定制 VLIW 结构实现四精度浮点基本函数

雷元武,窦　勇,倪时策,周　杰
(国防科学技术大学计算机学院,湖南长沙 410073)

摘　要： 本文针对科学应用中基本函数种类多、实现复杂、使用频率低的特点,提出一种定制 VLIW 结构四精度浮点基本函数协处理器(QPC-Processor).该结构通过显示并行技术挖掘基本函数实现算法的并行性,在同一硬件平台上通过元操作的不同组合来计算多种基本函数.同时,本文还提出基本函数元操作序列到定制 VLIW 指令的映射算法,指导基本函数的设计.最后,在 FPGA 平台上进行验证.实验结果表明,相对软件实现,单个 QPC-Processor 能够取得6倍以上的加速比,而且,QFC-Processor 在同一硬件平台上实现多种类型的算法,弥补单一算法的不足,获得较高的硬件资源利用率.

关键词： 四精度浮点算术；超长指令字；基本函数；CORDIC 算法
中图分类号： TN302　**文献标识码：** A　**文章编号：** 0372-2112 (2012)09-1715-08
电子学报 URL： http://www.ejournal.org.cn　**DOI：** 10.3969/j.issn.0372-2112.2012.09.003

A Special-Purpose VLIW Structure to Implement Quadruple Precision Floating-Point Elementary Functions

LEI Yuan-wu, DOU Yong, NI Shi-ce, ZHOU Jie
(*School of Computer, National University of Defense Technology, Changsha, Hunan* 410073, *China*)

Abstract: This paper proposes a quadruple precision floating-point elementary function co-processor based on very large instruction word (VLIW) structure (QPC-Processor), which exploits the parallelism through the explicitly parallel technology of the VLIW structure. Variety of quadruple precision elementary functions is evaluated via the different combination of basic operation in the unified hardware. Finally, we prototype the QPC-Processor units into FPGA chip. The experimental results show our design outperforms the software approach by a factor of more than 6. Moreover, high utilization of hardware resource can be obtained in QFC-Processor.

Key words: quadruple precision floating-point arithmetic; very long instruction word (VLIW); elementary function; CORDIC algorithm

1　引言

大部分科学和工程应用采用 IEEE-754 标准的双精度浮点算术实现.由于浮点运算过程中存在舍入误差,这种舍入误差的累积将会导致计算结果不精确、不可靠、甚至不正确.预测到“艾”量级(ExaScale:百万万亿次/秒)计算时代,双精度 LU 分解的计算结果仅有3位有效[1],这表明双精度浮点算术不能满足未来“艾”量级时代科学工程应用的精度要求.

高精度算术是解决科学计算精度问题的最直接、有效、可靠的方法,已经广泛应用于物理、数学等领域[2],如天气或气候模拟、超新星模拟、计算几何、数值理论等.鉴于高精度算术在提高计算结果精度、数值算法稳定性和结果再现性等优势,2008年,IEEE-754 标准中增加了128位的四精度(quadruple precision)浮点格式来支持高精度算术[3].目前,大部分四精度浮点算术使用软件模拟实现,如 MPFR[4]、Quad-Double(QD)[5]函数库.软件方法的最大缺点是计算性能低,相对于双精度浮点算术,四精度算术的性能下降了一个数量级以上,四精度 Linpack 测试时间为双精度的36倍[2].

一些研究学者设计专用硬件逻辑来支持四精度算术,克服软件方法的性能障碍.IBM S/390 G5[6]处理器在硬件上支持四精度浮点基本操作,A.AKKas 等设计了双模式的四精度加法[7]、乘法[8]和除法[9],支持一个四

收稿日期:2011-03-18;修回日期:2011-05-03
基金项目:国家自然科学基金(No.61125201)

精度浮点操作或两个并行双精度浮点操作. FPGA 平台在加速高精度科学应用中显示出较高的性能和良好的可扩展性[1,10]. 我们[1]在 FPGA 平台上设计了 Double-Double(DD)和 QD 类型的高精度乘累加单元,相对于通用 CPU,取得 30 倍以上的加速比.

但是,科学应用中还包含大量基本函数,对其提供相应的硬件支持能够提高整体计算性能[11]. 科学应用中基本函数具有种类多、实现复杂、使用频率低等特点,如何设计高效的四精度浮点基本函数单元,成为提高四精度浮点算术性能的挑战. 主要包括:(1)资源消耗过多:基于流水线技术的硬件设计方法能够获得较高的性能,但是这会消耗过多的硬件资源;(2)硬件资源利用率低:很难在同一芯片上实现多种四精度浮点基本函数单元,或者由于资源消耗过多而导致不能在同一芯片上集成大量基本操作单元,同时导致系统频率降低,从而降低整个芯片的性能和硬件资源利用率.

本文对 IEEE-754(2008)标准的四精度浮点基本函数的硬件实现进行研究,提出定制超长指令字(VLIW)结构四精度浮点基本函数协处理器(QPC-Processor),通过显示并行技术来挖掘算法的并行性,在同一硬件平台上通过元操作的不同组合来计算多种基本函数. 同时提出基本函数元操作序列到定制 VLIW 指令的映射算法,指导四精度浮点基本函数的设计.

2 研究背景

2.1 IEEE-754(2008)四精度浮点数据格式

IEEE-754-2008 标准的四精度浮点数据格式定义如图 1 所示,由 1 位符号(S)、15 位的偏移指数(E)和 112 位的尾数小数部分(F)组成. 规格化四精度数据尾数的整数位为 1,四精度浮点数据 x 表示数值为 $(-1)^S*(1.F)*2^{E-16384}$. 假定数据 x 的符号、指数和尾数分别表示为 S_x、E_x 和 F_x.

