面向间接数组索引的向量化方法

姚金阳 赵荣彩 王 琦 李颖颖
(数学工程与先进计算国家重点实验室 郑州 450001)

摘 要 对现有的编译器而言,间接数组索引不能被高效地向量化,这使得程序中包含有该类访存形式的间接数组索引不能利用SIMD扩展部件,这也是程序向量化研究中的热点问题。为了高效地利用SIMD扩展部件,充分挖掘程序中的向量化潜能,提出了一种对间接数组索引进行向量化的新方法,且提供了性能收益方法,分别对各种间接数组索引进行性能收益分析。实验结果表明,使用该向量化方法可以显著地提高程序的执行效率。

关键词 向量化,间接数组索引,收益分析,临时数组

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.09.036

Vectorization Methods for Indirect Array Index

YAO Jin-yang ZHAO Rong-cai WANG Qi LI Ying-ying
(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)

Abstract Indirect array index cannot be vectorized efficiently in the existing compiler. It makes the program which contains the indirect array index cannot take advantage of SIMD extension parts. It is a hot topic in research on procedure vectorization. In order to utilize the SIMD extension parts efficiently and excavate the vectorization potential in the program fully, a new vectorization method for indirect array index was proposed in this paper. The performance income method was provided so as to analyze the performance benefits for various indirect arrays index. The experimental results show that the vectorization method can significantly improve the efficiency of the execution of program.

Keywords Vectorization, Indirect array index, Cost-benefit analysis, Temporary array

1 引言

如今,主流处理器都已经集成了SIMD(Single Instruction Multiple Data)扩展部件,并使用该部件对程序进行加速。

使用SIMD扩展部件向量化程序的方式一般有两种:手工向量化和自动向量化[1]。手工向量化的操作较为复杂,不仅要求程序员熟练掌握不同应用程序的特点和不同处理器平台提供的SIMD扩展指令,还要求程序员掌握目标处理器的硬件结构,同时手工向量化后的程序的移植性、可读性也较差。相比而言,自动向量化方法不需要程序员考虑如何对程序进行向量化,减轻了程序员的负担。目前主流的程序向量化方法是自动向量化,许多编译器(例如INTEL的ICC编译器、GUN的GCC编译器以及ARM的RealView编译器)都可以支持自动SIMD代码的生成[2-4]。为了充分发挥向量化部件的优势,研究人员采取了多种方法[5-8]来分析程序并将可向量化的语句向量化。向量化程序时不仅要考虑语句间的依赖关系,还要考虑访存模式是否满足特定条件[9]。现有的通用编译器支持具有连续访存模式的向量化,但不对非连续访存和间接数组索引等做向量化处理;某些编译器即使能做向量化处理,也由于需要引入许多特殊的处理指令来加载有效数据到寄存器以及将寄存器中的数据存储到内存,从而耗时较长,严重降低了向量化的收益。

目前,已有一些针对间接数组索引向量化问题的研究[10-15]。文献[10]利用了处理器中特有的硬件来处理非对齐访存问题和不规则访存问题,以提高SIMD处理器的性能。若没有特有的硬件,则不规则访存问题依然不能得到解决。文献[11-12]利用处理器中集成的向量化指令集(例如向量加载、向量存储、混洗等)对非对齐访存和间接数组索引问题进行了处理,利用该方法可以向量化地计算数据地址,但是访问数据时仍为串行。文献[13]通过数据重组的方式对不规则访存进行向量化。文献[14]针对非连续或非对齐访存等问题提出了一种数学模型,并使用基于SLP的向量化代码生成方法、过程间数组填充等方式对该类问题进行优化。文献[15]针对结构体中存在的非连续和非对齐访存问题提出了结构体拆分模型,用来对存在于结构体数组中的非连续、非对齐访存问题进行向量化。

