软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(6):1306−1321 [doi: 10.13328/j.cnki.jos.004589]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一个浮点数学函数库测试平台*

许瑾晨, 黄永忠, 郭绍忠, 周 蓓, 赵 捷

(数学工程与先进计算国家重点实验室(解放军信息工程大学),河南 郑州 450002)

通讯作者: 许瑾晨, E-mail: atao728208@126.com

摘 要: 数学函数库作为 CPU 软件的重要组成部分,对于高性能计算机平台上的科学计算、工程数值计算起着极为关键的作用.现有的测试工具只能片面地对函数库进行测试,没有从正确性、精度和函数性能这 3 方面加以考虑,而且往往只针对一类目标体系结构,适用性有限.针对现有测试工具的缺陷,提出了面向多目标体系结构、全面可复用的一体化测试平台 BMltest(basic math library test).测试平台结合函数特征值、IEEE-754 特殊数以及利用浮点数生成规则实现的全浮点域指数分布的 IEEE-754 规范数构造了测试集,有效提高了测试集浮点数的覆盖率;提出了基于多精度库 MPFR(multiple-precision floating-point reliable library)的精度测试方法,提高了精度测试的可靠性;提出了基于代码隔离的性能测试方法,最大限度地降低了外部环境对性能测试的干扰.针对大量的浮点测试结果,给出了合理的结果评价方案.测试平台使用的测试集数据与函数做到了相关性的极大分离,保证了测试方法的普适性.通过对包括 GNU,Open64 及 Mlib 函数库内所有 855 个函数的测试结果表明:BMltest 平台的测试数据集更全面、有效,精度测试方法更可靠;与其他测试平台相比,性能测试结果更准确、稳定.

关键词: 数学函数库;测试平台;IEEE-754;精度测试;性能测试

中图法分类号: TP311

中文引用格式: 许瑾晨,黄永忠,郭绍忠,周蓓,赵捷.一个浮点数学函数库测试平台.软件学报,2015,26(6):1306−1321. http://www.jos.org.cn/1000-9825/4589.htm

英文引用格式: Xu JC, Huang YZ, Guo SZ, Zhou B, Zhao J. Testing platform for floating mathematical function libraries. Ruan Jian Xue Bao/Journal of Software, 2015,26(6):1306−1321 (in Chinese). http://www.jos.org.cn/1000-9825/4589.htm

Testing Platform for Floating Mathematical Function Libraries

XU Jin-Chen, HUANG Yong-Zhong, GUO Shao-Zhong, ZHOU Bei, ZHAO Jie

(State Key Laboratory of Mathematical Engineering and Advanced Computing (PLA Information Engineering University), Zhengzhou 450002, China)

Abstract: As one of the most important essentials of CPU, mathematical function libraries play a key role in scientific and engineering computing with high performance computers. Existing testing techniques and platforms can only evaluate function libraries from one or two aspects, therefore are unable to provide an evaluation result as a whole picture. Consequently, they are applicable for a specific targeting architecture and the scalability is restricted. To address this problem, this study proposes a novel testing platform BMltest (Basic math library test). It constructs the testing suite, which is composed of eigenvalues, IEEE-754 special values and IEEE-754 normalized values, to improve the cover rate of the floating numbers. A MPFR (multiple-precision floating-point reliable library) based precision test is introduced, and as a result, the reliability is improved. A code isolation based performance test is also described, so as to further eliminate the impact from enclosing circumstance. Some practical evaluating strategies are proposed to evaluate the test result. Such design makes the testing suite not correlated to mathematical functions, thereby ensuring the applicability. The experimental results show that, by testing 855 functions from various libraries, including GNU, Open64 and Mlib, the testing suite provided by BMltest is more

* 基金项目: 国家高技术研究发展计划(863)(2009AA012201)
收稿时间: 2013-04-25; 修改时间: 2013-12-09; 定稿时间: 2014-02-17

efficient and the precision test is more reliable. At the same time, compared with those of other testing platforms, the performance test is more stable.

Key words: mathematical function library; testing platforms; IEEE-754; precision test; performance test

当前,高性能计算机主要应用于航空航天、科学与工程计算、国防工业等领域,这些应用都会伴随超大数量级的数据处理、事务处理和信息服务,其中不乏对超大数量级或超高精度的浮点数据的处理,整个过程需要快速而精确的计算[1],而此类计算都需要数学函数库的支持.

数学函数库作为 CPU 软件的重要组成部分,是高性能计算机平台上科学计算、工程数值计算以及相关专用软件开发所必备的最基础、最核心的软件之一[2],主要包括三角类函数、指数类函数等初等函数[3−5].近些年来,学术界和工业界面向数学函数库的算法都展开了大量研究[6−11].虽然面向数学函数库算法的研究一直在延续,相应的应用也在不断面世[12−19],但在满足面向高性能计算领域的高可靠、高精度和高性能需求时,数学函数库的实现依然存在一些困难和挑战:

1. 浮点数的离散性

实数可以用来衡量连续变化的量.理论上,任何实数都可以用无限小数的方式表示,小数点右边是一个无穷数列.但在实际应用中,实数经常被近似成有限小数,尤其在计算机领域,由于计算机只能存储有限的小数位数,实数经常通过浮点数表示.

目前,浮点数都遵循由 KCS 组合(Kahn, Coonan and Stone)于 1985 年为 Intel 开发,并被 IEEE 组织采用的 IEEE-754 标准[20],该标准被大多数计算机支持,并从逻辑上用三元组{*s*,*e*,*m*}表示一个数 *n*,即:

$$n=(-1)^s\times m\times 2^e.$$

其中,

- *s*(sign)表示符号位,满足:*n*>0 时,*s*=0;*n*<0 时,*s*=1;
- *e*(exponent)表示指数位,位于 *s* 和 *m* 之间的若干位;
- *m*(mantissa)表示尾数位,位于 *n* 的末尾,*m* 也被称为有效数字位(significant)、系数位(coefficient),甚至“小数”.

这样的表示形式决定了浮点数在数轴上无法连续出现,呈现离散状态,即:实数无法和浮点数完全对应,计算的真实值可能需要通过舍入并以机器可表示的浮点数形式出现.IEEE-754 标准定义有 4 种舍入方式:就近舍入、向负无穷大舍入、向“0”舍入和向正无穷大舍入.在进行运算时,需要根据算法特性及函数功能选择适合的舍入模式.但无论何种舍入方式,都只能尽量减小舍入带来的误差,无法避免误差.这种数的近似表示给函数高精度要求带来了实现的困难.

2. 运算的局限性

数学函数库函数一般为初等函数,只能运用最基本的运算实现,所以数学库的函数实现和处理器的指令集密不可分,对于不同的指令集,相同函数的实现可能不同.同时,只能利用加减乘除、移位、跳转等简单的指令实现复杂函数算法,其难度是显而易见的.

3. 近似求解的计算方式

当前,函数计算的主要方法有级数法(多项式近似法)、迭代法、查表法[21]、有理数逼近法、逐位法[22]、CORDIC 法[23]等.无论是具有计算速度快这样优势的级数法和查表法,还是具有计算精度高这样特点的迭代法,都无可避免地存在无法精确计算的问题,都属于近似求解方法.虽然针对这一问题,在误差控制方面有较成熟的研究,如区间分析在浮点误差方面的应用[24]等,但这类研究并不能彻底解决近似求解问题,浮点误差可能带来的传递依然会存在,对于复杂计算问题甚至可能出现误差爆炸现象[25].

4. 实现语言的选择

数学函数库的实现通常要与具体应用平台挂钩,可以通过 C 语言,Fortran 语言等高级语言实现,也可以由汇编语言实现.采用汇编语言实现,更能满足函数的高性能要求,但也给实现带来了难度,需要编程人员了解更多底层体系结构的设计和对体系结构上汇编语言的熟练掌握,同时给后续函数分析加大了难度.

上述数学函数库实现过程中面临的困难,同样也是对数学函数库进行测试所面临的困难.虽然有学者认为,敏感的数值计算可以通过使用不同的算法或编码技术实现[26],但浮点运算在实现过程中非常容易出错,而且这样的操作需要耗费大量的人力和财力.而采用更高精度的算法是保证计算正确性的更容易、更便宜和更可靠的方法[26],但要验证计算(包括高精度算法)的正确性,需要大量可靠的测试才能完成.而相对于数学函数库算法和实现的发展,测试技术的研究却显得较为缺乏.

