面向RISC-V的基础数学库实现

李 飞[1,2],郭绍忠[1,2],郝江伟[1,2],侯 明[1,2],宋广辉[1,2],许瑾晨[1,2*]
(1. 信息工程大学,河南郑州 450002;2. 数学工程与先进计算国家重点实验室,河南郑州 450002)

摘 要: RISC-V指令集架构(Instruction Set Architecture,ISA)作为一种新兴的精简ISA,因免费、开源、自由等特点而得到快速发展.由于国内外对RISC-V的研究主要集中在硬件开发,软件生态相较于成熟ISA还很薄弱,实现一套RISC-V指令集高性能基础数学库可以进一步丰富RISC-V软件生态.本文基于自动化移植技术实现申威数学库到RISC-V的移植,为RISC-V指令架构提供首个使用向量指令优化的基础数学库系统.本文提出向量寄存器自动分支查表法与路径标记插入法,重点解决不同架构间寄存器映射过程中的寄存器复用问题,实现寄存器正确高效映射,并依据不同指令等价转换策略自动化移植数学函数69个.测试结果表明,RISC-V基础数学库函数可实现正确计算,最大误差为1.90ULP,函数性能平均为157.03节拍.

关键词: RISC-V;申威;汇编;向量;数学库;自动化移植
中图分类号: TP313 **文献标识码:** A **文章编号:** 0372-2112(2024)05-1633-15
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20220375

Basic Math Library Implementation for RISC-V

LI Fei[1,2], GUO Shao-zhong[1,2], HAO Jiang-wei[1,2], HOU Ming[1,2], SONG Guang-hui[1,2], XU Jin-chen[1,2*]
(1. *PLA Information Engineering University*, *Zhengzhou*, *Henan* 450002, *China*;
2. *State Key Laboratory of Mathematical Engineering and Advanced Computing*, *Zhengzhou*, *Henan* 450002, *China*)

Abstract: RISC-V instruction set architecture (ISA), as a new streamlined ISA, has developed rapidly due to its characteristics of free, open source, and freedom. Since the research on RISC-V at home and abroad mainly focuses on hardware development, the software ecosystem is still weak compared to mature ISAs. Implementing a set of high-performance basic math libraries for the RISC-V instruction set can further enrich the RISC-V software ecosystem. This paper realizes the transplantation of Sunway math library to RISC-V based on automatic transplantation technology, and provides the first basic math library system using vector instruction optimization for RISC-V instruction architecture. This paper proposes an automatic branch look-up table method and a path marker insertion method for vector registers, focusing on solving the problem of register multiplexing in the process of register mapping between different architectures, realizing the correct and efficient mapping of registers, and automatically transplanting 69 mathematical functions according to different instruction equivalence conversion strategies. The test results show that the RISC-V basic math library function can achieve correct calculation, the maximum error is 1.90ULP, and the average performance of functions is 157.03 beats.

Key words: RISC-V; Sunway; assembly; vector; math library; automatic porting

1 引言

近年来,在国家的大力推动下,自主研发的国之重器超级计算机从银河系列、天河系列到神威“太湖之光”得以快速发展[1].第一台亿次巨型计算机“银河-Ⅰ”的研发主要用于科学计算和军事用途,例如航空航天、大气海洋、地质勘探、气象卫星等诸多高性能计算应用领域.高性能基础数学函数库是科学计算所必备的基础核心软件[2],大多数处理器厂商都会根据其处理器架构研发具有较高效益的基础数学函数库[3],如Intel开发的MKL函数库[4]、IBM在POWER平台上开发的MASS库[5].

RISC-V的出现不仅避免了X86和ARM为了向后兼容而愈发臃肿的诟病,而且具有免费、开源、自由等特点,因此,在新一代信息技术领域中,广受产业界和

收稿日期:2022-04-08;修回日期:2022-10-25;责任编辑:王天慧
*通讯作者:许瑾晨

学术界关注[6]. 数学库作为计算机系统最基础的核心软件库之一,是RISC-V软件生态的重要组成部分. 实现一套RISC-V指令集高性能基础数学库,对RISC-V软件生态发展至关重要. 如果用汇编指令对RISC-V数学库进行新开发,则需要很大的时间成本和人工成本. 选择自动移植的方式无疑可以避开上述困难,但将会面临另外一个问题,即选择哪种指令集架构的数学库进行移植. 如果选择与RISC-V指令集架构设计方式相差巨大的指令集架构数学库,将无法充分发挥RISC-V指令集的优势,所以选择和RISC-V架构具有较高契合度的数学库作为移植源数学库成为关键.

国产申威ISA是由我国完全自主研发的处理器架构. 基于该架构的申威基础数学库在精度表现上不仅可以和高精度MPFR库[7]媲美,相较于Glibc数学库[8]性能也有一定的提升. 并且相比其他商用数学库源码的不开放,团队长期面向申威基础数学库进行研发,对其汇编源码具有一定的了解. 除此之外,该指令集架构采用数据并行级SIMD架构[9],有着与RISC-V向量架构类似的寄存器设计和指令集设计[10]. 所以将申威基础数学库作为移植的源数学库,不仅有利于数学库函数的移植,而且也能最大限度地发挥其性能优势. 然而,申威架构中的标量寄存器和向量寄存器是复用的,这与RISC-V的寄存器使用方式具有一定的差别,汇编指令的功能设计也有所不同. 因此,自动化移植方法所面临的困难,主要分为向量寄存器的自动化映射以及汇编指令的自动化转换. 设计正确高效的自动化移植方法,使对每一个函数的不同情况都能够进行准确移植是本文的研究重点.

因此,本文充分结合申威指令集和RISC-V指令集的特点,对现有自动化移植与性能优化工具(Automated Migration and Performance Optimization Tool for sunway math library, AMPOT)[11]进行面向RISC-V Vector 0.10版本的指令集扩展. 利用扩展后的AMPOT将申威数学函数准确、高效地移植为RISC-V数学函数,并保证移植后的函数能够在RISC-V环境下正确、高效地运行.

通过上述工作,本文取得以下两个方面的成果.

(1) 形成了首个使用向量指令优化的RISC-V双精度基础数学函数库,丰富了RISC-V软件生态. 该库由69个函数组成,包含三角函数、双曲函数、指数函数、对数函数、数值函数等,均已通过精度测试与性能测试. 该数学库(RV-Libm)源码我们已经开源在https://gitee.com/mathlib/RV-Libm.

(2) 实现了申威架构面向RISC-V自动化移植框架. 该框架符合高可拓展性、高通用性的设计标准. 对该框架进行扩展可以适用于任意架构之间汇编指令的自动化移植.

2 相关工作

申威基础数学函数库共150多个数学函数. 在这150多个数学函数的汇编代码中使用了大量的向量指令,以此来增加申威数学函数的数据并行性,从而提升数学函数性能[12]. 相较于其他成熟指令集架构,RISC-V诞生时间较短,其向量扩展指令集处于迭代完善阶段. 使用RISC-V标量指令去完成申威数学函数的向量指令功能,不仅烦琐而且移植后的函数性能会大幅下降. 2021年RISC-V Vector 0.10版本的发布,代表RISC-V向量扩展趋于稳定. 因此本文依据研究团队长期面向数学函数库的开发经验,进行基于申威基础数学函数库面向RISC-V向量扩展基础数学函数库的移植.

2.1 寄存器分配

寄存器分配有个重要的基本概念:生命期. 它是指寄存器从第一次到最后一次活跃的范围. 传统的寄存器分配算法有Chaitin等提出的图着色算法[13](Graph Coloring Register Allocation, GCRA)和Massimiliano等提出的线性扫描算法(Linear Scanning Register Allocation, LSRA)[14]. 图着色算法是通过对干涉图进行构造和化简冲突点完成寄存器的分配. 线性扫描算法是寄存器分配较快的算法,通过对待分配寄存器的一次性扫描,将各个变量生命周期的结束时间进行排序,完成寄存器的分配. 2017年王向前等[15]基于分簇结构提出了一种向量寄存器分配策略. 该策略首先为向量寄存器建立一个生命周期列表,其次依据生命周期列表建立冲突图,根据冲突图得到每个簇上允许分配的寄存器集合,进而完成分配. 2021年曹浩等[11]提出了一种基于全局式的寄存器分配方法. 该方法是一种源到源的寄存器转换方式,从整个生命周期出发,分析整个汇编程序的全文寄存器. 该方法虽然能够从全局考虑并更加充分利用寄存器资源,但是对于复杂的汇编程序,很难保证上下文寄存器的依赖关系正确. 寄存器依赖关系的改变会导致程序运行的绝对性错误,尤其是在向量寄存器和标量寄存器复用的情况中.

RISC-V Vector 0.10版本设计了32个128 bit可变长度的向量寄存器. 向量寄存器的位数可以通过寄存器分组的方式进行扩展[16]. 如果将八个向量寄存器分为一组,共有4个寄存器可供使用(v0、v8、v16、v24);如果将4个向量寄存器分为一组,共有8个寄存器可供使用(v0、v4、v8、v12、v16、v20、v24、v28). 在申威指令集架构中,共有32个向量寄存器,即$f0~$f31[17],长度均为256 bit. 如果对RISC-V向量寄存器进行分组来使两架

构的寄存器长度保持一致,则分组后的RISC-V向量寄存器少于申威架构中的向量寄存器个数. 在移植工作中,需要对向量寄存器进行生命周期结束后的再分配[18]. 寄存器的再分配会增加内存读写操作,显著降低RISC-V函数性能. 而本文不仅要保证移植后函数的功能正确,也要最大限度地保证性能高效,因此通过对向量寄存器分组来扩展寄存器长度的方法并不适用.

