科学计算程序语言的浮点数机制研究

王 力
(贵州大学电子科学与信息技术学院 贵阳 550003)

摘 要 浮点数运算存在精度方面、比较方面以及舍入误差等方面的问题,而这些问题直接影响到科学计算的准确性、可靠性和安全性等等。目前有关浮点数的中文资料很少,很多教科书上在谈到浮点数时都是浅尝辄止,本文以C语言浮点数机制为研究基础,对浮点数的格式、精度与应用等方面问题进行了实证研究,获得了一些有用的结果。

关键词 程序设计语言,浮点数机制,C语言

Research on Mechanism of Floating-point Value for Program Language in Science Computation

WANG Li
(School of Electronic Science and Information Technology, Guizhou University, Guiyang 550003)

Abstract There exist some problems in floating-point operation, such as problems of accuracy, comparison, rounding error, and so on. They have direct affects to accuracy, reliability and security of scientific computing etc. At present, there are only a few Chinese documents about floating-point value. It scratches the surface of floating-point value in many textbooks. The paper carries out a demonstration research on format, accuracy and application based on floating-point mechanism of C language and some useful results have been obtained.

Keywords Program design language, Mechanism of floating-point value, C language

浮点数运算是科学计算不得不面对的问题,但早由于计算机内部本身不能精确地处理某些整数或小数,在运算时可能存在较大的误差,运算结果影响到系统的可靠性、安全性等因素。由于C语言功能强大、程序设计灵活以及支持底应用,在数据处理、科学计算等领域中得到了广泛应用,但是C语言在浮点数运算方面也存在数据表示的不精确性、舍入误差等问题。国内权威的C语言教材没有对浮点数专门说明,很多教材虽述及浮点数,但也只是按IBM计算机浮点数格式讲解,没有正确给出浮点数编码格式及解码方案。很多程序员把教科书上的浮点数编码格式照搬到C语言中[3,4],按此格式进行科学计算,结果却是错误的,让程序员感到非常矛盾。即使查阅了国内大量的计算机基础、程序设计[3,4]、计算机原理[1,2]等书籍,也不能得到让人满意的结果。

本文先阐述IBM浮点数和一般教科书关于浮点数在计算机内部的编码及解码方法,然后对C语言浮点数格式、精度、表示范围及相关计算进入了深入的探讨,并提出了验证浮点数的实验方法。

1 浮点数的一般表示法

任何二进制实数可以写成以下表示形式:$\pm A * 2^{\pm B}$,A称为尾码,B称为阶码。IBM计算机浮点数的编码格式[1,2]如图1所示:

阶符	阶码	尾符	尾码
1	m	1	n-m-2

图1 浮点数格式

设浮点数由n个二进制位构成,阶码为m位,尾码为$n-m-2$位。尾码通常是一个规格化的定点小数,即小数点位于尾符与尾码之间,尾符占一比特,0表示正数,1表示负数;阶码是一个定点整数,小数点在阶码最低位,阶符占一比特,0表示正数,1表示负数。无论是阶码或尾码,都可以用原码、反码或补码表示,阶码和尾码的编码方式可以不同。

为了方便,设二进制浮点数用16比特表示,其中阶码和阶符共占7位,阶码用补码表示,尾码和尾符占9位,尾码用原码表示,求十进制数-11.25对应的浮点数表示。首先,先转换成对应的二进制数-1101.01,然后对其规格化后为$-0.110101 * 2^{+100}$,尾符和尾码的原码为1 11010100,阶码对应的7位阶符与阶码的原码为0 000100,补码为0 111100。从而-11.25的16位浮点数表示为:0 111100 1 11010100。如果把二进制浮点数转换成对应的实数,计算过程刚好相反。

浮点数的精度由尾码的位数决定,浮点数的取值范围为:

$$2^{-n} \cdot 2^{-(2^{m-1})} \leqslant |N| \leqslant (1-2^{-n}) \cdot 2^{(2^{m-1})}$$

2 C语言浮点数的标准

C语言所使用的浮点数符合IEEE754标准,该标准在1985年审核通过,目的是让遵守IEEE标准的机器之间运行的程序可以相互直接移植,另外也让程序员可以轻松写出有用的、鲁棒的浮点数应用程序。