当前的测试研究可以分为两类:

一类是利用形式化方法验证函数正确性的研究,如 Florian Benz 等人利用二进制翻译技术实现的动态程序分析方法[27];Boldo[28],Akbarpour[29]等人对初等函数正确性的证明;Coq[30−32]和 Gappa[33,34]工具在程序验证方面的应用等.这类方法理论性强,但实现困难.

另一类是针对浮点计算或部分函数实现的测试工具,如测试基于 C 语言或 Java 语言实现的部分初等函数的 ELEFUNT 工具[35];测试基于 C 语言和 Fortran 语言实现的基本运算和部分初等函数的 UCBTEST 工具[36];Berkeley 大学研发的浮点测试工具[37],Kahan 实现的 floating-point benchmark Paranoia 等[38,39].这些测试工具都只针对正确性和精度进行测试,属于 Cody[40]对精度测试技术的分类:基于表驱动技术、基于更高精度和基于预选恒等式的方法.此外,如 Olofsson[41]提出的针对 GNU 函数库实现的针对性测试,也同样存在应用的局限性,无法直接对不同的函数库进行测试.目前,大量用于测试函数技术指标的测试工具的测试并不全面;同时,基于本文的了解,在性能测试方面并没有发现相关的数学函数库测试工具.

针对上述测试方法普遍存在的应用局限性问题以及理论验证在实际应用中的困难,研究如何对不同目标体系结构的各种类型的库函数进行正确性测试、精度测试和性能测试,提供一体化的综合测试平台,显得尤为重要.鉴于此,本文实现了一个浮点数学函数库的一体化测试平台 BMltest,并给出了具体的测试方法及测试结果评价方案,主要贡献如下:

1. 当前,测试技术的研究都集中在精度测试方面,忽略了对性能测试的研究,而函数库在科学计算中的应用要求是高精度和高性能并存.如何准确有效地对函数性能进行测试,摆脱测试过程中的性能干扰,也是重要的问题.BMltest 平台在实现精度测试的基础上实现了相应的性能测试方法,确保了测试的完整性,形成了全面统一的测试平台.该平台可提供模块化可定制的测试流程,可根据需求对测试流程中的基本算法、数据集、结果评价等模块进行修改或扩充,增强了测试的可扩展性和可复用性;
2. 基于 IEEE-754 标准生成了符合浮点数分布的、规范化的全浮点域指数分布测试数据集,有效提高了精度测试中浮点数的覆盖率;同时,测试集的数据与函数做到了相关性的极大分离,保证了测试方法的适用性(可应用于不同语言不同体系结构的各类数学库);
3. 提出了基于多精度库 MPFR 的精度测试方法和基于代码隔离的性能测试方法,提高了精度测试的可靠性和性能测试的准确性;
4. 针对 BMltest 平台的测试结果,给出了基于雷达图的精度测试结果分析方法和基于 4D 检测的性能测试结果分析方法,以全面直观地衡量函数库运行指标,使整个测试更为完整,适用性更强.

本文第 1 节对一体化测试平台的测试框架进行阐述,并引出研究过程中的 4 点需要解决的问题.针对这 4 种问题,第 2 节对新的数据集进行详细说明,重点阐述基于 IEEE-754 浮点数分布的全浮点域指数分布测试集的生成方法.第 3 节给出具体的测试方法.针对其测试结果在第 4 节给出合理有效的结果评价方案.通过第 5 节的实验测试,分别验证新的数据集的有效性、精度测试方法的可靠性以及性能测试方法的准确性.最后,利用第 6 节对全文工作进行总结分析对比.

1 测试平台框架

本文研究的数学函数库测试平台的功能模块框图如图 1 所示.

该测试平台由待测函数集、测试程序生成模块、函数特征库、测试程序模板库、测试数据收集与分析模块和测试结果存储显示模块组成,主要模块功能如下:

1) 测试程序生成模块:结合测试程序模板库及函数特征库中的函数信息自动生成测试程序.该模块以待测函数集中的函数名称为基准,在函数特征库和测试程序模板库中分别查找相应的函数接口和测试程序模板,并生成相应的测试程序;
2) 测试数据收集与分析模块:统计测试程序生成的测试数据,根据测试内容,分别采用雷达图对精度测试结果、4D 检测法对性能测试结果进行分析;
3) 测试结果存储显示模块:对分析结果进行再整理,以易于观察和理解的方式呈现给程序员.

其中,函数特征库中的函数信息包括简单的显性特征,如函数输入/输出参数个数、参数类型、函数定义域,同时也包括相对复杂的隐性特征,如函数分支判断点、计算拐点、计算极值点、函数周期、关键路径区间.其中,函数分支判断点和关键路径区间的确定尤为困难,同时也是函数特征值中比较重要的部分.对函数代码进行静态控制流程图分析,是获取路径执行特点的有效方式,控制流程图能够清晰地描述出函数的分支判断条件、路径数目、每条路径的进入参数范围等特点.根据大量的数据测试以及对函数算法及执行指令的分析,可获取程序的关键路径,这方面也有很多研究,如王璐璐提出的全路径剖析方法[44]等.

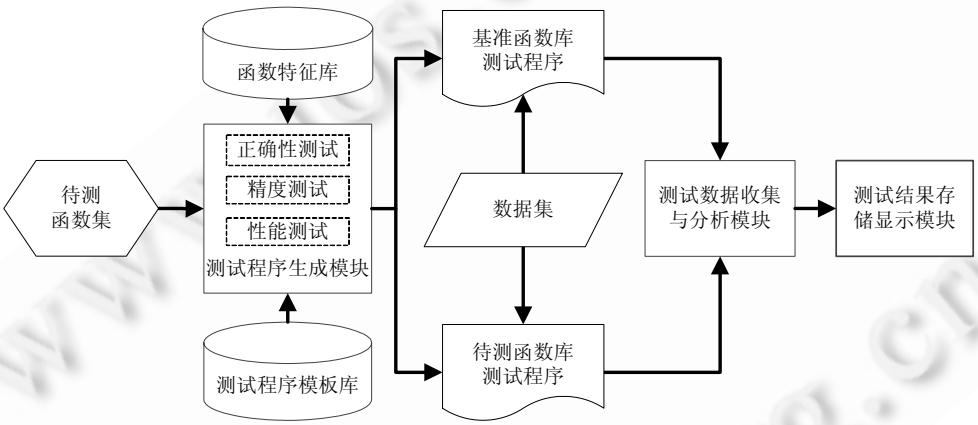


Fig.1 Prototype of the BMltest

图 1 测试平台程序功能模块框图

测试流程如下:

(1) 准备阶段:完成测试集和测试程序的生成.从数学函数库中选择待测函数,并从函数特征库中提取相应的函数特征,结合测试程序模板库及测试程序生成模块自动生成测试程序.与此同时,利用数据集生成方法生成测试数据集;
(2) 测试阶段:完成对基准函数库和被测函数库中函数的测试;
(3) 分析阶段:收集基准函数库和被测函数库测试程序生成的测试数据,利用测试结果分析模块进行测试结果分析.

在测试平台研究过程中,需要重点考虑几个问题:在测试数据集的生成方面,如何保证测试集的有效性(测试集中浮点数的覆盖率问题)?在精度测试方面,如何保证测试的可靠性(测试方法的选择问题)?在性能测试方面,如何保证测试的准确性及稳定性(测试过程中外部环境的干扰问题)?针对大量的浮点测试结果,采用何种分析方法更为合理,更直观?上述问题所对应的解决方案的好坏将直接影响函数测试结果的正确与否.为了有效解决上述问题,本文将从测试集的生成、测试方法的实现和测试结果的评价 3 个方面进行研究.

2 测试数据集

测试数据集是浮点测试的核心,测试数据集的合理性往往直接影响到测试的效果,甚至是测试正确性[42],是

函数正确性测试的基础.现有方法能够生成多种多样的测试数据集,但都无法直接满足数学库函数的测试.本文针对数学函数库函数分支多且分支判断点难以获取的特点,设计实现了由 IEEE-754 特殊数、函数特征值及 IEEE-754 规范数这 3 部分组成的浮点数据测试集.

