计算机科学
COMPUTER SCIENCE

面向申威平台的SIMD编程接口设计与研究

姜军, 顾晓阳, 徐坤坤, 吕勇帅, 黄亮明

引用本文

姜军, 顾晓阳, 徐坤坤, 吕勇帅, 黄亮明. 面向申威平台的SIMD编程接口设计与研究[J]. 计算机科学, 2025, 52(6): 66-73.

JIANG Jun, GU Xiaoyang, XU Kunkun, LYU Yongshuai, HUANG Liangming. Design and Research of SIMD Programming Interface for Sunway [J]. Computer Science, 2025, 52(6): 66-73.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240700009

面向申威平台的 SIMD 编程接口设计与研究

姜　军　顾晓阳　徐坤坤　吕勇帅　黄亮明
无锡先进技术研究院　江苏 无锡 214122
(goodsun_jj@163.com)

摘　要　在国产申威处理器中,申威 GCC 编译器在对程序进行向量化时,使用自动向量化和内嵌汇编的方式很难对某些复杂的程序进行向量化,阻碍了国产申威处理器的性能发挥。针对部分程序不能向量化的问题,在申威 GCC 编译器中进行 SIMD 编程接口的设计与研究。在申威向量指令的基础上,通过在申威 GCC 编译器中添加向量机器模式和向量数据类型,编译器可以对向量参数类型进行识别。根据向量指令的类型和复杂度,分别使用内建函数扩展、操作符扩展和高级语言扩展 3 种方式实现 SIMD 编程接口函数。在后端添加不同的指令模板,使接口函数可以匹配相应的指令模板,生成对应向量指令的汇编代码。通过对 FFTW 库和 Hyperscan 库进行测试和分析,相比优化前的程序,使用 SIMD 编程接口进行向量化后,FFTW 中 Double 类和 Float 类型程序的平均加速比分别为 1.97 和 2.13,Hyperscan 的平均加速比为 2.94。

关键词:向量化;SIMD 编程接口;向量指令;内建函数;指令模板
中图分类号　TP314

Design and Research of SIMD Programming Interface for Sunway

JIANG Jun,GU Xiaoyang,XU Kunkun,LYU Yongshuai and HUANG Liangming
Wuxi Institute of Advanced Technology,Wuxi,Jiangsu 214122,China

Abstract　In the domestically-produced Sunway high-performance systems,the Sunway GCC compiler finds it is challenging to vectorize complex programs using methods such as automatic vectorization and inline assembly during the compliation process,impeding the performance of domestically-produced Sunway processors. To address the issue of non-vectorizable programs,research and design of SIMD programming interfaces have been conducted within the Sunway compiler. By adding vector machine modes and vector data types in the Sunway GCC compiler based on Sunway vector instructions,the compiler can recognize vector parameter types. Depending on the type and complexity of the vector instruction,different vector instructions are expanded using intrinsic functions,operator expansion,and advanced language expansion,thereby implementing SIMD programming interface functions. Adding different instruction templates to the backend,so that the appropriate instruction templates can be matched,generating assembly code for the corresponding vector instructions. By testing and analyzing the FFTW library and Hyperscan library,it finds that after vectorizing the programs using SIMD programming interfaces,the average acceleration ratios for the FFTW library are 1.97 for the Double class and 2.13 for the Float type,while the average acceleration ratio for Hyperscan is 2.94.

Keywords　Vectorization,SIMD programming interface,Vector instruction,Instrinsic function,Instruction template

1　引言

在计算机领域的早期发展中,冯·诺伊曼首次提出了并行计算技术的概念[1]。然而,真正的飞跃发生在第一台高性能计算机 Cray-1 问世之时[2],这使得并行技术成为实现更高性能的主要途径。随着超级计算机如神威·太湖之光[3]等的出现,提供了超大规模的并行计算资源,其性能功耗比得以显著提高,推动了高性能计算的进一步发展。然而,随着程序规模的不断增大和计算需求的不断增加,如何通过各种并行技术更好、更充分地利用大规模并行计算资源成为了一个待解决的问题。

并行技术按照并行粒度通常分为指令级并行、线程级并行和数据级并行[4]。指令级并行一般指不存在相关性的指令序列中潜在的并行性[5]。线程级并行使 CPU 同时执行多个线程[6]。数据级并行指一次对多个数据进行相同处理,即单指令多数据(Single Instruction Multiple Data,SIMD)向量并行[7]。SIMD 向量并行的基本概念是将多个数据放入一个向量元素中,并应用特定的向量运算来计算结果,以实现数据级并行性。向量元素的数量表示可以同时执行多少个操作。向量的宽度、数据类型、向量寄存器的数量和算法的复杂性指定向量元素和对 SIMD 编程影响的重要因素。

当前各大主流处理器都具备了 SIMD 向量扩展。在 1996 年,Intel 首次提出了 MMX 指令集[8],并在处理器中集

到稿日期:2024-07-01　返修日期:2024-11-22
通信作者:黄亮明(liangming_huang@126.com)