2.2　数学库的向量方法研究

2019年周蓓等[9]提出了基于申威平台向量数学库的向量化方法. 根据"90-90"规则[19],即90%的资源其实用在了程序的10%代码段上,该方法将优化重心放在了性能消耗的"热点"上,通过大量的数据测试和算法分析找出关键路径,将关键路径上的标量指令代码段以逻辑等价的SIMD指令代码段进行替换,增加一次处理数据的长度. 此外,该方法设置了异常处理分支,当输入为非正常输入时则进入该分支,减少SIMD指令引发浮点算术异常所需的时间. 由于当前SIMD向量扩展部件对浮点超越函数的限制,2020年沈洁等[20]基于飞腾处理器实现了向量三角函数. 该函数的实现与优化方法是结合标量数学库分段计算与向量化的优势,增加了向量三角函数的分支处理. 在函数实现中整体采用"归约-逼近-重建"的实现算法,加之处理器向量部件的宽度优势,一次调用同时计算多个浮点操作数,因而理论性能是标量三角函数的数倍.

鉴于国产申威平台上的长向量函数只能通过循环调用标量函数进行间接实现,2021年刘聃等[21]提出了一种基于SIMD扩展部件的长向量超越函数实现方法. 该方法首先通过汇编来实现完全向量化的SIMD超越函数,然后采用高级语言进行循环调用,最后通过冗余项分类处理来实现长向量超越函数. 其中由于数学函数的基础性质不同,部分函数在进行归约之后分支个数不同. 对于两个分支的函数,刘聃等提出了浮点计算融合算法,将两个计算分支融合成一个分支. "算法归一"是浮点计算融合算法的核心,即通过对多项式近似计算进行分析与变形,最大限度地找出两个分支算法的一致性部分.

2.3　面向RISC-V函数的移植

2021年叶锡聪等[22]对ARM Compute Library函数库进行了面向RISC-V向量指令集的移植,其中提到用宏定义的方法把ARM定义的NEON向量类型替换成RISC-V中的向量类型. 但是本文基于申威数学库面向RISC-V进行移植,本质上是申威数学库函数汇编指令的移植. 由于申威基础数学库的函数数量庞大,汇编指令构成复杂,如果用宏定义的方法进行向量类型的替换,则难以准确、高效地移植所有函数.

2021年曹浩等[11]开发了一个自动化移植与性能优化工具(Automated Migration and Performance Optimization Tool for sunway math library, AMPOT). 该工具提供了一种面向RISC-V架构64位进行层次式标量指令选择算法. AMPOT以申威汇编指令为基础,对每一条指令进行绝对的等价变换,但是没有从全局角度去分析指令完成的功能. 在申威指令集架构向量扩展中,存在许多组合指令. 这类组合指令所完成的功能可以由RISC-V更少的指令完成. 如果对该类指令进行绝对的等价变换,会造成指令冗余,因此该方法对面向RISC-V向量汇编指令的自动化转换并不适用. 但是可以通过对该工具进行向量扩展,添加自动化移植向量扩展模块(Automatic Migration of Vector Extension Modules, AMVEM),以此满足本文工作的需要.

3　AMPOT的向量扩展实现

AMPOT实现的功能实质是将申威数学函数的汇编指令翻译为RISC-V汇编指令. 但是由于AMPOT不支持向量指令,而大部分申威数学函数为了提升函数的性能,使用了大量的向量指令,因此需对AMPOT进行向量扩展,使之适用于申威所有函数的移植.

向量扩展模块AMVEM主要有向量寄存器映射模块(Vector Register Mapping Module, VRMM)和向量指令转换模块(Vector Instruction Conversion Module, VICM)组成. 除此之外,还有指令类型分析模块和异常处理模块,向量寄存器映射模块依据RISC-V指令集架构可变长度的向量扩展部件,以及申威浮点寄存器特点进行设计,完成的功能是将获取到的申威浮点寄存器映射为RISC-V向量寄存器. 向量指令转换模块可以对申威向量汇编代码所完成的功能进行灵活判断,在面向RISC-V指令集正确性转换的前提下,将代码数量进行缩减,从而减少移植后函数的冗余代码,在保证RISC-V数学函数的正确性的同时,最大限度地保证函数的性能. AMPOT的向量扩展实现如图1所示.

首先经过指令预分析. 通过此步骤分析出申威数学函数汇编指令的类型. 若是标量指令则直接进入AMPOT标量指令转换分支,若是向量指令则进入下一步向量寄存器映射. 然后对寄存器类型进行预分析. 通过预分析可以对申威向量汇编指令中的不同类型寄存器进行判断和分离,对分离后的寄存器再分别进行映射. 最后将经VICM得到的RISC-V向量指令与经标量指令转换分支得到的RISC-V标量指令,汇聚成RISC-V数学函数指令流,进而生成RISC-V数学函数. 以申威atan数学函数节选代码段为示例进行详细阐述,如图2所示.

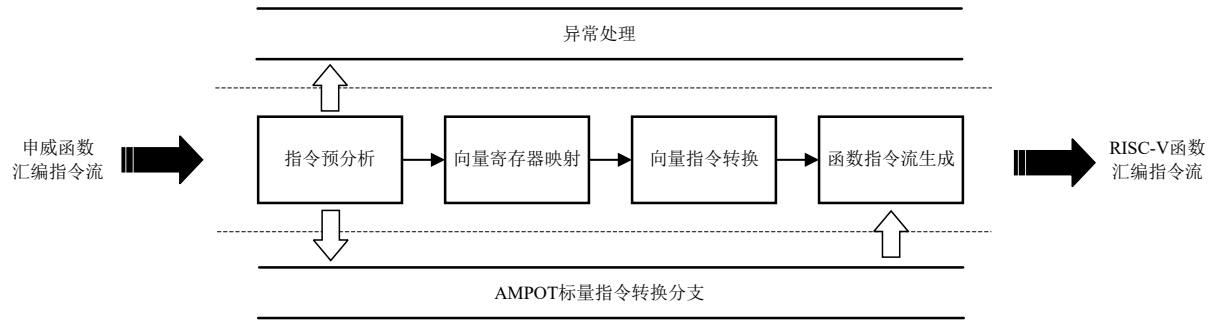


图1 AMPOT向量扩展流程示意图

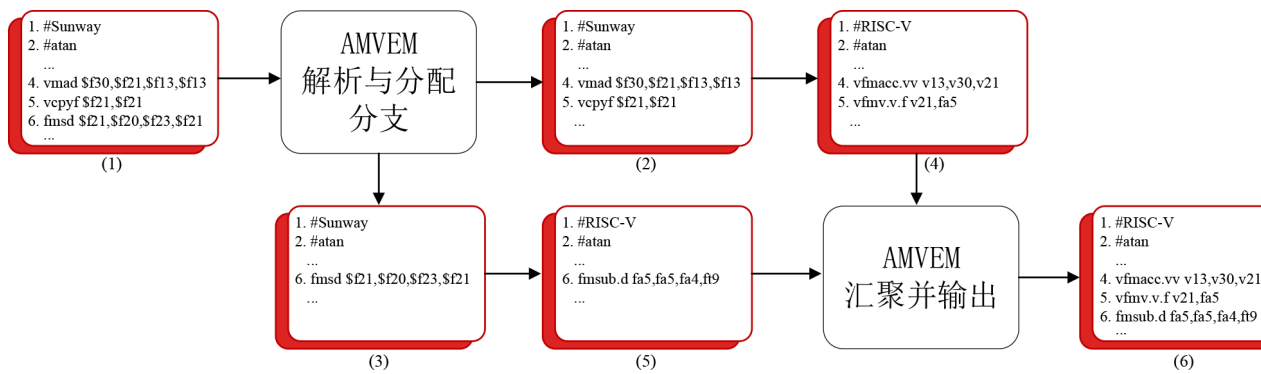


图2 申威atan函数节选代码段示例

图2中的代码(1)是申威数学函数atan的节选代码段,其中包含了标量指令类型(第6行)、向量指令类型(第4、5行). 此外,在向量指令类型中涉及的寄存器组合又分为混合寄存器组合(第5行)和向量寄存器组合(第4行),混合寄存器组合即在一条指令中同时出现了标量寄存器和向量寄存器. AMVEM中的指令预分析模块对输入的申威代码段逐一进行判断,将(1)中的代码段分别解析为向量代码段(2)和标量代码段(3),并为其分配不同的移植分支.

对向量代码段(2)中的指令进行寄存器类型分析可得,vmad指令中的寄存器都是向量寄存器,这种情况下可以直接进行向量寄存器映射. 然而vcpyf指令中的寄存器$f21则代表不同类型. 前者$f21代表的是标量寄存器,后者$f21代表的是向量寄存器,这种情况下需要将标量寄存器和向量寄存器进行分离才能进行下一步映射.

对映射后的RISC-V寄存器进行数量融合和顺序重组,保证RISC-V指令中的源寄存器和目的寄存器的操作顺序与申威保持一致. 进一步地对向量指令助记符进行等价转换,分别得到代码段(4)和代码段(5). AMPOT向量扩展模块与标量移植模块两者是并发的运行状态. 两模块每移植完一条指令便以上一条指令为起点,向下汇聚成RISC-V指令流,进而生成RISC-V数学函数程序,即代码段(6).

4 AMPOT向量扩展的主要功能模块

基于AMPOT高可扩展性的设计以及申威指令集架构和RISC-V指令集架构的特点,对AMPOT向量扩展的各个模块进行了实现. 下面对VRMM和VICM进行详细介绍,指令类型分析和异常处理两个模块较为简单,不作具体描述.