IEEE-754 特殊数(包括非数(QnaN,SNaN)、无穷数(Inf)和弱规范数(subnormal))的测试能够很好地反映函数的健壮性和对异常数的处理能力.函数特征值(既包括简单的显性特征,如参数类型、函数定义域等,也包括相对复杂的隐性特征,如函数分支判断点、计算拐点、计算极值点、函数周期、关键路径区间等)的测试能够很好地反映函数的相关特性和极端状态下的运算状态,通常可以由静态或动态程序分析[43]获得.两者都是测试数据集的重要组成部分,同时,其生成方法也比较成熟,能够有效满足数学函数库的测试,本文不作详述.

关于数据集中浮点数的覆盖率问题,理论上存在好的方法可以解决,如穷尽浮点域中的所有浮点数或分析获得全面的包含分支判断点的函数特征值,但这些方法只适于理论分析,在现实中难以实现.对于第 1 种方法,首先无法在现实中穷尽所有的浮点数,其次就算只关注机器可表示的浮点数,对于双精度形式也存在 2^{64} 个有限数,在实际应用中也难以实现;对于第 2 种方法,对于由汇编语言实现的高性能数学函数库而言,函数特征值中的分支判断点难以获取.再者,每一个函数的实现算法、语言、运行所在的体系结构等因素的变化都可能导致分支判断点的变化,即使分析获得分支判断点,也不具有普遍的适用性.

为了解决上述问题,本文深入研究浮点数的生成规则,摒弃测试数据全浮点域穷举(数据量大,实际应用中无法实现)及均匀随机(浮点数的实际覆盖率低)的方法,设计实现了一种新的可应用于实际的符合浮点数设计规则的 IEEE-754 规范数生成方法,提出了基于 IEEE-754 标准的全浮点域指数分布数据集生成方法,确保了测试数据集的有效性,提高了浮点数的覆盖率.

计算机中的浮点数一般严格以 IEEE-754 标准形式出现,将所有可在数轴上精确表示的数表示出来,可以发现:离零点越近,数越密集,可表示的两个浮点数之间的距离就越小;离零点越远,数越稀疏,可表示的两个浮点数之间的距离也就越大,如图 2 所示.

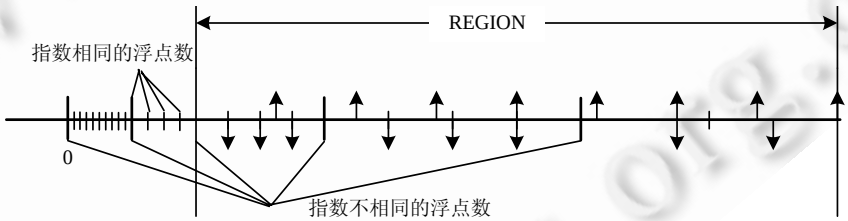


Fig.2 IEEE-754 floating numbers distribution and the data generation strategy

图 2 IEEE-754 浮点数分布及数据生成示意

假设针对某一函数在 REGION 区间内测试,在一般的测试方法中,倾向于在此区间内均匀生成测试数据,即,所生成的测试数据在此区间的分布呈均匀分布,如图 2 坐标轴上带有向上箭头的数.这种生成数据集的方法简单,但没有考虑到浮点数的分布,在浮点数密集区域的浮点数覆盖率低,可能导致浮点运算的测试结果与真实情况不符.

为了避免这一现象,从生成浮点数基本原理出发,将浮点测试数据集的生成方法也分为 3 个部分:符号位、指数和尾数.在待测区间内,针对 IEEE-754 标准中的指数的异同,将测试数据集划分为不同的区域,在每个指数相同的区域中再产生 0.5~1 之间的 N 个尾数,将产生的尾数与对应的指数构成测试数据集.其生成规则:(1) 符号位,根据测试区间,判断测试数据的正负属性;(2) 指数部分,覆盖测试区间内所有浮点数的指数;(3) 尾数部分,针对每一个指数值,产生 N 个 0.5~1 之间的均匀分布的随机数.具体到图 2 所示测试情况,测试集符号位为 1,指数部分包括 3 种不同的指数,在不同指数域内部再均匀生成 3 个尾数,构成完整的测试集(图中坐标轴上带有向下箭头的数).

这样,从指数相同的子区域来看,区域内部呈均匀分布;但是从整个测试区间来看,测试数据集呈指数分布.这种分布与 IEEE-754 的浮点数理论分布保持一致.这意味着,数轴区间上的浮点数越密集,按照此规则所生成的测试数据也就越多;数轴区间上的浮点数越稀疏,生成的测试数据越少,有效避免了测试用例在数值极小和极大情况下不平均的情况.同时,由于引起函数跳转的参数通常以浮点数指数进行区分(指数部分通常就决定了数的数量级和整数部分),测试区间内浮点指数的全覆盖,即,最大限度地保证了测试浮点数的覆盖率.

在尾数生成过程中,生成区间选择 0.5~1 而不是 0~1,这是由浮点数表示规则决定的.在 IEEE-754 规范中,浮点数需要规格化表示,而规格化的浮点数要求尾数的绝对值应大于或等于 0.5.为了使单独生成的指数和尾数可以直接组合成规格化的浮点数,避免规格化过程中尾数移位对指数的影响,将尾数生成区间限定在 0.5~1.

现有测试方法的均匀分布数据集通常采用将测试区间分成均匀的 N 等份的方法生成测试数据集,相比较而言,本文测试数据集在生成过程中考虑到了浮点数的具体分布状况,将生成过程分成符号位、指数及尾数等 3 部分实现,更加适应浮点函数测试的需要.

上述 3 个部分构成完整的测试数据集,保证了测试集的完整性(包括 IEEE-754 规范数、IEEE-754 特殊数及函数特征值)和有效性,为下一步测试做好了充分的数据准备.

3 测试方法

3.1 基于多精度库MPFR的精度测试方法

在详述精度测试方法之前,有必要先了解一下 ULP(unit at the last place).ULP 是浮点数最后一位的单位长度,比较有代表性的说法是:对于一个浮点数的 ULP 值,其对应的值是数轴上所能表示的这个数的两个浮点数的差值[45].一个浮点数在数轴上很可能不被精确表示,而需要对这个数进行舍入才能在数轴上找到其对应的符合 IEEE-754 浮点数规格的浮点数表示,那么这两个数轴上的规格浮点数之间的距离就被称为 ULP.图 3 表示了 ULP 的具体含义.

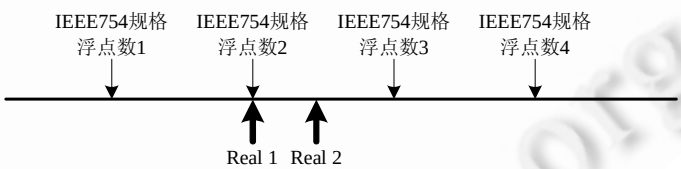


Fig.3 ULP

图 3 ULP 示意图

在图 3 中:

- 实数 real 1 能够用浮点数 2 精确表示,此时,ULP 的值为其到两侧浮点数 1 或者浮点数 3 的距离;
- 实数 real 2 在数轴上处于浮点数 2 和浮点数 3 之间,其 ULP 值即为浮点数 2 和浮点数 3 之差;同时,real 2 根据 IEEE-754 舍入方式表示为两个数中的一个.

目前,传统的精度测试工具[27−29]中的测试方法大多采用近似求解的方法计算函数真值,且计算精度一般限定在单精度或双精度范围内.这类方法计算速度快、测试时间短,但函数计算值的精度受限,无法精细化地计算函数实际精度情况,只能以 0ULP,1ULP,2ULP,…这样的整数形式表示函数精度(如第 5.2 节中的精度测试结果).这类方法计算的函数真实值与被测函数的精度相同(以双精度为例,都精确计算 53 位尾数,以下都以双精度为例),需要在第 53 位进行舍入处理等浮点操作,这就可能造成函数真实值的精度损失,导致测试误差.为了解决上述函数真实值精度损失的问题,本文引入多精度函数库 MPFR 进行函数真实值计算.MPFR 作为参照数学库[44],库中基本包括了 300 个以上的基础数学函数,并且在理论上提供对任意精度浮点函数的支持,利用此多精度库进行函数计算的唯一限制在于体系结构的内存大小,只要内存资源足够,可以支持任意精度的浮点函数计算.