成了 SIMD 向量扩展，支持 64 位整数向量操作。之后不断改进向量部件，推出一系列指令集，包括 SSE，AVX（Advanced Vector Extensions）[9]，AVX-512 等，这些指令集可以支持更大的向量长度，最高达 512 位。这些指令集广泛应用于各种应用领域，包括加密解码、科学计算优化等。此外，ARM 于 2017 年推出了适用于其体系结构的 SVE（Scalable Vector Extension）技术，使用可变向量长度寄存器[10]，支持可变的向量长度，以实现更好的性能。国产申威处理器也支持 SIMD 向量操作，采用 RISC 架构的申威 3231 处理器，支持向量宽度为 256 位的向量操作，极大地提升了处理器的性能。

目前实现 SIMD 向量并行主要有 3 种方式，自动向量化、内嵌汇编和 SIMD 编程接口[11]。3 种实现方式如图 1 所示。图 1(a)给出了使用自动向量化的源码和汇编代码；图 1(b)给出了使用内嵌汇编的源码和汇编代码；图 1(c)给出了使用 SIMD 编程接口的源码和汇编代码。

```
……                          ……
for(i=0; i<4; i++){         vlds $f12,32($30)
c[i]=a[i] * b[i];           vlds $f14,64($30)
}                           vmuls $f12,$f14,$f10
……                          vsts $f10,96($30)
                            ……
C 代码                          汇编
```

(a)自动向量化

```
……                                  ……
__asm__ __volatile__(
    "vlds $f12,0(%0)\n"
    "vlds $f14,0(%1)\n"
    "vmuls $f12,$f14,$f10\n"
    "vsts  $f10,0(%2)\n"
    :                               vlds $f12,32($30)
    :"r"(a),"r"(b),"r"(c)           vlds $f14,64($30)
    :                               vmuls $f12,$f14,$f10
);                                  vsts $f10,96($30)
……                                  ……
C 代码                                  汇编
```

(b)内嵌汇编

```
                                    ……
                                    vlds $f12,0($30)
                                    vlds $f14,32($30)
……                                  vmuls $f12,$f14,$f10
c=simd_vmuls(a,b);                  vsts $f10,0($1)
……                                  ……
C 代码                                  汇编
```

(c)SIMD 编程接口

图 1　3 种向量化方式

Fig. 1　Three vectorization methods

自动向量化在编译器中引入了 SIMD 指令的自动生成，目前大多数编译器，如 GCC（Gnu Compiler Collection）编译器和 LLVM（Low-level Virtual Machine）编译器[12]，都具有自动向量化能力。但是对一些复杂的算法和数据集，自动向量化就难以实现。

内嵌汇编是一种手动实现向量化的方法，是利用 ISA（Instruction Set Architecture）汇编语法[13]在 C/C++等高级语言中添加 SIMD 指令的低级编程方法。这种方法不受编译器的限制，能根据不同的任务编写特定的汇编代码，理论上可以实现最高程度的向量化。但是内嵌汇编需要深入了解目标硬件的 SIMD 指令集和汇编语言编程，代码编写困难，使用难度较高。

SIMD 编程接口提供了一个可以在高级语言编程中使用的函数接口，让编译器能够明确将函数映射到 SIMD 指令中，从而充分发挥 SIMD 的能力。与内嵌汇编相比，SIMD 编程接口更接近高级编程语言，程序员可以更容易理解和维护 SIMD 代码，而无需深入研究汇编语言。通常情况下，它在处理器的单个核心上能够显著提高性能。

目前国产申威处理器难以实现一些复杂程序的自动向量化，内嵌汇编的方法对程序员要求又相对较高，导致这些程序不能向量化，影响了国产申威处理器的性能。本文通过在国产申威编译器中实现 SIMD 编程接口，简化了并行计算的编程过程，不仅使程序高程度向量化，而且具有较高的可移植性，可以充分发挥国产申威处理器的并行计算能力。

2　研究背景

当前各大主流处理器平台，如 Intel，ARM 等，均在其架构中引入了 SIMD 编程接口。Intel 的多个 SIMD 扩展（如 SSE，AVX，AVX-512）使用 IPM（Intrinsic Programming Model）[14]编程模型来实现 SIMD 编程接口。IPM 利用 Intel 处理器的高级特性和指令集扩展来优化程序性能，通过提供一组专门的内建函数，使开发者能够直接调用底层处理器的指令和功能。Intel 的 SIMD 编程接口涵盖了广泛的运算、数据加载和存储操作，以及复杂的数据排列与整理操作。例如，_mm256_permute_ps 函数允许对向量中的元素进行灵活的重排。通过这些 SIMD 编程接口，Intel 平台能够加速多媒体处理、科学计算和机器学习等复杂应用。

ARM 架构同样在其处理器中集成了 SIMD 扩展，主要包括 NEON 和 SVE 指令集。ARM 通过 SIMD 编程接口，提供了便捷的方式来利用这些 SIMD 指令[15]。NEON 支持 128 位向量寄存器操作，其支持的编程接口如 vaddq_f32 允许在单条指令中对 4 个 32 位浮点数进行加法操作，广泛应用于图像处理、信号处理和其他嵌入式任务中。SVE 支持灵活的向量长度（128～2 048 位），支持的 SIMD 编程接口也可以自动适应不同向量的长度，使得开发者能够编写更加灵活和高效的并行代码。