2.1 单精度浮点数(float)格式及转换

单精度浮点数(float)格式如图2所示。其中,S是符号位,占一比特,0表示正数,1表示负数;E为指数或阶码,用8位表示,其值为指数值加一个偏离值Bias;F是为尾数部分的小数部分,其真值为1+0.F,其中1是隐含的,不出现在32位浮点数中。

一个单精度浮点数解码的计算公式是:

$$(-1)^S * 2^{(E-Bias)} * 1.F$$

王 力 副教授,博士,主要从事信息管理与信息系统、智能计算、模式识别研究。

公式中 Bias 恒等于 $2^{8-1}-1=127$。

图 2　IEEE754 单精度浮点数标准

可以根据公式转换浮点二进制数为十进制数，如二进制浮点数“111000111 1001011 11000110 1011100”的阶码为 E-127＝11000111-01111111＝72，尾码 F＝001011 11000110 1011100，然后由公式得到计算结果为

$(-1)^1 * 1.001011\ 11000110\ 1011100 * 2^{72} = -11.592 * 4.7224e21 = -7.51802e+21$

要把－11.25 的变成浮点数，先把－11.25 变成二进制－1101.01，规格化后为 $-1.110101 * 2^3$，再计算 E 的值为 3＋127＝130，对应二进制为 10000010，F 的值为 11010100000000000000000，S 的值为 1，所以－11.25 的二进制浮点数如图 3 所示。

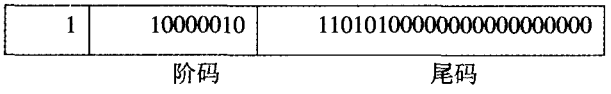

图 3　－11.25 的二进制浮点数

2.2　双精度浮点数(double)格式及转换

double 型数据格式如图 4，在内存中占 64 位共 8 字节，解码计算公式与 float 格式相同，但是 Bias 的值为 $2^{11-1}-1=1023$。

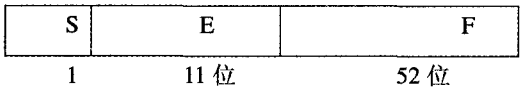

图 4　IEEE754 双精度实数格式

实数与二进制浮点数之间的转换算法与 float 的完全相同。

2.3　扩展双精度浮点数(long double)

C 语言的 long double 型数据对应 IEEE754 的扩展双精度数，但是在 IEEE754 标准中，并没有特别规定扩展双精度数的长度，也没有对扩展双精度数作详细说明，经过实证研究得到了 long double 的编码格式如图 4，与单度数与双精数不同的是，不存在隐含位的 1，而且显式地占据一个比特，其数值由 L 的值确定，阶码 15 位，尾码 63 位，占用 10 个字节 80 比特。解码公式为：

$(-1)^S * 2^{(E-Bias)} * L.F$，Bias 的值为 $2^{15-1}-1=16383$。

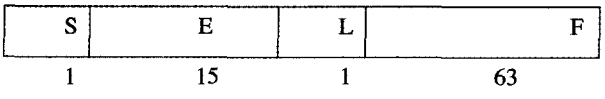

图 5　IEEE 754 长双精度实数

值得注意的是，Visual C++ 不支持扩展双精浮点数(long double)数据类型，当程序中出现 long double 时，与 double 类型含义完全相同。

3　C 语言浮点数科学计算中的问题

3.1　浮点数表示的“数”

浮点数表示数范围，不像整型那样简单。有意义的数由规格化数和非规格化数构成，此外还有一些特殊的数，如无穷数、非数、零值等。

阶码 E 不全为 0 或全为 1 的数称为规格化数(即 E>0&&E<255)。非规格化数界于最小数量规格化数和 0 之间，由阶码 E 为 0 的值和尾码 F 不为 0 的值表示(即 E==0&&V!=0)。非规格化数转换成十进制时，对以 float 数来讲，可以把尾数转换成一个整数 X，再乘以 2^{-149}(对于双精度数而言，要乘以 2^{-1074})，表 1 列出了单精度浮点数的表示范围。