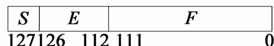


图1 IEEE-754(2008)标准的四精度浮点格式

2.2 基本函数硬件实现方法

基本函数的硬件实现算法可以分为两类:非迭代方法和迭代方法. 非迭代方法有直接查表法、多项式近似法. 直接查表法适合计算精度较低的基本函数计算,如单精度浮点. 多项式近似算法利用 Taylor 展开方式将基本函数转化为基本的乘法和加法运算,再根据精度要求截取展开式的前面部分项. 该方法需要展开次数较多,基本操作量大,执行时间长.

迭代方法又分为数字迭代算法和功能迭代算法,它适合低精度和高精度基本函数计算. 数字迭代算法,如 SRT[12]、Non-Restoring[13]、CORDIC[14]算法等,实现过程简单、硬件开销小,但是最主要缺点是该算法的收敛速度慢,仅为线性收敛,这导致算法的迭代次数与计算精度成正比,计算延时大. 因此,提出高基迭代算法和使用冗余算术的方法来减少计算延时、提高性能,但这些方法增加算法复杂度和硬件开销. 功能迭代算法,如 Newton[15]和 Goldschmidt 算法[16],也是采用乘法和加法操作实现,为二次收敛,对于高维算法,其收敛速度更快,但是以消耗硬件资源为代价的. 每次迭代的基本操作固定不变,但迭代次数与结果精度和初始近似值的精度有关,初始值越精确,所需迭代次数越少,算法计算延时也越小. 因此,该算法通常与查表法相结合,通过查表法得到一个低精度近似值,然后采用功能迭代算法快速计算精确的结果.

2.3 CORDIC 算法

CORDIC 算法能够使用同一硬件实现多种基本函数的计算. 迭代公式如下:

$$\begin{cases} X_{i+1}=K_m\cdot(X_i-m\cdot\sigma_i\cdot 2^{-S(m,i)}Y_i) \\ Y_{i+1}=K_m\cdot(Y_i+\sigma_i\cdot 2^{-S(m,i)}\cdot X_i) \\ Z_{i+1}=Z_i-\sigma_i\cdot\alpha_{m,i} \end{cases} \quad (1)$$

其中坐标系统 m、旋转角度 $a_{m,i}$、旋转序列 $S(m,i)$、旋转方向 σ_i 和扩展因子 K_m 的定义及 CORDIC 算法在不同坐标系统和不同工作模式下迭代结果及收敛范围如文献[17]所示.

CORDIC 算法每次迭代结果精确 1 位. 但是,对于某些基本函数的 0 点位置,该算法很难获得较小相对误差,如 $\ln(x)$,当 x 非常接近 1 时,很难保证 $\ln(x)$计算结果相对误差较小[17]. 这时可以使用多项式近似算法来计算,利用 Taylor 展开方法快速获得精确结果.

本文在同一硬件结构上使用多种方法实现多种四精度浮点基本函数,除法和开方函数使用基于查表法高维 Newton 迭代算法;指数、双曲正弦和双曲余弦函数直接使用 CORDIC 算法实现;自然对数、三角正弦和三角余弦函数使用 CORDIC 算法和 Taylor 展开相结合的方法实现.

3 基本函数算法分析

基本函数计算通常采用变元范围压缩方法将函数从定义域映射到一个较小的预定义范围内来提高计算精度和效率. 对于基本函数 $f(x)$的计算,实现步骤如下:

步骤 1 范围压缩(Range Reduction):将变元 x 映射到一个较小的预定义范围内的变元 y,并且函数 $f(x)$的值可以由 $f(y)$简单推导得到.

步骤 2 近似计算(Evaluation):通过适当的近似计

算算法得到 $f(y)$.

步骤 3　重构(Restruction):由 $f(y)$计算出 $f(x)$,并将其规格化为指定的数据格式.

$f(x)=a_0+a_1y^1+\cdots+a_ny^n$，其中$y=g(x)$.
计算过程:
S1：初始化，通过计算$T=y=g(x)$; $F=0$;
S2：循环计算近似结果
　for(i=1..n_T)
　begin
　　$S=a_i*T$;　▶乘法(1)
　　$F=F+S$;　▶加法(2)
　　$T=T*y$;　▶乘法(3)
　end
S3：规格化.

(*a*) 多项式近似(Taylor展开)

迭代公式如公式（1）所示.
计算过程:
S1：初始化，根据基本函数类型确定X、Y、Z、m的初值，选择方向σ判断方法、旋转序列S(m,i)和基本旋转角度$a_{m,i}$.
S2：CORDIC迭代计算
　for(i=1..n_C)
　begin
　　根据旋转模式和X、Y、Z的值确定σ为1或者-1.
　　$X_i=X$>>S(m,i);　▶移位(1)
　　$Y_i=Y$>>S(m,i);　▶移位(2)
　　$X=X-m*\sigma*Y_i$;　▶加法(3)
　　$Y=Y+\sigma*X_i$;　▶加法(4)
　　$Z=Z-\sigma*a_{m,S(m,i)}$;　▶加法(5)
　end
S3：规格化.