相比当前主流处理器平台，申威处理器目前尚未对 SIMD 指令集提供标准化的接口支持。本文旨在基于申威 SIMD 向量指令集，在申威处理器上设计并实现 SIMD 编程接口，不仅能充分挖掘该处理器的并行计算潜力，而且能够更加便捷地利用申威处理器的硬件优势，提升在多媒体处理、科学计算等领域的计算效率。

3　申威 SIMD 编程

3.1　申威 SIMD 指令集

申威处理器采用 256 位的向量寄存器，一次可以存储 8 个字或者 1 个 256 位 8 字整数类型数据，也可以存储 4 个单精度浮点或 4 个双精度浮点类型数据。申威 SIMD 指令集可

以分为4类，除了常用的SIMD存储装入指令、SIMD整数运算指令和SIMD浮点运算指令外，还包括SIMD整理指令等，可以应用于各种复杂情况的数据处理。

3.2 申威SIMD编程接口

申威处理器为实现SIMD编程接口，设计了高效简洁的SIMD扩展类型和扩展函数，编译过程中把对SIMD的处理融合进各级编译优化阶段，使程序获得SIMD编程和编译两重优化效果。图2(a)所示的代码是一个可以用来SIMD向量化的典型C代码。

```
float a[4],b[4];
for(I=0; i<4; i++) {
b[i]=sqrt(a[i]);
}
```

(a)C代码

```
floatv4 va,vb;
vb=simd_vsqrts(va);
```

(b)向量后C代码

图2 向量化前后C代码

Fig. 2 C code before and after the vectorization

图2(a)中C代码通过一个循环对4个单精度浮点类型的数据进行平方根运算。使用SIMD编程接口进行编程，向量化的代码如图2(b)所示。通过使用simd_vsqrts接口，可以把循环中的4次求平方根操作使用vsqrts向量运算一次完成，有效减少了指令数量，提高了资源利用率，从而提升了程序性能。

3.3 申威GCC向量代码生成

申威GCC编译器的代码生成主要涉及源代码、中间语言和汇编代码3部分。标量代码生成如图3所示，高级语言标量代码通过词法分析和语法分析把标量树节点转换为抽象语法树(Abstract Syntax Tree，AST)[16]。AST进一步规范并且转换为前端语言无关的表示，即GIMPLE，GIMPLE在进行一系列的优化后，使用机器描述文件中的标量RTL模板进行构造，生成寄存器传输语言(Register Transfer Language，RTL)[17]，RTL又通过匹配相应标量指令模板生成相应标量指令的汇编代码。

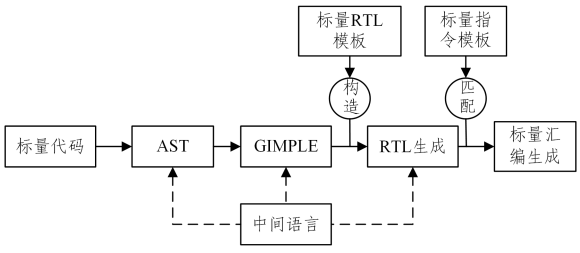

图3 标量代码生成

Fig. 3 Scalar code generation

向量代码生成需要在前端添加相应的向量节点，使GCC编译器可以对向量数据进行解析。其次在后端添加向量RTL模板和向量指令模板，GIMPLE根据向量RTL模板和向量指令模板进行构造和匹配，最终生成向量汇编代码。向量代码生成如图4所示。

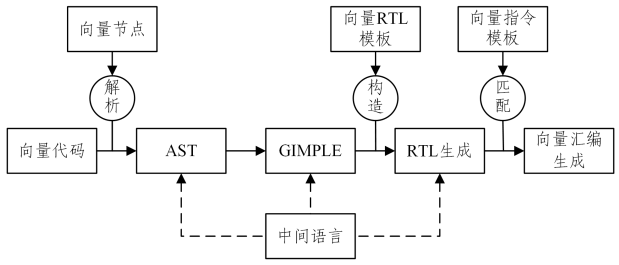

图4 向量代码生成

Fig. 4 Vector code generation

4 申威SIMD编程接口实现

在申威编译器中，SIMD编程接口主要通过内建函数扩展、操作符扩展和高级语言扩展实现。但在使用这3种方法前，要先在编译器中添加不同的向量数据类型，让SIMD编程接口可以识别向量参数。然后，根据向量指令的复杂度及其特性，选择不同实现方法。根据向量指令类型和向量指令的特性，对不同类型的向量参数进行类型转换。最后，在后端添加对应指令模板并构造INSN(在GCC中描述程序代码信息的RTL表达式被称为INSN)，生成汇编指令。其实现流程如图5所示。

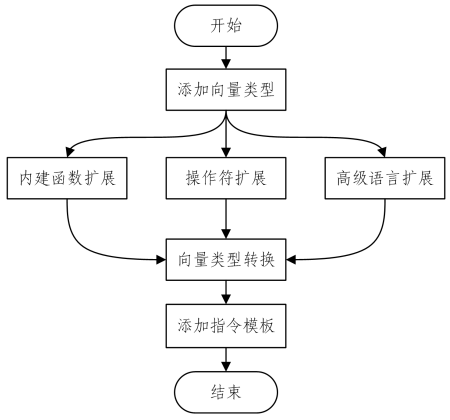

图5 SIMD编程接口实现流程

Fig. 5 SIMD programming interface implementation process

4.1 定义向量类型

因为向量指令使用的都是多数据的向量数据，所以需要对向量数据类型进行定义。对向量数据类型进行定义前，需要先定义向量机器模式[18]。在申威GCC中，向量机器模式一般有整型向量和浮点向量两种，其中整型向量机器模式有V8SI和V1OI，分别与字和8字标量机器模式对应。整型向量机器模式中V1OI对应的向量类型为int256，表示一次可以对256位的整型数据进行处理。浮点向量机器模式有V4SF和V4DF两种，对应单精度向量和双精度向量的机器模式。申威处理器需要根据不同的向量机器模式定义不同的向量数据类型，标量和向量类型的对应关系如表1所列。