到稿日期:2017-11-07 返修日期:2018-01-23 本文受国家重点研发计划"高性能计算"重点专项(2016YFB0200503)资助。

姚金阳(1992-),男,硕士生,主要研究方向为高性能计算、先进编译,E-mail:yaojy1024@126.com;**赵荣彩**(1957-),男,博士,教授,CCF高级会员,主要研究方向为高性能计算、先进编译、反编译技术;**王 琦**(1992-),男,硕士生,主要研究方向为高性能计算、先进编译;**李颖颖**(1984-),女,博士生,讲师,主要研究方向为高性能计算、先进编译。

本文提出了一种处理间接数组索引向量化问题的有效方法，首先将间接数组赋值到临时数组中，然后将临时数组中的数据加载为向量数据，以进行向量化运算，运算结束后如需将其存入间接数组，则首先将向量存储到临时数组，再对间接数组依次赋值。该方法不依赖于特殊的硬件和特殊的 SIMD 扩展指令集，且实验证明使用该方法可以将程序中的间接数组索引向量化，进而提高 SIMD 扩展部件的利用率。

2　问题描述

间接数组索引问题是指该数组的索引是经另一个数组或另一个变量计算而来。图 1 列举了间接数组索引的加载和存储过程。对于这种存在不规则访存问题的数组，其在内存中存放的位置不连续且很难找出存储位置的规律。而向量的访存（主要指向量的加载或向量的存储）都是对在内存中顺序存储的数据进行操作。出于保守考虑，传统编译器直接放弃对该形式代码的向量化，错失了对该部分进行向量化的机会。

LOAD	STORE

for(n=0;n<nri;n++)
{
…
is3=3 * shift[n];
shX=shiftvec[is3];
…
}
	for(n=0;n<nri;n++)
	{
	…
	is3=3 * shift[n];
	shiftvec[is3]=a[n]+b[n];
	…
	}

for(n=0;n<nri;n++)
{
…
is3=3 * n+1;
shX=shiftvec[is3];
…
}
	for(n=0;n<nri;n++)
	{
	…
	is3=3 * n+1;
	shiftvec[is3]=a[n]+b[n];
	…
	}

图 1　间接数组索引示例

Fig. 1　Examples for indirect array index

程序的向量化会带来额外的开销，在一些情况下甚至会抵消向量化所带来的收益或者导致负加速。收益分析是衡量某段代码向量化后效率是否有提升的技术，也是 SIMD 自动向量化优化技术的一类。通过建立代价模型，设定向量化条件，对向量化收益进行评估。若分析结果有收益，则进行后续自动向量化代码的生成；若无收益，则放弃该段代码的向量化，保持串行执行。

3　面向间接数组索引的向量化技术

3.1　编译框架

自主自动向量化工具 SW_VEC 是国产申威服务器平台上 SWCC 编译器的重要组成部分，其流程框架如图 2 所示。SWCC 编译器是由开源 GCC 编译器改进而来。该编译器一般在中间代码层次上进行向量化，所采用的方式有基于依赖的循环级向量化、基于基本块的 SLP 向量化[16]和两者相结合的 loop-aware-slp 向量化[17]。本文所采用的是基于依赖的循环级向量化方法。含有间接数组索引的循环体首先进入 VECT-LOOP 阶段，在该阶段分析循环中的语句类型是否合法并判断是否有阻止向量化的数据依赖，同时通过循环分布技术将能做向量化的语句和不做向量化的语句分开；接着做预优化，比如删除无用代码，将语句合法地转换成 SIMD 向量化语句；最后通过收益分析阶段的权衡来决定是否采用该向量化方案。如果收益分析表明可以向量化，则进入 Vectorization_Transform 阶段，将标量代码完全转换为向量代码。

