浮点数运算中的影响；了解 cache、数据存储与访问模式对程序性能的影响，掌握编写 cache 友好代码的基本原则。

实验任务：
1 学习 MOOC 内容
https://www.icourse163.org/learn/NJU-1449521162
第一周 实验与开发环境的安装和使用
第 2 讲 虚拟机、Linux 及其上实验环境的安装
第 3 讲 基本实验工具的使用
第二周 C 语言编程实践
第 1 讲 数据的位运算操作
第 2 讲 浮点数的精度问题
第 3 讲 Cache 友好代码

2 在自己的电脑上安装实验环境
安装虚拟机软件：VirtualBox（开源软件）或 VMware（商业软件）
安装 Linux 系统：Linux 32 位版本 Debian 或 Ubuntu
（注：Ubuntu 16.04.6 及之前版本支持 32 位）
熟悉软件工具：gcc，gdb，objdump

3 完成作业
3.1 编写 C 语言程序，不使用中间变量，交换变量 a 和 b 的值，已知变量的初始值为 a=2021，b=191，分析程序的反汇编代码，说明算法的基本原理。
3.1.1 程序代码和注释说明

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

int main()
{
  int a=2021,b=191;
  printf ("交换前: a=%d,  b=%d\n",a,b);
```

```
8.
9.     a = a^b;
10.    b = b^a;//b=b^(a^b)=a
11.    a = a^b;//a=(a^b)^a=b
12.    printf("交换后: a=%d, b=%d\n",a,b);
13. }
```

3.1.2 实验结果记录

3.1.3 结果分析与讨论

算法的核心在于对两个整数的异或操作。异或操作有以下特性：

- 异或同一个数两次，结果为原值。
- 异或操作满足交换律和结合律。

在这段代码中，通过三次异或操作，对这两个整数的值进行了重新排列，达到了交换的效果。

3.2 编写 C 语言程序，举一个例子，说明浮点数运算误差问题，并给出解决方案。

注：可以参考 kahan 累加算法的例子，MOOC 内容（第二周 第 2 讲 浮点数的精度问题）；也可以采用其他算例，分析运行效果，说明算法的基本原理。

3.1.1 程序代码和注释说明

```
1.  #include<stdio.h>
2.  void main()
3.  {
4.   float sum1 = 0;
5.   float sum2 =0;
6.   float c = 0;
7.   float y, t;
8.   int i;
9.   for (i=0;i<4000000;i++)
10.   sum1+=0.1;
11.
12.  for( i=0;i<4000000;i++)
13.  {
14.   y=0.1-c;
15.   t=sum2+y;
16.   c = (t-sum2) -y;
17.   sum2 = t;
18.  }
```

```
19.  printf ("sum1=%f\n" , sum1) ;
20.  printf ("sum2=%f\n" , sum2) ;
21. }
```

3.1.2 实验结果记录

