处理器中的浮点除法和平方根运算算法

李 蓉,于伦正
(西安微电子研究所,陕西 西安 710075)

摘 要:硬件设计中发展了许多除法运算算法,各算法在商收敛性速度、基本硬件单元和数学公式等许多方面均不相同。通过对现在较流行的浮点除法和平方根运算算法进行介绍,分析各浮点除法和平方根运算算法的思路和适合的不同场合,比较各自的优缺点。举例说明LSFT32处理器中浮点除法算法的选择。只有当算法的思路及其特点与运算器的结构相匹配时才能充分发挥速度和规模的优势,所选用的算法才是有意义的。

关键词:浮点;除法和平方根运算;算法;比较
中图分类号:TP332.2 **文献标识码**:A **文章编号**:1673-629X(2007)03-0109-03

Floating-Point Division and Square-Root Arithmetics on Processors

LI Rong, YU Lun-zheng
(Xi'an Microelectronics Technology Institute, Xi'an 710075, China)

Abstract: Many algorithms have been developed for implementing division in hardware. These algorithms differ in many aspects, including quotient convergence rate, fundamental hardware primitives, and mathematical formulations. The paper presents a study of floating-point division and square-root arithmetics. It analyzes the main idea and environment of each algorithm, also compares the advantages and disadvantages between several prevalent algorithms. This paper takes the LSFT32 processor for example and shows how to choose a suitable algorithm. While the ideas and characteristics of the arithmetic are matching to the arithmetic logical unit, high speed and small scale will be developed. The choice is just meaningful.

Key words: floating-point; division and square-root arithmetic; arithmetics; comparison

0 引 言

浮点运算用来处理数学中的实数,数据表示范围大,精度高,是最常用的数据类型之一。通过浮点运算器完成浮点运算,运算速度快。它广泛应用于科学计算、自动控制、信息处理甚至休闲娱乐等各个领域。

随着处理器的加减和乘运算性能的提高,除运算和平方根运算的耗时显得越来越大,这种性能差距限制了微处理器的发展[1],使除运算和平方根运算逐渐成为处理器算术领域研究的热点,诞生了大量提高运算速度的算法。

1 研究浮点除和平方根运算算法的重要性

近几年,由于90纳米以下CMOS工艺的出现,使得高性能处理器及浮点处理单元的性能取得了突破性的进展,但数据表明:虽然现在的浮点处理器乘法延时减少了,加法延时一般为2至4个时钟周期,乘法为2到8个时钟周期,但是浮点除法运算所花费的时间仍然较长,对于双精度浮点除法从8周期到超过60周期不等。在数学的四则运算中,除法是最复杂、也是最消耗时间的运算。浮点除法运算的速度将严重影响系统的性能。而在某些科学计算及工程应用中,浮点除法运算占的比例却很大。除运算和平方根运算存在类似的迭代关系,故一般放在一起研究。浮点运算在高级语言中很容易编程使用,但是通过硬件来实现就比较复杂,所以大多数的EDA软件目前还不支持浮点运算,浮点运算一般只作协处理器另行设计,其中主要考虑的是运算精度、运算速度、资源占用以及设计复杂度等。

2 浮点除和平方根运算算法的介绍

若将浮点除和平方根运算算法按照硬件操作的不同,可分为五类[2]:数值位循环(digit recurrence)、函数迭代(functional iteration)、高基数算法(very high

收稿日期:2006-06-01

作者简介:李 蓉(1981-),女,山东人,硕士研究生,研究方向为浮点除法处理单元的研究及浮点除法运算单元的设计,FPGA设计及验证;于伦正,研究员,硕士生导师,研究方向为航天计算机系统设计,弹载计算机设计,箭载计算机设计,能用型加固计算机系统设计。

radix)、查表法(table look - up)和可变延时算法(variable latency)。现在许多除和平方根运算算法并不是单纯的某一类,而是综合了几种类别,采取合理的配置以达到更高的效率。由于平方根运算理论是除运算理论的扩展,对除运算的大部分分析和结论同样适用于平方根运算,因此不再做专门的说明。

2.1 数值位循环(digit recurrence)

数值位循环是最简单且应用最广泛的除法算法。该算法采用每次取被开方数或被除数的一位或多位与除数进行加减运算,得到余数和商,下次再取被除数或被开方数的相同位数附于上次的余数末尾,开始新的迭代,直到被除数或被开方数的所有位取完为止。若商的精度要求不变,当进制由 r 增加到 r^f 时,算法迭代次数减少为原来的 $1/f$。但是进制增加时,商位选择逻辑会变得更复杂。