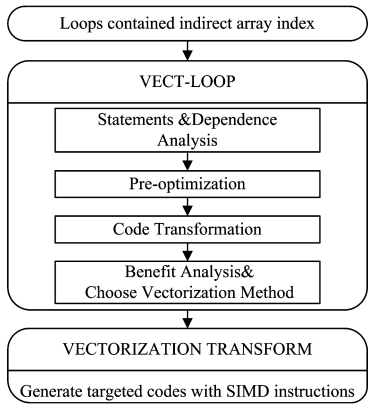

图 2　SW_VEC 上的 SIMD 代码生成流程

Fig. 2　SIMD code generation process on SW_VEC

为了最大程度地将程序向量化，使得含有间接数组索引的代码参与向量化操作，提出了一种有效的 SIMD 代码生成方法并将其应用在 VECT-LOOP 阶段。首先，在 VECT-LOOP 阶段保证即将向量化的具有间接数组索引的循环可以被正确执行，并判断向量化该段代码是否有收益；然后进入 Vectorization Transform 阶段，并在该阶段将待优化的标量代码转换为向量代码。

3.2　算法思想及原理

由于标量的间接索引数组的值在内存中存放的顺序是不连续的，并且向量访存指令大都是对内存中连续存放的数组进行存取操作，所以含有间接数组索引的数组在循环中不能直接地被向量化。如果将内存中不连续的数据进行一定的变换使之连续，那么就可以使用向量访存指令对其进行操作。

如今主流编译器 SIMD 扩展指令集早已提供了足够的数据整理指令对间接访存的数组进行整理，使之满足向量化的条件，但是这无疑增加了向量化的额外开销；另外，不同的处理器平台扩展的指令集也不同，增加了编译器上的自动向量化工具处理间接访存问题的难度。基于此，本文不使用额外的向量整理指令对间接数组进行处理，而在向量化间接数组之前使用标量的处理方式对其进行“预处理”。“预处理”之后的标量数据出现向量化机会时，我们就可以对其进行向量化。“预处理”针对的是处理间接数组的加载问题，而“尾处理”针对的是间接数组的存储问题。向量运算得出的结果在寄存器中是连续存放的，我们仍然不考虑使用向量整理指令整理其数据，而是将数据临时存放到寄存器中，使用标量的方式对间接数组索引问题进行处理。通过该方法，不仅提高了 SIMD 扩展部件对间接数组索引问题的向量化效率，而且降低了自动向量化工具对不同平台向量化间接数组索引的难度。

间接数组索引向量化的原理如图 3 所示，$A[b+index[i]]$数组在内存中是不连续存放的，假如每个值的存放位置间隔一个 A 数组类型的大小。图 3 中阴影部分表示 A 数组各个值之间在内存中的空隙，白格部分表示 A 数组的各个值

在内存中存放的位置。由于数据存放在内存中是不连续的，因此使用普通的向量加载指令取出的数据往往不是我们需要的。同理，将数据存放到内存中时，若需要不连续的数据存储，普通的向量存储指令并不能正确地存放数据。

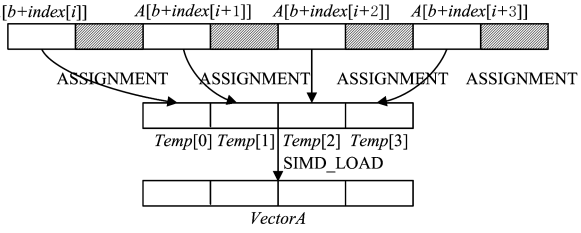

图 3 间接数组索引向量化的原理

Fig. 3 Principle for indirect array index vectorization

本文解决该类访存问题采用的方式是，首先申请一个空间大小合适的临时数组 *Temp*，然后依次将间接数组的值赋给临时数组 *Temp*。注意，此时该临时数组的值都是连续存放的，并且该临时数组可以被向量加载指令直接加载为向量数据。该向量数据可以直接参与之后的向量化运算。同理，在向量化运算结束之后，可以采用类似的方式处理间接数组存储向量化问题，即计算得出的结果可以首先存储到临时数组 *Temp* 中，而后临时数组 *Temp* 将值分别赋给间接数组。通过该方式对间接数组索引进行加载和存储两方面的处理，使得代码得到了向量化的机会，从而实现了包含有间接数组索引的循环代码的 SIMD 向量化代码的生成。