表1 标量和向量类型的对应关系

Table 1 Correspondence between scalar and vector types

标量类型	机器模式	向量类型
int	V8SI	intv8
float	V4SF	floatv4
double	V4DF	doublev4

4.2　内部接口实现

根据 SIMD 编程接口的通用性，不同的指令可以采用不同的 SIMD 接口实现方式。表 2 列出了 3 种方法涉及的接口和向量指令情况。

表 2　不同方法涉及的接口和指令

Table 2　Interfaces and instructions involved in different methods

实现方式	接口数量	指令数量
内建函数扩展	133	68
操作符扩展	49	46
高级语言扩展	12	0

4.2.1　内建函数扩展

对于其他功能相对复杂的向量指令，如向量求最值、向量插入指令等，采用经典的内部 builtin 函数方式实现支持，该方法虽然灵活性不高，但是能够实现比较复杂的功能，完成对复杂向量指令的支持。内建函数实现流程如图 6 所示。

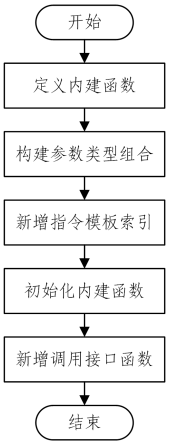

图 6　内建函数实现接口流程

Fig. 6　Process of implementing interfaces with built-in functions

使用内建函数实现 SIMD 编程接口需要先添加相应向量指令的内建函数的定义。对内建函数的定义如图 7 所示。

```
struct sw_builtin_def {
    const char * name;
    enum sw_64_builtin code;
    unsigned int target_ mask;
    enum rid keyword;
    enum sw_ftypef type;
    int arg_num;
}
```

图 7　内建函数定义

Fig. 7　Definition of built-in functions

以浮点向量平方根指令(vsqrts)为例，其内建函数声明如表 3 所列。

表 3　vsqrts 指令声明

Table 3　Declaration of vsqrts instruction

属性	vsqrts 声明
name	builtin_sw_vsqrts
code	SW_BUILTIN_VSQRTS_V4SF
target_mask	MASK_SW
rid	RID_SIMD_NAME
ftype	V4SF_FTYPE_V4SF
arg_num	1

从图 7 和表 3 可以看出，内建函数有 6 个属性，其中 name 表示内建函数的名称，对内建函数的处理都是通过 name 进行；枚举类型 code 作为索引获取内建函数对应指令的 INSN 代码，与后端指令模板的名称一一对应；target_mask 用于标记适用于哪个目标机器，如果内建函数的 target_mask 和当前机器的目标码不一样，则进行其他处理；rid 用于关键字的识别，一般变量或者函数为“RID_SIMD_NAME”，而访存类关键字为“RID_SIMD_LOAD”；ftype 定义了内建函数参数的数量和类型组合，2 个 V4SF 分别表示输入和输出的参数类型，其中 FTYPE 前的是输出参数；arg_num 表示输入参数的数量。

考虑对已有程序的兼容性问题，对于访存指令的接口函数，使用关键字的方式实现 SIMD 接口。其流程如图 8 所示。

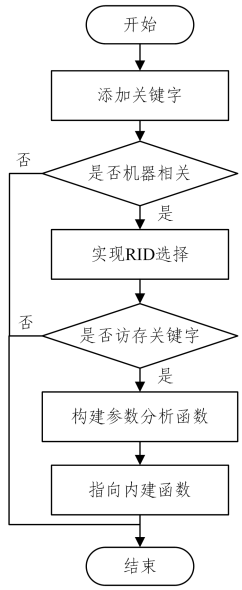

图 8　访存向量指令 SIMD 接口实现

Fig. 8　SIMD interface implementation for memory access instruction

在实现访存指令接口时，要添加对应访存关键字。如对向量装入指令接口 simd_load 的关键字定义为“builtin_sw_load”。申威 GCC 编译器在对访存关键进行处理时，添加机器相关的判断，并根据关键字名称实现 rid 掩码值的获取。根据 rid 掩码的值，实现分析函数，对关键字的参数列表进行分析，根据不同的参数类型获取对应参数类型的访存内建函数。

获取到内建函数后，可以根据内建函数的 code 寻找指令模板并构造 INSN，然后进行指令模板匹配，生成汇编代码。

4.2.2　操作符扩展

对常见的向量操作，如算术操作、移位操作、逻辑操作等，可以借用内部操作符完成对应的向量操作。例如，字向量加法的接口函数如图 9 所示。

```
static __inline intv8 __attribute__((__always_inline__))
simd_vaddw(intv8 __A,intv8 __B)
{
   return(intv8)(__A+__B)
}
```

