无需乘法计算 exp() 和 log()

Quinapalus Home :: <u>Things Technical</u> :: 无需乘法即可计算 exp() 和 log()

本页介绍了一些用于计算基本数学函数 log(x) (以 e 为底的对数) 和 exp(x) (e 的x次方) 的算法。该算法避免了乘法和除法运算,因此适用于在缺少此类指令的处理器上(或在指令速度较慢的情况下)的软件中或在可编程逻辑设备或专用芯片上的硬件中实现。

这些方法特别适合在移位便宜时使用:例如在 ARM 汇编代码中,移位通常可以作为另一条指令的一部分免费进行。我们可以使用 ARM 汇编代码在每位结果的几个周期内计算这些函数。为清楚起见,我们将以简单的 C 语言给出代码示例。

这里解释的想法可以扩展到实现其他基本函数,例如 sin(x) 或 arctan(x); 生成的算法类似于 CORDIC (坐标旋转数字计算机 - 是的,真的)方法,其描述可以在许多地方找到。

原则

有一些常数很容易相乘。例如,乘以 2 n (其中 n 是正整数或负整数) 可以通过简单地将数字移位 n 位来实现。 n 2 见移动将向左移动,如果 n 3 为负,则移动将向右移动。

乘以 $\pm 2^n \pm 1$ 形式的数字几乎同样容易。这些只涉及加法或减法和移位。例如,在 C 中, a = a + (a << 1)将(忽略溢出等)将a乘以3。类似地,a = -4 a + (a >> 4)将 a乘以1 $\pm 2 = 17/16$ 。

我们将调用易于乘以漂亮数字的数字。

相比之下,添加或减去某个任意数字 (例如 41256845) 通常不会比添加一个特殊数字 (例如 1) 慢。通常,将任意数字加在一起是 CPU 可以执行的最快操作之一。

现在我们将看看如何使用加法和乘法之间的这种区别来有效地计算 exp()和 log() 函数。

计算 exp()

假设我们要计算 $y = \exp(x)$ 。作为示例,我们将使用x = 4。该算法将为x和 y生成一系列值,我们将确保对于每一对

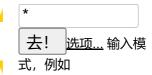
 $y \cdot \exp(x) = \exp(4)$

或者

 $y = \exp(4) \cdot \exp(-x)$

像这样的表达式,尽管所涉及的变量发生变化,但其值通过算法保持不变, 称为*不变量*。我们将把每一对写在一个表格的一行中,表格的开头是这样的:

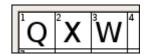
词匹配器



嗬嗬嗬

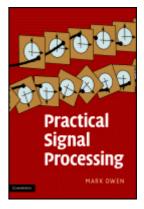
进入框中,然后单击 "开始!"查看匹配单 词的列表。<u>更多的...</u>

新: ARM Cortex-M7 循环计数和双问题组 合; 免费、快速、紧 凑的 ARM Cortex-M0 单精度和双精度浮点 库; 离线 SOWPODS 检查器



Qxw 是一个免费的 (GPL) 填字游戏构建程序。 新的! 适用于 Linux 和 Windows 的版本 20200708。非罗马字母、批处理模式、多路复用灯、答案处理、圆形和六角网格、混乱的条目等。 更多的...

您可以直接从CUP 或 通过 <u>Hive</u>、 <u>Amazon</u> UK或 <u>Amazon US</u> 订 购我的书"实用信号处 理"。



"可能是有史以来最好的关于信号处理的书"——Goodreads的评论。 怀丹妮·波尔斯基。

如果您发现此网站有 用或令人分心,请考 虑向 NASS (英国注

X	是
4	1

册慈善机构)、 KickAS (在美国)或 您所在国家/地区的类 似机构捐款。

请注意, $y \cdot \exp(x) = 1 \cdot \exp(4) = \exp(4)$ 根据需要。如果我们可以在保持不变量的同时 使x变为 0,那么y将由下式给出

版权所有©2004— 2022。 特此承认所有使用的商 标。

 $y = \exp(4) \cdot \exp(0) = \exp(4) \cdot 1 = \exp(4)$,

因此我们将在y中计算出所需的结果。

假设我们从x中减去某个值k。然后,为了保持不变性,新的y值y'必须满足

 $y' = \exp(4) \cdot \exp(-(x - k))$

 $=\exp(4)\cdot\exp(-x)\cdot\exp(k)$

 $= y \cdot \exp(k)$

换句话说,如果我们从x中减去k,我们必须将y乘以exp(k)。我们现在要做的就是确保 exp(k)是一个很好的数字,这样我们就可以轻松地乘以它,剩下的就很简单了。请注意,k本身不必很好,因为我们只是减去它,而不是乘以它。以下是 exp(k)的一些不错的值和k的相应(不一定不错)值。

ķ	exp(<i>k</i>)
5.5452	256
2.7726	16
1.3863	4
0.6931	2
0.4055	3/2
0.2231	5/4
0.1178	9/8
0.0606	17/16
0.0308	33/32
0.0155	65/64
0.0078	129/128