引入多精度函数库进行函数真实值计算,提高了精度测试的准确性,因为在测试过程中获得了没有精度损

失的函数真实值(保证53位尾数正确).但不可否认,此方法带来了新的问题:增加了测试时间,同时消耗了更多的存储空间.为了有效缓解基于多精度库的测试方法运行开销较大的问题,本文在精度测试过程中对函数真实值所需精度进行判断,以找到能够保证53位尾数正确的最小计算精度.通过测试分析,对于sin,cos,exp和log等三角函数、指数对数函数来说,该最小精度为60位(采用多精度库进行计算的过程中,精度达到60即可保证函数真实值的53位正确,没有精度损失);而对于数值类函数,则最小精度为53位.这样的处理使得计算的每一位函数真值都发挥了作用,没有冗余位,减少了测试时间和内存消耗,其具体实现算法如下:

//功能:实现对精确值计算精度 P 的判断,并计算函数精确值

//输入:测试数据集,被测函数,M=80;

//输出:精度位数 P 及函数精确值计算结果.

1. $mpfr\_inits_2(53,x_1,\ldots)$; $mpfr\_inits_2(M,x_0,\ldots)$; $p[M-53]=2$; //初始化
2. $mpfr\_FUNC(y_0,x_0,MPFR\_RNDN)$; //计算精度为 M 的函数计算值
3. $i=53$, $j=M$, $K=(M-53)/2+53$;
4. **while** (K!==M & K!==53) **do**
5. $mpfr\_inits_2(K,x_2,\ldots)$;
6. $mpfr\_FUNC(y_2,x_2,MPFR\_RNDN)$;
7. $p[K-53]=REULP(y_0,y_2)$; //测试精度为 K 的函数计算值的误差
8. **if** ($p[K-53]$==0) {$K=j$; $K=(K-i)/2+i$;}
9. **else** {$K=i$; $K=(j-K)/2+i$;}
10. **end**
11. **if** (K==53) {
12. $mpfr\_FUNC(y_1,x_1,MPFR\_RNDN)$;
13. $P[0]=REULP(y_0,y_1)$;
14. **if** ($p[0]$=0) P=53;
15. **else** P=54;}
16. $P=KMP(p[K-53],$‘10’)+54; //将数组 $p[\cdot]$看作字符串,并利用字符串匹配算法 KMP 搜索子串 10(二进制)的位置
17. **end**

其中,$REULP(\cdot)$函数为误差测试函数,返回值为0或1:0表示当前精度下计算的函数值的前53位完全正确,1表示前53位存在误差,有精度损失.

该算法首先选取一个较大的精度值 M 作为参照(M 应大于等于53,在此,选取经验值80作为 M 的初值),再利用二分法思想查找引起函数计算值前53位误差发生变化的精度值.其中,变化表现为相邻两个精度值的误差由1变为0,即算法步骤16中10的由来.

在此基础上,本文精度测试采用如下的计算过程,并以相对 ULP 作为衡量数学函数库与MPFR库计算结果差异的标准:

$$relativeULP_{Mlib}=\frac{|Function(x)_{Mlib}-Function(x)_{mpfr}|}{ulp_{mpfr}},$$

其中,$Function(x)_{Milb}$ 代表被测函数库中函数的计算结果,$Function(x)_{mpfr}$ 代表此函数在MPFR上的计算结果,ulp_{mpfr} 代表MPFR上的计算结果的 ULP 值.以相对 ULP 值的方式衡量两者的差值具有两个方面的好处:一是其不存在绝对误差并不能完全地表示近似值的好坏程度的问题,能够直观地反映计算值与精确值之间的近似程度;二是能够更直观地反映出浮点数误差的特点.

3.2 基于代码隔离的性能测试方法

一方面,现有测试工具并没有集成性能测试,降低了测试工具的适用性;另一方面,传统的性能测试方法在

进行数学函数库测试时不够准确.传统的测试方法只是简单地利用时间计数器的前后差值来计算函数运行时间,在测试过程中会受到函数外未完成的访存操作或函数首次加载的影响,导致测试结果的准确性降低.假设外部影响所消耗的时间为 *b*,而被测程序的实际时间消耗为 *a*,当 *a*>>*b* 时,我们可以认为忽略 *b* 对测试结果基本不造成影响,这也是目前多数大型程序测试还在延用此方法的原因所在;否则,这种影响将是致命的,会造成测试结果的严重偏差.不幸的是,数学函数库中的每一个函数都是这样的状况.

数学函数库往往有高性能需求,函数运行时间只有几十到几百拍不等,如本课题组实现的 Mlib 函数库中的 sin 函数性能为 78 拍,fabs 函数为 17 拍(运行平台的主频为 1000MHz,不同的体系结构下函数运行时间可能不同),所以外部环境的微弱影响都可能影响函数性能测试的准确性.鉴于此,本文利用函数预热、存储栏栅等代码隔离技术降低外部环境对函数测试的影响,以实现准确的性能测试.

性能测试主要采用指令 RTC 来读取计时器 TC 的内容,计时器(time counter,简称 TC)即周期计数器 PCC,长度为 64 位,主要用来存储处理器当前运行的节拍数.测试通过调用相应的被测函数,在调用前后分别记录相应的时间轴数值,两者的差值就是此被测试函数的性能开销,如图 4 所示.其中,*rpcc* 函数获取当前的节拍数,*aa* 和 *bb* 数组中分别存放每次调用函数前后的节拍数,在 *timeuse* 数组中存放两个节拍数的差值,也就是被测函数的性能开销.

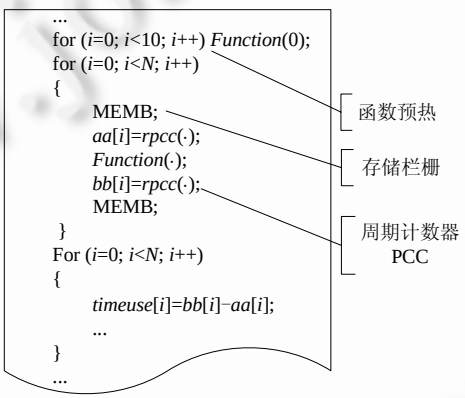


Fig.4 Kernel of the performance test

图 4 性能测试核心代码

测试首先进入函数预热阶段,此过程的目的主要是实现被测函数的预加载,以确保正式测试不会将函数加载所消耗的时间计算到函数运行时间内.此外,需要重点关注的是存储栏栅的使用,这是降低测试环境对性能测试干扰的有效策略.

没有存储栏栅 MEMB 指令时,其他处理核心观测到的本核心发出的对不同地址空间的存储器访问操作可能乱序执行.而 MEMB 指令则用于确保访存操作顺序进行,保证 MEMB 指令后的所有存储器访问指令不处理,直到 MEMB 指令之前的所有存储器访问处理完.MEMB 指令能够排除测试函数外部的访存操作消耗计算到最终结果的可能,能够有效降低外部环境对性能测试的干扰,确保测试的准确性.

采用多次循环的方式,主要考虑的就是测试的稳定性.由于硬、软件方面的原因,可能会造成所得到时间数据的波动[46],单独以某次或者某几次的测量数据去衡量函数性能是不科学的,采用多次循环的方法可以有效缓解这个问题,在一定程度上能够减小测试数据的波动,进一步提高测试的准确性.而对于多次循环后的测试结果的处理方法,将在第 4.2 节中进行详细阐述.

4 测试结果评价方案

测试总是希望能够通过测试结果反映被测对象的某些真实情况.通过精度测试,希望了解函数的计算精度;

通过性能测试,则希望了解函数运行性能.利用第 3 节的测试方法对函数进行测试,将产生大量的浮点测试结果,它们是如何反映函数精度的?又是怎样体现函数性能的?本文将采取下述方式对测试结果进行综合评价.

4.1 精度测试结果

对于精度测试,通过上节描述的测试方法,可以获取如下的 2 次计算测试信息:最大 ULP 值、平均 ULP 值及输出结果在各 ULP 区间的百分比,将测试结果进行分类的 ULP 区间为[0,0.5),[0.5,1),[1,2),[2,10),[10,*inf*)[47].

