计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201200058

面向64位RISC-V的基础数学库自动化移植

曹 浩 郭绍忠 刘 聃 许瑾晨
数学工程与先进计算国家重点实验室(信息工程大学) 郑州 450002
(2227078931@qq.com)

摘 要 受制于核心技术和知识产权等客观条件,国产自主芯片的研发困难重重。RISC-V作为一个开源指令集架构(ISA),具有简洁、模块化等优点,成为了国产处理器的新选择。基础数学库作为计算机系统最基础的核心软件库之一,对国产处理器的软件生态建设和健康发展尤为重要,而目前RISC-V还没有相关的基础数学库。因此,文中旨在将基于国产申威处理器的基础数学库移植到64位RISC-V平台。为了解决基础数学库的高效移植问题,首先设计了一个自动化移植框架,该框架通过功能模块间的松耦合,来实现高可扩展性;然后根据64位RISC-V指令集架构的特点,提出了基于全局的主动式寄存器分配方法和基于层次的指令选择策略;最后应用该框架,实现了对申威平台基础数学库中典型函数的移植,测试结果表明移植后函数功能正确且相对于GLIBC库在性能上有一定的提升。

关键词:RISC-V;基础数学库;汇编;自动化移植
中图法分类号 TP313

Automatic Porting of Basic Mathematics Library for 64-bit RISC-V

CAO Hao,GUO Shao-zhong,LIU Dan and XU Jin-chen
State Key Laboratory of Mathematical Engineering and Advanced Computing,PLA Information Engineering University,Zhengzhou 450002,China

Abstract Subject to the core technology and intellectual property rights and other objective conditions,the research and development of domestic independent chip is highly restricted. RISC-V has the advantages of simplicity and modularity as an open source instruction set architecture(ISA),and it will become a new choice of domestic processor. As one of the most basic core software libraries of computer system,basic mathematics library is particularly important to the software ecological construction and healthy development of domestic processors. However,RISC-V has no relevant basic mathematics library at present. Therefore,this paper aims at porting basic mathematics library based on domestic Shenwei processor to the 64-bit RISC-V platform. In order to solve the problem of efficient transportation of the library,an automatic porting framework is designed at first,which can achieve high scalability through loose coupling between functional modules. Secondly,based on the characteristics of 64-bit RISC-V ISA,a global active register allocation method and a hierarchical instruction selection strategy are proposed. Finally,the framework is applied to bring about the transportation of some typical functions in the Shenwei basic mathematics library. Test results show that the ported functions are working correctly and the performance is improved compared with GLIBC.

Keywords RISC-V,Basic mathematic library,Assembler,Automatic porting

1 引言

处理器作为电子设备的中枢,是网络信息领域最基础的核心技术,因此发展国产处理器必须走自主可控之路。处理器的灵魂是指令集架构,RISC-V作为一个全新的指令集架构,具有开源、模块化、极简等特点,为实现国产自主可控的高性能CPU创造了良好的条件。高性能硬件离不开高性能软件的支撑,RISC-V当前的软件生态以开源社区为基础,以开发环境级的软件为主,如编译工具链、模拟器等。因此,要想推动RISC-V处理器的发展,大量应用软件,尤其是系统级应用的支撑必不可少。对于一个处理器来说,核心软件库是必选项。基础数学库作为计算机系统最基础的核心软件库之一,其精度、性能和可靠性都要为上层软件的开发和应用提供基础支撑。Intel、AMD、SPARC以及申威等平台都拥有自己的基础数学库,而RISC-V目前还没有。

为了加强RISC-V生态建设,亟须构建一个高性能的RISC-V基础数学库,因此本文通过移植稳定高效的申威平台基础数学库,将研究成果快速扩展到64位的RISC-V平台。但是,手工移植汇编程序的工作量极大且易出错,产品的开发周期也很长,此外,目前尚没有相关的自动化移植工具可以准

到稿日期:2020-12-07 返修日期:2021-03-22
基金项目:国家自然科学基金(61802434)
This work was supported by the National Natural Science Foundation of China(61802434).
通信作者:许瑾晨(atao728208@126.com)

确高效地将申威汇编代码移植到 RISC-V 平台。

因此，本文的主要工作是设计一个自动化移植框架，使得申威平台基础数学库通过移植就能实现在 64 位 RISC-V 环境上运行。该框架能够根据申威及 RISC-V 指令集架构的特点，在主要功能模块中分别应用有针对性的算法和策略，以达到准确且高性能的移植效果。

通过上述工作，本文取得以下两个方面的成果：

(1)完成了对申威平台基础数学库中典型超越函数和数值计算函数的移植后，移植后函数功能正确，且相比于 GLIBC 库其性能有所提升，目前已将这些函数开源[1)]。典型函数的成功移植，是移植整个基础数学库的坚实基础。