数值位循环算法主要包含两类:恢复余数型、不恢复余数型。恢复余数算法速度太慢,因需要恢复余数的时间,已经淘汰。不恢复余数型中最典型的是 SRT 算法,已经出现了 2,4,8,16,64,128,256 进制的算法,主要都是依赖于专门的除和平方根部件实现。SRT 算法每次循环产生 $\log_2 r$ 位的结果(r 为算法的基数),同时生成相应的部分余数。根据 IEEE754 标准,单精度浮点数尾数为 23,双精度浮点数尾数为 52 位。若采用基数为 16 的 SRT 算法,实现单精度浮点数的除法至少需要 $24/\log_2 16 = 6$ 个指令周期,而实现双精度浮点数的除法至少需要 $52/\log_2 16 = 13$ 个周期。在高进制的 SRT 算法中,一般都引入余数和商的冗余形式,加快迭代的速度。除运算也是向更高进制和更高收敛速度的算法发展。

2.2 函数迭代(functional iteration)

函数迭代与数值位循环算法的主要不同在于它是对准确结果的逐步逼近,不是按位逐位进行的,而每次迭代的商的位数是相同的,并非随迭代次数的增加而商的位数不断增长。不足是缺少余数并且舍入困难。函数迭代是以乘运算作为基本操作的除法算法,二次收敛于商。主要有:Newton - Raphson,Goldschmidt 算法等。这些算法目前多采用专门的乘法器,以平方级数收敛于商,可以获得较高的运算速度。应用这种算法的处理器主要包括 Intel i860,IBM RS/6000,IBM 360/91 等。

● Newton - Raphson(牛顿迭代法)。

该方法以数学模式求得倒数,再应用高速的乘法器求得最后的商数。即利用乘法迭代的操作先求出除数的倒数,然后与被除数相乘来得到商。每次迭代可以得到双倍的精度,但除数倒数的初始近似值的精度决定了求倒数的迭代周期,近似值越高,所需的迭代次数就越少。例如,倒数要达到 53 位精度,起始精度为 1 位,则需要 6 次迭代:1→2→4→8→16→32→53;8 位起始精度时,减少为 3 次迭代;14 位时,只需 2 次迭代就可满足要求。但是,提高初始近似的精度的同时,导致初始近似值的生成逻辑复杂度直线上升。

● 级数展开算法。

也称 Goldschmidt 算法[3],是通过选择一系列乘数 $R_0, R_1, \cdots$,转化 N/D 到 $Q/1$,来获得商:$N/D = N_0/D_0 * R_0/R_0 = N_1/D_1 * R_1/R_1 = \cdots \rightarrow \sim Q/1$。通过选择每一个乘数因子 $R_K = 2 - D_K$,分母二次收敛到 1,分子近似 N/D(注:其中 N, D, Q 分别是被除数、除数、商)。

Goldschmidt 除运算算法的迭代过程如下:

$$\begin{cases} R_K = 2 - D_K \\ D_{K+1} = D_K \cdot R_K \\ N_{K+1} = N_K \cdot R_K \end{cases}$$

其中,$|1 - D_{K+1}| \leqslant 2^{-2^{K+1}}$,$R_K$ 是乘数因子,$K = 0, 1, \cdots$。

若迭代 M 次后,得到的 N_M 就是精度达到 2^{-2^M} 的商。除运算中累计有 $2M$ 次乘运算,M 次减运算。例如要达到 64 位精度时,则需迭代 6 次,此时要进行 12 次乘运算,6 次减运算。

可以根据精度要求选择适合的乘数因子[3],即对收敛项的次幂作出选择,使运算中乘运算次数最少,即通过选取适当的因子$(1 + X + \cdots + X^{K-1})$,使除数 D 的变化$(D \rightarrow D_1, D_2, \cdots)$ 由 $1 - X^2$ 变成 $1 - X^K (K > 1)$,则商不是二次收敛,而是 K 次收敛,故可以提高除运算的速度。

2.3 高基数算法(very high radix)

数值位循环仅适用于基值较小的除运算,反之,商位选择函数和除数倍乘逻辑会变得很复杂,同时芯片面积和延时都会增加。高基数算法是通过减少商位选择的规模和加快除数的倍数处理来实现面积和延时的缩小,达到可以接收的范围。延时较小并有余数。这里的高基数是指每次迭代的位数(基数)大于 10。高基数算法主要包括:

● 精确商近似(accurate quotient approximations)。

利用查找表和乘运算来得到商值,规模非常大,但迭代周期非常少。

● 短倒数(short reciprocal)。

是一种比较成熟的除法算法,是三种高基数算法中规模最小的,但每个迭代长度由精确商近似的一次乘操作增加到两次乘操作。应用于 Cyrix 83D87 处理

器中,其精度为 17,在 15 个周期内得到双精度结果。

● 舍入并预缩放(rounding and prescaling)。

舍入并预缩放的计算过程是:首先得到除数倒数的初始近似值,然后用来缩放除数和被除数,最后通过舍入来选择商,减小部分余数的多个迭代。该算法是前两种高基数算法在规模和迭代周期上的折中。