(*b*) 数字迭代算法(CORDIC算法)

$x_{j+1}=x_j(1+\varepsilon/2+3\varepsilon^2/8+5\varepsilon^3/16)$. 其中$x_0\approx x^{-0.5}$, $\varepsilon=1-x_j^2x$. 则$x_n=x^{-0.5}$, $x^{0.5}=x*x_n$.
计算过程:
S1：初始化，通过查表法得到$x_0\approx x^{-0.5}$;
S2：循环计算近似结果
　for(i=0..n_N)
　begin
　　$T_1=x_i*x_i$;　▶乘法(1)
　　$T_2=T_1*x$;　▶乘法(2)
　　$\varepsilon=1-T_2$;　▶加法(3)
　　$S_1=1+(\varepsilon>>1)$;　▶加法、移位(4)
　　$\varepsilon_2=\varepsilon*\varepsilon$;　▶乘法(5)
　　$T_3=(\varepsilon_2>>2)+(\varepsilon_2>>3)$;　▶加法、移位(6)
　　$S_2=S_1+T_3$;　▶加法(7)
　　$\varepsilon_3=\varepsilon_2*\varepsilon$;　▶乘法(8)
　　$T_4=(\varepsilon_3>>2)+(\varepsilon_3>>4)$;　▶加法、移位(9)
　　$S_3=S_2+T_4$;　▶加法(10)
　　$z_{i+1}=z_i*S_3$;　▶乘法(11)
　end
　Result$=x*x_n$;　▶乘法(12)
S3：规格化.

(*c*) 四维Newton迭代开方算法

实现方法	查找表	乘法	加法	移位
直接查表法	Yes			
多项式近似		Yes	Yes	
CORDIC算法	Yes		Yes	Yes
Newton迭代	Yes	Yes	Yes	Yes

(*d*) 基本函数实现算法的元操作组合

图2　基本函数近似计算算法描述

图 2 描述了三种典型的基本函数计算方法,分别为多项式近似算法(Taylor 展开)、数值迭代算法(CORDIC 算法)和功能迭代算法(四维 Newton 迭代算法).这些算法均采用循环实现,循环次数(num_iter)根据结果精度(prec_r)等相关参数确定.对于多项式近似算法,num_iter = prec_r/(c * lz_i),其中 c 表示多项式的收敛速度,lz_i 表示 y 值的首 0 个数;对于 CORDIC 算法,num_iter = prec_r/r,2^r 为算法的基数;对于功能迭代算法,收敛速度与算法的维数(d)有关,为 num_iter = $\log_d$(prec_r/prec_i).

上述算法均可以通过查找表和基本定点操作组合实现,图 2(d)总结了各种算法所需的基本操作.定义元操作集合{addsub, mult, shift, look_table},分别表示定点加/减法、乘法、移位和查询查找表操作.我们可以在同一硬件平台上实现这些算法,该平台包含上述元操作计算单元.

图 2 描述的算法可以并行执行的元操作,如图 2(b)CORDIC 算法中元操作(3)~(5)以及图 2(c)四维 Newton 迭代算法中操作(6)和(9).定制 VLIW 结构通过显示并行技术来挖掘算法的并行性,在 VLIW 的控制下,多个功能部件并行工作.该结构能够使用少量的元操作单元实现多种基本函数算法,结合各类算法的优势来实现某个基本函数,弥补单一算法的不足.该结构的扩展性较好,只需设计相应的 VLIW 指令就可以实现其它基本函数.同时,该结构还可以通过循环展开技术挖掘不同迭代之间并行性,进一步提高定制 VLIW 结构的性能.

4　定制 VLIW 硬件结构

本节描述定制 VLIW 结构四精度浮点基本函数协处理器(QFC-Processor).它能够实现基于查表法的功能迭代算法、多项式近似算法和 CORDIC 算法等.如图 3 所示,QFC-Processor 主要由并行计算阵列、控制状态机模块、VLIW 指令存储器和 VLIW 指令译码模块组成.

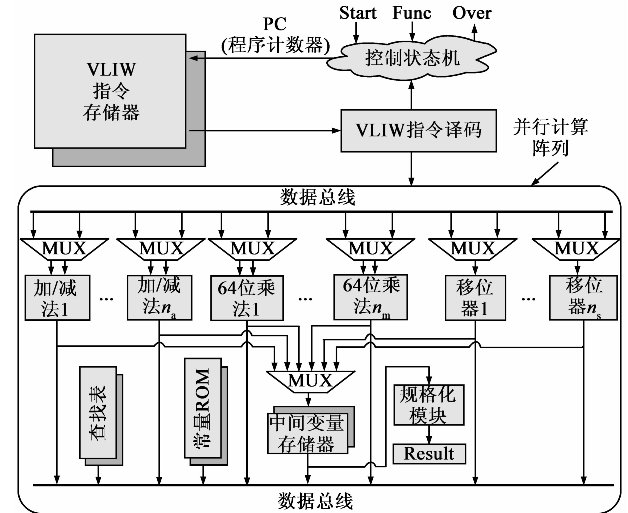


图3　QFC-Processor的总体结构

并行计算阵列模块由元操作计算单元(FU)和存储单元组成,负责基本函数的所有计算工作,通过多个元操作计算单元的显示并行方式来提高 QFC-Processor 的性能.该模块设置 n_m 个 64 位定点乘法模块、n_a 个 128 位定点加/减法模块、n_s 个 128 位的移位模块.以图 2 所示的三个算法为例,确定较优的参数 n_m、n_a、n_s.假定图 2 中三个算法的迭代次数分别为 $n_T=20$、$n_C=116$、$n_N=2$,乘法、加法、移位操作的执行周期分别为 4、1、1.图 4 描述计算阵列在不同配置下,三个算法的执行周期.配置方式是改变待评测操作单元的个数,其余操作单元不受限制,从图 4(b)和(c)可以看出 $n_a=3$、$n_s=2$ 为最优配置,这是因为在 CORDIC 算法每次迭代中,3 个加法和 2 个移位可以分别并行计算.从图 4(a)可以看出 $n_m=4$ 时,基于乘法的迭代算法能够取得最好的计算性能,4 个 64 位的定点乘法组合成一个全流水的 128 位定点乘法单元.

