向量数学库的向量化方法研究

周 蓓[1] 黄永忠[2] 许瑾晨[1] 郭绍忠[1]
(数学工程与先进计算国家重点实验室 郑州 450002)[1]
(桂林电子科技大学 广西 桂林 541004)[2]

摘 要 SIMD技术的出现使得基础数学库扩展到向量数学库成为必然趋势。基础数学库中多数函数存在代码实现复杂、分支判断多的特点,增加了向量化的难度,同时SIMD指令的不完备导致函数中的部分功能无法直接向量化,频繁的拆分和拼接操作降低了函数的性能。针对这些问题,提出了向量数学库的向量化方法,通过确定核心代码段、数据预处理过程向量化及指令向量化3个步骤,可以快速有效地对基础数学库进行向量化。实验表明,运用该方法,exp,pow,log10等典型函数的性能平均提高了24.2%。

关键词 SIMD技术,向量数学库,核心代码段,数据预处理,指令向量化
中图法分类号 TP313 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.01.050

Study on SIMD Method of Vector Math Library

ZHOU Bei[1] HUANG Yong-zhong[2] XU Jin-chen[1] GUO Shao-zhong[1]
(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China)[1]
(Guilin University of Electronic Technology, Guilin, Guangxi 541004, China)[2]

Abstract It's an inexorable trend from basic math library to vector math library with the occurrence of SIMD. But there are many difficulties because of complicated code and many branches of math library. On the other hand, SIMD instructions are not complete, so some functions are realized by frequent split and joint, which reduces the performance quickly. An effective vectoring method of vector math library was proposed in this paper. It consists of key code segment selection, data pre-processing vectoring and instruction vectoring. This method not only gets an effective performance improvement as much as possible, but also is a solid base for later depth optimization. The experimental results show that it can highly improve the functions' performance such as exp, pow and log10 up to 24.2% on average respectively.

Keywords SIMD technique, Vector math library, Key code segment, Data pre-processing, Instruction vectoring

1 引言

基础数学库是高性能计算机系统的核心基础软件。基础数学库主要完成科学和工程计算领域中常见的数值密集型运算,在气象、地震、油藏模拟等众多应用领域中具有举足轻重的作用。随着单指令流多数据(Single Instruction Multiple Data,SIMD)技术的提出,应用SIMD技术开发的向量数学库能够加速函数的性能,主要用来解决高度结构化、需要大量向量计算或者矩阵计算的问题[1-2]。

Intel的VML、AMD的ACML、Cell的XL Mass等是比较有代表性的向量数学库。这些向量数学库相对传统的标量计算而言,向量化带来的加速比较高,效率提升明显。由于不同处理器厂商的处理器体系结构不同、SIMD处理单元不同、指令集不同等,它们的性能在解决同一个问题时也存在着差异[3]。

2010年,中国科学院完成了基于龙芯3A国产四核CPU关键技术的研发,实现了龙芯3A体系结构的调研报告和优化技术报告。龙芯3A CPU的国产高性能扩展数学库CLeXML研究了自适应性能优化技术、递归分块技术、地址交错技术和多核并行化技术等,以提高数学库的性能[4]。针对申威1600处理器,文献[5]利用SIMD Intrinsic Functions[6]开发了长向量数学库软件包,并通过访存优化、化简向量分支、多线程等技术加以优化;文献[7]通过乘加指令、循环展开、指令重排、SIMD向量化运算、寄存器分块等技术对3级BLAS GEMM函数进行优化。

除了在数学库的应用外,近些年对SIMD的研究主要集中在SIMD编译优化等方面[8-10]。SIMD编译优化是指为程序源代码尽可能地生成优化的SIMD指令序列。SIMD编译优化也成为编译技术领域的一个研究热点。目前实现自动向

到稿日期:2018-01-26 返修日期:2018-03-06 本文受面向100P高效能计算机的基础数学库系统项目,国家重点研发计划"高性能计算"重点专项:E级计算机关键技术验证系统(2016YFB0200503)资助。

周 蓓(1977-),女,博士生,讲师,主要研究方向为高性能计算,E-mail:13653970052@163.com(通信作者);**黄永忠**(1968-),男,教授,博士生导师,主要研究方向为分布式计算、大数据处理等;**许瑾晨**(1987-),男,博士,主要研究方向为高性能计算;**郭绍忠**(1964-),女,教授,硕士生导师,主要研究方向为高性能计算。