表 1　单精度浮点数的表示范围

格式	阶码	尾码	正数范围(16 进制)	负数范围(16 进制)
0	全为 0	全为 0	0000 0000	8000 0000
非规格化数	全为 0	非零	0000 0001 至 007F FFFF	8000 0001 至 807F FFFF
规格化数	不全为 0 或 1	任意	0080 0000 至 7FF7 FFFF	8080 0000 至 FFF7 FFFF
无穷	全为 1	全为 0	7F80 0000	8F80 0000
NaN	全为 1	至少一位 1	7FC0 0000 至 7FFF FFFF	8FC0 0000 至 FFFF FFFF

单精度浮点数的非规格化数的绝对值取值范围 2^{-149}～$2^{-149} * (2^{23}-1)$，对应十进制数 1.4012985E－45～1.1754942E－38，规格化数的绝对值的取值范围为 2^{-126}～$2^{127}(2-2^{-23})$，对应十进制数 1.1754944E－38～3.4028235E＋38。很多教科书[3,4]认为单精度浮点数的取值范围是－3.4E－38 到 3.4E＋38，实际上只是一种近似的值。

无穷数，在数量上比最大规格化数大的值称为无穷，如表 1 所示，无穷数靠设置阶码全为 1 和尾码全为 0 来表示。如果符号位为 1，表示负无穷，否则表示正无穷。NaN(Not a Number)是 IEEE 对错误值的特定表示，可以是非法运算的结果，或当不正确地返回数值时由库函数返回的结果或者不确定的值。NaN 通过设置阶码的所有位为 1，及尾码至少一位为 1 来表示。零值(0)，把所有阶码和尾码部分都设置为 0 来表示，如果符号为 1，表示－0，否则表示＋0。＋0 与－0 在比较运算时是相等的。

3.2　精度及运算问题

3.2.1　浮点数不能精确表示十进制数

单精度数的有效精度可以表示到 7 或 8 位小数位，双精度数可以表示 16 位。如果试着读写超过有效小数位的值，最后的数位可能不包含有用的信息。

大部分浮点数运算不可能得到精确的数值，因为大部分实数不能被精确地采用二进制浮点数表示，而且浮点运算的结果常常被舍入到最邻近的数。即使是一些简单的分数也不能被精确表示，像 1/3 这个数，要精确表示需要无限位数，因为这个数是一个无限重复小数，对应十进制数 0.3333…，对应二进制数 0.010101…；很多能被有限几位十进数精确表示的数也不能被二进制精确表示，如 1/10，小数表示是 0.1，但二进制表示为 0.000110011001100…，因为像 1/3 和 1/10 这样的数不能被精确表示，所以没有浮点运算能在任何时候产生这些精确的值。示例见程序一。

程序一：

```
void test1( )
{
```

```
float x,y;
x=(float)0.1;
y=(float)(1.0/3.0);
printf("x=%.10f,y=%.10f\n",x,y);
}
```

程序一运行结果为：

x=0.1000000015,y=0.3333333433

从运行结果可以看出，0.1 的输出结果并不刚好是 0.1，1/3 的结果也不刚好是 0.3333…。虽然很多小数不能在浮点运算中精确地表示，但是小于 $1.6*10^7$ 的整数能被 float 数精确表示，任何 32 位的整数也能被 double 数精确表示。

3.2.2 舍入及误差

众所周知，由于不能表示所有十进制数值，所有浮点运算系统都是不精确的。当一个计算机系统不能精确表示一个数时，必须选择邻近的表示值，称为舍入。因为大多数浮点运算大部分时间产生舍入结果，不精确结果异常经常不被认为是一个错误。IEEE 标准对于所有运算而言，要求结果必须舍入到最邻近的值，这样 1 除以 3 的结果不等于 1/3。

除了舍入以外，当系统试图在十进制表示和二进制表示之间进行转换时，不精确也会发生。如在调用 C 语言的 scanf 和 printf 函数时，不精确结果就会发生。如果十进制表示的格式没有足够的空间去表示浮点数，从二进制到十进制转换，不精确会发生。如，在 printf 调用中，格式声明为 7.5f，但是待显示的值要求超过 5 个小数位置时，就会产生不精确结果。