(2)按照高通用性、高可扩展性的标准设计了一个自动化移植框架，解决了基础数学库从申威平台到 RISC-V 平台的高效移植问题。该框架有巨大的应用潜力，经过简单地扩展，可支持任意指令集架构的汇编程序移植到 RISC-V。

2 相关工作

本文在基于申威处理器的国产基础数学库上有着丰富的研发经验，该库包括以超越函数和数值计算函数为主的共 150 多个函数，由大量的申威指令集汇编代码编写。汇编代码的移植，宏观上是一个指令集架构到另一个指令集架构的转换，微观上是指令到指令的映射。一条指令包含操作码和操作数，因此这种映射实质上是操作码和操作数的映射，通俗来说就是要解决指令的对应关系和寄存器的对应关系。

2.1 不同指令集的特点

复杂指令系统计算机(Complex Instruction Set Computer，CISC)有较强的处理高级语言的能力，但复杂的指令系统会带来结构的复杂性[1]。为了尽量简化计算机指令功能，强调计算机结构的简单性和高效性，精简指令系统计算机(RISC)应运而生，如 MIPS、SPARC、申威、RISC-V 等。RISC-V 诞生于近些年，而其他大多数指令集架构诞生于 20 世纪 70 年代或 80 年代，因此 RISC-V 对以往的指令集架构充分去粗取精，其短小精悍的架构以及模块化的哲学，使得 RISC-V 架构的指令数目非常简洁。基本的 RISC-V 指令数目仅有 40 多条，加上其他的模块化扩展指令后也只有几十条指令[2]。

2.2 传统的寄存器分配方法

寄存器分配的实质是通过映射，将源指令集架构的寄存器映射到目标指令集架构。其最终目的是有效地利用目标指令集架构的寄存器集合，最小化访存指令数量[3]。

在编译器优化中，寄存器分配(register allocation)是将大量的源程序变量分配到少量的目标 CPU 寄存器的过程。传统的寄存器分配方法有图着色法(graph coloring register allocation)和线性扫描法(linear scanning register allocation)[4-5]。这两种分配方法通常针对中间语言(Intermediate Language，IL)执行，而本文需要汇编指令之间的直接转换，因此这两种分配方法并不适用。

另一种思路是将源指令集架构的寄存器全部映射到内存中，寄存器仅作为中间件暂存数据[6]。其好处是不用考虑平台间寄存器的差异，实现起来更为简单。因此，这种基于内存映射的寄存器分配方法也被广泛采用，可以充分简化工程实现。但是，该分配方法涉及大量的内存读写操作，而内存操作相对耗时，可能导致移植后的应用性能较差。

2.3 经典的指令选择算法

将 IL 操作映射到目标机操作的过程称为指令选择(instruction selection)[7]。经典的指令选择算法有树模式匹配(tree-pattern matching)[8]和窥孔优化(peephole optimization)[8]。这两种指令选择算法是在编译优化时，将中间语言的操作映射到目标机操作[9]。但在本文面向的汇编程序中，新的指令序列是由汇编程序员在编写代码的过程中依托自动化平台指定的，该生成过程不依赖于编译器的指令选择算法，因此在生成指令的使用上，程序员拥有更大的自由度，可以更加充分地利用各种资源，达到减少溢出、加快指令序列执行速度的目的。因此，经典的指令选择算法并不适用于自动化移植框架。

3 自动化移植框架的组成

自动化移植框架旨在实现申威平台基础数学库汇编代码自动化静态翻译，以解决申威指令集架构到 64 位 RISC-V 指令集架构的自动化移植问题。

如图 1 所示，为满足框架对扩展性的需求，进行松耦合的模块化设计，其主要移植过程为寄存器分配(register allocation)、指令选择(instruction selection)两个步骤，并分别由两个主要功能模块具体实现。

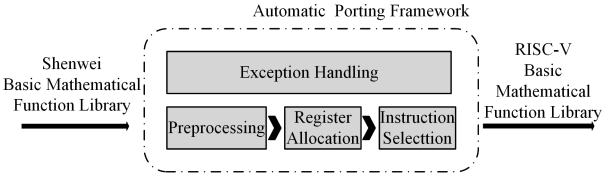


图 1 面向 64 位 RISC-V 的自动化移植框架组成示意图

Fig. 1 Schematic diagram of automatic porting framework for 64-bit RISC-V

其中寄存器分配模块的功能是将申威的寄存器映射到 RISC-V 的寄存器。寄存器的访问速度比内存快很多，因此尽可能地减少 Load/Store 指令数目，充分利用目标机的寄存器集合是实现高效的寄存器分配的基础。尤其对于高性能计算软件，如申威平台基础数学函数库，寄存器的使用需求量非常大，对于此类软件的移植，必须设计高效的算法来减少数据溢出(spilling out)[10]，其有利于目标代码执行效率的提高。