量化的方式主要有两种：传统的向量化方法和 SLP 算法[8]。

传统的向量化方法是一种基于数据相关性分析的编译优化技术，主要包括通过数据相关性分析最大限度的发现并向量化循环、代码规范化、循环变换等技术。

在超长指令字（Very Long Instruction Word，VLIW）思想的启发下，Larsen 等提出了 SIMD 编译优化的超字并行（Superword Level Parallelism，SLP）算法。SLP 算法侧重于在一段线性代码空间（例如基本块）中而不是在循环中来开发并行性。其主要思路是：在一个给定的代码空间范围内分析程序代码，找出满足一定条件的几条同构语句，将它们打包成一条 SIMD 指令。其中，同构语句是指两条语句的对应位置具有相同的操作符和操作数类型。对于程序中的循环，首先以一定的展开因子展开，使其成为一个基本块，再在基本块内进行分析，无需进行循环迭代依赖分析，简化了数据依赖分析的过程。

通过以上对现有常用的一些数学库和分析可以得知：目前大多数高性能数学库均为非开源的商业化产品，主要针对特定处理器进行优化，不具有普遍适用性。在基础数学库的向量化过程中，虽然可以借鉴 SIMD 编译优化中的一些向量化方法，但这些方法只是向量化中可以参考的一些技巧，还没有出现针对基础数学库的一种通用的向量化方法。

向量数学库的开发一般是在基础数学库的基础上，通过对程序中数据相关性的分析，尽可能地找到可并行处理的部分，应用 SIMD 技术进行最大限度的向量化处理。向量数学库具体可分为短向量数学库[4,11-12]（输入参数是一个向量寄存器长度，如 floatv4，doublev4 等）和长向量数学库[3,5]（输入参数是一个数组），短向量数学库是长向量数学库的基础，两者虽有一定联系，但处理的侧重点不同。本文主要讨论短向量数学库。

在研究数学函数的向量化过程中，函数代码复杂、分支判断多是导致向量化难度增加的主要原因。同时，由于多数处理器的指令设计采用 RISC 精简指令集，SIMD 指令的不完备[5,13]导致函数中的部分功能无法直接向量化，频繁的拆分和拼接操作降低了函数的性能。

针对 SIMD 指令的不完备问题，文献[11]提出了利用字向量逻辑左移和可重构逻辑运算指令实现向量逻辑左移的功能；文献[12]给出了用等价替换方法实现向量的算术右移操作，但操作步骤较繁琐。

需要指出的是，针对上述问题，本文提出了向量数学库的向量化方法。首先分析函数算法并结合测试，确定核心代码段，有针对性地进行向量化；然后对数据预处理过程进行向量化；最后利用逻辑功能等价替换的思路，将一些无法直接向量化的指令替换为向量化指令。鉴于条件所限，本文仅以申威 26010 众核处理器为例展开研究，实验测试结果表明，该方法可以对数学函数进行有效向量化。

本文第 2 节详细介绍了三步向量化方法；第 3 节给出实验结果及分析；最后对全文进行总结。

2　向量化方法

基础数学库中每个函数的特性不同，向量化实现的过程也不尽相同，但总体来说可以按照 3 个步骤进行：确定核心代码段、数据预处理过程向量化及指令向量化。

2.1　确定核心代码段

数学库中函数实现的特点是判断分支多，在函数运行过程中存在大量的跳转，这极大地增加了 SIMD 向量化的难度。同时，将过多的精力放在对跳转依赖的分析及细小分支的向量化过程中，不仅会大规模地增加工作量，同时还可能得不到理想的向量化效果。根据著名的“90-90”规则[14]，即程序 10％的代码量通常要消耗 90％的系统资源，可以通过程序剖分找出程序在性能消耗上的“热点”，从而有针对性地进行 SIMD 向量化，大幅提高函数性能。鉴于此，有必要在进行 SIMD 向量化前，对函数进行热点分析，找出函数执行过程中的关键路径。通过大量的数据测试以及对函数算法及执行指令的分析，获取函数的核心代码段。