图 9　字向量加法接口

Fig. 9　Add vector interface for word

从图 9 中可以看出，加法操作符“+”通过对两个 intv8 的字向量数据进行相加然后返回来实现字向量加法接口。此种方法相较于内建函数的方式相对灵活自由，效率较高。对一些不常见的简单操作，只要编译器本身包含对应物理意义的内部操作符，也可通过添加操作符实现对应的 SIMD 接口。

4.2.3 高级语言扩展

set 和 print 这类向量接口由于没有相应的向量指令，可以通过相关的高级语言来实现。根据向量类型的不同，需要编写不同向量类型的接口。intv8 类型的 set 接口函数如图 10 所示。

```
simd_set_intv8 (int__S,int__T,int__U,int__V,int__W,int__X,int__Y,
                int__Z){
union{
   int __a[8] __attribute__((aligned(32)));
   intv8 __v;
} __u;
__u.__a[0]=__S;__u.__a[1]=__T;
__u.__a[2]=__U;__u.__a[3]=__V;
__u.__a[4]=__W;__u.__a[5]=__X;
__u.__a[6]=__Y; __u.__a[7]=__Z;
return __u.__v;
}
```

图 10 set 接口函数

Fig. 10 Function interface of set

从图 10 中可以看出，set 函数接口是通过 C 语言的 union 结构为传进来的 8 个 int 类型数据分别赋值，达到向量 set 的目的。

4.3 向量类型转换

因为 SIMD 接口可以直接插在高级源代码中使用，在对 SIMD 向量接口的参数赋值时，传入接口的参数类型可能不同，所以需要对赋值的参数类型进行处理。

根据赋值前后参数数据类型是否相同，可以分为相同数据类型赋值和不同数据类型赋值两类。其中相同数据类型赋值主要是符号间的转换。不同数据类型赋值又可以分为向量间赋值、向量和标量间赋值两种类型。需要根据数据类型来选择用于类型转换的内建函数。其中标量到向量的赋值，是把标量进行复制扩展到相应的元素位置中，如图 11 所示。

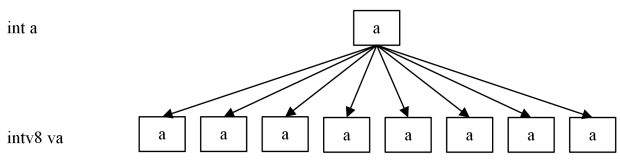

图 11 标量转向量

Fig. 11 Scalar convert to vector

图 11 中，int 类型的 a 赋值给了 intv8 类型的 va，编译器会把 a 同时扩展到 va 中对应的 8 个字元素位置中，最后 va 的值为{a,a,a,a,a,a,a,a}。

4.4 添加指令模板

在使用 SIMD 编程接口时，会根据内建函数的 code 找到对应的指令模板，使用构造函数构造 INSN，根据 INSN 进行模板匹配，生成汇编代码。vsqrts 指令模板如图 12 所示。

```
(define_insn "*sqrtv4sf2"=
[(set(match_operand:V4SF 0"register_operand="&f"=
  (sqrt:V4SF=
    (match_operand:V4SF 1"general_operand""f"))]=
  "flag_sw_sdsame == 0"=
  "vsqrts %R1,%0"=
  [(set_attr "type" "fsqrt")])
```

图 12 vsqrts 指令模板

Fig. 12 Template of the vsqrts instruction

根据向量求平方根内建函数的 fcode，获取对应指令模板的名称 vsqrtsv4sf2，通过指令模板名称在机器描述文件中找到图 12 中的 vsqrts 的指令模板，使用指令模板中的 RTL 模板进行操作数替换来构造 INSN，INSN 通过指令模板匹配完成代码生成，最终得到 vsqrts 的汇编代码。

5 实验结果与分析

5.1 实验配置

本文基于申威 SIMD 指令集在申威 GCC-8.3.0 编译器上实现申威 SIMD 编程接口，并在国产申威处理器上进行了优化效果评估。服务器采用国产多核 SW3231 处理器，软硬件参数如表 4 所列。

表 4 软硬件环境参数

Table 4 Software and hardware environment parameters

参数	参数值
处理器	SW3231
指令集架构	SW64
CPU 主频	2.4 GHz
核数	32
内存	256 GB
内核版本	4.19.180
操作系统	UOS Server 20 Military

5.2 测试程序

测试程序使用 FFTW(Fastest Fourier Transform int the West)库[19]和 Hyperscan[20]进行实验，通过使用申威 SIMD 编程接口，对两个程序中的标量代码进行向量化处理，来评估两者在不同任务中的性能表现。

5.2.1 FFT 库

快速傅里叶变换(Fastest Fourier Transform，FFT)是一种高效计算离散傅里叶变换(DFT)的算法，FFTW 是一个实现 FFT 算法的开源软件库，提供了用于计算实数和复数的一维、二维和三维 FFT 函数，在不同硬件架构上有着高效性和自适应性等特征。FFTW 在许多领域中得到了广泛的应用，如信号处理、图像处理、科学计算、数据压缩等，适用于各种复杂的计算任务。