从十进制转换成二进制时，不精确也会发生，因为 IEEE 浮点数格式不能表示所有小数值。例如，小数 0.1，因为 0.1 不能表示成有限个 2 的幂的累加和。

IEEE 提供了四种舍入模式，但是除了特定的数值分析的应用，舍入到近邻是最常用的舍入模式，而且也是程序设计时推荐的默认模式。程序二展示了浮点数计算时的误差。

程序二：

```
void test2( )
{
float i,a;
a=0.0;
for(i=1;i<=100000;i++)
    a+=(float)0.01;
for(i=1;i<=100000;i++)
    a-=(float)0.01;
printf("%.10f\n",a);
}
```

程序运行结果如下：

−0.0000382066

程序第一个循环是把 0.01 加了 10 万次，第二个循环又把 0.01 减了 10 万次，按理运算结果应该是 0，但是程序执行结果与我们预想的却不一样。原因是 0.01 不能被浮点数精确表示，其中存在着舍入误差，当进行了 10 万次累加后，其值并不刚好是 1000.0；当再次减去 10 万次 0.01 后，由于舍入误差的原因，前面的累加和不可能刚好被减为 0。

3.2.3 不正确的浮点运算的情况

当系统尝试完成一个无数值意义浮点数的解释时，会产生不正确的运算，主要有几种情况：(1)在有符号 NaN 上的运算；(2)无穷数(INF)的数量减；(3)与无穷的乘法；(4)0 与 0 的除法，无穷数与无穷数除法；(5)当 x 是无穷或 y 是 0 时，取 x，y 的余数；(6)取负值的平方根(负零除外)；(7)浮点数转换成整数时，当浮点数是无穷数或 NaN，或者转换结果在整数格式的范围之外。

当一个浮点应用程序执行时，它产生的结果可能不同于前面执行时的结果，或者结果可能不正确，大量因素归于差异或不准确。影响应用程序结果的因素可以是基本运算、库函数或其它系统相关因素，也可能浮点编码等原因。数学库同一标准函数也不总是产生同一结果，当从一个系统到另一个系统，或从一种语言到另一种语言时，或从一个软件版本到另一个版本时，结果可能不一样。

为相等或不等而测试两个浮点数或表达式，可能结果并不如程序员所愿。这个规则来自于这样的事实，两个给定的浮点数几乎不等，从纯数学的立场来看，即使程序员可能期望是相等的。在被比较的两个值在计算期间，因为舍入误差导致不等出现了。程序三说明了这个情况。

程序三：

```
void test3( )
{
double a,b,c,d,e,f;

a=1.01;b=2.02;c=3.03;d=4.04;
e=(a+b)*(c+d);
f=a*c+a*d+b*c+b*d;
if(e==f)
    printf("e=%.17lf is equal to f=%.17lf\n",e,f);
else
    printf("e=%.17lf is not equal to f=%.17lf\n",e,f);
}
```

运行结果：

e = 21.42210000000000400 is not equal to f = 21.42210000000000000

对于程序三，e、f 分别是同一个公式的两种计算方法，预想的结果是 e 与 f 相等。但是由于运算方法不同，e 和 f 得到的结果却不等。可见，浮点数在运算中，存在着很多问题，除了舍入误差外，直接用==号进行比较的结果可能是危险的。

4 浮点数研究实验

4.1 实验设计

对浮点数进行验证最有效的方法就是设计程序，通过程序的输入和输出来判定浮点数的精度、取值范围和应用等方面的诸多问题。要看到浮点数的二进制，有两种方法，一种是根据浮点数编码来确定，另外一种是利用 C 语言的联合体来实现，因为联合体中的数据成员共享内存，这样可以把浮点数看成是其它类型的数，如整型、长整型、字符型等，再利用 printf 的%x 格式符即可按十六进制形式观察浮点数的二进制表示，本实验采用自定义函数的方法实现。

```
typedef union{
        REAL F;
unsigned int ST[N/2];
}REALDATA;
```