以一个具体的实例即 SPEC2006 测试集中的 416. GAMES 来说明问题。图 4 中 SOURCE CODE 的代码是该应用中某段核心循环代码的简化形式，VECTORED CODE 是源代码向量化之后对应的手工向量化代码，变量 *shX* 是一个具有间接索引的数组通过赋值得到的。在循环内使用本文提出的方法首先将变量 *shX* 赋值给临时数组 *tmp*，之后再使用向量加载指令将临时数组加载为向量数据。变量 *shX* 由之前的不能进行向量化转换成了变量 *vshX* 参与向量运算，从而实现了间接数组索引的向量化。最后，算法 1 给出了该方案具体的向量化代码生成过程。

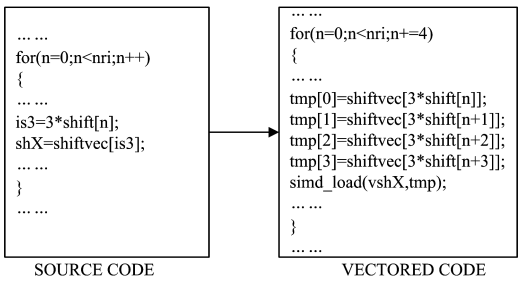

图 4 间接数组索引的向量化代码生成

Fig. 4 Vectorization code generation for indirect array index

算法 1 向量化代码生成算法

输入：串行代码

输出：使用本算法生成的 SIMD 向量化代码

```
1. PRODUCEDURE SIMDCodeGen(G=(N))
2.    N<- {n|n∈N,n is one node in loop}
3.    IA // The indirect array
4.    TA //The temporary array
5.    NDi <- Ø //Vi candidates to SIMD for indirect array access
                nodes
6.    NDf <- Ø //Vr candidates to SIMD for formal nodes
7.    B //B candidates to BENEFIT for special SIMD code generation
8.    for n∈N do
9.      if n∈IA then
10.       NDi <- NDi∪{n}
11.     else
12.         NDf <- NDf∪{n}
13.     end if
14.   end for
15.   for n∈N do
16.    if n∈NDi then
17.     if B>0 then
18.       // The special SIMD code generation method
19.       IA->TA
20.       SCALAR LOAD -> SIMD LOAD
21.       SCALAR STORE -> SIMD STORE
22.       SCALAR COMPUTE -> SIMD COMPUTE
23.     else
24.       //The serial code generation method
25.       SCALAR LOAD
26.       SCALAR STORE
27.       SCALAR COMPUTE
28.     end if
29.   else if n∈NDf then
30.       //The conventional SIMD code generation method
31.     SCALAR LOAD -> SIMD LOAD
32.     SCALAR STORE -> SIMD STORE
33.     SCALAR COMPUTE -> SIMD COMPUTE
34.   else
35.     continue
36.   end if
37. end for
38. END PRODUCEDURE
```

3.3 间接数组索引向量化的收益分析

在使用间接数组索引向量化方法后，虽然循环内总的指令条数减少了，但是引进了一些对临时数组的赋值指令和向量加载、向量存储指令，而且在某些特定的程序中间接数组索引重用率低、核心循环中间接数组参与的运算操作较少等问题也会降低该向量化方法的加速效果。因此，为了向量化间接数组索引而引入的这些指令无疑会增加时间开销。这里使用收益分析的方式来权衡向量化和串行时的开销，并判断是否对目标代码进行向量化。对于循环中间接数组索引重用率高且核心循环中有大量运算操作的情况，该方法有非常明显的效率提升效果；而对于核心循环中含有大量间接数组索引且间接数组索引参与的运算操作较少的情况，该方法对程序执行效率的提升较小。

