第43卷 第6A期
2016年6月

计 算 机 科 学
Computer Science

Vol. 43 No. 6A
June 2016

基于符号执行和人机交互的自动向量化方法

陈 勇[1] 徐 超[2]

(中国电子科技集团公司第十四研究所 南京 210039)[1] (南京审计大学 南京 210029)[2]

摘 要 自动向量化技术是一种针对单指令多数据(SIMD)向量化计算单元的并行编译优化技术,它能够自动将源程序中多个相同标量操作合并为一个向量操作,从而提升系统吞吐量。随着SIMD向量化计算单元的广泛应用,自动向量化技术已经成为学术界和商业界的研究热点。针对现有自动向量化技术可向量化模块识别难、向量化优化方案选择难、可移植性差等问题,提出了一种基于符号执行和人机交互的自动向量化方法。首先借助于符号执行技术,获得较好的可移植性和较高的可向量化模块识别率;然后利用人机交互技术选择出理想的向量化方案。应用示例及实验结果表明,该方法具有较好的可操作性,能够有效提升自动向量化技术的优化效果和可移植性。

关键词 自动向量化,符号执行,人机交互,可移植性,单指令多数据

中图法分类号 TP391 文献标识码 A

Symbolic Execution and Human-Machine Interaction Based Auto Vectorization Method

CHEN Yong[1] XU Chao[2]

(The 14th Research Institute, CETC, Nanjing 210039, China)[1] (Nanjing Audit University, Nanjing 210029, China)[2]

Abstract Auto-vectorization is a parallel compiling optimization technology for SIMD vector computing units. It combines multiple same operations into one SIMD instruction which can significantly improve the output of the system. As SIMD vector computing units are used widely, auto-vectorization technology has become the hot topic in both academic and commerce world. Focusing on the shortcoming of current auto-vectorization technology such as the difficulty to get the code that can be vectorized, the difficulty to select the best optimization schema and poor portability, we proposed a new vectorizing method based on the symbolic execution and human-machine interaction. The method contains two phases. At first, based on the symbolic execution technology, it recognizes the vectorizable code as much as possible. Then, the human-machine interaction technology is used for determining the exactly code to be vectorized. At the same time, the method has portability that can be used for other architectures by only modifying the pattern file. Application example shows that our new technology is feasible and effective.

Keywords Auto-vectorization, Symbolic execution, Human-machine interaction, Portability, SIMD

1 引言

自动向量化技术是编译器提供的一种目标代码优化技术,它能够在不改变高级语言源代码的基础上,自动识别程序中的可向量化部分,并使用对应的SIMD指令集生成相应的目标代码。随着单指令多数据(SIMD)向量化计算单元广泛应用于现代处理器(如Intel SSE/AVX[1],ARM NEON[2],IBM AltiVec/VMX[3]等)中,自动向量化技术已经成为国内外学者和商业机构的研究热点。

Konrad. T等人[4]研究了如何把循环自动向量化和其他循环优化映射到多面体模型中,并在这一模型中构建统一代价模型,最后通过对代价模型的评估选择出最优的优化方案,以提高自动向量化效果。Wang等人[5]针对SIMD的数据对齐问题,提出了一种新的循环转换方法。它通过对数据依赖关系的分析构建数据偏移信息的数学模型,并以此为指导逐层对齐每个循环的数据,有效减少了SIMD向量化优化时数据不对齐访问的情况,提高了多核处理器的并行性。Christopher Haine等人[6]提出了一种针对SIMD向量化效果的二进制分析工具,它能够对现有的二进制代码进行分析,找出其中可能的向量化部分以帮助程序员改进其代码。Ramanarayanan等人[7]介绍了两种在open64中实现的向量化方法:基于同构树匹配的算法和基于动态规划的方法,并评估了这两种方法的时间开销和向量化效果。Gueltons等人[8]探讨了python程序中特殊应用的自动向量化方法。徐金龙[9]等人结合基于循环的向量化算法和超字并行向量化算法的特点,设计了一种混合向量化框架,以便于针对不同应用的特点选择不同的向量化方案。此外,Zhu等人[10]针对GCC编译器的自动向量化方法,Ojha D K[11]、李春江[12]等人对主流编译器(如GCC、LLVM、ICC、PGI等)的自动向量化效果以及存在的问题,进行了评估和探讨,为进一步研发高效的自动向量化编译方法提供了参考。