以平方根函数 *sqrt* 为例，此函数的运算流程如图 1 所示。

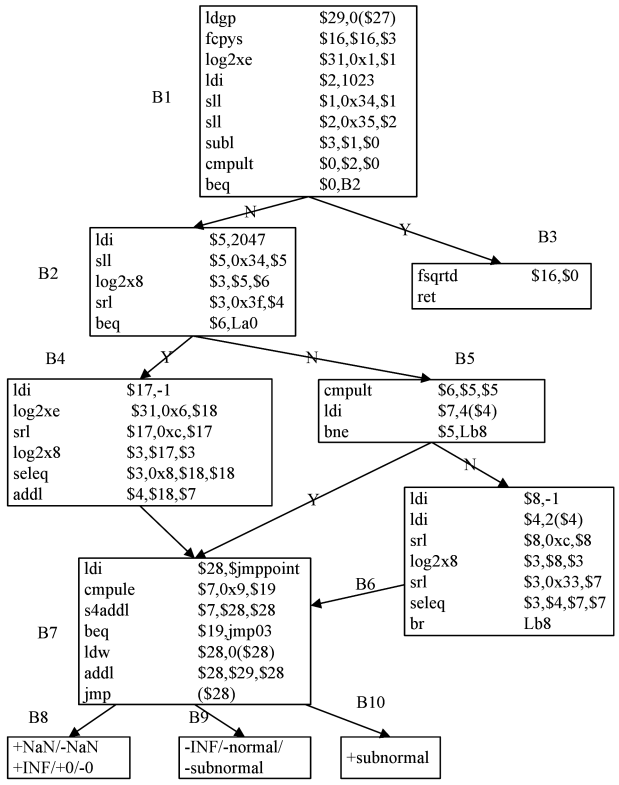

图 1　*sqrt* 函数的运算流程图

Fig. 1　Operational flowchart of *sqrt*

经过统计分析可知，图 1 的频繁执行路径由基本块 B1、B3 组成，核心基本块只占了函数 10 个基本块中的 2 个，代码量为总代码量的 14％，但通过测试可以发现此段代码占到整个函数运行时间 50％以上的性能消耗，同时函数定义域内 95％以上数据的计算通过此段代码实现。通过对函数代码的分析发现，剩余基本块的功能都是对 IEEE 754 标准定义的特殊数及函数定义域之外的数进行处理，这些基本块在函数实际应用过程中使用频率较低，因此通过直接对核心段代码进行向量化，可减少大量的工作量，同时大幅提升函数的性能。

2.2　数据预处理过程向量化

由于 SIMD 指令集的浮点运算指令中加入了对 IEEE 754 标准定义的特殊浮点数的检查功能，特殊数参与浮点运

算会引发浮点算术异常，如 IEEE 标准定义的 5 种浮点算法异常：无效操作（Invalid Operation）、除数为“0”（Division by Zero）、上溢（Overflow）、下溢（Underflow）和非精确结果（Inexact Result），从而使得函数运算过程中断，降低数学库的高可靠性及性能。因此，在进入函数核心运算阶段前，需要判断函数的向量输入参数是否存在非正常输入（包括 IEEE 754 标准中定义的特殊数和函数定义域外的输入），即判断输入参数是否会在后面的计算中引发某类异常，若存在非正常输入，则无法直接向量化，进行特殊处理分支，否则进行正常处理。这个数据预处理过程通常在函数中属于核心代码段最开始的一部分。

向量数学库中对输入参数做预处理的典型过程如下：

Step 1 程序初始化完毕后，将参数放入某个浮点向量寄存器中暂存，作为保存参数的副本。

Step 2 将这个副本向量参数拆分成 4 个分量。

Step 3 分别判断这 4 个分量是否为非正常输入，并将判断结果放入临时寄存器中暂存。

Step 4 将 4 个判断结果相与，即可得出向量参数中是否存在非正常输入，若存在，则函数进入特殊处理分支，若不存在，则进入正常求解分支。

对上述算法流程进行分析，可以发现该处理过程共存在 5 次判断操作，前 4 次是分别对向量参数中的 4 个分量进行是否为非正常输入的判断，最后一次判断操作是根据前 4 次的判断结果判断向量参数中的 4 个分量是否存在非正常输入。