该向量化过程的 SIMD 收益分析用向量执行代价 *Cost_vec* 和串行执行代价 *Cost_ser* 来描述，如式(1)所示。

$$BENEFIT=Cost\_ser-Cost\_vec \tag{1}$$

向量执行代价 $Cost\_vec$ 指使用向量的方法执行完整个循环所需要的时间。每条向量指令的执行时间为 $Cost\_vec_i$，循环迭代的次数为 N，每个循环中向量指令的条数为 num，则向量执行的代价 $Cost\_vec$ 为：

$$Cost\_vec=N*\sum_{i=1}^{num}Cost\_vec_i \tag{2}$$

串行执行代价 $Cost\_ser$ 指使用串行的方法执行完整个循环所需要的时间。每条标量指令的执行时间用 $Cost\_ser_i$ 表示，循环迭代次数为 N，每个循环中的标量指令条数为 num，则串行执行的代价为：

$$Cost\_ser=N*\sum_{i=1}^{num}Cost\_ser_i \tag{3}$$

利用 SIMD 收益分析，在真正使用间接数组索引向量化方法之前，通过式(1)来计算 $BENEFIT$ 的值，以确定是否对该段程序实施向量化。若 $BENEFIT>0$，则表明程序向量化后总指令条数减少且向量化的额外开销较少，向量化该段代码有收益，可以对该段代码实施 SIMD 向量化代码生成；若 $BENEFIT\leqslant 0$，则表明向量化该段代码后指令总条数减少所获得的收益不能抵消向量化该段代码所引入的额外开销(当 $BENEFIT<0$ 时)，或者是向量化该段代码后指令总条数减少所获得的收益恰好可以抵消向量化该段代码所引入的额外开销(当 $BENEFIT=0$ 时)，那么向量化该段代码没有收益，不对该段代码做 SIMD 向量化代码生成。

4　测试结果与分析

4.1　平台与测试用例

本文测试是在国产处理器“申威·太湖之光”上实现的，使用中国自主研发的“申威 26010”众核处理器，64 位自主申威指令系统，核心工作频率为 1.45GHz。该处理器的运算控制核心和运算核心内部都扩展了 SIMD 扩展部件，且运算控制核心和运算核心中的 SIMD 扩展结构均为相同宽度，即均为 256 位。宽度为 256 位的向量寄存器可以同时处理 4 个 double 型数据、4 个 float 型数据或 8 个 int 型数据。提取 SPEC2006 测试集中相关测试程序的核心代码和高精度有限体积方法(Compact high order finite volume method on unstructured grids，CFV)中的核心代码 RECONST_GS 作为评估本算法的测试用例。表 1 列举了本实验所使用的测试程序。

表 1　测试集

Table 1　Benchmarks

APPLICATION	KERNEL	EXTRACTED FUNCTION
SPEC2006 416. GAMESS	FORMS	FORMS()
SPEC2006 435. GROMACS	INL	INL1130()
SPEC2006 444. NAMD	CPEF	CALC_PAIR_ENERGY_FULLELECT()
CFV	RECONST_GS	RECONST_GS()

本文所提方法的目的是解决实际应用程序中间接数组索引的向量化问题，因此选取了 SPEC2006 测试集中的 416. GAMESS 的核心代码 FORMS、435. GROMACS 的核心代码 INL、444. NAMD 的核心代码 CPEF 以及 CFV 中的核心代码 RECONST_GS 作为测试用例。这些测试用例都包含有间接数组索引，使用本文提出的算法对 4 个核心代码分别进行测试。