以上这些技术有效促进了自动向量化优化的发展。但由

陈 勇 工程师,主要研究方向为编译优化、嵌入式系统等,E-mail:cyong1000@163.com;徐 超 副教授,主要研究方向为编译优化、嵌入式系统等。

于主流编译器(如 GCC 等)的中间代码主要是为通用计算机体系结构的优化而设计的,对以 SIMD 指令集为主要目标的自动向量化技术考虑较少,现有的自动向量化技术优化效果十分有限,存在可向量化部分识别不精确、向量化方案难以确定、数据对齐导致的优化效率低下、可移植性差等不足,亟需一种新型的自动向量化优化技术,以适应快速增长 SIMD 架构的普及与应用。为此,本文将结合符号执行和人机交互技术,来解决自动向量化操作识别率低、优化效果不理想的问题,并通过可配置的面向 SIMD 指令识别的模板,使其能够很方便地扩展 SIMD 指令集,从而有效提高该方法的可移植性。

2 基于符号执行和人机交互的自动向量化方法

为弥补当前主流自动向量化技术优化效果不理想、可移植性差等不足,本文将设计一种基于符号执行和人机交互的自动向量化方法。该方法在主流编译器框架(如 GCC 等)的基础上,一方面利用符号执行和人机交互技术地增强编译器对源程序的分析能力,从而更精准地识别源程序中的可自动向量化模块,生成更加高效的目标代码;另一方面借助于向量指令匹配模板,使得该方法能够很方便地识别不同架构下的新增向量指令,从而使得该方法具有良好的可移植性。该方法的总体框架如图 1 所示。