```
sanfenbai@ubuntu:~/Desktop/计算机系统/实验1$ gcc 2_2.c -o 2_2
sanfenbai@ubuntu:~/Desktop/计算机系统/实验1$ ./2_2
sum1=384524.781250
sum2=400000.000000
sanfenbai@ubuntu:~/Desktop/计算机系统/实验1$
```

3.1.3 结果分析与讨论

第一个值为直接累加得到的，然而，由于浮点数精度的限制，最终的结果可能不会精确等于正确结果。第二个值为 kahan 算法累加得到的，这里的 c 是累计产生的误差，y 经过误差修正后的加数，t 是经过本次累加后的和，t-sum 为本次累加实际加上的加数。

总的来说，浮点数在计算机中的表示是有限的，对于某些十进制小数，无法精确表示为有限的二进制小数。这种精度损失在复杂计算中可能会积累，导致输出结果与预期值有微小的差异。

3.3 编写 C 语言程序，实现两个 1024*1024 的浮点数矩阵相乘，采用不同的循环顺序，比较运行效果，并分析导致差异的原因。

注：参考 MOOC 内容（第二周 第 3 讲 Cache 友好代码）。

3.3.1 程序代码和注释说明

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <sys/time.h>
4.  #include <time.h>
5.
6.  void multMat1( int n, float *A,float *B, float *C ) {
7.      int i,j,k;
8.      //ijk
9.      for(i=0;i<n;i++)
10.         for(j=0;j<n;j++)
11.             for(k=0;k<n;k++)
12.                 C[i+j*n] += A[i+k*n] *B[k+j*n] ;
13. }
14.
15. void multMat2( int n,float *A,float *B, float *C ) {
16.     int i,j,k;
17.     //ikj
18.     for(i=0;i< n; i++ )
19.         for(k=0;k<n;k++)
20.             for(j=0;j<n;j++)
21.                 C[i+j*n] += A[i+k*n] *B[k+j*n] ;
22. }
23.
```

```
24. void multMat3( int n,float *A,float *B, float *C ) {
25.     int i,j,k;
26.     //jik
27.     for(j=0;j<n;j++)
28.         for(i=0;i< n; i++ )
29.             for(k=0;k<n;k++)
30.                 C[i+j*n] += A[i+k*n] *B[k+j*n] ;
31. }
32.
33. void multMat4( int n,float *A,float *B,float *C ) {
34.     int i,j,k;
35.     //jki
36.     for(j=0;j< n; j++ )
37.         for(k=0;k<n;k++)
38.             for(i=0;i<n;i++)
39.                 C[i+j*n] += A[i+k*n] *B[k+j*n] ;
40. }
41.
42. void multMat5( int n,float *A,float *B,float *C ) {
43.     int i,j,k;
44.     //kij
45.     for(k=0;k<n;k++)
46.         for(i=0;i< n; i++ )
47.             for(j=0;j<n;j++)
48.                 C[i+j*n] += A[i+k*n] *B[k+j*n] ;
49. }
50.
51. void multMat6( int n,float *A,float *B,float *C ) {
52.     int i,j,k;
53.     //kji
54.     for(k=0;k<n;k++)
55.         for(j=0;j<n;j++)
56.             for(i=0;i< n; i++ )
57.                 C[i+j*n] += A[i+k*n] *B[k+j*n] ;
58. }
59.
60. int main( int argc, char **argv ) {
61.     int nmax = 1024,i,n;
62.
63.     void (*orderings[]) (int,float *,float *,float *)= { &multMat1 , &multMa
    t2, &multMat3,&multMat4 , &multMat5 , &multMat6} ;
64.     char *names[] = {"ijk","ikj", "jik" ,"jki", "kij", "kji"};
65.
66.     float *A = (float *)malloc( nmax*nmax * sizeof (float)) ;
```

```
67.     float *B = (float *)malloc( nmax*nmax * sizeof (float)) ;
68.     float *C = (float *)malloc( nmax*nmax * sizeof (float)) ;
69.
70.     struct timeval start, end;
71.
72.     //用随机数填充矩阵
73.     for(i=0;i<nmax*nmax;i++) A[i]=drand48() *2-1;
74.     for(i=0;i<nmax*nmax;i++) B[i]=drand48() *2-1;
75.     for(i=0;i<nmax*nmax;i++) C[i]=0;
76.
77.     for(i=0;i<6;i++)
78.     {
79.         //矩阵相乘，测量时间
80.         gettimeofday( &start,NULL ) ;
81.         (*orderings[i]) ( nmax, A, B, C ) ;
82.         gettimeofday( &end, NULL ) ;
83.
84.         //将时间转换为Gflop/s
85.         double seconds =( end.tv_sec - start.tv_sec)+ 1.0e-
    6*(end.tv_usec - start.tv_usec) ;
86.         printf( "%s:\tn = %d,  %.3f s\n", names[i] ,nmax, seconds ) ;
87.     }
88. }
```

3.3.2 实验结果记录

```
sanfenbai@ubuntu:~/Desktop/计算机系统/实验1$ gcc 3_3.c -o 3_3
sanfenbai@ubuntu:~/Desktop/计算机系统/实验1$ ./3_3
ijk:    n = 1024,  6.294 s
ikj:    n = 1024,  6.332 s
jik:    n = 1024,  6.384 s
jki:    n = 1024,  2.902 s
kij:    n = 1024,  6.511 s
kji:    n = 1024,  3.144 s
sanfenbai@ubuntu:~/Desktop/计算机系统/实验1$
```

3.3.3 结果分析与讨论

在 C 语言中，二维矩阵采用的是行优先存储，也就是依次顺序地存储每一行的数据。当我们访问相邻数据时，应当尽可能地遵从空间局部性和时间局部性。

当按照 ijk 顺序循环时，矩阵 B 中第 j 行与矩阵 A 中第 i 列对应的元素相乘并累加，得到矩阵 C 中的第 j 行第 i 列元素的值，可以看到矩阵 A 的访问不是顺序的，而是跨越了一行的数据，因此如果矩阵过大，Cache 放不下整个矩阵的数据，矩阵 A 的访问就会很慢。

与 ijk 类似，按照 jik 顺序循环时，矩阵 B 中第 j 行，与 A 中的第 i 列对应的元素相乘并累加，得到 C 中的第 j 行第 i 列元素的值。矩阵 A 的访问不是顺序的，而是跨越了一行的数据，因而它的访问速度也不快。

按照 jki 循环时，矩阵 B 中第 j 行第 k 列元素分别与矩阵 A 的第 k 行元素相乘，得到 C 中的第 j 行元素的部分值。此时对 A 和 C 的访问是顺序的，对 Cache 的利用最好，因此 jki 的运行速度最快。