函数中的判断指令会打断指令流水，从而降低函数的性能。在通常情况下，编译器的做法是在每一个判断指令后插入 3 条空操作指令 NOP，即上述过程中针对前 4 次判断总共额外插入 12 条空操作指令 NOP，4 次打断了指令流水，降低了该代码段的性能。

而对于一个逻辑功能来讲，通常有多种指令实现方式。由于每种指令的运行节拍不同，因此，对于性能较低的指令或者代码段，可以使用功能等价但性能较高的指令来替换，以提高函数的性能。对于上述过程，消除 12 条空操作指令 NOP 是提升函数性能的关键。

由于前 4 次的判断操作只是分别判断向量中的 4 个分量是否为非正常输入，并没有进行相应的跳转，因此可以通过向量化操作，一次性同时判断 4 个分量。其工作流程如图 2 所示。

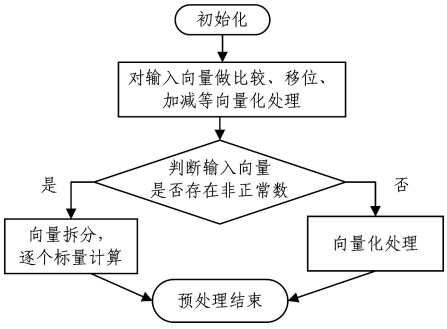

图 2 利用向量操作优化后的数据预处理流程图

Fig. 2 Flowchart of data pre-processing after vectoring

2.3 指令向量化

指令的向量化过程，是在深入分析函数算法的基础上，充分利用 SIMD 指令对函数中可以向量化的部分进行向量化，以提高执行效率。通常的做法是把基础函数的每条指令替换为对应的 SIMD 指令，但多数处理器采用 RISC 指令集结构，向量指令数目有限，部分指令没有对应的 SIMD 指令。在这种情况下，一般将向量拆分成若干标量，执行对应的标量操作，最后进行向量拼接。这种做法由于增加了大量的向量拆分与拼接操作，使函数性能损失较大。

针对该问题，与数据预处理过程的向量化思路相同，采用逻辑功能等价的 SIMD 指令代码段替换标量指令代码段。下面以函数中经常出现的几种指令为例说明指令向量化过程。

2.3.1 移位指令的向量化

移位指令在计算过程中经常使用，常用的有逻辑左移、逻辑右移和算术右移。指令集中只有字向量（字为 32 位）移位指令，没有长字向量（长字为 64 位）移位指令。由于处理思路基本相同，这里仅以逻辑左移指令的向量化为例，分情况进行说明。

第 1 种情况：如果向量寄存器的每个长字分量在移位后与把它看作两个字移位后的结果一致，则可以使用字向量的移位指令。

如图 3 所示，向量寄存器 $v1$ 中有 4 个长字，从低到高位分别为 $a0$，$a1$，$a2$，$a3$，值都为 0x5f，需要完成向量逻辑左移 4 位的任务。经过分析，由于每个长字 ai 的高位是 0x0，移位前后值不发生改变，ai 的低位逻辑左移 4 位后仍在低 32 位以内，因此，可以使用字向量的逻辑移位指令完成长字向量的逻辑移位功能。

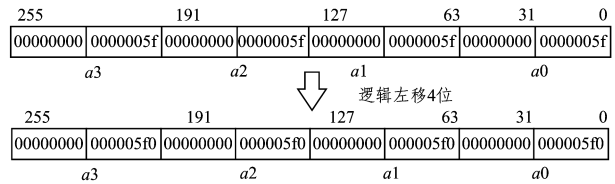

图 3 利用字向量移位指令完成向量移位操作的示意图

Fig. 3 Diagram of realizing long word vector shift function using word vector shift instruction

第 2 种情况：如果不满足第 1 种情况，则可以利用逻辑功能等价替换思路，有两种替换方法。文献[12]给出了用 8 倍字整数逻辑移位指令和向量逻辑运算指令的替换方法。本文给出另一种实现方法：通过字向量移位指令、向量逻辑运算指令和混洗指令完成长字向量的移位操作。下面给出一个具体的等价替换示例。

例：把向量 va 的每个长字分量逻辑左移 46 位。

相当于把 va 的每个长字分量的第 0 到 17 位（采用小端地址格式，最低位在最右边为第 0 位，最高位为第 63 位）移至最高位，后面补 0。向量化的思路为：