2.4　查表法(table look - up)

在函数迭代和高基数算法中,计算开始前,都需要一个较精确的初始近似值[4]。初始近似值可以通过某些算法(如:线性近似法、部分积阵列法等)计算得到。但在现代的浮点运算器的设计中,为了缩短浮点除法的延时,获得初始近似值最常用的方法是通过查询初始近似值表。该表通常用 ROM 或 PLA 的形式实现,里面存放着预先计算的初始近似值。使用时通过将操作数的前几位作为表的一个入口地址,这个地址所指示的位置存放着满足一定精度的初值近似值。

查找表的优点是不需要额外的算术计算就可以快速地获得初始近似值。缺点是查找表的容量与初始近似值的精度有关。初始近似值精确度越高,需要的迭代周期越少;然而精度越高,容量越大。在具体实施中,需要在二者之间找一个平衡点。

2.5　可变延时算法(variable latency)

可变延时算法的迭代周期数目不是固定的,而是与操作数相关。有的操作数的计算时间可以更短或可以从以前的计算中复用某些操作数。例如:计算 $D = A \div B \div C$,假设以牛顿迭代计算,每次乘法需两个周期,整个迭代需 16 个周期,在 32 个周期后才能得到结果 D。如果设置一个 cache,存放最近计算结果的倒数,这里存入 $A \div B$ 的倒数 F,那么再只需两个周期就可以得到结果 D。平均每个除法的延时是 9 个周期,性能提高了近一倍。目前的可变延时算法有:自定时算法(self - timing)、cache 结果算法(result caches)和跳位算法(bit - skipping)、商位预测算法(speculation of quotient digits)。可变延时算法自动匹配乘法序列长度,表现了未来在同时缩小面积和延时的发展倾向。

3　LSFT32 处理器中浮点除法算法的选择

航天 771 研究所根据 SPARC V8 IP 核所设计出来的 LSFT32 国有化抗辐照航天处理器是一款高性能抗辐射加固型 32 位 RISC CPU,具有高性能的定点数处理单元(IU)及并行的浮点处理单元(FPU)[5]。在 LS - FT32 的第一个版本中,没有实现 FPU,但为以后扩展方便在 IU 设计中预留了 FPU 接口。目前综合考虑卫星、飞船、空间站等航天飞行器对浮点处理的需求,课题——"SPARC V8 处理器浮点处理算法设计和验证仿真"是一项并行浮点处理器(FPU)的专门研究课题,需要实现一种快速地能够更好满足 LSFT32 的双精度浮点运算部件,其浮点数标准和 IEEE - 754 标准完全一致,从而进一步提高 LSFT32 处理器的浮点处理能力,通过将大部分浮点操作指令设计成在单周期完成,可以大幅度提高科学计算的速度。

在以上各硬件浮点除运算算法中,数值位循环和函数迭代算法具有实现面积小的优点。高基数算法速度快,但硬件规模太大,不符合航天处理器的小型化和轻量化要求,故不予考虑。可变延时算法,是未来除运算算法发展的趋势。

数值位循环算法是以加减运算为基本操作的除运算算法,实现简单,结构规则。主要缺点是商线性收敛,速度较慢,无法流水化作业,而快速的流水化、并行的浮点运算是提高 LSFT32 性能的关键环节,因此并不适合。函数迭代的主要优点是将除法运算乘法化,乘法运算是其核心,商以平方级数收敛。算法结构比较复杂,需要设计乘法器和查找表(用于迭代的初值查询),但其收敛速度快,精度高,可扩展性好,除法器可以兼作乘法器使用,便于流水化作业。函数迭代中,Goldschmidt 算法与 Newton - Raphson 算法性能差不多,但是它一次迭代中的两个乘操作与 Newton - Raphson 算法的两个乘操作不同,相互之间是不相关的,可以并发产生,因此可以利用流水式乘法器,来提高除运算的性能。因此 Goldschmidt 算法更适合浮点指令的并行化和流水化,故选为 LSFT32 处理器的浮点除运算算法。

4　结束语

实现除和平方根运算的硬件算法很多,它们在商收敛速度、基本硬件单元和数学公式等许多方面都不尽相同;在系统时钟周期,周期延时,规模大小上也各异。要选择出面向应用最佳的算法来,是一件非常艰巨的任务,需根据计算机的体系结构和指令格式,以及用户的侧重点等要求来选择适合的算法,并需要作出必要改进,以求改进的优化算法比传统算法效率更高。

尽管现在已有许多算法被开发,然而兼具简单易实现、速度快、精度高以及占据空间小等优点成为浮点算法的普遍目标。

参考文献:

[1]　Oberman, Flynn M J. Design Issues in Division and Other Floating - Point Operations[J]. IEEE Trans Computers, 1997,46(2):154 - 161.