5.2.2 Hyperscan 库

Hyperscan 是一款由 Intel 开发的高性能正则表达式匹配库，专为需要处理大量数据和高吞吐量应用而设计。其核心优势在于，能够在现代处理器上利用 SIMD 向量指令集，如 AVX2 和 AVX-512，从而大幅提升正则表达式匹配的速度和效率。

5.3　实验数据分析

5.3.1　FFTW 实验结果

实验主要对一维 FFT 函数进行测试分析，使用 SIMD 向量化方法在申威平台上对不同数据点的 FFTW 库进行 Double 和 Float 类型向量化实现和测试，对标量代码和使用 SIMD 编程接口的向量代码进行对比分析，其中每个数据点表示多个 Double 或 Float 类型数据的集合。程序编译时采用-O3 选项编译，对比分析优化前后相同数据点的 GFLOPS (Giga Floating-point Operations Pre Second，每秒 10 亿次浮点计算)，并计算加速比。FFTW 库 Double 和 Float 类型优化前后不同规模的 GFLOPS 如图 13 和图 14 所示。

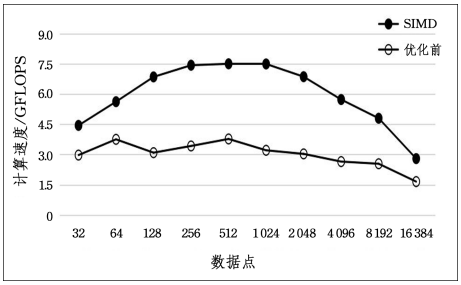

图 13　Double 类型优化前后 GFLOPS

Fig. 13　GFLOPS before and after optimization for Double type

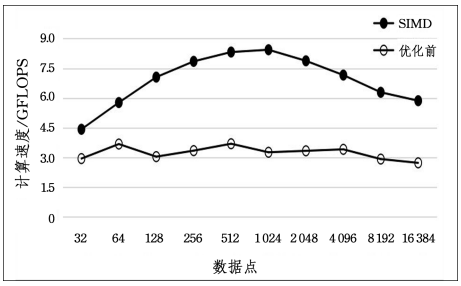

图 14　Float 类型优化前后 GFLOPS

Fig. 14　GFLOPS before and after optimization for Float type

从图 13 和图 14 中可以看出，Double 类型和 Float 类型的性能表现基本相同，优化前 GFLOPS 都成小幅度下降趋势，优化后的 GFLOPS 则随着数据点数量增大出现峰值，然后缓慢下降。图 15 和图 16 给出了两种类型的加速比。

从图 15 和图 16 可以看出，优化后的程序都有很高的加速比，其中 Double 类型和 Float 类型数据的平均加速比为 1.97 和 2.13，并且都在 1024 个数据点下有着最高加速比，分别为 2.34 和 2.58。两种类型程序的加速比会随着数据点的数量增加，呈现先上升再缓慢下降的趋势。

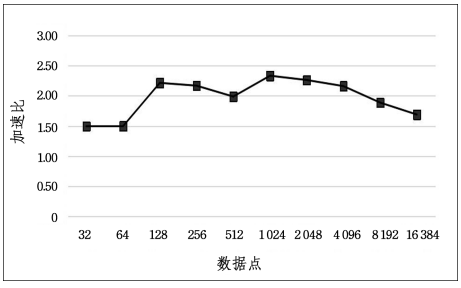

图 15　Double 类型不同数据点加速比

Fig. 15　Acceleration ratio for different data points in Double type

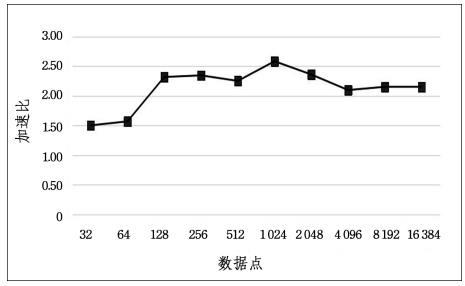

图 16　Float 类型不同数据点加速比

Fig. 16　Acceleration ratio for different data points in Float type

使用 gprof 性能分析工具对加速比最高的 3 种数据点数量的 Double 和 Float 类型程序进行分析，对比核心函数时间占比和所包含的浮点计算数量，Double 类型和 Float 类型的核心函数分析如表 5 和表 6 所列。

表 5　Float 类型的核心函数分析

Table 5　Analysis of core functions for Float type

数据点个数	加速比	核心函数	时间占比/%
256	2.34	t2fv_16	57
1024	2.58	t2fv_64	63
2048	2.36	t2fv_64	42

表 6　Double 类型的核心函数分析

Table 6　Analysis of core functions for Double type

数据点个数	加速比	核心函数	时间占比/%
128	2.22	n2fv_16	50
1024	2.34	t1fv_64	60
2048	2.26	t1fv_64	40

从表 5 和表 6 可以看出加速比最高的数据点，其程序核心函数拥有最多的浮点运算，并且同一核心函数的时间占比也高于其他数据点，因此使用 SIMD 编程接口对浮点运算进行向量化后，其加速效果更为明显。