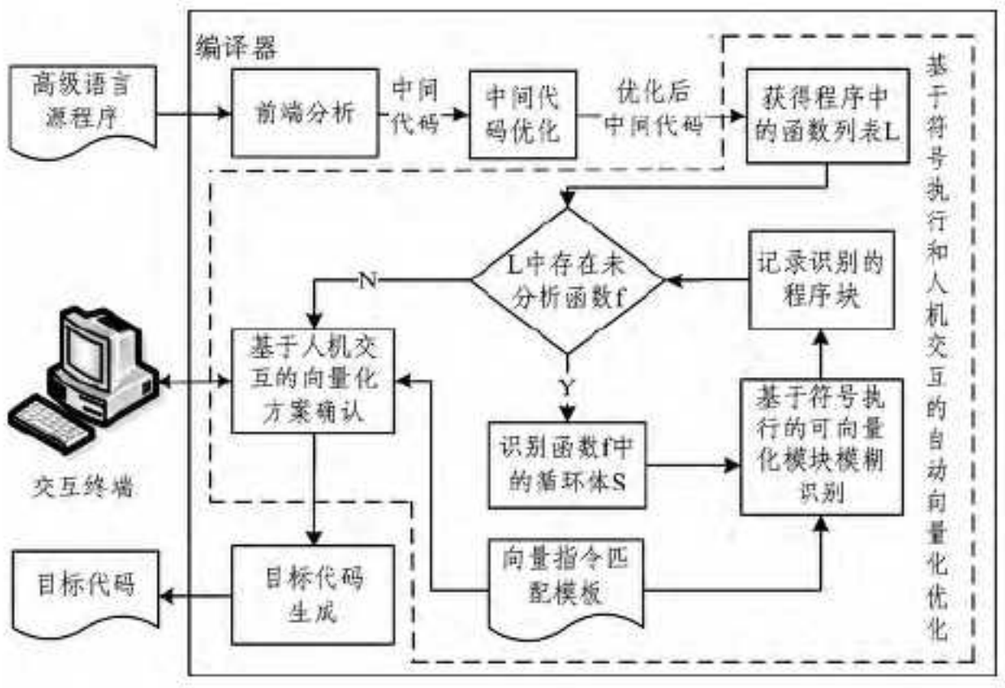

图 1 基于符号执行和人机交互的自动向量化优化

它以编译器优化后的中间代码作为输入,依次对每个函数进行分析。由于自动向量优化的目标主要是针对循环进行的,因此首先遍历函数体,识别出其中的循环体;然后利用符号执行技术,根据向量指令模板对循环体进行分析,识别其中可向量化部分并记录;接着将记录的匹配循环映射结果传递给人机交互系统,人机交互系统根据向量指令模板,将记录的匹配循环映射信息以自然语言的方式展示给程序员以确认具体的向量化方案;最后,人机交互系统将根据程序员的确认结果,生成最终的向量化优化后的代码。

由以上框架不难看出,相对于目前广泛使用的自动向量化方法,本方法主要增加了 3 个部分:向量指令匹配模板、基于符号执行的可向量化模块模糊识别、基于人机交互的向量化方案确认。

2.1 向量指令匹配模板

面向向量化识别的向量指令匹配模板一方面用于提高该方法的可移植性,使得该方法在其配合下能够很方便地对新增复杂向量指令进行识别;另一方面用于指导该方法更有效地识别源程序中可向量化的部分。由于向量指令通常是同时对多个数据进行处理,且以输入作为处理对象,以输出处理结果,具有固定的格式,因此能够方便地表达和新增指令。本文将采取条目的形式作为该匹配模板的基本形式,每个条目由固定数目的输入数据、输出数据构成。同时,由于本文设计的自动向量化方法将结合使用符号执行和人机交互技术,因此要利用该模板指导对向量化模块的识别,该模板还需要为符号执行和人机交互提供必要信息。

对于符号执行系统,其最为突出的即为路径爆炸问题。符号执行系统中的路径爆炸问题主要是由于符号常量的取值范围不确定造成的,如果能够将符号常量限定为某个具体的常量值,则将大幅度减少系统的路径,因此本模板为每条向量指令提供了一个典型的输入数据和对应的输出结果,以便于在识别该条向量指令时能够直接使用这个典型输入输出进行测试,缓解符号执行导致的路径爆炸问题。为了能够提供友好的人机交互界面,本模板还将对每条指令提供必要的说明信息,以方便用户进行选择。该模板的具体形式如图 2 所示。

```
指令名:[输入数据类别数 m:输出数据类别数 n:循环执行次数 N]
    #指令描述信息
    [第 1 类输入数:类型,大小(典型输入数据)]
    …
    [第 m 类输入数:类型,大小(典型输入数据)]
    [第 1 类输出数:类型,大小(典型输入的预期输出)]
    …
    [第 n 类输出数:类型,大小(典型输入的预期输出)]
指令名:[输入数据类别数 m:输出数据类别数 n:循环执行次数 N]
    #指令描述信息
    [第 1 类输入数:类型,大小(典型输入数据)]
    …
    [第 m 类输入数:类型,大小(典型输入数据)]
    [第 1 类输出数:类型,大小(典型输入的预期输出)]
    …
    [第 n 类输出数:类型,大小(典型输入的预期输出)]
    …
```

图 2 向量指令匹配模板的基本形式

2.2 基于符号执行的可向量化模块模糊识别

符号执行技术是 20 世纪 70 年代提出的一种路径敏感静态分析方法[13],已经被广泛应用于程序分析中[14-17]。其基本思想是以符号的形式替换输入,根据程序中每条语句的具体语义,模拟程序的执行。当遇到分支条件时,则将该条件增加到对应的后续语句的执行约束中。其分析的最终结果为程序中每条执行路径及其需要满足的约束条件映射表。由于符号执行能够模拟程序的执行过程,虽然其存在路径爆炸的问题,但只要能够很好地限制符号执行的输入数据,就能够做到对复杂功能进行识别,这对于缺乏直接中间表示的复杂向量指令识别是很有利的。因此,本文在自动向量化识别阶段借助了符号执行技术,设计了基于符号执行的可向量化模块模糊识别方法。