(下转第 115 页)

3　系统运行实例

为了证明 FKB 系统的可行性及推理效率，下面给出一个完整的摩托车设计类实例，设计类的对象及其相似性关系见表 1～3。

表 1　工作存储器中的摩托车设计类对象

objectID	top－velocity	exhaust	…	Object membership degree
P1	veryhigh	veryBig		1
P2	high	medium		0.8
P3	verylow	verysmall		0.2
P4	medium	big		0.7

表 2　power 的相似性关系

top－velocity	veryhigh	high	medium	low	verylow
veryhigh	1.0	0.9	0.6	0.2	0.2
high	0.9	1.0	0.7	0.2	0.2
medium	0.6	0.7	1.0	0.2	0.2
low	0.2	0.2	0.2	1.0	0.5
verylow	0.2	0.2	0.2	0.5	1.0

表 3　exhaust 的相似性关系

exhaust	veryBig	big	medium	small	verysmall
veryBig	1.0	0.9	0.6	0.2	0.2
big	0.9	1.0	0.7	0.2	0.2
medium	0.6	0.7	1.0	0.2	0.2
small	0.2	0.2	0.2	1.0	0.5
verysmall	0.2	0.2	0.2	0.5	1.0

```
Defrule:(R－1)X .combination([veryGood],Y,Z):－
        Power(X,0.5)
        X.top－velocity ([veryHigh],0.6)
        X.exhaust ([veryBig],0.6)
    (R－2)X .combination([good],Y,Z):－
        Power(X,0.5)
        X.top－velocity ([high],0.6)
        X.exhaust ([big],0.6)
```

根据表 1～3 中的摩托车设计类的对象及其相似性关系等已有事实和规则库中的规则(R－1)、(R－2)按照 FKB 系统的模糊推理机制检索组合结论是 veryGood 的对象，并计算出结论的隶属度。

```
Query:Svearch the moto objects with a veryGood power
    Select X from Power(X,0.5)
    Where X.combination([veryGood],Y,0.5)
```

首先，检查已有事实，根据已有事实从知识库中取出尚未匹配成功的规则与已有事实进行匹配并推定该为知识库中的一条规则；然后，按要求将所需对象调入到 FKB 系统中，power(X,0.5)指定了对象的隶属度大于或等于 0.5，若满足该条件，则对象被调入 FKB 存储器；接着，从知识库中得到结果，在回答集中将对象返回；最后，向用户界面提交该结果。该摩托车设计类实例结论如下：

Conclusion－1：μ_{final} = Max(1,0.6) = 1. p1 与 p2 的组合结论是 veryGood 且隶属度为 1；

Conclusion－2：μ_{final} = Max(1,0.6) = 1. p1 与 p4 的组合结论是 veryGood 且隶属度为 1。

4　结　论

提出一个具有模糊推理机制的新型模糊知识库(FKB)系统。系统的推理机制是基于传统 RETE 算法的扩展，通过相似性来处理模糊问题，同时使用哥德尔的模糊蕴涵函数来建立隶属度，提高了结论隶属度计算效率。另外，在推理中用户可自行设置阈值来“过滤”掉一些可能性极小的结论，提高了推理的效率和准确性。鉴于以上优点，该系统适用于智能信息检索、故障诊断系统和设计类专家系统等。

参考文献：

[1] Zedeb L A. The Role of fuzzy logic in the management of uncertainty in expert systems[J]. Fuzzy Sets and System, 1983, 11:199－227.

[2] Sosnowski Z A. Activation of fuzzy rules in RETE network [C]//in Proc. 4th Int. Conf. Flexible Query Answering Systems (FQAS'2000), Advances in Soft Computing. Warsaw, Poland:[s.n.],2000:200－209.

[3] 向　艳，王洪元. 基于模糊推理模型的专家系统的研究与应用[J]. 计算机工程，2005，31(10)：180－181.

[4] 吴　钊，尹朝庆. 基于模糊技术的推理机设计[J]. 计算机应用与软件，2004，21(7)：14－15.

[5] 向　艳. 基于不精确推理模型的诊断型专家系统研究[J]. 江苏石油化工学院学报，2002，14(3)：46－49.

(上接第 111 页)

[2] Oberman, Flynn M J. Division algorithms and implementations [J]. IEEE Trans Computers, 1997, 46(8):833－854.

[3] Ecgovac M, Imbert L, Matula D, et al. Improving Goldschmidt Division, Square Root and Square Root Reciprocal[J]. IEEE Trans Computer, 2000, 49(7):759－763.

[4] Sarma D, Matula D. Measuring the Accuracy of ROM Reciprocal Tables[J]. IEEE Trans Computers, 1994, 43(8):932－940.

[5] SPARC International Inc. The SPARC Architecture Manual [M]. Version 8. [s.l.]:[s.n.], 1991:11－13.