4.1 向量寄存器的自动化映射

申威指令集架构设计了32个256 bit的浮点寄存器($f0-$f31),RISC-V指令集架构同样设计了32个浮点标量寄存器(fa0~fa7, fs0~fs11, ft0~ft11)和32个VLEN默认值为128 bit的向量寄存器(v0-v31). 在第2.1节中我们提到,如果使用RISC-V向量寄存器组来保持和申威向量寄存器长度的契合,则RISC-V可使用的向量寄存器数量少于申威向量寄存器. 由于RISC-V向量寄存器采用可变长度的硬件设计,因此本文通过添加指定编译选项--varch="vlen:256,elen:64,vstartalu:0"将RISC-V向量寄存器VLEN设置为256 bit,一次处理的元素长度为64 bit. 这样就能建立等量配对的向量寄存器映射表和标量寄存器映射表,如表1和表2所列.

4.1.1 向量寄存器自动分支查表法

在申威指令集架构中,$f0~$f31这32个浮点寄存器既当作向量寄存器使用,又当作标量寄存器使用,即

表1　向量寄存器映射表

序号	申威	RISC-V
1	$f0	v0
2	$f1	v1
3	$f2	v2
4	$f3	v3
…	…	…
32	$f31	v31

表2　标量寄存器映射表

序号	申威	RISC-V
1	$f0	fa7
2	$f1	ft0
3	$f2	ft1
4	$f3	ft2
…	…	…
32	$f31	fs11

两者是复用的. 复用又分为局部复用和全局复用. 局部复用是指一条汇编指令中同时存在标量寄存器和向量寄存器,即混合寄存器组合. 如果只获取寄存器的名字,无法分析出两者的区别,那么将会导致寄存器的映射发生错误. 例如申威中vcpyf $f1,$f2指令,这条指令完成的功能是将标量寄存器$f1的双精度浮点数据复制为4个相同的数据放入向量寄存器$f2. 当面向RISC-V进行映射时,无论查阅标量寄存器映射表还是向量寄存器映射表,将会被错误地映射为vfmv.v.f ft1,ft0或者vfmv.v.f v2,v1. 然而,正确的应该是vfmv.v.f v2,ft0. 如果开发人员每当遇到这种问题进行人工分析,那么移植工作不仅效率低而且易出错. 所以针对申威寄存器局部复用的情况,我们提出了向量寄存器自动分支查表的方法. 分支自动查表映射的主要功能是由指令解析单元(Instruction Parsing Unit,IPU)完成,如图3所示.

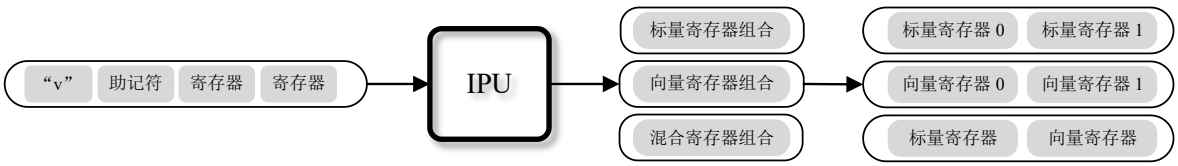


图3　指令解析单元执行示意图

将一条汇编指令中的助记符和寄存器以及立即数等元素当作一个集合,通过分析集合中助记符的首字母是否为"v"来判断是向量指令类型或是标量指令类型. 当申威指令类型被分析出是标量指令时,此时IPU会收到上一步生成的数字1. 根据申威指令集特点,标量指令中的寄存器只有标量寄存器,IPU直接对集合中的所有寄存器进行提取,可得到标量寄存器0和标量寄存器1. 如果IPU收到数字0,则代表申威指令类型为向量指令. 此时,需要结合指令操作码判断该指令对哪种寄存器进行的操作,分为两种情况:一是向量寄存器组合,二是混合寄存器组合. 若是向量寄存器组合,提取过程和标量指令类型下的寄存器提取方法同理. 若是混合寄存器,需要依据寄存器在指令中的位置,对不同类型的寄存器进行提取,实现寄存器的分离,可得到向量寄存器和标量寄存器.

提取出的不同类型寄存器分别查阅寄存器映射表可得到RISC-V寄存器. 最后对RISC-V寄存器进行顺序重组和数量融合,完成寄存器的映射,如图4所示.

IPU对混合寄存器实现判断与映射如算法1所示.

对某条汇编指令的寄存器完成提取映射后,便将本次的提取策略写入IPU. 当再次遇到相同指令时,直接依赖上次写入的提取策略就能完成寄存器的提取和

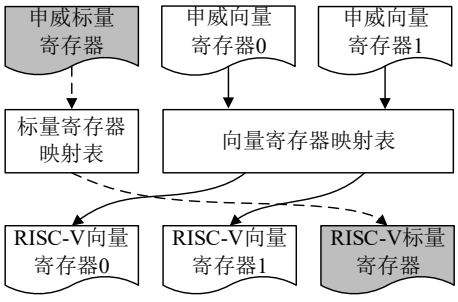


图4　自动分支查表原理图

映射,以此实现寄存器分支查表映射的自动化. 这种一劳永逸的方式,无疑提升了数学函数的移植效率,尤其在工作量大的情况下,这种方法更能体现出优势.

4.1.2　向量寄存器路径标记插入法

全局复用是指在程序的上下文中,宏观分析所有申威浮点寄存器都是向量寄存器,微观分析向量寄存器的低64 bit位也当作标量寄存器使用,但是RISC-V的向量寄存器和标量寄存器都是单独设计的. 如果只做简单的寄存器映射,那么在RISC-V汇编函数程序中,上文的向量寄存器和下文的标量寄存器是无关联的. 向量寄存器路径标记插入法从全局出发,动态分析寄存器的依赖关系. 通过标记需要恢复依赖关系的位置,然后在标记处进行寄存器插入,以此恢复RISC-V上下文向量寄存器和标量寄存器的关联性.

算法1 混合寄存器判断与映射

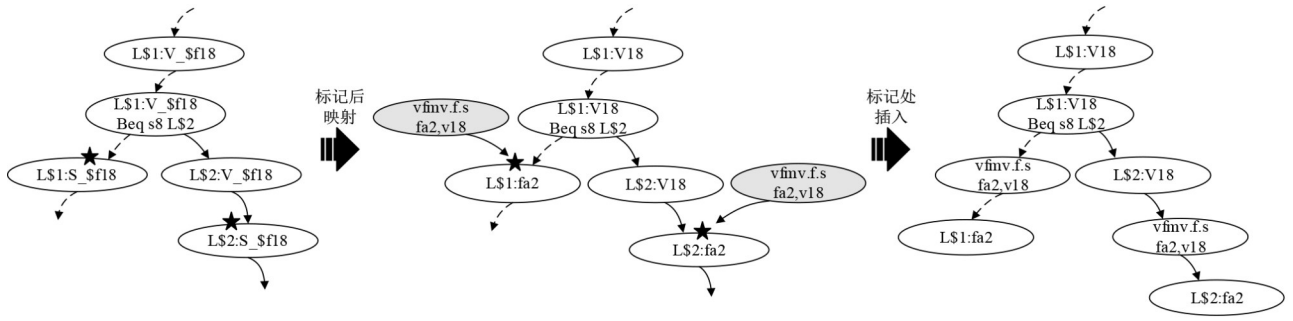
输入: sw_regs //申威寄存器

输出: risc-v_regs //RISC-V 寄存器

```
1.FOR sw_reg in sw_regs:
2.   sw_reg=sw_reg.strip(·)
3.   regs.append(reg_dict.reg_dict.get(sw_reg, sw_reg))//标量寄存器映射表
4.   regs.append(v_reg_dict.v_reg_dict.get(sw_reg, sw_reg))//向量寄存器映射表
5.IF (sw_op == 0 and sw_op == mix_op):
6.      part1 = sw_regs[0][0:5]//提取标量寄存器
7.      part2 = reg_dict.reg_dict.get(part1, part1)//标量寄存器进行映射
8.      tmp1 = regs[1]//向量寄存器进行映射
9.      regs.append(tmp1)
10.      regs.append(part2)
11.      ret_regs.append(regs)
```

向量寄存器路径标记插入法原理如图5所示. 首先是跟踪申威数学函数中作为目的寄存器的向量寄存器编号的路径,即跟踪图5(a)A中V_$f18向量寄存器的路径,并在由该编号代表的向量寄存器变为标量寄存器处注上标记. 例如图5(a)中Beq s8 L$2指令,程序会根据寄存器s8的值进行分支跳转,此时需要分别同时跟踪两个分支中$f18寄存器的路径. 根据图5($a$)中的左分支可以发现,向量寄存器V_$f18变为标量寄存器S_$f18. 两寄存器的关系是S_$f18的数据等于V_$f18寄存器的[63:0]位的数据,此时则需要在此处注上标记. 之所以只跟踪作为目的寄存器的向量寄存器路径,是因为当某个向量寄存器首次出现时,其一定是目的寄存器,否则该向量寄存器的值一定为空.

基于第4.1.1节申威寄存器面向RISC-V寄存器映射之后,将得到RISC-V寄存器路径,如图5(b)所示. 在该路径中,上文的向量寄存器和下文的标量寄存器失去了图5(a)中的寄存器关联性. 因为在RISC-V架构中,向量寄存器和标量寄存器是独立无关联设计,即子图B左分支中的fa2寄存器的数据不等于上文向量寄存器v18的[63:0]位的数据. 因此,在申威寄存器路径标记处,需要插入vfmv.f.s fax,vx指令. 该指令所完成的功能就是将RISC-V向量寄存器的[63:0]位元素复制进标量寄存器,通过这样的方式建立等同于申威寄存器的关联性. 完成插入指令后的RISC-V寄存器路径如图5(c)所示.