而指令选择模块的功能是将寄存器分配后的中间指令映射能成功一致的 RISC-V 指令，中间指令的结构及操作码仍然是申威的，而操作数是 RISC-V 的，因此本文依然将其视为申威指令。RISC-V 是追求 ISA 简洁性的精简指令集架构，因此通常是多条 RISC-V 指令对应一条申威指令。而指令选择模块可以保证新的指令序列在功能相同的情况下拥有较高的执行效率。

除了主要功能模块，自动化移植框架还包括预处理(preprocessing)和异常处理(exception handling)两个辅助模块，

1) https://github.com/mathlib-cn/riscv-mathlib

将协助主要模块共同完成移植工作。本文以 sin 函数的部分代码为例，阐述自动化移植框架的工作流程，如图 2 所示。

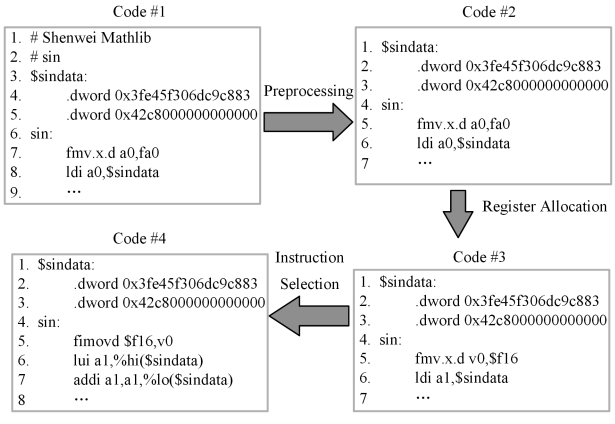


图 2　自动化移植框架流程示例

Fig. 2　Porting example of automatic framework

(1)预处理模块主要进行伪指令的转换、注释的删除、代码格式的调整和寄存器的统计，使处理后的汇编源程序满足后续处理的要求。Code＃1 是申威平台基础数学库函数 sin 函数的部分代码，作为输入经过预处理后生成 Code＃2。可以看到，预处理删去了 Code＃1 中的注释，并将申威伪指令“.dword”转换为功能相同的 RISC-V 伪指令“.quad”。

(2)寄存器分配模块将申威寄存器映射到 RISC-V 寄存器。Code＃2 通过寄存器分配生成 Code＃3，申威寄存器 a0 和 fa0 分别被替换成 RISC-V 的 v0 和 $f16。

(3)指令选择模块按照一定的规则用 RISC-V 的指令对寄存器分配后的申威指令做相应替换。Code＃3 通过指令选择生成 Code＃4，其中指令 fmv.x.d v0，$f16 替换成了 RISC-V 指令 fimovd $f16，v0，指令 ldi a1，$sindata 替换成了 Code＃4 中的第 6、第 7 行的 RISC-V 指令序列。

(4)异常处理模块主要处理自动化移植框架的异常情况。当其他 3 个模块出现不能处理的异常，如遇到无法识别的指令时，异常处理模块将会接管并做相应处理。因此，加入了异常处理模块后，有助于增强框架的鲁棒性，减轻程序员 debug 的工作量。

4　自动化移植框架的主要功能模块

按照自动化移植框架的设计，针对平台的具体特点，对各个模块进行了具体实现。下面对寄存器分配、指令选择这两个主要功能模块的设计与实现做具体介绍。预处理和异常处理这两个辅助模块的功能简单明确，本文不做具体描述。

4.1　寄存器分配

4.1.1　基于全局的主动式寄存器分配方法

相比内存，寄存器的访问速度要快得多，尽可能多地利用寄存器资源，减少访存的次数，是提高汇编程序执行效率的重要方式。不同的指令集架构之间寄存器的数量和用途有很大的区别。申威平台和 RISC-V 都是典型的精简指令集架构，而且都拥有包括整数寄存器和浮点寄存器在内的大量通用寄存器[11]。

传统的寄存器分配是被动的分配模式，变量先被分配到伪寄存器，再由编译器分配到物理寄存器，其更多地依靠编译器去实现。但是，在本文面向的 RISC-V 汇编程序中，程序员有更大的操作空间，可以自主分配寄存器和内存资源，以实现主动的寄存器分配。因此，本文提出基于全局的主动式寄存器分配方法(Global Active Register Allocation Method，GARA Method)，该方法从程序的整个生命周期出发，站在全局角度上统筹需要用到的资源，按照非保存寄存器和保存寄存器、内存的优先级顺序，主动进行寄存器分配，生成寄存器的映射关系。GARA Method 可以更加充分地利用寄存器资源，从而减少数据溢出。