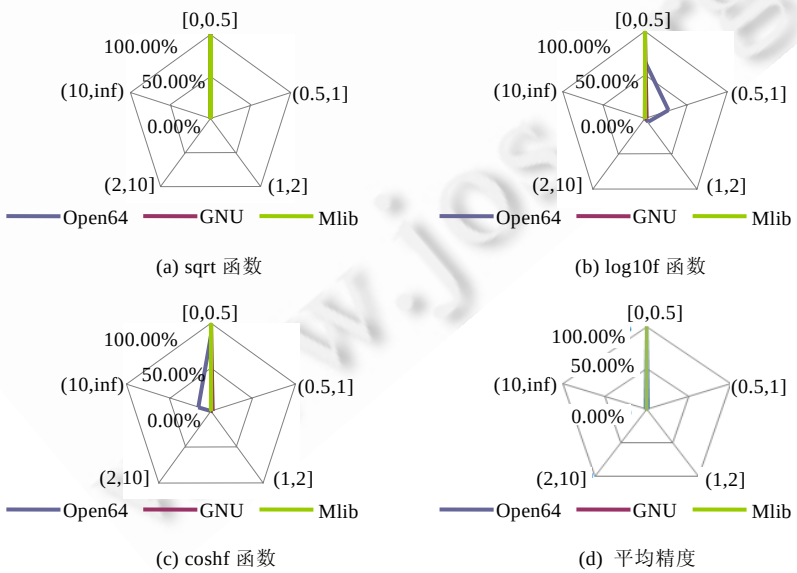
以本课题组实现函数库 Mlib、开源函数库 GNU 和 Open64 为例,其测试结果见表 1.

Table 1 Results of precision test of the mathematical functions (%)

表 1 函数库精度测试结果(%)

函数	[0,0.5)			[0.5,1)			[1,2)			[2,10)			[10,*inf*)		
	Open64	GNU	Mlib	Open64	GNU	Mlib	Open64	GNU	Mlib	Open64	GNU	Mlib	Open64	GNU	Mlib
acos	99.97	100.0	99.22	0.03	0.0	0.78	0.00	0	0	0.00	0	0	0	0	0
acosf	92.35	99.75	100.0	1.04	0.25	0.00	0.94	0	0	1.69	0	0	3.97	0	0
asin	99.99	100.0	99.95	0.01	0.00	0.04	0.00	0	0.01	0.00	0	0	0.00	0	0
asinf	100.0	99.78	99.97	0.00	0.22	0.03	0.00	0	0	0.00	0	0	0.00	0	0
exp	99.23	99.95	99.95	0.12	0.05	0.05	0.00	0	0	0.00	0	0	0.65	0	0
expf	94.07	99.62	99.61	0.25	0.38	0.39	0.00	0	0	0.01	0	0	5.67	0	0.01
atan	99.98	100.0	100.0	0.02	0.00	0.00	0.00	0	0	0.00	0	0	0.00	0	0
atanf	100.0	99.85	99.96	0.00	0.15	0.04	0.00	0	0	0.00	0	0	0.00	0	0
cosh	97.87	99.82	99.94	0.23	0.18	0.06	0.00	0	0	0.00	0	0	1.90	0	0
coshf	84.94	98.88	99.68	0.27	1.12	0.32	0.00	0	0	0.00	0	0	14.79	0	0
log10	71.13	99.86	100.0	26.29	0.13	0.00	2.58	0	0	0.00	0	0	0.00	0	0
log10f	66.11	98.59	100.0	28.77	1.36	0.00	5.12	0.05	0	0.00	0	0	0.00	0	0
sqrt	100.0	100.0	100.0	0.00	0.00	0.00	0.00	0	0	0.00	0	0	0.00	0	0
sqrtf	100.0	100.0	100.0	0.00	0.00	0.00	0.00	0	0	0.00	0	0	0.00	0	0

表中数据表示某一函数分别在每个函数库中对应区间下测试结果占总结果的百分比.单纯地从大批量的测试结果很难全面地评价测试结果,为了更直观地体现函数库整体精度,了解函数精度在各精度测试区间的状况,本文研究的测试平台将精度测试结果以雷达图的形式呈现,更方便于进行精度分析.以 sqrt,log10f,coshf 函数和平均情况为例,形成的雷达图分布如图 5 所示.



(a) sqrt 函数 (b) log10f 函数

(c) coshf 函数 (d) 平均精度

Fig.5 Evaluation results of the precision test of mathematical functions

图 5 函数精度测试结果评价图

从图 5 中可以分析得到:3 个库中,sqrt 函数精度相同;log10f 和 coshf 函数的精度从高到低为 Mlib,GNU 和 open64;平均看来,Mlib 和 GNU 函数库精度基本保持一致,均高于 Open64 函数库.

通过雷达图数据的具体分布,可以通过如下的评价指标确定 *A* 函数库的某一函数精度高于 *B* 函数库:

(1) *A* 的数据相比于 *B* 更接近于[0,0.5]坐标轴,重合为最优,如图 5(a)所示;

(2) *A* 的数据线在顺时针方向与[0,0.5]坐标轴形成的封闭区域面积小于 *B*,如图 5(b)所示;

(3) *B* 的数据线在逆时针方向与[0,0.5]坐标轴形成的封闭区域面积大于 *A*,如图 5(c)所示;

(4) 当第(2)、第(3)种情况同时出现,则优先考虑情况(3).

4.2 性能测试结果

为了提高性能测试的准确性,测试中的性能数据平均值取算术平均值.但实际测试的数据中常常存在偏差比较大的数据,从而实际性能测试结果与理论值偏差比较大.虽然在第 3.2 节中利用多种代码隔离技术以求测试结果尽量接近或等于理论值,但在实际运行过程中,终究无法百分之百保证测试结果不出现偏差大的数.

为了减小偏差较大的测试数据对性能测试结果的影响,需要对测试数据进行一定的处理,剔除掉偏差较大的数据.比较常用的方法是 4D 检验法,虽然该方法实现简单,但其能够准确地实现数据的偏差检查,可满足函数测试数据分析的要求.假设测试迭代 15 次产生的数据 $a_1,a_2,\ldots,a_{15}$,按照从小到大排列.对于 a_1,a_2,a_{14},a_{15} 分别进行如下检测:计算 $a_3,a_4,\ldots,a_{13}$ 共 11 个数据的平均值 p 和平均偏差 $d=(|a_3-p|+\ldots+|a_{13}-p|)/11$.例如对 a_1 进行检测:若 $|a_1-p|>4\times d$,则 a_1 偏差较大,应舍去;同样对其他需检测的数据进行处理,剩余的数据再取算术平均值.

以 cos 函数为例,进行 100 次性能测试,测试结果如图 6、图 7 所示.

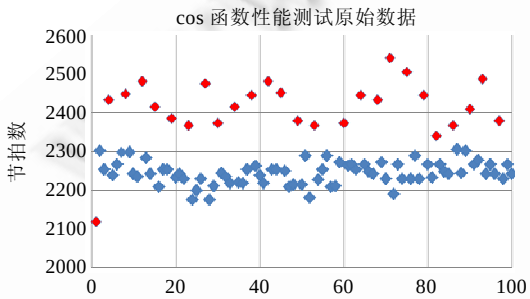


Fig.6 Original data distribution of performance test of function cos

图 6 cos 函数性能测试原始数据

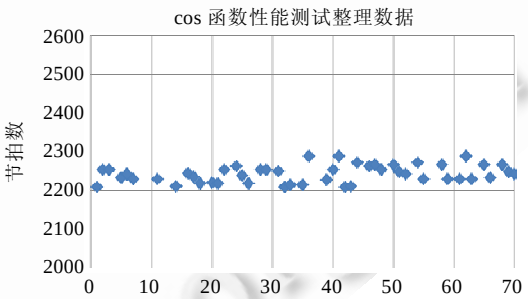


Fig.7 Data distribution of performance test of function cos after deviation check

图 7 偏差检测后 cos 函数性能测试数据

图 6 为性能测试的原始数据,从图中可以发现,部分测试结果分布并不集中,此部分数据若参与最终性能测试结果的计算,将影响到测试结果的准确性.图 7 为经过 4D 检测法处理之后的数据,处理后数据分布更为集中,更有利于对测试结果的评价.通过 4D 检测法,可以将测试结果的方差从原始的 87.79 降到 22.74,有效排除了性能测试结果中可能存在的异常点,使得测试数据更为稳定,更有利于准确地评价测试结果.