对 Float 类型 t2fv_64 核心函数进行分析，其部分标量代码和对应向量化后的代码如图 17 所示。

```
E T7,Ti,Tn,Tq;
T7=T1+T6;
Ti=Tc+Th;
ri[WS(rs,2)]=T7-Ti;
ri[0]=T7+Ti;
Tn=Tk+Tl;
Tq=To+Tp;
ii[0]=Tn+Tq;
ii[WS(rs,2)]=Tq-Tn;
```

(a)标量代码

```
V Ta,Tb;
Ta=VADD(T1,T3);
Tb=VADD(T6,T8);
ST(&(x[WS(rs,2)]),VSUB(Ta,Tb),ms,
&(x[0]));
ST(&(x[0]),VADD(Ta,Tb),ms,&(x[0]));
```

(b)向量代码

图 17　核心函数标量代码和向量代码对比

Fig. 17　Comparison between scalar and vector code for core function

图 17(b)中，VADD 和 VSUB 被定义为 simd_vadds 和 simd_vsubs，是申威 SIMD 编程接口中关于单精度向量加法和单精度向量减法的接口。从图 17 中可以看出，相比使用标量代码，使用 SIMD 编程接口向量化后，可以一次性计算多个数据，缩短程序执行时间，有效提升程序的性能。

对 Float 类型和 Double 类型的数据进行向量化后，程序的理想加速比应为 4，但是由于程序在运行中需要执行多个

函数，每个函数中还存在一些其他开销，从核心函数的时间占比来看，程序的加速比在 2 左右是较为合理的。

5.3.2 Hyperscan 实验结果

实验主要利用申威 SIMD 编程接口对 Hyperscan 程序中的多个标量模块进行优化。从 Hyperscan 程序中选取了 10 个标量模块，并使用申威 SIMD 编程接口对不同类型的标量模块代码进行向量化。在程序编译时，采用了-O3 优化选项，并对比分析了优化前后标量模块的加速比。图 18 给出了 10 个标量模块优化前后的加速比。

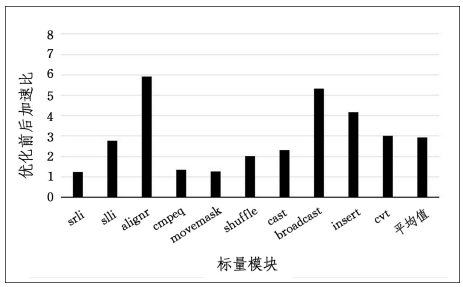

图 18 不同模块优化加速比

Fig. 18 Acceleration ratio of different module optimizations

从图 18 中可以看出，这 10 个标量模块在优化后均实现了正向加速，平均加速比为 2.94。其中，有 7 个标量模块加速比超过了 2，alignr 和 broadcast 接口模块加速比超过了 5。对 alignr 标量模块进行分析，其部分标量代码和对应向量化后的代码如图 19 所示。

```
……
for(i=0; i < 16-imm8; ++i) {
    ret.s_8[i]=b.s_8[i+imm8];
}
for(j=0; j < imm8; ++j,++i) {
    ret.s_8[i]=a.s_8[j];
}
……
```

(a)标量代码

```
……
ve=simd_set_int256(vb.i1[0],vb.i1[1],va.i1[0],va.i1[1]);
vg=simd_srlow(ve,imm8*8);
……
```

(b)向量代码

图 19 alignr 模块标量和向量代码对比

Fig. 19 Comparison of scalar and vector code for the alignr module

从图 19 中可以看出，这段代码主要是对 128 位数组 a 和 b 的提取合并操作，把 b 的高 8 字节放到 ret 的低 8 字节，并把 a 的低 8 字节放到 ret 的高 8 字节。优化后通过 simd_set_int256 和 simd_srlow 向量接口提取合并 a 和 b 的值，从而得到对应的 128 位 ret 数组。通过使用赋值和逻辑右移的 SIMD 编程接口，能够在一次操作中处理多个数据，有效地缩短 alignr 标量模块的运行时间，并显著提高程序的执行效率。

结束语 本文针对申威 GCC 编译器不能对某些复杂程序进行向量化的问题，基于申威 SIMD 向量指令，进行 SIMD 编程接口的设计与研究。通过实现不同的向量类型，使 SIMD 编程接口可以正常使用向量类型的参数。根据不同的向量指令，使用内建函数扩展、操作符扩展和高级语言扩展 3 种方式实现向量接口函数的构建。通过在后端添加相应向量指令的指令模板，使向量接口在程序编译过程中能够正常生成汇编代码。实验结果表明，本文实现的 SIMD 向量接口充分利用了并行计算资源，提高了国产申威处理器的计算能力。下一步的工作主要是继续对接口进行完善，使不同的程序可以更充分的向量化。

参 考 文 献

[1] ARIKPOI I, OGBAN F U, ETENG I E. Von neumann architecure and modern computers[J]. Global Journal of Mathematical Sciences, 2007, 6(2): 97-103.

[2] RUDSINSKI L, PIEPER G W. Evaluating computer program performance on the CRAY-1: ANL-79-9; TRN: 79-008828[R]. Argonne, IL: Argonne National Lab., 1979.

[3] DONGARRA J. Report on the Sunway TaihuLight System: UT-EECS-16-742 [R]. University of Tennessee, 2016.

[4] ASANOVICK, BODIK R, DEMMEL J, et al. A view of the parallel computing landscape[J]. Communications of the ACM, 2009, 52(10): 56-67.