(a) 申威 (b) RISC-V (c) RISC-V

图5 向量寄存器路径标记插入流程图

在申威数学函数中,汇编代码量大,分支跳转复杂. 如果对每一个函数里面所有向量寄存器以人工的方式进行路径分析,不仅工作量大,而且极易出错. 为了减少了冗余跟踪,路径标记插入法仅对首次作为目的寄存器出现的向量寄存器进行路径跟踪. 如果该向量寄存器的值在中间过程有变化,但是还没有当作标量寄存器出现时,则将值变化之后的向量寄存器更新为路径的起点. 如果跟踪的向量寄存器在跟踪过程中不以目的寄存器出现时,则不计入该向量寄存器的路径里. 以这样的方式对向量寄存器路径进行跟踪及插入,不仅节省人工成本,而且具有准确性和高效性.

4.2 向量汇编指令的自动化转换

RISC-V指令集架构与申威指令集架构虽然设计类似,但是指令数量和种类以及完成的功能依然有所区别. 不同的指令在实际移植过程中遇到的问题也会有所不同,仅靠人工分析每一个函数的每一条指令进行移植,工作量无疑是巨大且烦琐的. 所以本文在AMPOT的基础上实现向量汇编指令自动化转换模块,通过对该模块的逐步完善,使之适用于所有函数汇编指令的自动化转换.

本文采用等价功能转换的方法实现向量汇编指令自动化转换模块,如图6所示. 等价功能转换方法首先分析申威数学函数汇编指令SWIx所完成的功能,其次分析RISC-V可以完成同样功能的指令RVIy,最后将对于申威指令的分析策略与RISC-V指令的生成策略写入汇编指令自动化转换模块里,可表示为SWIx = RVIy. 当再次遇到SWIx汇编指令时,可以直接经上述策略得

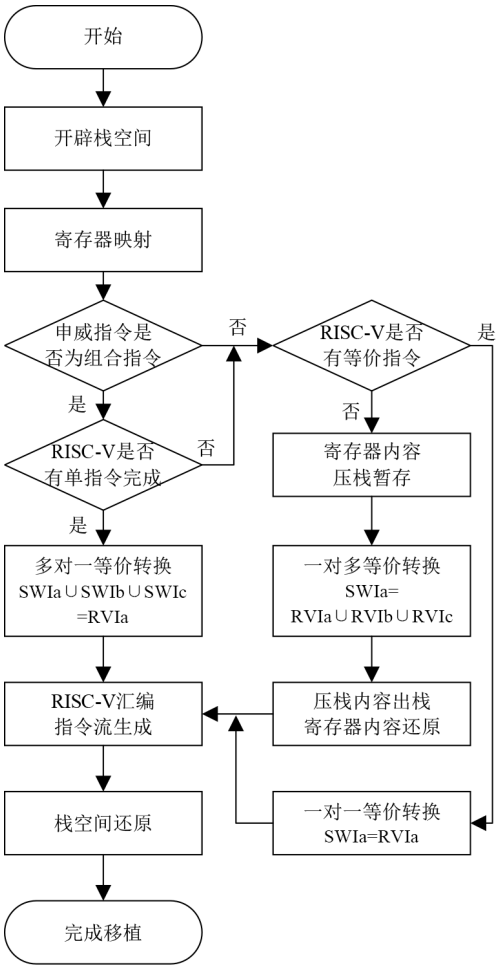


图 6　向量汇编指令自动转换流程

到 RVIy. 通过将转换策略对 VICM 的一次写入,而可以无限次利用,这无疑大大提升了移植效率.

对于 SWIx = RVIy 又分为以下三种情况.

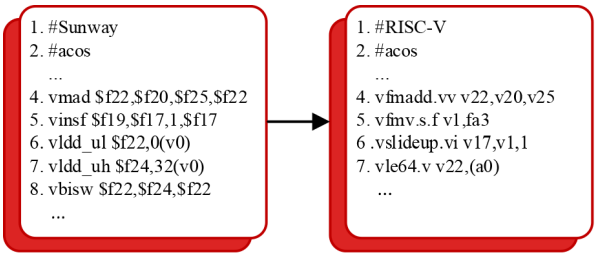
(1) RISC-V 有与申威相同向量指令功能且向量寄存器一致的情况. 此类向量指令在自动化转换模块里对汇编指令进行一对一等价变换,即 SWIa = RVIa.

(2) RISC-V 有与申威相同向量指令功能但是向量寄存器不一致和 RISC-V 没有对应申威的向量指令功能的情况. 此类情况需要对汇编指令进行一对多等价变换,即 SWIa=RVIa∪RVIb∪RVIc.

(3) 申威向量组合指令完成的功能可以由 RISC-V 单条向量指令完成的情况. 这需要对汇编指令进行多对一等价变换,即 SWIa∪SWIb∪SWIc=RVIa.

以申威 acos 数学函数节选代码段为示例,详细阐述向量汇编指令自动化转换三种情况具体完成的功能(图 7).

(1) 图 7 的代码 1 中第 4 行指令 vmad 是向量乘加指令. 根据图 6 的向量汇编指令自动转换流程,该指令非申威向量组合指令,并且 RISC-V 有等价功能的指令,即遵循一对一等价转换 SWIa=RVIa. 因此,代码 1 中的申威指令 vmad $f22,$f20,$f25,$f22 被转换成代码 2 中的 RISC-V 指令 vfmadd. vv v22,v20,v25.



(代码1)　　(代码2)

图 7　申威 acos 数学函数节选代码段示例

(2) 代码 1 中第 5 行指令 vinsf 是浮点向量元素插入指令,即将标量寄存器 $f19 的数据插入到向量寄存器 $f17 的[127:64]位置. 由于 RISC-V 没有完成这种功能的向量指令,所以按照向量汇编指令自动转换流程,该指令遵循一对多等价转换 SWIa=RVIa∪RVIb∪RVIc. 因此,代码 1 中的申威指令 vinsf $f19,$f17,1,$f17 被转换成代码 2 中的第 5、第 6 行指令. 此转换过程使用了 AMPOT 中借助辅助寄存器完成转换的方法,借助向量辅助寄存器 v1 来完成转换. 如图 6 所示,进行一对多等价转换之前,需要将 v1 寄存器的值进行压栈,进一步地将 fa3 的值复制进 v1 向量寄存器的[63:0]位置,通过 vslideup.vi v17,v1,1 指令将 v1 的[63:0]数据移动到向量寄存器 v17 的[127:64]位置. 最后需要将向量辅助寄存器 v1 的值从栈中取出还原,以恢复 v1 寄存器现场.

(3) 在申威指令集架构中,存在一些向量组合指令,例如图 7 代码 1 中的第 6、第 7、第 8 行. 由于申威架构中数据在内存中的存储地址是对界的,因此需要通过 vldd_ul,vldd_uh,vbisw 三条向量指令将 256 bit 浮点数据从内存中加载到向量寄存器 $f22 中. 而在 RISC-V 架构中数据存放时不要求地址对界,因此只需要一次加载 256 bit 浮点数据即可完成申威指令的组合功能. 按照图 6 自动转换策略,该组合指令遵循多对一等价转换 SWIa∪SWIb∪SWIc=RVIa,即自动化转换为代码 2 中的 vle64.v v22,(a0).

5　RISC-V 基础数学库函数测试

在 AMPOT 移植工具的基础上,通过面向 RISC-V 进行向量扩展,成功移植了 69 个 RISC-V 数学函数,形成了首个使用向量指令优化的 RISC-V 双精度基础数学函数库. 选择领域内权威、合适的测试方法[23],对 RISC-V 数学函数进行精度测试与性能测试,以验证 RISC-V 数学函数库的可用性与 AMPOT 向量扩展的有效性. 通过

对测试数据进行统计与分析，为RISC-V数学函数的进一步优化提供数据支撑.

5.1 测试函数

申威基础数学函数库共包含152数学函数，其中双精度和单精度各76个. 由于双精度函数中$j0$、$j1$、$y2$、$y0$、$y1$、$y2$、drem这7个函数是高级语言实现，因此本文仅对69个汇编实现的双精度数学函数进行了面向RISC-V指令集架构的自动化移植，移植的69个函数中函数类型多样，如图8所示. RISC-V基础数学函数库包含9个三角函数、8个反三角函数、3个双曲函数、3个反双曲函数、4个指数函数、5个对数函数、32个数值函数以及5个其他函数. 其中复杂超越函数伽马(lgamma)函数分支高达39个，此类函数算法逻辑和寄存器依赖关系复杂. 一些数值函数的分支跳转和访存机制较为简单，如fabs函数和fp_class函数. 上述RISC-V基础数学函数库覆盖了绝大部分汇编指令类型，这也从侧面验证了经向量扩展后的AMPOT自动化移植能力更强，适用范围更广.

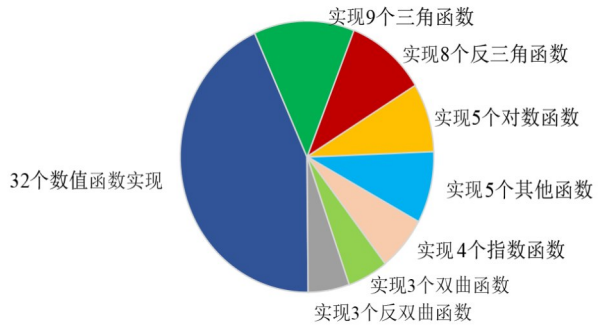


图8 RISC-V基础数学库函数类型统计

5.2 测试环境

由于目前缺乏支持RISC-V Vector 0.10的硬件环境，所以本文使用由RISC-V基金会支持的指令模拟器SPIKE[24]进行精度测试，采用时序精确的gem5模拟器[25]进行性能测试，测试环境的搭建如图9所示.