并行计算阵列模块中的存储单元分为查找表、常量存储器 ROM、中间变量存储器.查找表存储基于查表法实现的基本函数的近似值,如除法算法中保存除数的倒数的近似值,在开方算法中保存开方倒数的近似

值.常量存储器 ROM 是一个双端口 ROM,保存所有基本函数计算所需的常数;中间变量存储器是一个简单双端口 RAM(一个读端口和一个写端口),保存输入数据、中间的计算结果.所有存储单元的端口宽度为 128 位.

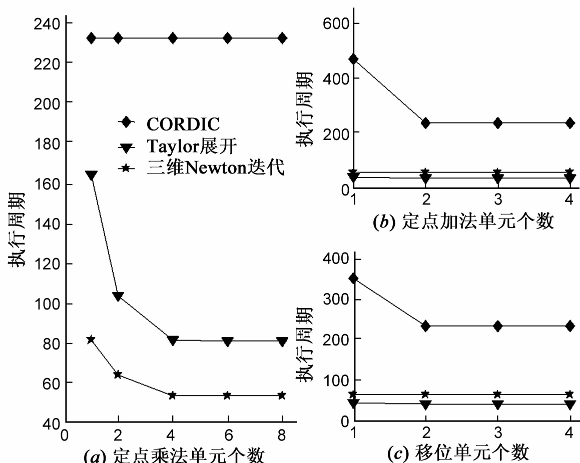


图4 不同配置下三个算法的执行周期

VLIW 指令存储器是一个 2048 * 120 的单端口存储器.指令格式如图 5 所示,每条 VLIW 指令包括每个计算单元数据端口的选择域(sel_ a、sel_ b)、移位器的移位位数域(num)和移位方向域(dir)、存储器的地址域(addr_ a、addr_ b)和控制字(Control),其中 Control 控制着 QFC-Processor 的启动、完成、循环、分支等处理.VLIW 指令译码模块负责将 VLIW 指令转化为上述域.

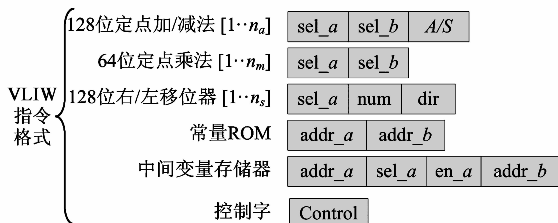


图5 VLIW指令格式

控制状态机模块控制着整个协处理器的运行,通过控制和改变 PC(程序计数器)值,访问不同 VLIW 指令序列,完成对应基本函数的计算.具体执行过程如下:

步骤 1 启动

在接收到 Start 信号后,协处理器开始运行,将初始数据写入到中间变量存储器,根据基本函数选择信号(Func)确定该函数对应的 VLIW 指令序列的起始地址(Init_ addr).

步骤 2 计算

从 Init_ addr 开始读取 VLIW 指令,同时根据 VLIW 指令中的 Control 字改变 PC 值,控制计算模块的运行,实现对应基本函数.

步骤 3 完成

基本函数计算完成后,启动规格化模块,将计算结果转化为四精度浮点格式,将其写入结果寄存器,同时产生 Done 信号.

5 四精度浮点基本函数设计方法

本节提出一种在定制 VLIW 结构上实现四精度浮点基本函数的设计方法,分为 3 个步骤:基本函数算法设计与分析、算法到定制 VLIW 指令的映射及 VLIW 指令序列生成.本节以四精度对数函数为例来说明基本函数的设计过程.同时,介绍常用基本函数的实现算法.

5.1 基本函数算法设计与分析

四精度基本函数设计的第一步是选择实现算法.分析各种算法的收敛性、收敛范围、优势和不足,选择最佳算法或者多种算法的组合,对其进行精度分析,确定初始值、迭代次数等参数,然后使用元操作实现.

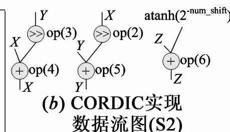
四精度浮点对数函数 $z = \ln(x)$ 可以表示为: $z = \ln(x) = E_x \cdot \ln(2) + \ln(F_x)$,其中 $x > 0$.采用 CORDIC 算法计算 $\ln(F_x)$,在 F_x 接近 1 时,结果相对误差较大,此时更适合采用多项式近似方法,公式如下:

$$\ln(F_x) = \ln(1 + f_x) = \sum_{j=0}^{n} \frac{(-1)^j (f_x)^{j+1}}{j+1} \qquad (2)$$

该多项式展开的项数 num_ iter = 112/prec_ i,其中

```
S1-1: F1 = Fx - 1
S1-2: Count leading zero (prec_i) in the fractional
    part of F1. if(prec_i>4) goto S2-1; else goto S3-1;
S2-1: Set num_iter=0; X=F1; Y=Fx + 1;
      Z=0; num_shift=0; σ=-1;
S2-2: Xt=X>>num_shift;
S2-3: Yt=Y>>num_shift;
S2-4: X = X + σ*Yt;
S2-5: Y = Y + σ*Xt;
S2-6: Z = Z - σ*atanh(2^(-num_shift));
S2-7: num_iter++;
    if((X>0)&(Y>0)|(X<0)&(Y<0)) σ=-1; else σ=1;
    if(num_iter=117) {Z=Z*2; goto S4-1};
    else if((num_iter=4)|(num_iter=14)|(num_iter=42))
        {num_shift=num_shift; goto S2-2;}
    else {num_shift++; goto S2-2;}
S3-1: Set num_iter=2; X= Y= Z=F1; σ=-1; prec=prec_i;
S3-2: Y=Y*X;
S3-3: T = Y *(1/num_iter);
S3-4: Z = Z + σ*T;
S3-5: if(prec>113) {goto S4-1};
        else {num_iter++; prec=prec+prec_i;
            σ=(-1)*σ; goto S3-2}
S4-1: T2 = Ex*ln(2);
S4-2: Normalization z according Z and T2.
```

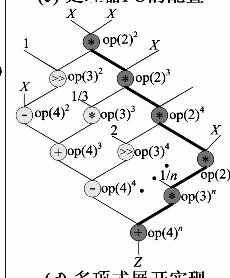
(a) 四精度浮点对数算法



(b) CORDIC实现数据流图(S2)

FU	Num	Cycle
Mult	1	4
Add/Sub	3	1
Shift	2	1

(c) 处理器FU的配置



(d) 多项式展开实现数据流图(S3)

周期	Mult	A/S1	A/S2	A/S3	Shift1	Shift2
1					$op(2)^1$	$op(3)^1$
2		$op(4)^1$	$op(5)^1$	$op(6)^1$		
...						