5 实验结果与分析

对于 BMltest 平台的相关技术指标的测试在众核异构结构处理器上进行,能够同时对主核及从核的数学函数库进行测试.测试在单个节点进行,主核处理器主频为 1000MHz,内存主频 400MHz,内存容量 1.8GB;从核处理器主频为 1200MHz,内存主频为 800MHz,内存容量 1.8GB.

测试主要分为 3 个方面:数据集的有效性测试、精度测试方法测试和性能测试方法测试.其中,

- 数据集的有效性测试指通过与均匀分布数据集(现有测试工具大都采用均匀随机生成器生成的均匀分布数据集)进行对比测试;

- 精度测试方法测试指通过与基于单/双精度精确值计算的方法(UCBTEST,Berkeley 大学研发的测试工具等都采用了此方法,该方法通过近似计算,如连分式逼近方法,实时计算在单精度/双精度范围内的精确值)进行对比测试,以验证精度测试方法的可靠性;
- 性能测试方法测试指通过与基于计时器的性能测试方法(不具备代码隔离及偏差检测技术的传统测试方法)进行对比测试,以验证性能测试方法的准确性和稳定性.

5.1 数据集的有效性测试

本文从对精度测试结果的分析及测试数据能够包含的函数分支数等方面,对数据集的有效性进行测试分析.测试采用相同的精度测试方法,对 Mlib 数学函数库中各类函数(三角类、指数类、误差类、双曲类、对数类、数值运算类)中的部分典型函数进行测试,数据集中的测试数据为 2 100 万,测试结果见表 2.

Table 2 Efficiency test of testing suite

表 2 数据集有效性测试

函数名	函数分支数	均匀分布数据集			本文数据集		
		[0,0.5) (%)	测试覆盖分支数	覆盖率(%)	[0,0.5) (%)	测试覆盖分支数	覆盖率(%)
cos	14	99.82	12.5	89.29	99.61	14	100.00
cosf	13	100.0	11.5	88.46	99.99	13	100.00
asin	6	99.99	5.5	91.67	99.95	6	100.00
asinf	7	99.97	6	85.71	99.97	7	100.00
exp	9	99.98	8.5	94.44	99.95	9	100.00
expf	9	99.62	7.5	83.33	99.61	9	100.00
erf	8	100.0	7	87.50	96.58	8	100.00
erff	8	99.81	7	87.50	99.24	8	100.00
Cosh	9	99.97	9	100.00	99.94	9	100.00
Coshf	9	99.78	9	100.00	99.68	9	100.00
log1p	10	100.0	10	100.00	99.98	10	100.00
log1pf	10	100.0	10	100.00	99.97	10	100.00
sqrt	7	100.0	6	85.71	100.0	7	100.00
sqrtf	8	100.0	6	75.00	100.0	8	100.00
Average	9.07	99.92	8.25	90.94	99.61	9.07	100.00

其中,对于测试覆盖分支数的测试采用了一些常规的程序分析技术,以统计数据集中每个数据经过的函数分支.测试覆盖分支数出现小数形式,是因为在部分函数分支中存在多个跳转判断,当所有测试数据经过某个判断时出现跳转,则当前分支下该判断之后的程序段将一直无法进行测试,出现这种情况则统计为 0.5 个分支.

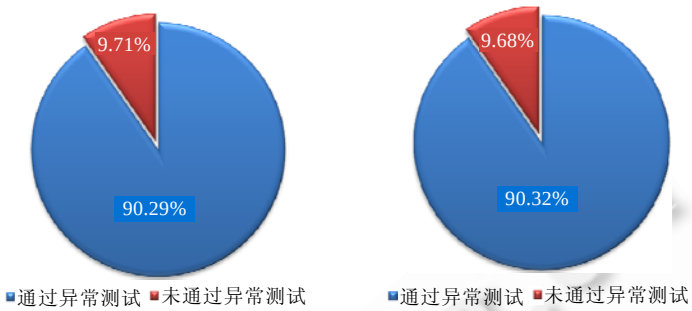
以 cos 函数采用均匀分布数据集进行测试为例,[0,0.5)区间的精度测试结果表明:在 2 100 万的测试数据集中,有 99.82%的数据作为 cos 函数的输入计算后的结果为精确值,其余 0.18%的值计算的结果则出现了误差.测试覆盖分支数为 12.5 表明:利用均匀分布数据集进行测试能够测试到函数中的 12 个完整的分支,外加一个没有实现完整测试的分支.覆盖率计算方法为测试覆盖分支数除以函数分支数.

从测试数据所覆盖的函数分支看,采用本文数据集进行测试平均能够多覆盖 10%左右的函数分支(覆盖率从均匀分布数据集的 90.94%提高到 100.00%),以此保证能够测试到函数的所有分支.从[0,0.5)区间的精度测试结果看,采用本文数据集进行测试平均能够多发现 0.31%的函数潜在精度损失(函数精度从 99.92%降低到 99.61%),能够更真实的测试出函数的实际精度.

进一步利用本文方法实现的测试数据集对 GNU 函数库(包括主核函数 124 个、从核函数 103 个)进行了测试,测试中主要关注了函数库的健壮性及异常处理能力,测试结果如图 8 所示.

测试结果表明:GNU 函数库能够实现浮点异常处理的函数个数在主核上仅占函数数量的 9.68%,在从核上仅占 9.71%.因此可以判断 GNU 函数库大多数函数无法通过浮点异常处理测试,函数的可靠性相对较差.而如果利用单一的均匀分布数据集进行测试,测试结果则显示为全部通过,将无法测试出函数真实情况.

通过对函数分支覆盖率、函数精度及异常处理能力测试的分析可以发现:本文实现的数据集在测试过程中能够发现更多函数可能存在的潜在问题,数据集更有效.



(a) 从核测试 (b) 主核测试

Fig.8 Exception test of GNU function library

图 8 GNU 函数库异常处理测试

5.2 精度测试方法的可靠性测试

测试采用相同的数据集(2 100 万),对数学函数库中各类函数中的部分典型函数进行精度测试.首先,以 cos 函数为例,采用两种方法进行精度测试,测试结果如图 9、图 10 所示.

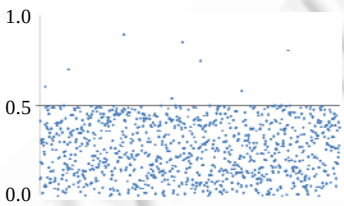


Fig.9 Precision test result using our method

图 9 本文方法测试的精度结果

Fig.10 Precision test result of double precision testing method

图 10 双精度精确值计算的方法测试的精度结果

测试结果表明:采用本文方法对函数进行精度测试的结果主要集中在 0ULP~0.5ULP 和 0.5ULP~1ULP 的精度范围,能够精确反映函数的实际精度;而由于单/双精度精确值计算的测试方法存在尾数长度受限、舍入模式异常等问题,造成精度误差较大,无法保证测试结果的可靠性.在测试过程中,将部分精确值(0~0.5*ULP* 区间)测试为非精确值(0.5ULP~1ULP)、非精确值测试为精确值.

通过对部分典型函数的测试,同样可以得到上述结论,测试结果见表 3.