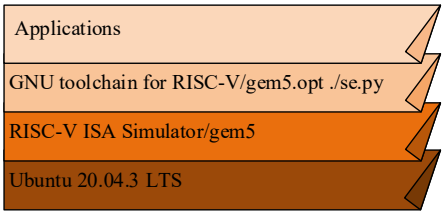


图9 RISC-V程序测试环境

最底层使用的是Ubuntu 20.04.3 LTS[26]，精度测试通过在SPIKE功能模拟器上搭建GNU toolchain for RISC-V[27]来完成对RISC-V程序的编译支持，所有RISC-V数学函数程序通过添加-march=rv64gcv编译选项进行编译执行. 性能测试通过在Ubuntu上搭建gem5对应的RISC-V架构进行系统模拟，经riscv64-unknown-elf-gcc编译之后，使用“./gem5.opt./se.py--cpu-type RiscvAtomicSimpleCPU”指令完成执行.

5.3 精度测试

为保证对RISC-V数学函数进行有效性测试，本文按照IEEE-754浮点数分布的全浮点域指数分布测试集的生成方法[23]，在函数定义域内生成5 000 000个双精度浮点测试样本，最大限度地保证函数分支覆盖率，确保每一个分支都能够进行正确性运算.

精度测试是和高精度MPFR数学库计算相对误差(relative ULP，relULP)实现，并以相同的方式测试申威数学函数、Glibc数学函数的精度结果. MPFR库是一个可以提供多精度浮点计算的库，在理论上提供对任意精度浮点函数的支持，是领域内进行精度测试对比的权威数学库. 本文精度测试采用如下计算过程，并以相对ULP作为衡量数学函数库的精度情况：

$$\mathrm{relULP}=\frac{\left|E(x)_{\mathrm{RISC-V/Sunway/Glibc}}-E(x)_{\mathrm{mpfr}}\right|}{\mathrm{ulp}(x)_{\mathrm{mpfr}}}$$

在相同测试样本下，$E(x)_{\mathrm{RISC\text{-}V/Sunway/Glibc}}$代表RISC-V数学函数、申威数学函数、Glibc数学函数计算得到的结果，$E(x)_{\mathrm{mpfr}}$代表此函数在MPFR上计算得到的结果，$\mathrm{ulp}(x)_{\mathrm{mpfr}}$代表MPFR计算得到结果的ULP值.

采用上述的测试方法，统计各函数的最大ULP值以及输出结果在0ULP的百分比，测试结果如表3所列，表中黑色三角符号表示Glibc函数库中没有对应的函数. 以这种方法计算得到relULP值，并与成熟的数学函数进行比较，更能精细化反应出RISC-V数学函数的精度情况.

以5 000 000个输入为测试样本，对表3中69个RISC-V数学函数进行了精度测试，并与MPFR高精度数学库进行了对比，据结果统计，RISC-V函数库最大ULP平均为0.504 279，0ULP占比为97.888 81%. 申威函数库最大ULP平均为0.518 492，0ULP占比为97.974 62%. Glibc函数库最大ULP平均为0.578 289，0ULP占比为97.905 33%. 经测试验证，绝大部分RISC-V数学函数产生的相对误差处于0ULP区间，且最大相对误差平均值均小于申威数学库和Glibc数学库对应的值. 虽然0ULP占比稍逊于另外两个数学库，但从整个数学库的角度出发，RISC-V数学函数的精度结果符合预期精度范围.

表3　RISC-V和Sunway及Glibc基础数学库函数精度测试果

序号	函数	RISC-V		Sunway		Glibc		函数类型
		Max ulp	0ulp [0,0.5)	Max ulp	0ulp [0,0.5)	Max ulp	0ulp [0,0.5)	
1	acos	0.775 757	99.517 60%	0.775 765	99.213 87%	0.499 998	100.000 00%	反三角函数
2	acosd	0.775 045	99.202 80%	0.859 601	98.213 42%	▲	▲	反三角函数
3	acosh	1.054 710	99.487 60%	1.056 113	99.373 01%	0.983 602	80.875 74%	反双曲函数
4	addtc	0.000 000	100.000 00%	0.000 000	100.000 00%	▲	▲	调用函数
5	asin	0.550 706	99.995 60%	0.574 605	99.996 62%	0.500 000	100.000 00%	反三角函数
6	asind	1.517 407	76.817 20%	1.602 849	76.341 67%	▲	▲	反三角函数
7	asinh	0.995 009	99.752 00%	0.998 804	99.689 02%	1.507 567	90.100 15%	反双曲函数
8	atan	0.510 000	99.997 14%	0.522 575	99.995 13%	0.500 000	100.000 00%	反三角函数
9	atan2	1.389 864	90.310 80%	0.534 729	98.683 20%	0.500 000	100.000 00%	反三角函数
10	atand	1.730 000	66.886 60%	1.762 473	66.873 10%	▲	▲	反三角函数
11	atand2	1.610 000	76.890 80%	1.630 684	76.395 10%	▲	▲	反三角函数
12	atanh	0.554 198	99.993 80%	0.561 236	99.993 68%	1.500 083	99.338 68%	反双曲函数
13	cabs	1.000 000	99.869 00%	1.000 000	99.853 20%	0.000 000	100.000 00%	数值函数
14	cbrt	0.525 989	99.290 20%	0.523 941	99.730 56%	3.425 189	45.247 58%	数值函数
15	ceil	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
16	copysign	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
17	cos	0.541 737	99.447 60%	0.540 653	99.444 98%	0.500 000	100.000 00%	三角函数
18	cosd	0.525 616	99.985 40%	0.525 616	99.985 40%	▲	▲	三角函数
19	cosh	0.999 965	99.899 75%	0.999 866	99.970 86%	1.499 737	99.907 56%	双曲函数
20	cot	0.535 427	99.792 80%	0.535 263	99.785 80%	▲	▲	三角函数
21	cotd	1.500 000	99.463 80%	1.500 000	99.534 60%	▲	▲	三角函数
22	cvttqc	0.000 000	100.000 00%	0.000 000	100.000 00%	▲	▲	调用函数
23	erf	0.894 361	97.145 20%	1.558 380	96.367 76%	1.027 441	96.315 71%	误差函数
24	erfc	1.901 567	99.894 60%	1.901 567	99.920 25%	2.679 938	99.900 96%	互补误差函数
25	exp	0.999 983	99.933 13%	0.999 910	99.975 15%	0.999 910	99.975 57%	指数函数
26	exp2	0.999 860	99.975 16%	0.999 830	99.975 17%	0.999 830	99.975 16%	指数函数
27	expm1	0.501 019	99.999 47%	0.503 974	99.999 65%	0.788 579	99.954 39%	指数函数
28	fabs	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
29	fdim	0.500 000	99.944 80%	0.500 000	99.940 00%	0.500 000	99.944 80%	数值函数
30	finite	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
31	floor	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
32	fma	0.499 996	100.000 00%	1.638 449	87.656 17%	0.499 996	100.000 00%	数值函数
33	fmax	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
34	fmin	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
35	fmod	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
36	fp_class	0.000 000	100.000 00%	0.000 000	100.000 00%	▲	▲	数值函数
37	frexp	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
38	hypot	0.951 289	99.652 00%	1.004 020	86.706 68%	0.980 823	87.728 98%	数值函数
39	ilogb	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
40	isnan	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
41	ldexp	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
42	lgamma	1.792 103	85.437 57%	1.792 103	86.169 60%	1.826 136	85.895 01%	伽马函数
43	log	0.504 946	99.999 00%	0.533 355	99.999 07%	0.500 000	100.000 00%	对数函数
44	log10	0.499 999	99.998 75%	0.534 180	99.998 61%	1.564 316	99.776 30%	对数函数
45	log1p	0.749 075	99.987 00%	0.749 452	99.982 46%	0.750 976	99.931 26%	对数函数

续表

序号	函数	RISC-V		Sunway		Glibc		函数类型
		Max ulp	0ulp [0,0.5)	Max ulp	0ulp [0,0.5)	Max ulp	0ulp [0,0.5)	
46	log2	0.500 000	99.998 78%	0.532 236	99.998 63%	1.593 188	99.885 53%	对数函数
47	logb	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	对数函数
48	lrint	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
49	lround	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
50	modf	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
51	nearbyint	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
52	nextafter	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
53	nint	0.000 000	100.000 00%	0.000 000	100.000 00%	▲	▲	数值函数
54	pow	0.502 440	99.998 87%	0.502 440	98.268 04%	0.4999 99	99.102 23%	指数函数
55	rint	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
56	round	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
57	scalb	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
58	scalbln	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
59	scalbn	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
60	sin	0.531 143	99.429 93%	0.544 457	99.447 27%	0.500 000	100.000 00%	三角函数
61	sincos	0.541 653	99.450 00%	0.541 504	99.461 00%	0.518 140	99.923 60%	三角函数
62	sind	0.723 339	94.684 60%	0.761 942	94.184 60%	▲	▲	三角函数
63	sinh	0.500 216	99.999 35%	0.507 713	99.999 54%	1.660 434	99.614 43%	双曲函数
64	sqrt	0.499 997	100.000 00%	0.500 000	100.000 00%	0.500 000	100.000 00%	数值函数
65	tan	0.530 166	99.789 20%	0.540 427	99.792 75%	0.500 000	100.000 00%	三角函数
66	tand	1.540 000	72.413 98%	1.541 481	99.336 20%	▲	▲	三角函数
67	tanh	0.536 371	99.996 00%	0.583 767	99.997 13%	2.078 258	99.305 05%	双曲函数
68	trunc	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数
69	unordered	0.000 000	100.000 00%	0.000 000	100.000 00%	0.000 000	100.000 00%	数值函数