其分配方法如图 3 所示。假定在整个程序的生命周期中，申威平台(SW)用到的寄存器数量为 M，M 的值由预处理模块统计得到，目标 64 位 RISC-V(RV)用到的寄存器数量为 N，N 的值根据需要由程序员定义，通常小于通用寄存器的总数 64，则：

(1)当 $M>N$ 时，在程序入口将 RV 的保存寄存器全部入栈保存，那么在程序的生命周期这些保存寄存器可以当作临时寄存器使用，程序出口再将其值恢复，接着将 SW 的 N 个寄存器映射到 RV 的寄存器上，最后开辟一段栈空间，专门用来模拟 SW 剩余的 $M-N$ 个寄存器。

(2)当 $M\leqslant N$ 时，如果非保存寄存器的数量满足映射需求，则按照非保存寄存器优先的原则将 SW 的 M 个寄存器映射到 RV 的寄存器上；如果非保存寄存器不够用，则将用得到的保存寄存器先入栈后再做映射。

```
1. # define SRC_ARCHITECTURE SW
2. # define DST_ARCHITECTURE RV
3. # define SRC_REGISTER_NUMBERS M
4. # define DST_REGISTER_NUMBERS N
5. Allocation Registers(ASM)
6. {
7.     if (M>N)
8.     {
9.       push_all_svRegs() //push all saved registers on the stack
10.      map(N)//map SW's registers to RV's registers
11.      spill(M-N)//spill left M-N registers to memory
12.    }
13.    else //M≤N
14.    {
15.      if(is_Callers_Enough())
16.      {
17.        map(M)//map SW's registers to RV's registers
18.      }
19.      else
20.      {
21.        push_needed_svRegs()
22.        //push needed saved registers on the stack
23.        map(M)//map SW's registers to RV's registers
24.      }
25.    }
26. }
```

图 3　基于全局的主动式寄存器分配方法

Fig. 3　GARA method

基于全局的主动式寄存器分配方法，既充分利用了有限的寄存器资源，又灵活利用了内存空间，解决了寄存器不足的问题。虽然当寄存器不足时会使用内存进行映射，但 64 位的 RISC-V 通用寄存器较多，往往不会借用太多内存来模拟寄存

器。该方法大大简化了寄存器的分配过程，使处理后的代码更为易读，同时不会损耗太多的性能。

4.1.2 寄存器分配模块的设计与实现

国产申威平台基于精简指令集架构，拥有大量的通用寄存器，且寄存器的用途和数量与 64 位的 RISC-V 相似。与 RISC-V 不同的是，申威平台的临时寄存器和参数寄存器更多，而保存寄存器比 64 位的 RISC-V 少。

因此，根据“通用寄存器优先”的原则，使用基于全局的寄存器分配方法，首先分配整数寄存器。如图 4 所示，除了保存寄存器，RISC-V 其他类型的寄存器不足。因此，将 RISC-V 剩余未使用的保存寄存器作为补充，用来给申威平台的临时/参数寄存器进行映射，其中 64 位 RISC-V 可用保存寄存器总数为 $ISsum$，则满足：

$$iss1 \cup iss2 \cup iss3 \cup iss4 = ISsum$$

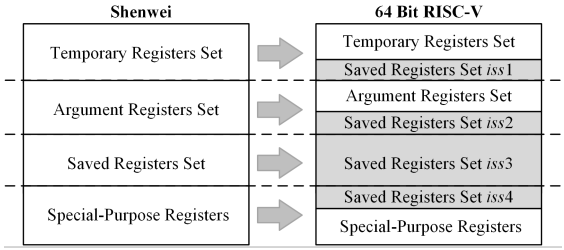


图 4 整数寄存器分配示意图

Fig. 4 Allocation of integer registers

不同于整数寄存器，两个平台的浮点寄存器都是通用寄存器，但是 RISC-V 的临时寄存器不足(见图 5)，将剩余可用的参数寄存器和保存寄存器作为补充，用来给申威平台的临时寄存器进行映射，其中 64 位 RISC-V 可用保存寄存器总数 $FSsum$ 和参数寄存器总数 $FAsum$，则满足：