Step 1 生成一个向量 vb，每个分量的值为 0x3ffff，即第 0 到 17 位为 1，其他位为 0。

Step 2 vb 与 va 进行向量逻辑与操作。

Step 3 向量 va 的每个分量都在一个字范围内，因此利用字向量移位指令逻辑左移 14 位。

Step 4　使用混洗指令把向量 va 的每个分量的高 32 位与低 32 位对调。

对应的汇编代码如下：

```
ldi vb,0x3ffff
vshff vb,vb,0x0,vb          //对应 Step1
vlog2x8 va,vb,va            //对应 Step2
vsllw va,0xe,va             //对应 Step3
ldi vb,0x67452301
vshfw va,va,vb,va           //对应 Step4
```

指令说明：ldi 为立即数载入指令；vlog2x8 为向量逻辑与指令；vshff 和 vshfw 分别为浮点向量和字向量混洗指令；vsllw 为字向量逻辑移位指令。

图 4 是按照以上思路，对向量 va（每个分量为 0xxxxxxxxxxfffff(x 为 16 进制表示的某个值)）进行逻辑左移 46 位的实现过程。

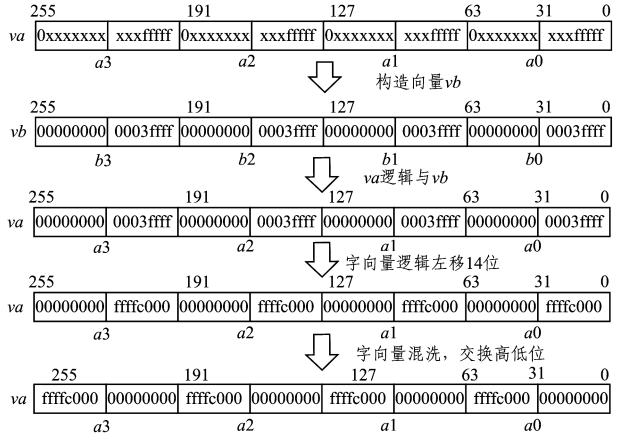

图 4　利用字向量移位、字向量混洗、向量逻辑运算等指令完成向量移位操作的流程图

Fig. 4　Diagram of realizing long word vector shift function using word vector shift, shuffle and logic instructions

2.3.2　fcvtds 指令的向量化

fcvtds 是一条标量指令，把双精度浮点数转为单精度浮点数。由于没有对应的 SIMD 指令，需要首先把向量拆分为 4 个标量，执行 4 次 fcvtds 指令，再进行 4 次合并，共需 12 条指令。

根据浮点数在寄存器中的存放格式，fcvtds 是把双精度浮点的低 29 位置 0，第 30 位根据第 29 位进行舍入。因此，可以对第 29 位加 1 完成舍入处理，再把低 29 位置 0。把向量 va 中的 4 个双精度浮点数转为单精度浮点数的具体过程是：

Step 1　生成一个向量 $v1$，每个分量的值为 0x10000000，即低 28 位全为 0，第 29 位为 1。

Step 2　生成一个向量 $v2$，每个分量的值为 0x1fffffff，即低 29 位全为 1。

Step 3　把两个向量 va 和 $v1$ 相加，得到 va。

Step 4　对向量 va 与向量 $v2$ 做逻辑与非操作得到向量 vb。

图 5 中，左图采用拆分合并操作，右图是向量化的实现，可以看出，处理过程从 12 步简化为 4 步，简化了代码，提高了性能。

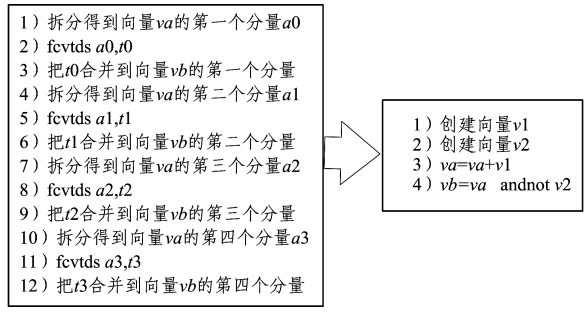

图 5　fcvtds 指令的向量化

Fig. 5　Vectoring of fcvtds instruction

以上通过移位指令、双精度转单精度指令的向量化方法给出了利用逻辑功能等价把标量指令替换为向量指令的思路。在函数的向量化过程中，采用同样的思路和方法，可以对其他标量指令进行相似的向量化工作。