从表3可知，RISC-V的互补误差函数erfc和复杂超越函数伽马(lgamma)函数，最大ULP分别为1.901 567和1.792 103，相较于其他函数的最大ulp显得异常．为了分析该测试数据的出现，对申威lgamma函数和erfc函数与对应的RISC-V函数进行了分支跟踪比较．

采用传统的方法跟踪lgamma函数和erfc函数产生较大误差对应的分支跳转路径，如图10和图11所示．图中节点序号的最大值代表函数分支的总个数，即lgamma函数共39个分支，erfc函数共9个分支．无跳出箭头的节点代表函数的返回结果分支，橙色箭头代表两函数产生较大误差对应的路径．

RISC-V基础数学库是基于申威架构数学库进行移植的并与MPFR函数进行精度对比，因此基于RISC-V平台、申威平台和MPFR平台对橙色路径产生的结果进行比较，以此分析RISC-V对应的两个函数在橙色路径中是否存在误差传播，比较结果如表4所列．以分支跳转是橙色路径的输入为测试样本，测试结果表明(蓝色变量x代表结果最后一位不同)，在相同测试样本下，RISC-V平台和申威平台在橙色路径产生的结果相同，

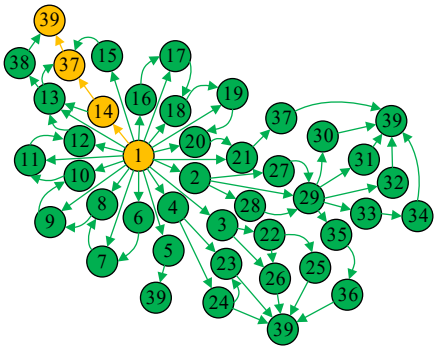


图10 lgamma函数产生较大误差的输入分支跳转路径

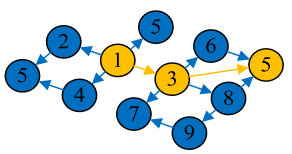


图11 erfc函数产生较大误差的输入分支跳转路径

与MPFR的结果相差2. 由于申威源函数这两个分支上的算法实现复杂，“归约-逼近-重建”的方法导致该测试

表4　lgamma函数与erfc函数不同平台测试结果

lgamma					erfc				
输入(0x)	输出(0x)	RISC-V	申威	MPFR	输入(0x)	输出(0x)	RISC-V	申威	MPFR
3ff000125a9ccc8a	bee53017169b851*x*	8	8	6	3ff0001292121f0e	3fc42223b7a8f87*x*	e	e	c
3ff0001282a66c56	bee55e4efb9e3aa*x*	a	a	8	40000012817bc636	3f7328321e14b93*x*	2	2	0
403000125e5add21	403be668ff82665*x*	a	a	8	3ff000122440b7d4	3fc42225245660a*x*	c	c	e
40300012404218ec	403be668ad0364b*x*	9	9	b	3ff0001285d11771	3fc42223e05a3d1*x*	6	6	8
3ff000128a5ecc1d	bee5673874cb534*x*	4	4	2	400000127d8b31bd	3f73283247c380b*x*	6	6	4

分支出现了误差累积,因此移植后的RISC-V函数和申威函数存在相同的误差传播.

5.4　性能测试

RISC-V和Glibc基础数学库函数的性能测试均基于gem5模拟器进行,以保证环境一致性. 测试区间来自Inter® oneAPI Math Kernel Library[28],以此得到的数据更有参考价值. 由于申威基础数学库函数受平台的限制,因此在"太湖之光"超级计算机上完成测试. 以上均通过获取1 000个测试样本的平均节拍实现. 为了保证测试的节拍结果准确,需要对函数进行预热,以此降低外部环境对测试的影响,各函数的平均节拍通过式(1)计算,数学库的总平均节拍通过式(2)计算:

$$\mathrm{averCYC}=\frac{1}{mn}\sum_{i=1,j=1}^{m,n}\mathrm{cycle}_{ij} \tag{1}$$

$$\mathrm{AVER\_CYC}=\frac{1}{k}\sum_{l=1}^{k}\mathrm{averCYC}_{l} \tag{2}$$

式(1)中,cycle_{ij}表示样本x_j第i次测试得到的节拍数;m表示每个测试样本的测试次数,n表示测试样本的个数,在本文中m取10,n取1 000. 式(2)中$\mathrm{averCYC}_l$表示各函数的平均节拍,k表示测试的函数个数. 根据式(1)计算得各函数的平均节拍,测试结果如图12所示.

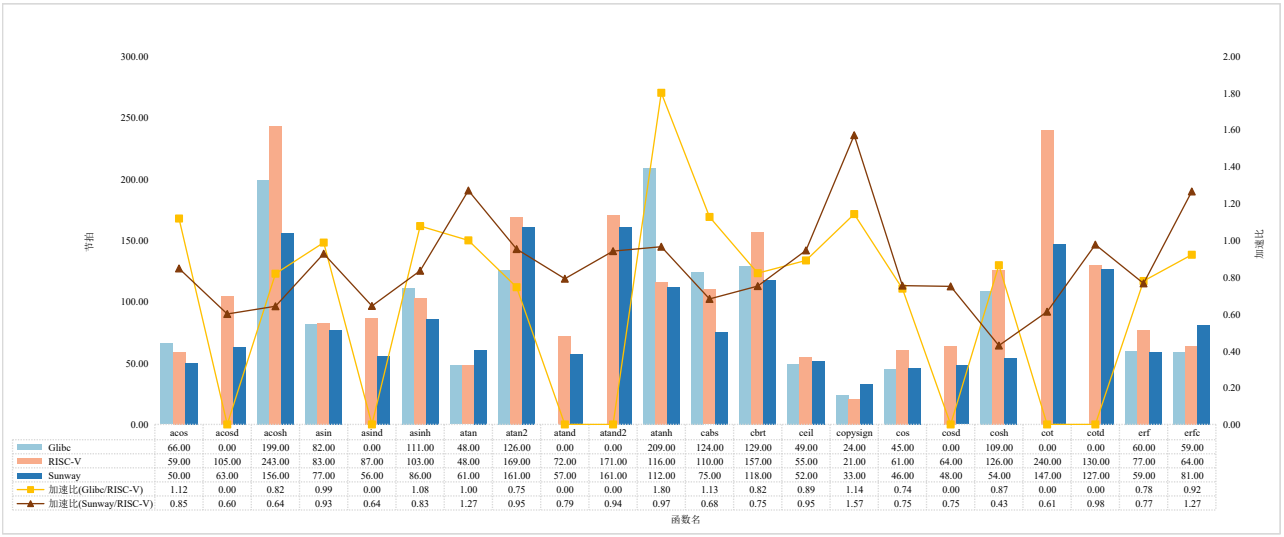
经测试,RISC-V的69个函数中addtc函数和cvttqc函数属于内部调用函数,不进行性能测试,据式(2)计算,测试的67个函数性能平均为157.03节拍. 由于Glibc库无cot、addtc、cvttqc、fp_class、nint、acosd、asind、atand、atand2、cosd、cotd、sind、tand这13个函数,据式(2)计算,测试的56个函数性能平均为181.71节拍. 对于申威数学库,除2个内部调用函数不进行性能测试,据式(2)计算,其余67个函数的性能平均为131.22节拍. 在3种数学库中fmod函数的性能表现明显低于其他函数,经测试,Glibc为5 317节拍,RISC-V为4 627节拍,申威为3 537节拍.

根据测试结果不难发现,相较于Glibc,RISC-V不同函数有不同的性能表现. 算法逻辑复杂的exp2函数,性能提升较为明显,加速比达2.49倍. 而简单的数值函数,加速不明显,比如sqrt和ilogb等函数,部分函数性能低于Glibc函数. RISC-V的sinh和cosh函数,性能明显低于申威源函数,这是由于申威函数中含有太多需要一对多等价转换(SWIa=RVIa∪RVIb∪RVIc)指令. 一对多等价转换会增加对内存的读写操作,加之申威函数的性能测试是基于成熟的硬件环境完成的,因此其性能表现低于申威源函数.

为进一步分析函数指令复杂度,本文对部分函数的运算指令、访存指令、移位指令、传送指令以及其他指令(逻辑运算、跳转等)进行了数量统计,如表5所示. 各函数的访存指令和运算指令占比较高,由于对内存进行读写比对寄存器进行读写的速度慢很多,运算指令所需的节拍数一般也多于其他指令,因此,访存指令和运算指令对函数性能会产生一定的影响. 下一步将对该类型指令进行优化.