$$fss1 \cup fss2 = FSsum \text{ 且 } fas1 \cup fas2 = FAsum$$

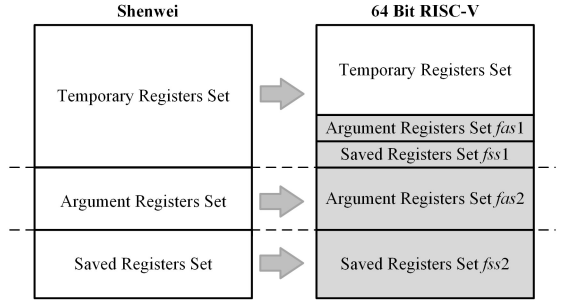


图 5 浮点寄存器分配示意图

Fig. 5 Allocation of floating point registers

4.2 指令选择

4.2.1 指令选择算法

为了追求简洁和可扩展性，RISC-V 指令集小而精，每条指令尽可能地完成基础操作。申威平台虽然也是精简指令集架构，但是在多年的发展中其指令集规模越来越大，许多单一指令能够完成复杂的操作。因此，对于申威指令集的一条指令，通常需要多条 RISC-V 指令来实现相同的功能，而且往往这个指令序列并不是唯一的。此外，指令序列的执行时间会受到操作码的类型、访存时数据在内存中的位置等情况的影响，针对如何才能在这些备选的指令序列中挑选出最优的方案，必须构建一种行之有效的指令选择策略来解决上述问题。

根据 RISC-V 指令集架构的特点，在进行指令选择时，为了尽可能最小化操作序列执行周期，本文采取一种基于层次的指令选择策略。

其流程如图 6 所示，该策略分为 3 层结构：

(1)在选择过程中，首先寻找与申威指令功能相同的 RISC-V 指令替换，如申威指令是加减乘除、平方根等，可以找到功能相同的 RISC-V 指令直接替换；

(2)如果不能用功能相同的指令直接替换，则优先使用编程语言辅助实现，如可以用编程语言来判断源指令的操作数在 RISC-V 环境下是否合法；

(3)最后若还是不能生成目标指令序列，则追加使用额外的寄存器来辅助实现，如 RISC-V 通常仅支持 12 位的立即数操作，而申威指令集支持的立即数操作大于 12 位，这时就需要借助额外的寄存器来辅助实现指定功能。

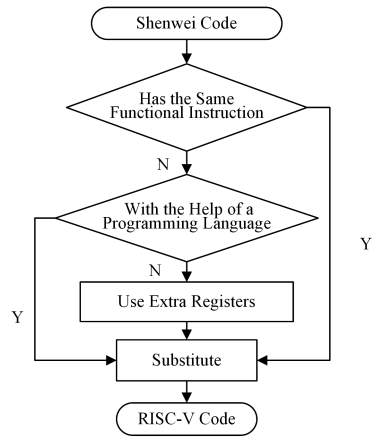


图 6 基于层次的指令选择策略

Fig. 6 Hierarchical instruction selection strategy

4.2.2 指令选择模块设计与实现

在指令选择模块中，应用基于层次的指令的选择策略，完成申威指令和 RISC-V 指令之间的转换。下面将列举 3 个典型的案例，分别对应逐层选择策略的 3 个层次。

(1)用功能相同的 RISC-V 指令直接替换申威指令。对于申威平台的移动 mov、加字 addl、减字 subl、双精度浮点乘 fmuld、双精度浮点减 fsubd 等指令，RISC-V 可以找到功能相同的指令来直接替换。如图 7 所示，按照 RISC-V 的指令标准，替换指令并改变相应操作数的位置，实现一对一的指令映射。

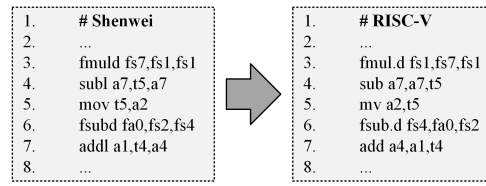


图 7 指令选择示例 1

Fig. 7 Instruction selection example 1

(2)编程语言辅助实现。RISC-V 的部分立即数操作通常只支持包含符号位扩展在内的 12 位，而申威平台对应指令的立即数操作大于 12 位。为了实现指令的高效转换，如图 8 所示，利用 Python 编写的 isLegal 函数对立即数的位数进行快速判断。如果不大于 12 位，则使用功能相同的指令直接替换；如果大于 12 位，则进入下一层次，通过追加寄存器来实

现。除此之外，对于一些较为复杂的位操作，也可以通过 Python 来辅助实现。

```
1.    # PYTHON
2.    def isLegal(strInput):
3.        if(strInput.find('0x')!=-1):
4.            if(-2048<=int(strInput,
              16)<=2047):
5.                return True
6.            else:return False
7.        elif(strInput.lstrip('-').isdigit()):
8.            if(-2048<=int(strInput)<=
              2047):
9.                return True
10.           else:return False
11.       else: return False
```