现在让我们试一试。在算法的每个步骤中,我们将从x中减去上表中最大的 k,我们可以在不发送x负数的情况下,然后将y乘以相应的 $\exp(k)$ 。

步骤 0. x = 4,我们可以减去的最大k是 2.7726,我们必须将y乘以16。到目前为止的结果:

X	是
4	1
4-2.7726=1.2274	1.16=16

步骤 1. *x* =1.2274, 我们可以减去的最大*k*是 0.6931, 我们必须将*y*乘以2。 到目前为止的结果:

X	是
4	1
1.2274	16
1.2274-0.6931=0.5343	16·2=32

步骤 2. *x* =0.5343, 我们可以减去的最大*k是 0.4055, 我们必须将y*乘以 3/2。到目前为止的结果:

X	是
4	1
1.2274	16
0.5343	32
0.5343-0.4055=0.1288	32·3/2=48

步骤 3. *x* =0.1288, 我们可以减去的最大*k*是 0.1178, 我们必须将*y*乘以 9/8。到目前为止的结果:

X	是
4	1
1.2274	16
0.5343	32

0.1288	48
0.1288-0.1178=0.0110	48-9/8=54

步骤 4. *x* =0.0110, 我们可以减去的最大*k*是 0.0078, 我们必须将*y*乘以 129/128。到目前为止的结果:

X	是
4	1
1.2274	16
0.5343	32
0.1288	48
0.0110	54
0.0110-0.0078=0.0032	54·129/128=54.42

在我们的k 表中有更多条目,我们可以继续;但结果已经相当准确:exp(4)的正确值为 54.598。

示例 C 代码

这是使用上述算法计算 exp()的示例 C 函数。该代码假定整数至少为 32 位长。(非负)参数和函数的结果都表示为具有 16 个小数位的定点值。请注意,在算法的 11 步之后,所涉及的常数变为这样,代码只是进行乘法运算:这在下面的注释中进行了解释。

否定论点的扩展留作练习。

```
int fxexp(int x) {
整数 t,y;

y=0x00010000;
t=x-0x58b91; 如果(t>=0) x=t,y<<=8;
t=x-0x2c5c8; 如果(t>=0) x=t,y<<=4;
t=x-0x162e4; 如果(t>=0) x=t,y<<=2;
t=x-0x0b172; 如果(t>=0) x=t,y<<=1;
t=x-0x067cd; 如果(t>=0) x=t,y+=y>>1;
t=x-0x03920; 如果(t>=0) x=t,y+=y>>2;
t=x-0x01e27; 如果(t>=0) x=t,y+=y>>3;
t=x-0x00f85; 如果(t>=0) x=t,y+=y>>4;
t=x-0x007e1; 如果(t>=0) x=t,y+=y>>6;
t=x-0x001fe; 如果(t>=0) x=t,y+=y>>6;
t=x-0x001fe; 如果(t>=0) x=t,y+=y>>7;
如果(x&0x100) y+=y>>8;
```

```
如果(x&0x080) y+=y>>9;
如果 (x&0x040) y+=y>>10;
如果 (x&0x020) y+=y>>11;
如果(x&0x010) y+=y>>12;
如果(x&0x008) y+=y>>13;
if(x&0x004) y+=y>>14;
if(x&0x002) y+=y>>15;
如果 (x&0x001) y+=y>>16;
返回 y;
```

请注意,试验减法中涉及的常数在每一步都减少了小于或等于 2 的因子。 这意味着永远不需要两次测试相同的常数。

关于残差的说明

最终答案的误差取决于 x中的残差值;事实上,相对误差是 exp(x)。由于对于较小的x, exp(x)大约为 1+ x,因此可以通过将其乘以 1+ x来纠正最终答案.在上面的示例中,54.42·(1+0.0032)=54.594,这与我们的中间结果四舍五入到小数点后四舍五入的预期一样准确。应用校正的好处是它使答案的准确位数大致翻了一番;缺点是它需要一般乘法。在软件实现中,这是否值得取决于该算法和乘法指令的相对速度。在硬件实现中不太可能值得。

计算日志 ()

现在让我们尝试计算 $y = \log(x)$ 。例如,我们将使用x = 54。(因为我们从上面知道 $\exp(4) = 54.598$,所以我们期望的答案是略小于 4。) 至于 $\exp()$,算法为x和y生成一系列值。这次我们的不变量是

 $\exp(y) \cdot x = 54$

或者

 $y = \log(54/x)$

在这种情况下,我们的表格是这样开始的:



请注意,根据需要 $y = \log(54/x) = \log(1) = 0$ 。我们的目标是在保持不变性的同时使x变为 1。然后y将由

 $y = \log(54/1) = \log(54)$

我们想要的答案。

假设我们将x乘以某个数字k。然后为了保持不变量,新的不变量,新的y值 y '必须满足

 $y' = \log(54/kx)$

 $=\log(54/x) + \log(1/k)$

 $= y - \log(k)$

换句话说,如果我们将x乘以k,我们必须将 -log(k)添加到y。我们确保k是一个很好的数字,所以我们可以很容易地乘以它。log(1/k)不必很好,因为它只涉及加法。这里有一些不错的k值和对应的,不一定是不错的 log(k)值;这只是与以前相同的表,但交换了列。