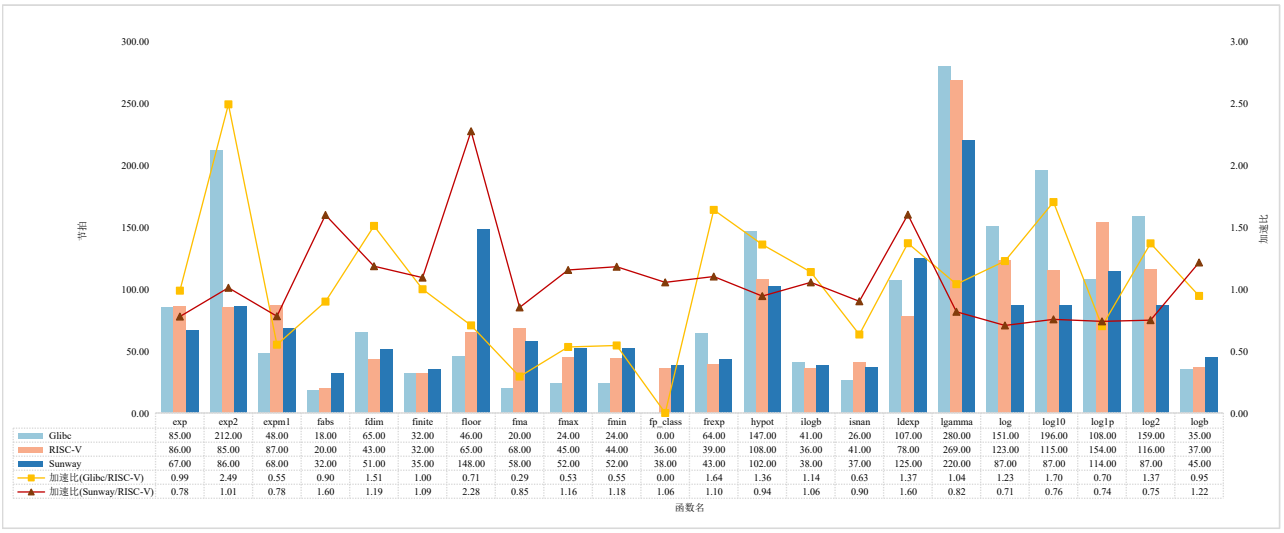
6　结束语

本文根据申威指令集架构和RISC-V指令集架构的特点,提出了向量寄存器自动分支查表法与路径标记插入法,实现了申威向量寄存器面向RISC-V向量寄存器的自动化映射;依据不同指令等价转换策略设计了向量汇编指令自动化转换模块,节省了大量人工成本,使函数移植工作更加准确高效;通过自动化移植的方式,完成了69个申威基础数学库函数面向RISC-V的移植;实现了首个使用向量指令优化的RISC-V指令集基础数学库,丰富了RISC-V软件生态. 测试结果表明,RISC-V基础数学库函数的精度结果最大ULP平均为0.504 279,0ULP占比为97.888 81%,函数性能平均为157.03节拍,验证了自动化移植框架AMPOT的正确性. 依据科学的测试方法对函数完成测试,并针对测试结果作了较详细的分析,得到的数据具有一定的参考价值. 由于本文工作优先考虑函数的高精度实现,所以部分函数的性能还有一定的优化空间. 下一步,将对RISC-V基础数学库函数进行性能优化. 此外,经过向



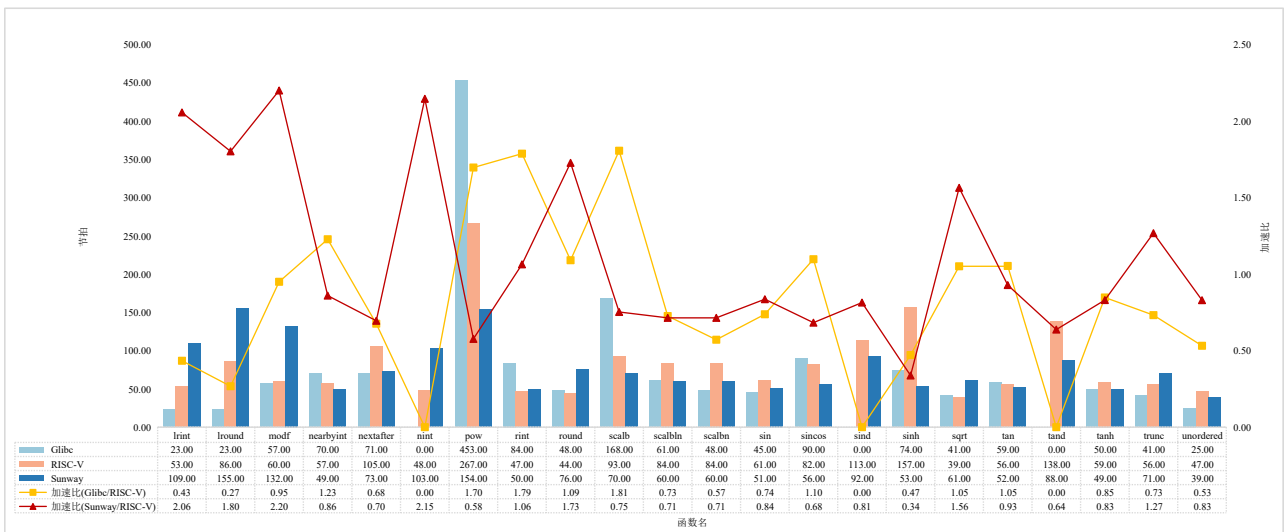
	acos	acosd	acosh	asin	asind	asinh	atan	atan2	atand	atand2	atanh	cabs	cbrt	ceil	copysign	cos	cosd	cosh	cot	cotd	erf	erfc
Glibc	66.00	0.00	199.00	82.00	0.00	111.00	48.00	126.00	0.00	0.00	209.00	124.00	129.00	49.00	24.00	45.00	0.00	109.00	0.00	0.00	60.00	59.00
RISC-V	59.00	105.00	243.00	83.00	87.00	103.00	48.00	169.00	72.00	171.00	116.00	110.00	157.00	55.00	21.00	61.00	64.00	126.00	240.00	130.00	77.00	64.00
Sunway	50.00	63.00	156.00	77.00	56.00	86.00	61.00	161.00	57.00	161.00	112.00	75.00	118.00	52.00	33.00	46.00	48.00	54.00	147.00	127.00	59.00	81.00
加速比(Glibc/RISC-V)	1.12	0.00	0.82	0.99	0.00	1.08	1.00	0.75	0.00	0.00	1.80	1.13	0.82	0.89	1.14	0.74	0.00	0.87	0.00	0.00	0.78	0.92
加速比(Sunway/RISC-V)	0.85	0.60	0.64	0.93	0.64	0.83	1.27	0.95	0.79	0.94	0.97	0.68	0.75	0.95	1.57	0.75	0.75	0.43	0.61	0.98	0.77	1.27

(*a*) RISC-V 和 Sunway 及 Glibc 基础数学库函数性能测试结果一



	exp	exp2	expm1	fabs	fdim	finite	floor	fma	fmax	fmin	fp_class	frexp	hypot	ilogb	isnan	ldexp	lgamma	log	log10	log1p	log2	logb
Glibc	85.00	212.00	48.00	18.00	65.00	32.00	46.00	20.00	24.00	24.00	0.00	64.00	147.00	41.00	26.00	107.00	280.00	151.00	196.00	108.00	159.00	35.00
RISC-V	86.00	85.00	87.00	20.00	43.00	32.00	65.00	68.00	45.00	44.00	36.00	39.00	108.00	36.00	41.00	78.00	269.00	123.00	115.00	154.00	116.00	37.00
Sunway	67.00	86.00	68.00	32.00	51.00	35.00	148.00	58.00	52.00	52.00	38.00	43.00	102.00	38.00	37.00	125.00	220.00	87.00	87.00	114.00	87.00	45.00
加速比(Glibc/RISC-V)	0.99	2.49	0.55	0.90	1.51	1.00	0.71	0.29	0.53	0.55	0.00	1.64	1.36	1.14	0.63	1.37	1.04	1.23	1.70	0.70	1.37	0.95
加速比(Sunway/RISC-V)	0.78	1.01	0.78	1.60	1.19	1.09	2.28	0.85	1.16	1.18	1.06	1.10	0.94	1.06	0.90	1.60	0.82	0.71	0.76	0.74	0.75	1.22

(*b*) RISC-V 和 Sunway 及 Glibc 基础数学库函数性能测试结果二



	lrint	lround	modf	nearbyint	nextafter	nint	pow	rint	round	scalb	scalbln	scalbn	sin	sincos	sind	sinh	sqrt	tan	tand	tanh	trunc	unordered
Glibc	23.00	23.00	57.00	70.00	71.00	0.00	453.00	84.00	48.00	168.00	61.00	48.00	45.00	90.00	0.00	74.00	41.00	59.00	0.00	50.00	41.00	25.00
RISC-V	53.00	86.00	60.00	57.00	105.00	48.00	267.00	47.00	44.00	93.00	84.00	84.00	61.00	82.00	113.00	157.00	39.00	56.00	138.00	59.00	56.00	47.00
Sunway	109.00	155.00	132.00	49.00	73.00	103.00	154.00	50.00	76.00	70.00	60.00	60.00	51.00	56.00	92.00	53.00	61.00	52.00	88.00	49.00	71.00	39.00
加速比(Glibc/RISC-V)	0.43	0.27	0.95	1.23	0.68	0.00	1.70	1.79	1.09	1.81	0.73	0.57	0.74	1.10	0.00	0.47	1.05	1.05	0.00	0.85	0.73	0.53
加速比(Sunway/RISC-V)	2.06	1.80	2.20	0.86	0.70	2.15	0.58	1.06	1.73	0.75	0.71	0.71	0.84	0.68	0.81	0.34	1.56	0.93	0.64	0.83	1.27	0.83