[5] REDDY V, SUDHAKAR A, SIVAKUMAR P. Computing Performance Enhancement of VLIW Architecture Using Instruction Level Parallelism[J]. International Journal of Innovative Science and Research Technology, 2020, 5(9): 431-435.

[6] YIAPANIS P, BROWN G, LUJAN M. Compiler-Driven Software Speculation for Thread-Level Parallelism[J]. ACM Transactions on Programming Languages and Systems, 2015, 38(2): 1-45.

[7] LIMOUSINC, SEBOT J, VARTANIAN A, et al. Architecture optimization for multimedia application exploiting data and thread-level parallelism[J]. Journal of Systems Architecture, 2005, 51(1): 15-27.

[8] RAMAN S K, PENTKOVSKI V, KESHAVA J. Implementing streaming SIMD extensions on the Pentium III processor[J]. IEEE Micro, 2000, 20(4): 47-57.

[9] CEBRIANJ M, NATVIG L, JAHRE M. Scalability analysis of AVX-512 extensions[J]. The Journal of Supercomputing, 2020, 76(3): 2082-2097.

[10] ODAJIMA T, KODAMA Y, SATO M. Power performance analysis of ARM scalable vector extension[C]// IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS). IEEE, 2018: 1-3.

[11] GAO W, ZHAO R C, HAN L, et al. Research on SIMD Auto-Vectorization Compiling Optimization[J]. Journal of Software, 2015, 26(6): 1265-1284.

[12] FENG J G, HE Y P, TAO Q M. Evaluation of compilers' capability of automatic vectorization based on source code analysis [J]. Scientific Programming, 2021, 2021: 1-15.

[13] KONG M, VERAS R, SADAYAPPAN P. When polyhedral transformations meet SIMD code generation[C]// Proc. of the

34th ACM SIGPLAN Conf. on Programming Language Design and Implementation. ACM，2013：127-138.

[14] AMIRI H，SHAHBAHRAMI A. SIMD programming using Intel vector extensions[J]. Journal of Parallel and Distributed Computing，2020，135：83-100.

[15] BRAMASB. A fast vectorized sorting implementation based on the ARM scalable vector extension(SVE)[J]. PeerJ Computer Science，2021，7：e769.

[16] RACORDON D. From ASTs to Machine Code with LLVM [C]// Companion Proceedings of the 5th International Conference on the Art，Science，and Engineering of Programming. New York：ACM，2021：68-76.

[17] WANG X W，WANGK X，YANG Q S. Research and Development of Computer Based on GCC[M]// Recent Advances in Computer Science and Information Engineering. Berlin：Springer，2012：809-814.

[18] NOVILLO D. GCC an architectural overview，current status，and future directions[C]// Proceedings of the Linux Symposium. Ottawa：Linux Symposium，2006：185.

[19] FRIGO M，JOHNSON S G. FFTW an adaptive software architecture for the FFT[C]// Proceedings of the 1998 IEEE International Conference on Acoustics，Speech and Signal Processing. IEEE，1998：1381-1384.

[20] WANGX，HONG Y，CHANG H，et al. Hyperscan：A fast multi-pattern regex matcher for modern CPUs[C]// 16th USENIX Symposium on Networked Systems Design and Implementation. USENIX Association，2019：631-648.

JIANG Jun，born in 1980，M. S，senior engineer. His main research interests include compiler optimization and architecture-oriented performance analysis and optimization.

HUANG Liangming，born in 1988，Ph.D，engineer. His main research interests include compiler optimization and architecture-oriented performance analysis and optimization.

（责任编辑：喻藜）

欢迎申报 CCF 高性能计算专委会“青年学者激励计划”

由 CCF 高性能计算专委会发起的“CCF 高性能计算专委会青年学者激励计划”评选是为表彰年龄在 40 周岁（含 40 周岁）以下，在国内高等院校、科研院所、企业等单位从事高性能计算技术研究、教学、产品开发和工程应用并取得突出的成就的 CCF 高性能计算专业委员会委员。

2022 年，由 CCF 高性能计算专业委员会发起、设立，并负责承办的高性能计算“卓越青年”评选，按照 CCF 政策要求，2025 年度更名为“CCF 高性能计算专委会青年学者激励计划”，作为中国首个高性能计算技术领域的专业人才项目，即日起正式开通申报。

CCF 高性能计算专委会“青年学者激励计划”评选是为表彰年龄在 40 周岁（含 40 周岁）以下，在国内高等院校、科研院所、企业等单位从事高性能计算技术研究、教学、产品开发和工程应用并取得突出的成就的 CCF 高性能计算专业委员会委员。

旨在充分调动该领域内广大科技工作者的积极性、创造性和主动性，推动我国高性能计算技术的教育、研究开发与应用。

评选结果经公示无异议后将在 2025CCF 全国高性能计算学术年会（CCF HPC CHINA 2025）现场举办发布颁奖仪式。

申报说明

申报截止日期：7 月 20 日 24：00

申报方式：按要求填写申请表，发送至 LIXIDAI@ICT. AC. CN

联系人及联系方式：李希代　13693056420

“CCF 高性能计算专委会青年学者激励计划”评选将面向全体 CCF 高专委委员，根据如上标准遴选，欢迎申报！

据 CCF 专业委员会