4.2　测试结果分析

测试结果如图 5 所示，其中横轴代表测试所选取的核心循环，纵轴代表加速比。通过以下 3 种不同方式对本文提出的向量化算法进行评估。

1)SER：编译时关闭向量化选项，程序串行执行；

2)Conventional SIMD：编译时使用传统方式进行向量化，程序向量化执行；

3)Special SIMD：使用本文所提方法进行向量化，程序向量化执行。

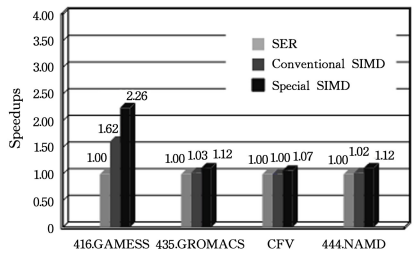

图 5　使用 3 种方法测得的加速比

Fig. 5　Speedups measured with three methods

通过测试结果可知，对于 SPEC2006 中的 416. GROMACS、435. NAMED 和 444. NAMD 核心循环以及 CFV 核心循环而言，本文所提出的向量化方案都有加速。416. GROMACS 程序得到的加速效果最好，原因是在其核心循环中间接数组索引在计算过程中的重用率高，循环中的大部分指令都为运算操作，因此使用该方法进行向量化可以获得得很好的效果。而 CFV 等的核心循环中间接数组索引的重用率低，且循环中运算操作所占比例小、间接数组索引赋值操作多，因此不能充分发挥该方法的优势。但是总体来看，使用本文提出的向量化方法解决间接数组索引问题时，向量化收益分析中 $BENEFIT$ 的结果都大于 0，因此编译器认为可以向量化。

从实验结果中可以看出，采用该方法对含有间接数组索引的循环进行向量化后得到了一定的加速比，而且该加速比相比传统向量化方式的效果更好。在循环中使用该方法对含有间接数组访问且拥有较高计算密度和较高重用性的程序进行向量化时可以得到较好的效果；而对于 CFV 等，核心循环中的间接数组的重用性、计算密度不及 416. GROMACS，使用本文提出的方法进行向量化也会得到加速效果，但不如后者的加速比高。因此，在实际的应用程序中，应该结合程序自身的特点来选择是否采用该方式对间接数组索引进行向量化。

结束语　间接数组索引在实际应用中十分常见，由于访存不规则，传统的自动向量化方法不能发掘其向量化机会。对于间接数组索引问题，本文提出了相应的解决方案。

针对间接数组索引向量化问题，本文提出了一种新的向量化算法。该算法根据收益分析判断代码是否存在向量化的

(下转第 236 页)

[17] SHI K Q. Function inverse P-sets and information law fusion [J]. Journal of Shandong University (Natural Science), 2012, 47(8):73-80. (in Chinese)
史开泉. 函数逆 P-集合与信息规律融合[J]. 山东大学学报(理学版),2012,47(8):73-80.

[18] SHI K Q. Function inverse P-sets and the hiding information generated by function inverse P-information law fusion [C]// Proceedings of Digital Services and Information Intelligence on e-Business, e-Services, and e-Society. Berlin: Springer-Verlag, 2014:224-237.

[19] TANG J H,CHEN B H,ZHANG L,et al. Function inverse P-sets and the dynamic separation of inverse P-information laws [J]. Journal of Shandong University (Natural Science),2013, 48(8):104-110. (in Chinese)
汤积华,陈保会,张凌,等. 函数逆 P-集合与逆 P-信息规律动态分离[J]. 山东大学学报(理学版),2013,48(8):104-110.

[20] SHI K Q. Inverse P-sets[J]. Journal of Shandong University (Natural Science),2012,47(1):98-109. (in Chinese)
史开泉. 逆 P-集合[J]. 山东大学学报(理学版),2012,47(1):98-109.

[21] FAN C X,HUANG S L. Inverse P-reasoning discovery identification of inverse P-information[J]. International Journal of Digital Content Technology and Its Applications,2012,6(20):735-744.

[22] SHI K Q,TANG J H,ZHANG L. Intelligent fusion of inverse P-information and recessive transmission of information intelligent hiding[J]. Systems Engineering and Electronics,2015, 37(3):599-605. (in Chinese)
史开泉,汤积华,张凌. 逆 P-信息智能融合与信息智能隐藏的隐性传递[J]. 系统工程与电子技术,2015,37(3):599-605.