(e) 基于CORDIC对数的VLIW指令填充

周期	Mult	A/S1	Shift1
1	$op(2)^2$		
5	$op(2)^3$		$op(3)^2$
6		$op(4)^2$	
9	$op(2)^4$		
10	$op(3)^3$		
13	$op(2)^5$		$op(3)^4$
...			

(f) 基于多项式展开VLIW指令填充

图6 四精度浮点对数算法

prec_ i 表示 f_x 的首 0 个数.这表明 F_x 越接近 1,多项式收敛越快,计算时间越短.因此,我们采用 CORDIC 算法和多项式近似相结合的方法来计算 $\ln(F_x)$,算法如图 6(a)所示:

步骤 1　确定 f_x 的首 0 个数,当 $f_x \leqslant 2^{-4}$时,采用步骤 3 的多项式近似方法计算 $\ln(F_x)$,否则采用步骤 2 的 CORDIC 算法来计算 $\ln(F_x)$.

步骤 2　由于 $\ln(F_x) = 2 \cdot \tanh^{-1}(\frac{F_x + 1}{F_x - 1})$,令 $X = F_x - 1, Y = F_x + 1, Z = 0$,在向量模式和双曲坐标下,直接由 CORDIC 计算得到 $\tanh^{-1}(\frac{F_x + 1}{F_x - 1})$).由于 $f_x > 2^{-4}$,CORDIC 计算结果中首 0 个数最多为 4,而 CORDIC 算法线性收敛,每次迭代结果精确一位,因此迭代次数设置为 117 次.

步骤 3　多项式近似如式(2),多项式项数每增加一次,计算精度约提高 prec_ i 位,需要计算的项数为 112/prec_ i.

步骤 4　计算 $E_x \cdot \ln(2)$,再与 $\ln(F_x)$相加,并将结果规格化为四精度浮点格式 z.

```
步骤1    正序和逆序遍历数据流图G，计算每个元操作节点的最早启动时间EET
和最晚启动时间LET；同时每个元操作节点的自由度DOF(i)=LET(i)-EET(i)，其中
i为图G的操作节点.
步骤2    根据EET和LET建立数据流图G中的关键路径，
      为CriticalPath (G) = {i| EET(i)==LET(i)且i∈G}.
步骤3    根据LET和DOF对数据图G的节点进行排序，生产元操作序列：
Order_List = {序列··i··j··满足LET(i)<LET(j)或LET(i)=LET(j)且DOF(i)≤DOF(j)}.
步骤4    填充VLIW指令槽.
  while (not empty of  Order_List)        //循环执行，直到所有节点填充完成
   begin
      curr_node = select(Order_List);   //选择填充节点
      cycle = EET(curr_node);           //从选择节点的最早启动时间开始位置填充
      while (TURE)
      begin
         if(((check_read(curr_node, cycle)==0)&((check_unit(curr_node, cycle)==0)
                 &((check_write(curr_node, cycle)==0))
              {填充节点curr_node到VLIW指令槽的第cycle条指令中；如果
cycle>LET(curr_node)，则需要从该节点开始正序遍历图G，更新对应节点的EET
和LET数值，同时根据第三步的规则更新序列Order_List；  break;}
          else  {cycle++; continue;}
       end
   end
步骤5    生成处理器VLIW指令.
```

图7　基本函数到定制VLIW指令的映射算法

5.2　基本函数到定制 VLIW 指令的映射算法

四精度基本函数设计的第二步是将元操作表示的算法映射到定制 VLIW 指令槽中.这种映射算法是在定制 VLIW 结构上通过显示并行技术来挖掘算法的并行性,通过循环展开技术挖掘不同迭代之间并行性.该映射算法的设计目标是在满足元操作之间的数据相关性前提下,尽可能减少 VLIW 指令槽的深度,使得基本函数计算能在最少的时钟周期内完成.执行周期取决于基本函数的实现算法及元操作的计算延时.

映射算法采用基于周期和基于操作相结合的调度策略,如图 7 所示.其主要思想是首先调度基本函数算法所对应的数据流图 G 中的关键路径,然后按照关键路径优先和贪心原则对所有的节点进行调度.这样可以保证关键路径上的操作被优先调度,同时为每个时钟周期安排尽可能多的不相关的元操作.假设定制 VLIW 结构有 m 种元操作单元,每种元操作单元的个数分别为 num(1),…,num(m),执行周期分别为 cycle(1),…,cycle(m).