ķ	日志 (k)
16	2.7726
4	1.3863
2	0.6931
3/2	0.4055
5/4	0.2231
9/8	0.1178
17/16	0.0606
33/32	0.0308
65/64	0.0155
129/128	0.0078

此表中的所有k值都大于 1。因此,我们必须从x小于 1 开始,因此我们首先将x乘以1/256(其他不错的数字也可以),并且为了保持不变量,添加log(1/256)=5.5452 到y;这实际上只是输入的缩放和y的初始化。在这个准备步骤之后,我们的表格如下所示:

X	是
54	0
54/256=0.2109	5.5452

步骤 0. x = 0.2109,我们可以乘以的最大k是 4 (保持x < 1) ,我们必须将 -1.3863 添加到y。到目前为止的结果:

X 是

0.2109	5.5452
0.2109*4=0.8436	5.5452-1.3863=4.1589

步骤 1. x = 0.8436, 我们可以乘以的最大*k是 9/8, 我们必须将 - 0.1178 添加到y*。到目前为止的结果:

X	是
0.2109	5.5452
0.8436	4.1589
0.8436*9/8=0.9491	4.1589-0.1178=4.0411

步骤 2. *x* =0.9491, 我们可以乘以的最大*k是 33/32, 我们必须将 -0.0308 添加到y*。到目前为止的结果:

X	是
0.2109	5.5452
0.8436	4.1589
0.9491	4.0411
0.9491*33/32=0.9788	4.0411-0.0308=4.0103

步骤 3. x = 0.9788,我们可以乘以的最大 k 是 65/64,我们必须将 - 0.0155 添加到 y。到目前为止的结果:

X	是
0.2109	5.5452
0.8436	4.1589
0.9491	4.0411
0.9788	4.0103
0.9788*65/64=0.9941	4.0103-0.0155=3.9948

真正的答案是 3.9890。答案中的绝对误差是x中残差的对数,在本例中为 $\log(0.9941)$ 。对于较小的x, $\log(1-x)$ 大约等于 -x,因此这种情况下的绝

对误差约为 1-0.9941=0.0059。从最终的y值中减去它,我们得到 3.9889: 非常好!

请注意,因为我们在答案中获得了绝对误差,所以最终的校正 (大约使结果中的准确位数增加一倍) 不涉及乘法。

示例 C 代码

这是使用上述算法计算 log()的示例 C 函数。该代码假定整数至少为 32 位 15 长。参数(小于 2)和函数的结果都表示为具有 16 个小数位的定点值,尽管中间值保持 31 位精度以避免移位期间的精度损失。在算法的 12 步之后,应用上述校正。

```
int fxlog (无符号整数 x) {
无符号整数 t;
整数y;
y=0xa65af;
如果 (x<0x00008000) x<<=16, y=0xb1721;
如果 (x<0x00800000) x<<= 8, y-=0x58b91;
如果 (x<0x08000000) x<<= 4, y-=0x2c5c8;
if(x<0x20000000) x<<= 2, y=0x162e4;
如果 (x<0x40000000) x<<= 1, y-=0x0b172;
t=x+(x>>1): 如果((t&0x80000000)==0) x=t,v=0x067cd:
t=x+(x>>2); if((t&0x80000000)==0) x=t,y=0x03920;
t=x+(x>>3); if((t&0x80000000)==0) x=t,y=0x01e27;
t=x+(x>>4); if((t&0x80000000)==0) x=t,y=0x00f85;
t=x+(x>>5): if((t&0x80000000)==0) x=t,v==0x007e1:
t=x+(x>>6); if((t&0x80000000)==0) x=t,y=0x003f8;
t=x+(x>>7); 如果((t&0x80000000)==0) x=t,y=0x001fe;
x=0x80000000-x;
y=x>>15;
返回 y;
 }
```

实施问题

此处提供的 C 代码示例用于任何用途,不提供任何担保。它们需要进行修改以适合您的应用程序。如果您想要更高的准确性,您可能需要扩展它们;相反,如果您希望以牺牲准确性为代价获得更快的速度,您可能希望删除一些步骤。您还应该检查该函数是否涵盖了您将遇到的所有可能的输入值;这些示例根本不包括任何此类检查。

您可能会发现,由于舍入,这些算法的实现往往会出现系统误差。您可以通过在 exp() 函数的参数或 log() 函数的结果中添加一个小的正或负常数来获得更好的整体准确性,但可能以不再获得 log() 的精确结果为代价。 1) 和 exp(0)。

这些算法的 ARM 汇编器实现特别优雅。上面 C 代码中的每一行翻译成大约 3 或 4 条指令;这意味着 log()算法大约每两个周期产生一个结果位。如果您对针对特定应用程序的这些算法的经过测试的 ARM 汇编器实现感兴趣,请通过主页上的电子邮件地址与我联系。

此页面最近更新时间为 2022 年 2 月 4 日星期五 16:49:52 GMT