(*c*) RISC-V 和 Sunway 及 Glibc 基础数学库函数性能测试结果三

图 12 RISC-V 和 Sunway 及 Glibc 基础数学库函数性能测试结果

表5　RISC-V数学函数各类型指令数

序号	函数	运算指令	访存指令	移位指令	传送指令	其他指令	总指令数	序号	函数	运算指令	访存指令	移位指令	传送指令	其他指令	总指令数
1	acos	127	257	42	82	189	697	28	isnan	1	0	2	1	12	16
2	acosh	37	135	8	30	118	328	29	ldexp	25	89	16	19	103	252
3	asin	104	225	30	83	205	647	30	lgamma	133	365	101	105	597	1 301
4	asinh	60	97	33	8	32	230	31	log	70	209	23	33	156	491
5	atan	47	49	7	16	75	194	32	log10	69	211	29	45	182	536
6	atan2	157	442	35	65	349	1 048	33	log1p	83	185	26	37	204	535
7	atanh	82	178	12	20	92	384	34	log2	69	211	29	46	183	538
8	cbrt	55	97	16	33	72	273	35	logb	13	86	4	15	37	155
9	ceil	10	20	7	12	33	82	36	lrint	21	75	11	21	48	176
10	copysign	0	0	0	2	5	7	37	lround	18	88	11	38	65	220
11	cos	154	196	16	46	158	570	38	modf	10	0	6	20	37	73
12	cosh	82	159	10	16	86	353	39	nearbyint	12	2	6	11	29	60
13	cot	213	285	46	75	220	839	40	nextafter	45	100	16	56	239	456
14	erf	122	332	13	29	175	671	41	nint	10	8	6	18	29	71
15	erfc	145	290	33	40	239	747	42	pow	143	424	48	127	548	1 290
16	exp	35	126	28	39	107	335	43	rint	15	42	4	7	34	102
17	exp2	42	142	18	32	119	353	44	round	10	8	6	18	29	71
18	expm1	64	150	10	18	109	351	45	scalb	51	188	35	62	237	573
19	fabs	0	0	0	0	3	3	46	scalbln	25	87	16	20	104	252
20	finite	1	0	1	1	8	11	47	scalbn	25	87	16	20	104	252
21	floor	15	58	7	14	47	141	48	sin	153	165	16	47	181	562
22	fma	44	219	20	57	197	537	49	sinh	82	123	10	12	82	309
23	fmod	158	420	53	162	425	1218	50	sqrt	1	0	2	3	26	32
24	fp_class	3	0	5	5	29	42	51	tan	174	266	49	73	218	780
25	frexp	2	4	7	11	30	54	52	tanh	101	190	8	12	93	404
26	hypot	55	150	33	43	192	473	53	trunc	4	8	5	14	26	57
27	ilogb	5	11	7	10	49	82	54	unordered	2	0	2	6	20	30

量扩展后的AMPOT依然具有高可扩展性,后续可以对其进行单精度指令集扩展,实现RISC-V单精度基础数学函数库. 本文工作是在RISC-V模拟环境下完成的,相关条件具备后,可以将数学库部署在RISC-V硬件环境下.

参考文献

[1] HU Y M, YANG H L, LUAN Z Z, et al. Massively scaling seismic processing on sunway taihulight supercomputer[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(5) : 1194-1208.

[2] QI H Y, XU J C, GUO S Z. Detection of the maximum error of mathematical functions[J]. The Journal of Supercomputing, 2018, 74(11): 6275-6290.

[3] 郭绍忠, 许瑾晨, 陈建勋. 一种改进的超越函数通用算法[J]. 计算机工程, 2012, 38(15): 31-34.
GUO S Z, XU J C, CHEN J X. Improved transcendental function general algorithm[J]. Computer Engineering, 2012, 38(15): 31-34. (in Chinese)

[4] INTEL CORPORATION. Intel math kernel library[EB/OL]. (2020)[2022]. https://software.intel.com/en-us/mkl.

[5] CORPORATION IBM, The mathematical acceleration

subsystem (MASS) library [EB/OL]. (2019)[2022]. https://www.ibm.com/docs/en.

[6] 刘畅, 武延军, 吴敬征,等. RISC-V指令集架构研究综述[J]. 软件学报, 2021,32(12): 3992-4024.
LIU C, WU Y J, WU J Z, et al. Survey on RISC-V system architecture research[J]. Journal of Software, 2021, 32(12): 3992-4024. (in Chinese)

[7] MPFR. The multiple precision floating-point reliable library[EB/OL]. (2021)[2022]. https://www.mpfr.org/.

[8] GLIBC. The GNU C library [EB/OL]. (2018)[2022]. http://www.gnu.org/software/libc/.

[9] 周蓓, 黄永忠, 许瑾晨, 等. 向量数学库的向量化方法研究[J]. 计算机科学, 2019, 46(1): 320-324.
ZHOU B, HUANG Y Z, XU J C, et al. Study on SIMD method of vector math library[J]. Computer Science, 2019, 46(1): 320-324. (in Chinese)

[10] DAVID P, ANDREW W. The RISC-V Reader[M]. Berkeley: Strawberry Canyon, 2017. 73-82.

[11] 曹浩, 郭绍忠, 刘聃, 等. 面向64位RISC-V的基础数学库自动化移植[J]. 计算机科学, 2021, 48(6): 41-47.
CAO H, GUO S Z, LIU D, et al. Automatic porting of basic mathematics library for 64-bit RISC-V[J]. Computer Science, 2021, 48(6): 41-47. (in Chinese)

[12] 曹代, 郭绍忠, 张辛. 基于申威26010处理器的扩展函数库实现与优化[J]. 计算机工程, 2017, 43(1): 61-66, 71.
CAO D, GUO S Z, ZHANG X. Implementation and optimization of extended function library based on SW26010 Processor[J]. Computer Engineering, 2017, 43(1): 61-66, 71. (in Chinese)

[13] CHAITIN G J. Register allocation and spilling via graph coloring[C]//Proceedings of the SIGPLAN Symposium on Compiler Construction-SIGPLAN'82. New York: ACM, 1982: 98-101.

[14] MASSIMILIANO P, VIVEK S. Linear scan register allocation [J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1999, 21(5): 895-913.

[15] 王向前, 王昊. 分簇结构向量寄存器分配策略研究[J]. 单片机与嵌入式系统应用, 2017, 17(7): 10-12.
WANG X Q, WANG H. Research on vector register allocation method for clustering[J]. Microcontrollers & Embedded Systems, 2017, 17(7): 10-12. (in Chinese)

[16] KRSTE A, ANDREW W. RISCV-v-spec-1.0[EB/OL]. (2021)[2022]. https://github.com/riscv/riscv-v-spec/releases/tag/v1.0.

[17] 夏军. 基于申威架构实现RISC-V浮点指令[D]. 合肥: 安徽大学, 2020.
XIA J. Implementation of RISC-V Floating Point Instructions Based on Shenwei Architecture[D]. Hefei: Anhui University, 2020. (in Chinese)

[18] FLORIAN G, LUKAS G, GUILLERMO P V. Evolutionary algorithms for instruction scheduling, operation merging, and register allocation in VLIW compilers[J]. Journal of Signal Processing Systems, 2020, 92(7): 655-678.

[19] LUK C K, MOWRY T C. Compiler-based prefetching for recursive data struchtures[C]//Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 1996: 222-233.

[20] 沈洁, 龙标, 姜浩, 等. 飞腾处理器上向量三角函数的设计实现与优化[J]. 计算机研究与发展, 2020, 57(12): 2610-2620.
SHEN J, LONG B, JIANG H, et al. Implementation and optimization of vector trigonometric functions on phytium processors[J]. Journal of Computer Research and Development, 2020, 57(12): 2610-2620. (in Chinese)

[21] 刘聃, 郭绍忠, 郝江伟, 等. 基于SIMD扩展部件的长向量超越函数实现方法[J]. 计算机科学, 2021, 48(6): 26-33.
LIU D, GUO S Z, HAO J W, et al. Implementation of transcendental functions on vectors based on SIMD extensions[J]. Computer Science, 2021, 48(6): 26-33. (in Chinese)

[22] 叶锡聪, 庄灿锋, 王宇木, 等. RISC-V向量指令集的Compute Library函数库移植[J]. 单片机与嵌入式系统应用, 2021, 21(1): 8-13.
YE X C, ZHUANG C F, WANG Y M, et al. Transplantation of compute library of RISC-V vector instruction set [J]. Microcontrollers & Embedded Systems, 2021, 21(1): 8-13. (in Chinese)

[23] 许瑾晨, 黄永忠, 郭绍忠, 等. 一个浮点数学函数库测试平台[J]. 软件学报, 2015, 26(6): 1306-1321.
XU J C, HUANG Y Z, GUO S Z, et al. Testing platform for floating mathematical function libraries[J]. Journal of Software, 2015, 26(6): 1306-1321. (in Chinese)

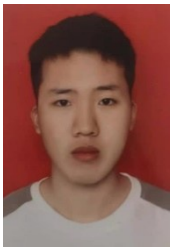
[24] SPIKE. The RISC-V ISA simulator[EB/OL]. (2017)[2022]. https://github.com/riscv/riscv-isa-sim.

[25] GEM5. The gem5 simulator[EB/OL]. (2016)[2022]. https://github.com/plctlab/plct-gem5.

[26] UBUNTU. The Ubuntu 20.04.3 LTS [EB/OL]. (2020)

[2022]. https://ubuntu.com/download/desktop.
[27] PALMER D. GNU toolchain for RISC-V[EB/OL]. (2015)[2022]. https://github.com/riscv-collab/riscv-gnu-toolchain.
[28] INTER. Inter one API math kernel library[EB/OL]. (2017)[2022]. https://software.intel.com/content/www/us/en/develop/documentation/onemkl-vmperfdata/top.html.

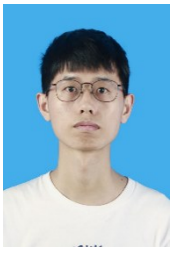
作者简介



李　飞　男,1996年出生,河南商丘人. 信息工程大学硕士研究生. 主要研究方向为高性能计算.

郭绍忠　女,1964年出生,安徽合肥人. 信息工程大学教授. 主要研究方向为高性能计算,分布式处理.

郝江伟　男,1995年出生,陕西清涧人. 信息工程大学博士研究生. 主要研究方向为高性能计算.



侯　明　男,1994年出生,周口太康人. 信息工程大学硕士研究生. 主要研究方向为高性能计算.

宋广辉　男,1997年出生,河南商丘人. 信息工程大学硕士研究生. 主要研究方向为高性能计算,先进编译技术.

许瑾晨　男,1987年出生,江苏泰州人. 信息工程大学讲师. 主要研究方向为高性能计算.