3　实验及结果分析

为了验证本文方法的有效性，以神威太湖之光计算机系统的申威 26010 众核处理器作为测试平台。申威 26010 众核处理器是异构众核结构，从核的向量化指令比主核的向量化指令更为精简，向量化难度更大。本文的测试都是针对从核的向量化函数进行的。

测试使用原有的方法 1（指标量指令用对应的向量化指令替换，不能直接向量化的指令采取拆分合并的方法）与本文提出的方法 2（指本文提出的向量化方法）进行对比。

实验从正确性和性能两方面进行测试。正确性测试使用相同的测试用例分别对方法 1 和方法 2 的运行结果进行对比；性能测试包括两项：1）对每种向量化方法进行逐一测试，并与方法 1 对比；2）综合使用方法 2 对部分函数的整体性能提升进行测试。

采用的性能测试方法是：计算被测函数运行的节拍数。输入取函数常用区间的随机数，运行 200 次，取平均值。性能提升的计算公式如下：

$$\text{性能提升}=(\text{方法 1 性能}-\text{方法 2 性能})\ /\ \text{方法 1 性能}\times 100\%$$

3.1　各种向量化方法的正确性测试

分别使用随机产生的小数据（1 万）和大数据（2100 万）作为测试用例，方法 2 的运行结果与方法 1 的结果保持一致。该测试结果表明：方法 2 与方法 1 的逻辑功能等价。

3.2　单项向量化方法的性能测试

关于数据预处理过程向量化的测试，有两点需要说明。1）对结果向量的判断中，主核提供了整数向量比较和数 1 指令，非常方便获取输入中是否存在非正常输入，而从核不支持这些指令，仍需要进行拆分操作；2）每个函数的预处理过程不同，预处理在整个函数性能中所占比例越大，向量化后效果越明显。因此，该项目的性能测试是在主核上针对最简单的预处理进行的，目的是反映拆分拼接方法与向量化方法的性能差异。

从表 1 中的测试结果可知，各项向量化方法都比之前的拆分拼接方法的性能有不同程度的提高。

表 1 单项向量化方法对函数性能的提升

Table 1 Performance improvement of functions using single item vectorization method

no	item	clocks (before)	clocks (after)	improvements/%
1	data pre-processing	58	50	13.79
2	shift function	32	21	34.38
3	fcvtds function	37	21	43.24

3.3 综合使用多种向量化方法的测试

数学库中函数类型众多，下面以 pow，exp，log，sqrt，log10，asinf，tanh，round，atan 共 9 个典型函数的从核向量化版本进行性能测试，结果如表 2 所列。

表 2 部分典型函数综合使用多种向量化方法后性能的提升

Table 2 Performance improvement of some typical functions using multiple vectorization methods

simd-function	clocks (before)	clocks (after)	improvements/%	optimizations
simd_exp	260	92	64.6	data pre-processing, shift, fcvtds
simd_pow	390	247	36.7	data pre-processing, shift, fcvtds
simd_asinf	184	147	20.1	data pre-processing, shift, fcvtds
simd_tanh	247	236	3.64	shift
simd_log10	222	189	14.9	shift
simd_sqrt	58	55	5.17	data pre-processing
simd_round	71	72	−1.41	data pre-processing
simd_log	175	190	−8.57	data pre-processing
simd_atan	147	153	−4.08	data pre-processing

从表 2 的测试结果来看，大致分为以下几种情况：

1）exp，pow，asinf 等函数的性能明显提升。这些函数中存在许多移位指令、fcvtds 指令，在把拆分合并操作改为向量化实现后，性能有大幅提升。

2）tanh，log10，sqrt 等函数的性能略有提升。这些函数使用的指令基本都有对应的 SIMD 指令支持，只对少量的移位指令做了向量化处理，因此，性能改进不明显。