Table 3 Reliability test result of some representable function's precision tests (%)

表 3 典型函数精度测试方法的可靠性测试结果(%)

函数名	单/双精度精确值计算的方法			本文方法		
	0	1	2	[0,0.5)	[0.5,1)	[1,2)
cos	100.0	0	0	99.61	0.39	0
cosf	100.0	0	0	99.99	0.01	0
asin	99.96	0.04	0	99.95	0.04	0.01
asinf	99.97	0.03	0	99.97	0.03	0
exp	99.95	0.05	0	99.95	0.05	0
expf	99.62	0.34	0.04	99.61	0.39	0
erf	97.82	2.18	0	96.58	3.42	0
erff	99.85	0.15	0	99.24	0.76	0
cosh	99.96	0.04	0	99.94	0.06	0
coshf	99.88	0.12	0	99.68	0.32	0
log1p	100.0	0	0	99.98	0.02	0
log1pf	100.0	0	0	99.97	0.03	0
sqrt	100.0	0	0	100.0	0.00	0
sqrtf	100.0	0	0	100.0	0.00	0

由于采用多精度库测试导致运行开销提升,因此本文对单/双精度精确值计算的方法和本文方法的测试时

间及占用的存储空间进行了测试,该测试在同平台下对相同的函数采用相同的测试数据集(数据集大小为 2 100 万)进行测试,以双精度精确值计算方法的测试结果为例,结果如图 11、图 12 所示.

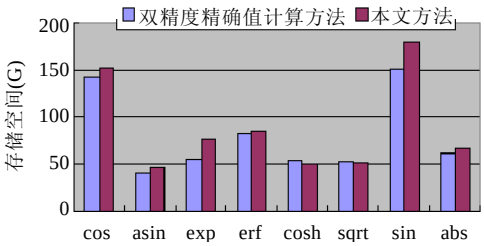


Fig.11 Temporary comparison of precision testing methods

图 11 不同精度测试方法的时间开销

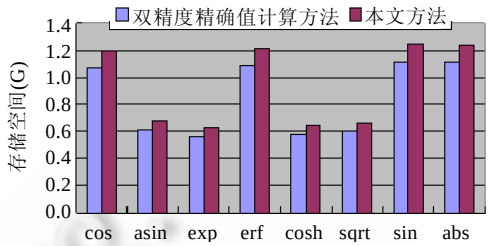


Fig.12 Spatial comparison of precision testing methods

图 12 不同精度测试方法的空间开销

通过对数学函数库中各类函数的部分典型函数的测试,本文方法在测试时间上平均比双精度精确值计算方法多 8.65s;同时,平均多消耗 0.1G 的存储空间.对于高性能计算平台硬件而言,这样的开销完全可以接受.

5.3 性能测试方法的准确性及稳定性测试

为了验证本文性能测试方法的准确性和稳定性,本文采用相同的数据集(10 万),对数学函数库中各类函数中的部分典型函数进行性能测试,并计算每一个被测函数的性能测试结果的方差,以衡量测试结果的准确性及稳定性,测试结果见表 4.其中,传统方法为基于计时器的性能测试方法.

Table 4 Performance Variance of some representable performance testing methods

表 4 典型函数性能测试方法的性能结果方差

函数名	主核		从核	
	传统方法	本文方法	传统方法	本文方法
sin	9 149	1	7 717	53
log	12 422	24	5 950	17
ceil	1 779	221	129	6
cos	7 280	1	9 546	28
exp	8 923	1	3 338	57

从测试结果可以发现:利用本文的方法进行性能测试后,测试结果的方差远小于利用传统方法测试后的方差,充分体现了本文方法的准确性和稳定性.

6 总 结

对函数算法及实现的研究固然重要,但测试(包括验证)也是不容忽视的重要环节.只有通过测试,才能保证函数的可靠性和精度以及实现的优劣.在程序验证相对困难的情况下[48],测试是确定函数是否可靠和高效的有效途径.

本文实现了浮点数学库的统一测试平台,完整地包含了正确性测试、精度测试和性能测试这 3 个方面.该测试平台将测试分为多个子模块,提高了测试准确性的同时降低了实现的难度,并针对测试数据集和结果分析方案作了较详细的阐述.BMltest 平台采用新的测试数据集生成方法,有效缓解了正确性测试和精度测试中测试用例无法覆盖函数各分支的问题,提高了测试浮点数的覆盖率.同时,在精度测试方面引入了多精度库,提高了测试的可靠性.在性能测试过程中,利用代码隔离和偏差检测技术在函数关键路径区间内进行测试,降低了外部环境对测试的干扰,提高了性能测试的准确性.与目前存在的应用较广泛的测试工具相比,BMltest 平台测试范围更广、功能更多、扩展性更好,见表 5.

Table 5 Comparison of testing platforms and methods

表 5 不同测试工具的指标对比

测试工具	测试范围	测试难易度	测试功能	兼容性	扩展性
BMltest	所有基础函数	易	正确性测试、精度测试、性能测试	Fortran、C、汇编	强
UCBTEST	部分函数	难	正确性测试、精度测试	Fortran, C	弱
ELEFUNT	少数函数	易	正确性测试	C,Java	弱
Berkeley	少数函数	较难	正确性测试、精度测试	Fortran、C、汇编	弱

本文从测试前测试集的生成到测试中测试方法的选择,再到测试后测试结果的评价,都进行了详尽的分析研究,为各类函数库的测试提供了一个有效的实现方案.通过该方案,能够准确得到测试结果并进行合理的分析,具有较强的实际应用能力.

致谢 在此,向评阅本文的审稿专家表示衷心的感谢,向对本文工作给予支持和建议的同行,尤其是课题组的同学和老师表示感谢.

References:

[1] Liu CG. Floating Point Calculation: Programming Principle, Realization and Application. Beijing: China Machine Press, 2008. 130−132 (in Chinese).

[2] Xu JC, Guo SZ, Wang L. Optimization technology in SIMD mathematical functions based on vector register reuse. In: Proc. of the 2012 IEEE 14th Int'l Conf. on High Performance Computing and Communications (HPCC 2012). Liverpoor: IEEE Computer Society, 2012. 1102−1107. [doi: 10.1109/HPCC.2012.161]

[3] Daramy C, Defour D, de Dinechin F, Muller JM, Arenaire P. CR-LIBM: A correctly rounded elementary function library. In: Proc. of the Optical Science and Technology, SPIE's 48th Annual Meeting. Int'l Society for Optics and Photonics. 2003. 458−464. [doi: 10.1117/12. 505591]

[4] Wu XY, Xia JL. New vector forms of elemental functions with Taylor series. Applied Mathematics and Computation, 2003,141(2): 307−312. [doi: 10.1016/S0096-3003(02)00255-2]

[5] Tang PTP. A Portable Generic Elementary Function Package in Ada and an Accurate Test Suite. Department of Defense, 1990. [doi: 10.1145/123533.123573]

[6] Manos P, Turner LR. Constrained Chebyshev Approximations to Some Elementary Functions Suitable for Evaluation with Floating Point Arithmetic. NASA, 1972.

[7] Abramowitz M, Stegun IA. Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. Courier Dover Publications, 1964.

[8] Andraka R. A survey of CORDIC algorithms for FPGA based computers. In: Proc. of the '98 ACM/SIGDA 6th Int'l Symp. on Field Programmable Gate Arrays. ACM Press, 1998. 191−200. [doi: 10.1145/275107.275139]

[9] Muller JM. Elementary Functions: Algorithms and Implementation. Springer-Verlag, 2006.

[10] Baboulin M, Buttari A, Dongarra J, Kurzak J, Langou J, Langou J, Luszczek P, Tomov S. Accelerating scientific computations with mixed precision algorithms. Computer Physics Communications, 2009,180(12):2526−2533. [doi: 10.1016/j.cpc.2008.11.005]

[11] Bailey DH, Barrio R, Borwein JM. High-Precision computation: Mathematical physics and dynamics. Applied Mathematics and Computation, 2012.

[12] Numerical Computation Guide-Sun Studio 11. 2005 (in Chinese). http://www.sun.com/hwdocs/Numerical Computation Guide-Sun Studio 11.pdf

[13] Li Y, Zhang YQ, Wang K, Guan WH. Heterogeneous multi-core parallel SGEMM performance testing and analysis on Cell/B.E processor. In: Proc. of the IEEE NAS 2010. Macau, 2010. 202−207. [doi: 10.1109/NAS.2010.48]

[14] Yang C, Zhang YQ, Li LG. Numerical simulation of the thermal convection in the Earth's outer core. In: Proc. of the 12th IEEE Int'l Conf. on High Performance Computing and Communications (HPCC 2010). Melbourne, 2010. 552−555. [doi: 10.1109/HPCC.2010.70]

[15] Yuan L, Zhang YQ, Sun XZ, Wang T. Optimizing sparse matrix vector multiplication using diagonal storage matrix format. In: Proc. of the 12th IEEE Int'l Conf. on High Performance Computing and Communications (HPCC 2010). Melbourne, 2010. 585−590. [doi: 10.1109/HPCC.2010.67]

[16] Yang CQ. Key compilation techniques for high productivity computing: Precision, performance and power consumption [Ph.D. Thesis]. Changsha: National University of Defense Technology, 2007 (in Chinese).