数据结构 REALDATA 定义为联合体，其中的 F 和 ST 共享内存空间，目的是把 REAL 数解释为 $N/2$ 个 16 位无符号整型数。N 为常量，值可以是 4、8、10，分别表示 float、double 和 long double 三种浮点数数据类型的长度。

4.2 相关算法

自定义函数 FloatToBin 把浮点数变成二进制字符串，为了验证浮点数二时制编码与浮点数的十进制值的关系，编写了自定义函数 BinToFloat 把二进制浮点数转换成十进制值；函数 power 用于求 x 的 y 次幂，y 是整数；函数 DeNormal 用于检验数的类别，返回值为 0 时表示规格化数，返回值为 1 时表示非规格化数；0，NaN，无穷等数被当作规格化数进行处理。

算法 1 浮点数变成二进制形式

```
void FloatToBin(REALDATA F, unsigned char B[])
```

(下转第 291 页)

超过路由规则中阈值小于 Volume 的数据包，提前提前预警与处理。同时通过订阅检查，清除超时没有重置的订阅规则，防止恶意增加路由路径导致网络带宽消耗的拒绝服务攻击。

协议对于伪造，篡改和重发消息内容类型的攻击防范尚不严密，数字签名或时间戳的方法会因服务器资源耗费大造成拒绝服务攻击。

5.2 测试验证

为验证协议的可行性进行模拟测试，模拟测试的环境包含 5 个安全数据采集模块作为消息发布者，2 个计算模块作为数据采集模块信息的订阅者和显示模块的消息发布者，分别部署在 7 台 P4 2.2G，内存 256MB 的 linux 平台服务器上，Bloom filter 算法中的参数选取为 $m=140, n=8, k=12$，采用选定散列函数组，采用预先映射属性值与对应的 Bloom 编码，实际 Bloom 编码作|运算。在每秒 100 条安全事件的流量下的情况下，CPU 占用率为 13%～18%，当流量增大时，CPU 占用率增长不明显，只有内存占用增加，这是可以接受的。如果全部原文订阅开关设置为关闭状态，则当接收流量超过 51000 条/秒时，带宽占用才达到 1M。由于 Bloom filter 本身的出错概率限制，实际发送记录与接受记录相比大概在 1:004:1，在订阅开关关闭的情况下，这影响到了部分计算的准确率，在实际应用时需要在准确率和带宽占用之间做权衡。

总体来说，模拟试验从性能上验证了本文提出协议的可行性。

结束语 本文提出了一种适合大规模分布式的安全管理平台体系，其消息传输路由协议在其他应用场合也有着良好的应用前景，在一定程度上是一个泛应用平台。同时，作为一个安全管理平台，其所具备的安全、隐私保护特性、良好的可扩展性和动态适应性，是对于现有平台体系的一个革新。

参 考 文 献

1 Khurana H, Koleva R. Scalable security and accounting services for content-based publish/subscribe systems. In: proceedings of the 2005 ACM symposium on Applied computing, 2005. 801～807
2 ShamirA . How to share a secret[J]. Communications of the ACM, 1979, 22(11): 612～613
3 李慧贤，程春田，庞辽军. 一般访问结构上的多秘密共享方案. 华南理工大学学报(自然科学版)，2006，34(6)：95～98
4 穆成坡，黄厚宽，田盛丰. 入侵检测系统报警信息聚合与关联技术研究综述. 计算机研究与发展，2006(1)：1～8
5 张慧敏，钱亦萍，郑庆华，董世杰，管晓宏. 集成化网络安全监控平台的研究与实现. 通信学报，2003(7)：155～163
6 Locasto M E, Parekh J J, Keromytis A D, Stolfo S J. Towards Collaborative Security and P2P Intrusion Detection. In: proceedings of the 2005 IEEE Workshop on Information Assurance and Security, 2005. 30～35
7 张志伟，郭长国，曹贺锋，王伟球，王睿. 基于发布/订阅通信的动态数据集成模型. 计算机科学，2005，25(31)：124～126
8 马建刚，黄涛，汪锦岭，徐肛，叶丹. 面向大规模分布式计算发布订阅系统核心技术. 软件学报，2006，17(1)：134～147
9 Fiege L, Zeidler A, Buchmann A, Kehr R K, Mhl G. Security Aspects in Publish/Subscribe Systems. In: proceedings of the Third International Workshop on Distributed Event-Based Systems (DEBS), 2004
10 Koloniari K, Pitoura E. Bloom filters for hierarchical data. In: Proc. of the 5th Int'l Workshop on Distributed Data and Structures (WDAS), 2003
11 Little MC, Speirs NA, Shrivastava SK. Using bloom filters to speed-up name lookup in distributed systems. The Computer Journal, 2002, 45(6): 645～652
12 Chin-Chen C, Tian-Fu L, Jyh-Jong L. Partition search filter and its performance analysis. Journal of Systems and Software, 1999, 47(1): 35～43
13 Bloom B. Space/time trade offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 1970. 422～426
14 肖明忠，代亚非. Bloom Filter 及其应用综述. 计算机科学，2004，31(4)：180～183
15 DMTF. CIM Specification. http://dmtf.org/spec/cims.html