3）round，log，atan 等函数的性能有所下降。这些函数只做了预处理的向量化。方法 1 的预处理过程是先拆分，再处理，逐个判断是否存在特殊数；方法 2 的预处理过程是对逐个处理操作进行向量化，然后通过拆分判断是否存在特殊数（从核不支持向量数 1 指令）。相比于方法 1，方法 2 中的向量化操作会提升性能，而提升多少取决于操作的个数和复杂度，通常来说，操作越多，性能提升越大；并且方法 2 中的拆分操作会降低性能。正如 round，log，atan 函数，拆分对性能的影响超过向量化操作对性能的影响，因此总体性能有所下降。对于这类函数，实际测试结果说明预处理不适合做向量化。

综合以上情况的分析，可以得出：移位、fcvtds 等指令向量化对性能的提升比较明显，预处理是否向量化与函数的预处理流程有关，需要进行实际分析和测试。一般来说，预处理过程越复杂，涉及的操作越多，越适合做向量化；反之，预处理过程越简单，涉及操作越少，不适合做向量化。

结束语 本文通过对多种典型函数的向量化工作，提出了一种既有效又通用的向量化方法，该方法包括确定核心代码段、数据预处理过程向量化及指令向量化 3 个步骤。该方法在快速进行向量化的同时，保证了向量化函数的高性能。下一步工作将主要围绕更多指令向量化的自动化工作展开。

参 考 文 献

[1] LIU Y，ZHANG D H，ZHAO X B，et al. A Rapid Parallel ART Based on SIMD Technology[J]. Journal of Image and Graphics，2007，12(1)：73-77.（in Chinese）
刘远，张定华，赵歆波，等. 一种基于 SIMD 技术的快速并行代数重建算法[J]. 中国图像图形学报，2007，12(1)：73-77.

[2] VAN DER HOEVEN J，LECERF G，QUINTIN G，et al. Modular SIMD arithmetic in Mathemagix [J]. ACM Transactions on Mathematical Software，2014，43(1)：5.

[3] XIE Q C，ZHANG Y Q，WANG K，et al. Research of the SIMD and Vector Math Library[J]. Computer Science，2011，38(7)：298-301.（in Chinese）
解庆春，张云泉，王可，等. SIMD 技术与向量数学库研究[J]. 计算机科学，2011，38(7)：298-301.

[4] ZHANG Y Q，SUN J C，YUAN G X，et al. Perspectives of China's HPC system development：a view from the 2009 China HPC TOP100 list[J]. Frontiers of Computer Science in China，2009，4(4)：437-444.

[5] 解庆春，张云泉，鲁永泉，等. SW_VML：基于神威蓝光处理器的向量数学软件包[C]// 2013 全国高性能计算学术年会论文集. 桂林：中国计算机学会，2013.

[6] PARRI J，SHARIRO D，BOLIC M，et al. Returning Contrl to the Programmer：SIMD Intrinsics for Virtual Machine[J]. Communications of the ACM，2011，54(4)：38-43.

[7] LIU H，LIU F F，ZHANG P，et al. Optimization of BLAS Level 3 Functions on SW1600[J]. Computer Systems and Applications，2016，25(12)：234-239.（in Chinese）
刘昊，刘芳芳，张鹏，等. 基于申威 1600 的 3 级 BLAS GEMM 函数优化[J]. 计算机系统应用，2016，25(12)：234-239.

[8] WANG D. The Research on SIMD Compilation Optimization [D]. Hangzhou：Zhejiang University，2008.（in Chinese）
王迪. SIMD 编译优化技术研究[D]. 杭州：浙江大学，2008.

[9] ZHAO W X，ZHANG X D，LEMIRE D. A General SIMD-Based Approach to Accelerating Compression Algorithms[J]. ACM Transactions on Information Systems，2015，33(3)：1-28.

[10] ZHOU H，XUE J L. A Compiler Approach for Exploiting Partial SIMD Parallelism[J]. ACM Transactions on Architecture and Code Optimizaiton，2016，13(1)：1-26.

[11] 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.（in Chinese）
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.

[12] 郭绍忠，许瑾晨，陈世淼. SIMD 优化中的指令等价替换实现方法[C]// 河南省计算机学会 2011 年学术年会. 2011.

[13] ASHER Y B，ROTEM N. Hybrid Type Legalization for a Sparse SIMD Instruction Set[J]. ACM Transactions on Architecture and Code Optimizaiton，2013，10(3)：11.

[14] 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. 1996：222-233.