[17] Allen E, Chase D, Hallett J, Luchangco V, Maessen JW, Ryu S, Steele Jr. GL, Tobin-Hochstadt S. The Fortress language specification. Sun Microsystems, 2005,139:140.

[18] Yang C, Li LG, Zhang YQ. Development of a scalable solver for the Earth's core convection. In: Proc. of the HPCA 2009. LNCS 5938. Springer, 2010. 497−502. [doi: 10.1007/978-3-642-11842-5_69]

[19] Zhang YQ, Sun JC, Yuan GX, Zhang LB. Perspectives of China's HPC system development: A view from the 2009 China HPC TOP100 list. Frontiers of Computer Science in China, 2009,4(4):437−444. [doi: 10.1007/s11704-010-0372-0.2009]

[20] IEEE 754-1985: IEEE Standard for Binary Floating-Point Arithmetic. New York: IEEE, 1985. [doi: 10.1109/MCSE.2005.52]

[21] Green R. Faster math functions. In: Proc. of the Game Developers Conf. 2003.

[22] Koren I. Computer Arithmetic Algorithms. 2nd ed., Prentice Hall, 2001. 225−246.

[23] Juang TB. Para-CORDIC: Parallel CORDIC rotation algorithm. IEEE Trans. on Circuits and Systems I: Regular Papers, 2004,51(8): 1515. [doi: 10.1109/TCSI.2004.832734]

[24] Moore RE. Interval Analysis. Englewood Cliffs: Prentice-Hall, 1966.

[25] de Weerdt E, Chu QP, Mulder JA. Continuous state and action advantage-learning using interval analysis and neural networks. In: Proc. of the AIAA Guidance, Navigation and Control Conf. and Exhibit. South Carolina: Hilton Head, 2007.

[26] Bailey DH. High-Precision floating-point arithmetic in scientific computation. Computing in Science and Engineering—C in S&E, 2005,7(3):54−61. [doi: 10.1109/MCSE.2005.52]

[27] Benz F, Hildebrandt A, Hack S. A dynamic program analysis to find floating-point accuracy problems. In: Proc. of the PLDI 2012. 2012. [doi: 10.1145/2254064.2254118]

[28] Boldo S, Nguyen TMT. Hardware-Independent proofs of numerical programs. In: Proc. of the 2nd NASA Formal Methods Symp. Washington: NASA Conf. Publication, 2010.

[29] Akbarpour B, Paulson L. Metitarski: An automatic prover for the elementary functions. In: Proc. of the Intelligent Computer Mathematics. LNCS 5144, Springer-Verlag, 2008. 217−231. [doi: 10.1007/978-3-540-85110-3_18]

[30] Boldo S, Melquiond G. Flocq: A unified library for proving floating-point algorithms in Coq. In: Proc. of the 20th IEEE Symp. on Computer Arithmetic. 2011. [doi: 10.1109/ARITH.2011.40]

[31] Ayad A, Marché C. Multi-Prover verification of floating-point programs. In: Giesl J, Hähnle R, eds. Proc. of the 15th Int'l Joint Conf. on Automated Reasoning. Edinburgh: Springer-Verlag, 2010. [doi: 10.1007/978-3-642-14203-1_11]

[32] Foussea L, Zimmermanna P. A formal proof of demmel and Hida's accurate summation algorithm. Elsevier Science, 2004.

[33] Boldo S, Filliatre JC, Melquiond G. Combining Coq and gappa for certifying floating-point programs. In: Proc. of the 16th Symp., 8th Int'l Conf. on Held as Part of CICM 2009 on Intelligent Computer Mathematics. Springer-Verlag, 2009. 74. [doi: 10.1007/978-3-642-02614-0_10]

[34] de Dinechin F, Lauter C, Melquiond G. Certifying the floating-point implementation of an elementary function using Gappa. IEEE Trans. on Computers, 2011,60(2):242−253. [doi: 10.1109/TC.2010.128]

[35] Cody WJ. ELEFUNT test results under AST Fortran V1.8.0 on the sequent symmetry. Technical Report. United States: Mathematics and Computer Science Div, 1990.

[36] UCBTEST suite. http://www.netlib.org/fp/ucbtest.tgz

[37] Liu ZA. Berkeley elementary function test suite [MS. Thesis]. Berkeley: Computer Science Division, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1987.

[38] Kuliamin VV. Standardization and testing of implementations of mathematical functions in floating point numbers. Programming and Computer Software, 2007,33(3):154−173. [doi: 10.1134/S036176880703005X]

[39] Verdonk B, Cuyt A, Verschaeren D. A precision-and range-independent tool for testing floating-point arithmetric I: Basic operations, square root, and remainder. ACM Trans. on Mathematical Software (TOMS), 2001,27(1):92−118. [doi: 10.1145/382043.382404]

[40] Cody WJ. Algorithm 714: CELEFUNT: A portable test package for complex elementary functions. ACM Trans. on Mathematical Software (TOMS), 1993,19(1):1−21. [doi: 10.1145/151271.151272]

[41] Olofsson P. Improved testing of an arbitrary-precision arithmetic library using artificially small words. Bachelor's Thesis in Computer Science, Stockholm University, 2012.

[42] Li HF, Wang SQ, Liu C, Zheng J, Li Z. Software reliability model considering both testing effort and testing coverage. Ruan Jian Xue Bao/Journal of Software, 2013,24(4):749−760 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4257.htm [doi: 10.3724/SP.J.1001.2013.04257]

[43] Wang LL, Li BX, Zhou XY. Profiling all paths. Ruan Jian Xue Bao/Journal of Software, 2012,23(6):1413−1428 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4102.htm [doi: 10.3724/SP.J.1001.2012.04102]

[44] Fousse L, Hanrot G, Lefèvre V, Pelissier P, Zimmermann P. MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Trans. on Mathematical Software (TOMS), 2007,33(2):13. [doi: 10.1145/1236463.1236468]

[45] Muller JM. On the Definition of *Ulp*(*x*). France: Laboratoire de l'Informatique du Parallélisme, 2005.

[46] Ding HW. A design and implementation of decimal floating-point multiplication unit based on SOPC. In: Proc. of the 3rd Int'l Conf. on Digital Manufacturing and Automation. 2012. 36−41.

[47] Enenkel R. A comprehensive test environment for mathematical functions. IBM Technical Report, TR-74.200, IBM Corp., 2004.

[48] Schryer NL. A test of computer's floating-point arithmetic unit. Computer Science Technical Report, AT&T Bell Labs., 1981.

附中文参考文献:

[1] 刘纯根.浮点计算编程原理、实现与应用.北京:机械工业出版社,2008.130−132.

[12] Sun 数值计算指南. 2005. http://www.sun.com/hwdocs/Sun 数值计算指南.pdf

[16] 杨灿群.面向高效能计算的编译关键技术:精度、性能与功耗[博士学位论文].长沙:国防科学技术大学,2007.

[42] 李海峰,王栓奇,刘畅,郑军,李震.考虑测试工作量与覆盖率的软件可靠性模型.软件学报,2013,24(4):749−760. http://www.jos.org.cn/1000-9825/4257.htm [doi: 10.3724/SP.J.1001.2013.04257]

[43] 王璐璐,李必信,周晓宇.全路径剖析方法.软件学报,2012,23(6):1413−1428. http://www.jos.org.cn/1000-9825/4102.htm [doi: 10.3724/SP.J.1001.2012.04102]

许瑾晨(1987−),男,江苏泰州人,博士生,主要研究领域为高性能计算.

周蓓(1977−),女,讲师,主要研究领域为高性能计算.

黄永忠(1968−),男,博士,教授,博士生导师,主要研究领域为高性能计算,大数据处理.

赵捷(1987−),男,博士生,主要研究领域为先进编译技术.

郭绍忠(1964−),女,副教授,主要研究领域为高性能计算,分布式处理.