图 8　指令选择示例 2

Fig. 8　Instruction selection example 2

(3)追加寄存器辅助实现。如图 9 中箭头左侧所示，申威指令 ldi，将寄存器 s11 的值加上 17 666，将结果赋值给寄存器 s11。通过层次策略的前两层无法实现指令选择，此时就需要借助额外的寄存器间接实现转换。如图 9 中箭头右侧所示，立即数 17 666 首先存储到辅助寄存器 s8 上，RISC-V 通过 li 和 add 的指令组合，完成申威指令的 ldi 功能。s8 在使用完之后，将原先的值从内存中取回。

```
1.    # Shenwei
2.    ...
3.    ldi s11,17 666(s11)
4.    ...
```

```
1.    # RISC-V
2.    ...
3.    sd s8,80(sp)
4.    li s8,17 666
5.    add s11,s11,s8
6.    ld s8,80(sp)
7.    ...
```

图 9　指令选择示例 3

Fig. 9　Instruction selection example 3

5　移植函数的测试

应用自动化移植平台，对申威平台基础数学库的典型函数进行移植。借鉴申威平台基础数学库的测试经验，对移植后的函数进行正确性和性能的测试，以验证自动化移植框架的有效性。另外，测试的最终结果可以反向指导自动化移植平台中各种方法和策略的选择，为各模块的功能优化提供数据支撑。

5.1　测试函数

申威平台基础数学库共有 150 多个函数，以超越函数和数值计算函数为主，还包括少量特殊函数。超越函数和数值计算函数往往包括复杂的算法逻辑和指令序列，而特殊函数功能简单且指令序列简短。

因此，本文选取部分典型的超越函数和数值计算函数作为移植测试函数。这几个函数如表 1 所列，sin，cos，sinh，expm1 是超越函数，rint，ceil 是数值计算函数。表 1 中，○表示该函数包含此类指令，×表示不包含。这 6 个函数涵盖了转移指令(branch)、存储装入指令(load/store)，以及绝大部分的运算指令(arithmetic)。对于没有涵盖到的指令类型，如字节操作指令(byte)，可以通过“基于层次的指令选择策略”由 RISC-V 指令组合实现；对于读写 FPCR 指令(r/w fpcr)，可以由 RISC-V 寄存器控制指令实现；此外，申威基础数学库没有使用到系统调用指令(system call)、特权指令(privileged)和杂项指令(miscellaneous)。因此，选取的测试函数覆盖的指令类型较广，生成的目标函数包含的指令集扩展为 RV64IMFD，都具有代表性，其他同类型函数和特殊函数也能够通过类似的方法进行移植。

表 1　测试函数覆盖指令类型统计

Table 1　Statistics of correctness test

Function	Code Type															
	Basic Operations											Extended Operations				
	Arithmetic	Logical	Shift	Compare	Convert	Byte	Sign Copy	Register transfer	MAD/FMAD	Conditional Selection	R/W FPCR	Branh	Load/Store	System Call	Privileged	Miscellaneous
sin	○	○	○	○	○	×	○	○	○	○	×	○	○	×	×	×
cos	○	○	○	○	○	×	○	○	○	○	×	○	○	×	×	×
sinh	○	○	○	○	×	×	×	○	○	○	×	○	○	×	×	×
expm1	○	○	○	○	×	×	×	○	○	○	×	○	○	×	×	×
rint	○	○	○	○	×	×	○	○	×	×	×	○	○	×	×	×
ceil	○	○	○	○	×	×	○	○	×	×	×	○	○	×	×	×

5.2　测试样本

为确保测试集的完整性和有效性，参考基于 IEEE. 754 标准的全浮点域指数分布数据集生成方法[12]生成测试样本，该样本总共包含 8 380 416 个 64 位双精度浮点数，其浮点指数能够全部覆盖，最大限度地保证了测试浮点数的覆盖率。

5.3　测试环境

目前尚没有支持 Linux 的 64 位 RISC-V 硬件环境，因此使用软件环境进行模拟测试。如图 10 所示，最底层是 64 位 x86 操作系统，通过虚拟机 QEMU[13]安装支持 64 位 RISC-V 的 Fedora[14]，所有 RISC-V 应用将在 Fedora 上运行。

选择以 QEMU 为核心搭建的 RISC-V 测试环境，而不使用 RISC-V 基金会支持的指令模拟器 SPIKE[15]，原因如下：1)SPIKE 返回的时钟周期计数不准确，对测试结果的影响较大，但 QEMU 返回的计数值是准确的；2)QEMU 具备稳定的性能及接近实机速度的仿真能力，使最终的测试结果更具有参考价值。

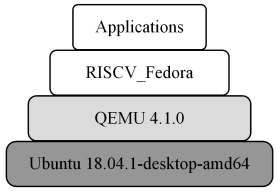


图 10　RISC-V 模拟环境

Fig. 10　RISC-V simulation environment

5.4　正确性测试

正确性测试通过与高精度 MPFR 库[16]的比较来实现[17]，流程如图 11 所示，将 MPFR 库的函数计算结果作为参考值，计算与移植函数得到的双精度 64 位结果的绝对误差，