(上接第 223 页)

机会,之后只需在代码中加入临时数组变量,而不依赖于某一平台特殊的数据整理指令就可以解决间接数组索引的向量化问题。使用在 SPEC2006 等测试集中提取的核心循环代码进行测试,结果表明该方法对含有间接数组索引的循环代码可以向量化,且具有比传统向量化方式更好的加速效果;尤其当循环中的间接数组索引重用率高且计算密度大时,使用本文方法的加速效果更明显。后续将继续深入研究向量化过程中的访存优化问题,基于收益分析模型提出更通用的 SIMD 向量化技术。

参 考 文 献

[1] 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. (in Chinese)
高伟,赵荣彩,韩林,等. SIMD 自动向量化编译优化概述[J]. 软件学报,2015,26(6):1265-1284.

[2] BIK A J C,GIRKAR M,GREY P M,et al. Automatic Intra-Register Vectorization for the Intel® Architecture[J]. International Journal of Parallel Programming,2002,30(2):65-98.

[3] ZHU Q S. Improving Program Performance via Auto-Vectorization of Loops with Conditional Statements with GCC Compiler Setting[J]. Applied Mechanics & Materials, 2013,433-435: 1410-1414.

[4] YIU J. Getting Started with the ARM RealView Development Suite[M]// The Definitive Guide to the ARM Cortex-M0. Elsevier Inc.,2011:361-384.

[5] KANSAL R,KUMAR S. A vectorization framework for constant and linear gradient filled regions[J]. The Visual Computer,2015,31(5):717-732.

[6] CHEN J Z,LEI Q,MIAO Y W,et al. Vectorization of line drawing image based on junction analysis[J]. Science China Information Sciences,2015,58(7):1-14.

[7] SUI Y,FAN X,ZHOU H,et al. Loop-oriented array-and field-sensitive pointer analysis for automatic SIMD vectorization[J]. Acm Sigplan Notices,2016,51(5):41-51.

[8] WEINHARDT M,LUK W. Pipeline vectorization[J]. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems,2001,20(2):234-248.

[9] KENNEDY K,ALLEN J R. Optimizing compilers for modern architectures: a dependence-based approach[M]// Optimizing compilers for modern architectures. Morgan Kaufmann Publishers,2002.

[10] CHANG H,SUNG W. Efficient vectorization of SIMD programs with non-aligned and irregular data access hardware[C]// Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems. ACM,2008: 107-176.

[11] LI P,ZHAO R,ZHANG Q,et al. An SIMD Code Generation Technology for Indirect Array[J]. International Journal of Computer Theory and Engineering,2016,8(3):218-222.

[12] ALEEN F,ZAKHARIN V P,KRISHANIYER R,et al. Automated compiler optimization of multiple vector loads/stores[J]. International Journal of Parallel Programming,2018,46(2):47-503.

[13] KIM S,HAN H. Efficient SIMD code generation for irregular kernels[J]. Acm Sigplan Notices,2012,47(8):55-64.

[14] WEI S,ZHAO R C,YAO Y,et al. Data Regroup Alignment Optimization Based on SIMD [J]. Chinese Journal of Computers, 2012,39(2):305-310. (in Chinese)
魏帅,赵荣彩,姚远,等. 面向 SIMD 的数组重组和对齐优化[J]. 计算机科学,2012,39(2):305-310.

[15] YU H N,HAN L,LI P Y,et al. Structure Optimization for Automatic Vectorization [J]. Chinese Journal of Computers,2016, 43(2):210-215. (in Chinese)
于海宁,韩林,李鹏远,等. 面向自动向量化的结构体优化[J]. 计算机科学,2016,43(2):210-215.

[16] LARSEN S,AMARASINGHE S. Exploiting superword level parallelism with multimedia instruction sets[J]. Acm Sigplan Notices,2000,35(5):145-156.

[17] NUZMAN D. loop aware SLP in GCC[C]// GCC Developers Summit. 2007.