步骤 1　顺序和逆序遍历数据流图 G,计算每个元操作节点的最早启动时间 EET、最晚启动时间 LET 和自由度 DOF.EET 表明该节点在这个时刻之后启动就可满足元操作之间的数据相关性,LET 表明该节点在这个时刻之前启动均不会增加算法的执行周期,DOF 表明该节点元操作可以启动的周期个数.EET、LET 和 DEF 定义如下:

$$\mathrm{EET}(i) = \begin{cases} 0, & i\text{ 为根节点} \\ \max\{\mathrm{EET}(j) + \mathrm{cycle}[\mathrm{op}(j)]\}, & j\text{ 为 }i\text{ 的父节点} \end{cases}$$

$$\mathrm{LET}(i) = \begin{cases} \max\{\mathrm{LET}(j)\}, & i,j\text{ 为叶节点} \\ \min\{\mathrm{LET}(j) - \mathrm{cycle}[\mathrm{op}(i)]\}, & j\text{ 为 }i\text{ 的子节点} \end{cases}$$

$$\mathrm{DOF}(i) = \mathrm{LET}(i) - \mathrm{EET}(i), \quad i \in G$$

其中,op(j)表示节点 j 的操作类型.

步骤 2　根据 EET 和 LET 得到数据流图 G 的关键路径.关键路径为 DOF 等于 0 的节点,如果关键路径上的元操作启动时间延迟一拍,那么该节点后续元操作都需延迟一拍,这将增加 VLIW 指令槽的深度,而非关键路径上的节点在 EET 和 LET 之间启动是不会影响 VLIW 指令槽的深度.因此,首先考虑将关键路径上元操作映射到 VLIW 指令槽中.

步骤 3　根据节点的 LET 和 DOF 对数据流图 G 的节点进行排序,确定填充次序.根据节点的 LET 从小到大,再根据节点 DOF 从小到大依次排序.按照上述方案进行调度可以保证关键路径上的节点优先填充.

步骤 4　VLIW 指令槽填充.通过函数 select(Order_List)选取元操作序列中的第一个节点,并将其从序列 Order_List 中删除.调用函数 check_read、check_unit 和 check_write 用来检测将元操作节点 node 填充到 VLIW 指令槽的 cycle 位置是否会发生读数据冲突、运算单元冲突、写端口冲突.这个过程的基本思想是首先填充最关键的节点,从该节点的 EET 时刻开始,逐步尝试,最大程度上减少 VLIW 指令槽的深度.当节点填充完成后,需要根据填充时刻更新剩余节点的 EET、LET、DOF 及 Order_List.一旦数据流图 G 的所有节点填充完毕,算法终止.

步骤 5　将 VLIW 指令槽转换为 VLIW 指令序列,并将其集成到 VLIW 指令存储器中.

图 6 以四精度浮点对数函数为例,说明基本函数到

定制 VLIW 指令的映射算法.对数函数实现算法如图 6(a)所示,计算可以采用 CORDIC 算法实现,对应的数据流图如图 6(b),通过映射算法得到的 VLIW 指令填充槽如图 6(e)所示,两个移位操作和三个加法操作并行执行.当算法采用 Taylor 展开方法实现时,我们采用循环展开技术来挖掘更多的数据并行,数据流图如图 6(d)所示.根据映射算法得到关键路径的元操作为图中灰色节点,得到的 VLIW 指令填充槽如图 6(f)所示,其中 $\mathrm{op}(m)^n$ 表示第 n 次迭代中元操作 op(m).

5.3 常用基本函数实现算法

5.3.1 四精度浮点除法算法

四精度浮点除法($z=x/y$)可以表示为:$z=(-1)^{S_x-S_y}\times 2^{E_x-E_y+16384}\times(F_x/F_y)$.采用基于查表法的高维 Newton 迭代方法来计算定点尾数除法($F_z=F_x/F_y$).

步骤 1 查表得到近似值 $z_0\approx 1/F_y$,作为高维 Newton 迭代的起始值;

步骤 2 高维 Newton 迭代计算 $z_n=1/F_y$,迭代公式为:$z_{i+1}=z_i(1+\varepsilon_i+\cdots+\varepsilon_i^{15})$,这里 $\varepsilon_i=1-z_i*F_y$.高维 Newton 迭代是以 16 次方速度收敛,即每次迭代计算精度增加 16 倍.因此,当近似值 z_0 的精度高于 7 位,一次迭代即可获得 112 位精确的结果;

步骤 3 计算 $F_z=F_x*z_n=F_x/F_y$.并将 F_z 规格化为四精度浮点结果 z.

5.3.2 四精度浮点开方算法

与四精度除法算法相似,开方算法($y=x^{1/2}$)采用基于查表法的 4 维 Newton 迭代方法.计算算法如图 2(c)所示,四维牛顿迭代是以 4 次方速度收敛,即每次迭代计算精度增加 4 倍.因此,通过查找表得到 x_0 的精度为 8 位,两次迭代就可以获得 128 位精度的结果.

5.3.3 四精度浮点指数算法

四精度浮点指数函数(e^x)直接使用 CORDIC 算法实现.CORDIC 算法在双曲坐标和旋转模式下,计算得到双曲正弦 $\sinh(x)$和双曲余弦 $\cosh(x)$,则 $e^x=\sinh(x)+\cosh(x)$.

步骤 1 变元范围压缩.对 x 进行变换,使其映射到区间$[0,\ln 2]$,即将 x 分解为 q 和 d,其中 q 为整数,$d\in[0,\ln 2]$且 $x=q*\ln 2+d$.

步骤 2 CORDIC 计算.令 $X=K_{-1}*(1+2^{-2q})$,$Y=K_{-1}*(1-2^{-2q})$,$Z=d$ 作为 CORDIC 算法初始值,在双曲坐标和旋转模式下得到 CX 和 CY.

步骤 3 结果重构.$\cosh(x)=2^{q-1}*CX$,$\sinh(x)=2^{q-1}*CY$,$e^x=2^{q-1}*(CX+CY)$.

5.3.4 四精度浮点正弦和余弦函数算法

与对数算法相似,使用 CORDIC 算法计算三角正弦 $\sin(x)$和三角余弦 $\cos(x)$时,分别在 x 接近 $k*\pi$ 和$(k+0.5)*\pi$ 时(k 为整数),结果相对误差较大.采用 CORDIC 算法和多项式近似相结合方法解决.