如果绝对误差小于等于 1ULP(Unit in the Last Place),则可以认为结果正确。

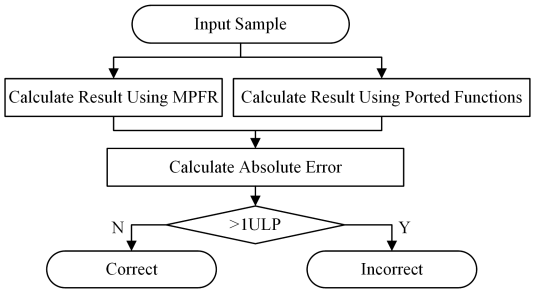


图 11 正确性测试流程

Fig. 11 Process of correctness test

对于 8380416 个输入,通过与 MPFR 库的计算结果进行比较,测试结果表 2 所列,移植函数绝大部分的比较结果小于 1ULP,小部分等于 1ULP,不存在大于 1ULP 的情况,因此可以认为通过该框架移植的函数结果是正确的。

表 2 正确性测试结果的统计

Table 2 Statistics of correctness test

(单位:%)

Functions	ULP		
	等于 1.0	小于 1.0	大于 1.0
sin	0.55	99.45	0.00
cos	0.55	99.45	0.00
sinh	0.00	100.00	0.00
expm1	0.00	100.00	0.00
rint	0.00	100.00	0.00
ceil	0.00	100.00	0.00

5.5 性能测试

5.5.1 内存读写性能

由于内存比寄存器的读写速度慢很多,因此基于全局的主动式寄存器分配方法的核心思路是尽可能地减少内存操作次数。本文将其与主流的基于内存映射的分配方法的内存读写次数进行比较,如图 12 所示,基于全局的主动式寄存器分配方法大量减少了内存读写比例,该现象在实现较为复杂的函数如 sin 和 cos 时更为明显。因此,基于全局的主动式寄存器分配方法非常有利于提升移植程序的性能。

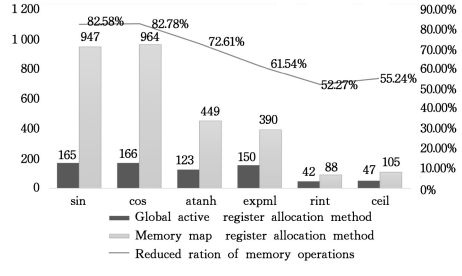


图 12 寄存器分配内存读写统计图

Fig. 12 Statistics of memory operations after register allocation

5.5.2 函数总体性能

函数总体性能的测试流程如图 13 所示。以 GLIBC[18] 数学库(GLIBC 2.27)为基准,计算移植函数相对于 GLIBC 数学库的平均加速比,测试结果如图 14 所示。以 GLIBC 库为基础可以看到通过该框架移植的数学函数中,除 expm1 提升不明显外,其他 5 个函数的提升效果明显。

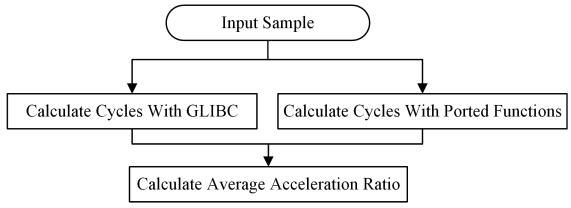


图 13 性能测试流程

Fig. 13 Process of performance test

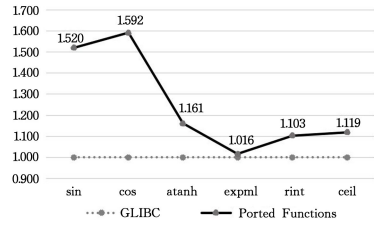


图 14 函数加速比统计图

Fig. 14 Statistics of acceleration ratio

5.6 小结

测试结果表明,移植后的函数功能准确,而且相比于 GLIBC 库其性能有所提升。可见通过寄存器分配和指令选择,确保了移植的正确性和一定的性能提升。虽然测试工作是在 RISC-V 的模拟环境下进行的,但是测试结果具有可参考、比较的意义,其验证了自动化移植框架的可用性,测试结果符合预期。下一步,将利用该框架完成国产申威平台基础数学库所有函数的移植工作。