(上接第 287 页)

```
{
   for (j=0;j<N/2-1;j++)
      for(k=0;k<16;k++)
      { 取浮点数的第 j 字节的第 15-k 位 c
        把 c 存到字符串的 B[j*16+k]中
      }
}
```

算法 2 二进制变成十进制浮点数

```
REAL BinToFloat(unsigned char B[])
{
   求符号 S
   if(B 是规格化数) /*规格化*/
   {
     求 Bias
     求阶码 E
     E=E-Bias
     求 EV=2^E
     计算尾数 Fv
   if((是扩展双精数且前导符为 1)||(不是扩展双精度数)) Fv=1
   +Fv;
   计算浮点数 result=s*Ev*Fv;
   }
   else /*计算非规格化数*/
   { 求出尾码表示的整数 F
     计算浮点数 result=S*F/2^k/*k 分别为 -149、-1074、
     -16445*/
     }
     return result;
}
```

4.3 结果分析

程序分别在 Turbo C 2.0(TC)和 Visual C++ 6.0(VC)环境中运行，在 Turbo C 中的运行结果与在 Visual C++中运行基本一致。由于 VC 的 int 型为 4 字节，因此，程序要稍作调整后才能正常运行。实验结果验证了 C 语言的 float 和 double 型符合 IEEE754 的格式，每种格式的精度和数据的取值范围、0、NaN、无穷数等都与表 1 总结的结果相同。

在 VC 中虽然可以使用 long double 型数据，但是经测试不是扩展双精度数，实质还是 double 数据类型。由于在文[5,6]等很多参考文献中没有对于 long double 存储格式的描述，所以根据文[5]的描述，经过反复试验，得到 3.3 节的 long double 格式。在解码时，除要计算出尾码对应的定点小数外，还要显式地加上前导位，即第 64 位的值，其它计算方法与 float 或 double 数据类型相同。

结论 本文深入剖析了 IEEE754 浮点数格式，以及浮点数在表示十进数时的精度、取值范围，以及对浮点数在运算时的舍入误差进行了分析。最后通过实验验证了 C 语言浮点数的格式、取值范围，对 long double 扩展双精度的长度、存储格式以及二进制转换成十进制的方法等进行了研究，对高级语言在浮点数科学计算中可能碰到的问题进行了深度探讨，对于编写高安全性和高可靠性的程序有一定的参考意义。

参 考 文 献

1 王书增. 计算机原理[M](第 2 版). 北京：电子工业出版社，2006，9
2 李敬章. 计算机原理与体系结构[M]. 广州：中山大学出版社，1998，2
3 谭浩强. C 语言程序设计. 北京：清华大学出版社，2000
4 郑莉，等. C++语言程序设计(第 3 版). 北京：清华大学出版社，2003，12
5 IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Standard 754-1985. Institute of Electrical and Electronics Engineers, August 1985
6 Goldberg D. What Every Computer Scientist Should Know About Floating-Point Arithmetic [M]. Association for Computing Machinery. Issue of Computing Surveys, March 1991