步骤 1 变元范围压缩.将 x 映射到区间$[0,\pi/2]$,即将 x 分解为 q 和 d,其中 q 为整数,$d\in[0,\pi/2]$且 $x=q*(\pi/2)+d$.对于 $\cos(d)$且 d 与 $\pi/2$ 的相对误差小于 2^{-3}时,令 $q=q+1$ 且 $d=\pi/2-d$,或者对于 $\sin(d)$且 d 小于 2^{-3}时,采用 Stage2 的多项式近似方法计算 $\sin(d)$,令 d 的首 0 个数为 $prec_i$,否则使用 Stage3 的 CORDIC 迭代算法计算.

步骤 2 多项式近似公式为 $\sin(d)=\sum_{j=0}^{n}\frac{(-1)^j(d)^{2j+1}}{2j+1}$,展开次数每增加一次,计算精度约提高 $2*prec_i$ 位,因此需要的展开次数为 $56/prec_i$.

步骤 3 在圆周坐标和向量模式下,令 $X=K_1$,$Y=0$,$Z=D$,直接由 CORDIC 计算得到 $\sin(D)=Y_n$,$\cos(D)=X_n$.CORDIC 算法的迭代次数为 117 次.

步骤 4 重构.根据 q 的值选择计算结果($\pm\sin(d)$或 $\pm\cos(d)$).

6 实验结果

我们在 FPGA 平台上验证上述设计.所有的模块采用 Verilog 语言描述,使用 ISE11.3 工具进行综合.选取的 FPGA 芯片为 Xilinx Virtex6 XC6VLX760-2FF1760,它包含 948480 个 Register,474240 个 LUT,864 个 DSP48E,25920Kbits 片内存储器.我们使用 MPFR 函数库(MPFR 3.0.0)来对比性能和结果精度,精度设置为 113 位.软件平台包括四核处理器(2.33GHz Intel Core2 Quad Q8200)、4GB DDR3 1333MHz 内存.

表 1 FPGA 资源使用情况

	类型	REG	LUT	BRAM	DSP	频率(MHz)
基本运算部件	Addsub	128	128	0	0	287.0
	Mult	363	442	0	12	294.4
	Shift	128	908	0	0	351.9
	Normal	1182	2839	0	0	280.4
QFC-Processor		4374 (1%)	14090 (2%)	22 (2%)	48 (5%)	216.2

6.1 FPGA 资源利用

表 1 描述了基本运算部件及四精度浮点基本函数协处理器(QFC-Processor)的综合结果.可以看出,DSP48E 资源使用率最高,达到了 5%,这种资源将会成为四精度科学计算加速器设计的瓶颈,一块 FPGA 芯片最多能够集成 20 个 QFC-Processor.DSP48E 用于构建定点乘法单元,利用 FPGA 提供的较高乘法性能来实现四精度浮点应用.

FPGA 片内存储资源在 QFC-Processor 设计中起着重要作用,分为嵌入式块 RAM(18Kbits)和分布式 RAM.使

用 8 个块 RAM 实现 1024×128 的常量存储器 ROM,使用 14 个块 RAM 实现 2048×120 的 VLIW 指令存储器.其它较小的存储模块,如查找表和中间变量存储器,均采用分布式 RAM 实现,这使 FPGA 的布局布线更加灵活,克服块 RAM 位置和结构固定的不足,同时这也能提高 FPGA 片内存储器的利用率.但是,分布式 RAM 消耗 FPGA 的逻辑资源.

表 2　与 MPFR 函数库的性能对比

类型	实现方法	硬件执行		MPFR 函数库	
		周期	时间	时间	加速比
除法	Newton	41	0.19	3.0	15.8
开方	Newton	54	0.25	1.522	6.1
指数	CORDIC	247	1.14	31.98	28.1
双曲正弦	CORDIC	247	1.14	36.16	31.7
双曲余弦	CORDIC	247	1.14	36.16	31.7
自然对数	CORDIC	254	1.17	69.44	59.4
	Tailoy	$10n+22$	0.7	136.4	194.9
三角正弦	CORDIC	260	1.20	28.78	24.0
	Tailoy	$10n+20$	0.42	29.1	69.5
三角余弦	CORDIC	260	1.20	17.39	14.5
	Tailoy	$10n+20$	0.42	39.95	95.1

6.2　性能比较

表 2 对比了单个 QFC-Processor 与 MPFR 函数库的性能,其中时间单位为 μs.n 表示多项式展开次数,相应的输入为 $x=3^{0.5}-1$,$y=5^{0.5}$;采用 Tailoy 展开计算对数和正弦时输入 $x=1.001$,余弦的输入 $x=1.572$,此时 $prec_i=9$.

对比更高精度 MPFR 实现结果表明,除法和开方操作能够获得正确舍入的结果,对于其他超越函数舍入误差小于 1ulp(unit in last place:最小单元).

对于四精度除法和开方算法,采用查表法与高维 Newton 迭代相结合的算法,收敛速度快.这种方法能够充分利用 FPGA 提供的乘法性能,相对于数字迭代算法,如 SRT 除法算法(58 周期)和 Non-restoring 开方算法(115 周期)[1],QFC-Processor 具有性能优势,相对于 MPFR 函数库,单个 QFC-Processor 就可取得 15.8 和 6.1 倍的加速比.

其他基本函数采用 CORDIC 算法和多项式近似相结合的方法实现,解决 CORDIC 算法在计算结果接近 0 时结果的相对误差较大问题.实验结果表明采用 CORDIC 算法实现基本函数时,单个 QFC-Processor 单元能够取得 14 到 60 倍的加速比;在计算结果接近 0 时,对应基本函数的多项式展开收敛速度较快,仅需要展开较少次数.由第 5 节的实现算法可知对数函数 Tailoy 展开次数为 $112/prec_i=13$,正弦和余弦函数 Tailoy 展开次数为 $56/prec_i=7$,此时加速比可以达到 70 倍.