结束语 本文设计了一个高可扩展性的自动化移植框架,实现了申威平台基础数学库到 RISC-V 的快速准确移植。根据申威和 RISC-V 的特点,提出了基于全局的主动式寄存器分配方法,充分利用有限的寄存器资源,并通过内存空间的灵活使用来解决寄存器不足的问题;提出了基于层次的指令选择策略,使源汇编指令映射到目标机操作的过程中,最小化操作序列执行周期。目前利用自动化移植框架成功实现了申威平台基础数学库中典型函数的移植。自动化移植框架虽然解决了申威平台基础数学库的移植问题,但其功能有很大的扩展空间。通过模块化的改进升级,其有潜力成为任意两个指令集架构有源汇编程序的自动化移植平台。

虽然基础数学库的移植工作已初见成效,但是后续的工作仍有待完成,自动化移植框架也有很大的扩展改进空间。1)当前只完成了典型函数的移植,验证了移植的可行性,下一步将通过自动化框架实现整个基础数学库的移植。2)目前工作的重心是确保移植函数功能正确,在此基础上虽然性能有一定的提升,但仍有改进空间。下一步,计划在自动化框架中加入指令调度模块,采用合适的指令调度算法重排指令序列让函数性能得到显著的提升。此外,寄存器分配、指令选择和指令调度是可以相互配合的,可通过共同作用来提高程序的执行效率,因此后续将对主要模块的功能算法和策略进行优化,在保证框架可扩展性的基础上加强模块间的协同工作。3)受限于缺少支持 Linux 的 64 位 RISC-V 硬件,因此当前工作主要在模拟环境中进行。在未来的相关条件具备时,将进

行在真实硬件环境下的验证实验。

参 考 文 献

[1] ISEN C, JOHN L K, JOHN E. A Tale of Two Processors: Revisiting the RISC-CISC Debate [C] // Computer Performance Evaluation and Benchmarking. Berlin, Heidelberg: Springer, 2009.

[2] The RISC-V Instruction Set Manual. Volume I: User-Level ISA; Volume II: Privileged Architecture [EB/OL]. https // riscv.org/specifications/. 238 pages. 2019.

[3] CHAITIN G. Register allocation and spilling via graph coloring [C] // ACM. 2004: 66-74.

[4] DAS D, AHMAD S A, VENKATARAMANAN K. Deep Learning-based Hybrid Graph-Coloring Algorithm for Register Allocation [EB/OL]. [2020-01-16] https // www. researchgate. net.

[5] POLETTO M, SARKAR V. Linear scan register allocation [J]. ACM Transactions on Programming Languages and Systems, 1999, 21(5): 895-913.

[6] CAO H Y, ZHANG Y. Highly Portable Light-Weight x86 Emulator [J]. Computer Systems & Applications, 2011, 20(5): 101-104, 143.

[7] BLINDELL G H, CARLSSON M, LOZANO R C, et al. Complete and Practical Universal Instruction Selection [J]. Acm Transactions on Embedded Computing Systems, 2017, 16(5s): 1-18.

[8] COOPER K D, TORCZON L. Engineering a Compiler, Second Edition [M]. USA: Morgan Kaufmann, 2003.

[9] SURHONE L M, TENNOE M T, HENSSONOW S F. Instruction Selection [M]. Switzerland: Springer International Publishing, 2016.

[10] GUO Z H, GUO S Z. Register Allocation Strategy for Hierarchy Structure in Base Mathematics Library [J]. Computer Engineering, 2012, 38(24): 266-268.

[11] PATTERSON D, WATERMAN A. The RISC-V Reader [M]. USA: Strawberry Canyon, 2017.

[12] XU J C, HUANG Y Z, GUO S Z. Testing Platform for Floating Mathematical Function Libraries [J]. Journal of Software, 2015, 26(6): 1306-1321.

[13] QEMU. A generic and open source machine emulator and virtualizer [CP/OL]. [2020-01-16], https // www. qemu. org/.

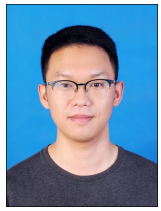
[14] FEDORA/RISC-V, a complete Fedora experience on the RISC - V (64 bit, RV64GC) architecture [CP/OL]. [2020-01-16], https // fedoraproject. org/wiki/Architectures/RISC-V.

[15] Spike, the RISC-V ISA Simulator [CP/OL]. [2020-01-16]. https // github. com/riscv/riscv-isa-sim.

[16] MPFR, the Multiple Precision Floating-Point Reliable Library [CP/OL]. [2020-01-16], https // www. mpfr. org/.

[17] QI H Y, XU J N, GUO S Z. Detection of the maximum error of mathematical functions [J]. Supercomput, 2018, 74: 6275-6290.

[18] GLIBC. The GNU C Library [CP/OL]. [2020-01-16]. http // www. gnu. org/software/libc/.



CAO Hao, born in 1991, postgraduate. His main research interests include high performance computing and so on.

XU Jin-chen, born in 1987, Ph.D, lecturer, is a member of China Computer Federation. His main research interests include high performance computing.