表 2 描述的是单个 QFC-Processor 的性能,可以在同一芯片上集成更多的 QFC-Processor,并行工作,这样来获得更高的计算性能.而且,QFC-Processor 在同一硬件平台上,使用几个基本操作单元来实现多种基本函数算法,弥补单一算法的不足,获得较高的硬件是使用效率.

7　总结

本文针对科学应用中基本函数种类多、实现复杂、使用频率低的特点,提出一种基于定制 VLIW 结构四精度浮点基本函数协处理器(QFC-Processor),通过显示并行技术来挖掘算法的并行性,在同一硬件平台上通过元操作的不同组合实现多种 IEEE-754(2008)标准的四精度浮点基本函数的计算.同时提出基本函数到定制 VLIW 指令的映射算法来指导基本函数的设计,获得较好的扩展性.最后在 FPGA 平台上进行验证,结果表明,该协处理器能弥补单一实现算法的不足,获得较高的性能和硬件利用率.

参考文献

[1] Y Dou, Y Lei, G Wu, et al. FPGA accelerating double/quad-double high precision floating-point application for exascale computing[A]. Proceedings of 24th International Conference on Supercomputing[C]. Tsukuba: ACM Press, 2010. 325 – 336.

[2] D H Bailey. High-precision floating-point arithmetic in scientific computation[J]. Computing in Science and Engineering, 2005, 7(3): 54 – 61.

[3] ANSI/IEEE Std 754-2008, Standard for Binary Floating Point Arithmetic ANSI/IEEE Standard 754-2008[S].

[4] L Fousse, G Hanrot, V Lefevre, P Pelissier. MPFR: A multiple-precision binary floating-point library with correct rounding[J]. Transactions on Mathematical Software, 2007, 33(2): 1 – 15.

[5] Y Hida, X S Li, D H Bailey. Quad-Double Arithmetic: Algorithms, Implementation, and Application [R]. Berkeley: Lawrence Berkeley National Laboratory, 2000.

[6] E M Schwarz, R M Smth, C A Krygowski. The S/390 G5 floating point unit supporting hex and binary architectures[A]. Proceedings of the 14th IEEE Symposium on Computer Arithmetic[C]. Adelaide: IEEE Press, 1999. 836 – 841.

[7] A Akkas. Dual-mode quadruple precision floating-point adder [A]. Proceedings of 9th Euromicro Conference on Digital System Design[C]. Dubrovnik: IEEE Press, 2006. 211 – 220.

[8] A Akkas, M Schult. Dual-mode floating-point multiplier architectures with parallel operations[J]. Journal of Systems Architecture, 2006, 52(10): 549 – 562.

[9] A Isseven, A Akkas. A dual-mode quadruple precision floating-point divider[A]. Proceedings of Fortieth Asilomar Conference on Signals, Systems and Computers[C]. Pacific Grove: IEEE

Press,2006.1697 – 1701.

[10] Ej-Araby, E Gonzalez, I El-Ghazawi. Bringing high performance reconfigurable computing to exact computations[A]. Proceedings of International Conference on Field Programmable Logic and Applications[C]. Amsterdam: IEEE Press,2007.79 – 85.

[11] J Detrey, F Dinechin, X Pujol. Return of the hardware floating-point elementary function[A]. Proceedings of the 18th IEEE Symposium on Computer Arithmetic[C]. Montepellier: IEEE Press,2007.161 – 168.

[12] K K Parhi, H R Srinivas. A fast radix-4 division algorithm and its architecture[J]. IEEE Transactions on Computers,1995,44(6):826 – 831.

[13] Y Li, W Chu. Parallel-array implementations of a non-restoring square root algorithm[A]. IEEE International Conference on Computer Design[C]. Austin: IEEE Press,1997.690 – 695.

[14] J S Walther. A unified algorithm for elementary functions [A]. Proceedings of AFIPS Joint Computer Conference[C]. New York: IEEE Press,1971.379 – 385.

[15] P Soderquist, M Leeser. Division and square root: Choosing the right implementation[J]. IEEE Micro,1997,17(4):56 – 66.

[16] M D Ercegovac, L Imbert, et al. Improving goldschmidt division, square root and square root reciprocal[J]. IEEE Transactions on Computers,2000,49(7):759 – 763.

[17] J Zhou, Y Dou, Y Lei, et al. Double precision Hybrid mode floating-point FPGA CORDIC co-processor[A]. Proceedings of the 10th IEEE International Conference on High Computing and Communications[C]. Dalin: IEEE Press,2008.182 – 189.

作者简介

雷元武 男,1982 年 1 月出生于湖南桂阳,2007 年于国防科技大学获计算机科学与技术专业硕士学位,现为 08 级博士研究生.主要研究方向:高性能计算机系统结构.

E-mail:yuanwulei@nudt.edu.cn

倪时策 男,1985 年 7 月出生于浙江台州,2009 年于国防科技大学获计算机科学与技术专业硕士学位,现为 2010 级博士研究生.研究方向:高性能计算机系统结构.

窦 勇 男,1966 年 12 月出生于吉林省吉林市,博士,教授,博士生导师,中国计算机学会高级会员(E200009248),1995 年于国防科技大学获计算机科学与技术专业博士学位.主要研究领域为高性能计算机体系结构,可重构计算等.

E-mail:yongdou@nudt.edu.cn

周 杰 男,1980 年 12 月出生于山西运城,2011 年于国防科技大学获计算机科学与技术专业博士学位.研究方向:高性能计算机系统结构.