该方法以函数为单位,为函数的参数赋予不同的符号常量,然后按照每条中间代码的语义逐条执行,并记录各个程序点的状态。为解决符号执行过程中的路径爆炸问题,该方法利用向量指令匹配模板,对可能出现可向量化模块的部分(即循环体部分)使用初值替换对应的符号常量,然后再继续执行相应的循环体,并在循环体的出口处同匹配模板中的期望值

进行比较，从而根据比较结果初步识别该循环体是否是可向量化模块。该方法的具体步骤如下。

(1)根据识别的循环体 S，分析获得 S 中的迭代数组 A、出口点的所有活跃变量 Le，以及 S 中所有定义的变量 D。迭代数组是指在循环体中，其数组元素依次被访问的数组，如对于图 3 所示的实例，其迭代数组即为 a。对于一个变量从某时刻 q 起，到下一个定义点止，其间若有对该变量的使用，则称该变量在 q 是活跃的，否则称该变量在 q 是非活跃的，所有的全局变量总被认为是活跃的，函数内的局部变量在其最后一次使用后是不活跃的。如对于图 3 所示实例，在 for 循环的出口处，对于变量 i，参数 a 及其所指向的数据在该循环以后不再使用，因此它不是活跃变量；而对于参数 $result$ 指向的数据，由于其在循环后重定义，因此它也不是活跃的，所以在 for 循环的出口处，其活跃变量为：$\{index, min, result\}$。定义的变量是指该程序块中被赋值的变量，如对于图 3 所示实例，for 循环中定义的变量为 $\{i, min, index\}$。

```
int Min(unsigned int * result, unsigned int*a, int size)
{
  unsigned int min;
  int index, i;
  min=a[0];
  index=0;
  for(i=1;i<size;i++){
    if(a[i]<min){
      min=a[i];
      index=i;
    }
  }
  *result=min;
  return index;
}
```

(a)源程序

```
1. ([], a, 0, min)
2. (=, 0, null, index)
3. (=,1,null,i)
4. (label,L0,null,null)
5. (jge,i,size,L1)
6. ([],a,i,tmp1)
7. (jge,tmp1,min,L2)
8. (=,tmp1,null,min)
9. (=,i,null,index)
10. (Label,L2,null,null)
11. (+,i,1,i)
12. (jmp,L0)
13. (label,L1,null,null)
14. (=,min,*,result)
15. (ret,index,null,null)
```

(b)中间表示代码

图 3　示例程序

(2)在函数的入口处，以符号常量的形式为参数赋初值。如果参数为指针类型，则认为其可能为数组，对其本身及其对应的数组分别赋值，得到函数入口的初始状态。如对于图 3 所示实例，共有 3 个参数，其中前两个为指针类型，因此设置其初始值为：$[result=x_0, *(result+i)=r_i, a=x_1, *(a+i)=t_i]$，其中 $0<i<N$，N 为该体系结构中单条 SIMD 指令能够操作的最大数据长度。最后一个参数为普通值，直接使用符号变量对其赋值，即 $[size=s]$，获得初始状态 $[result=x_0, *(result+i)=r_i, a=x_1, *(a+i)=t_i, size=s]$。

(3)上步骤赋值的符号常量作为初始状态，逐语句模拟程序的执行，并同时设置每个程序点的状态以及约束条件。如对于图 3 中的第一条语句 $(min=a[0])$，根据当前状态 $[result=x_0, *(result+i)=r_i, a=x_1, *(a+i)=t_i, size=s]$，将 $*(a+0)$ 的值 t_1 代入，获得该语句的执行结果为：$min=t_1$，并使用该符号执行结果更新当前状态为：$[result=x_0, *(result+i)=r_i, a=x_1, *(a+i)=t_i, size=s, min=t_1]$。

(4)依次分析向量化指令集合 V 中的每条指令，进行如下操作：

a)以迭代数组 A 为向量指令输入数据长度，以既在 D 中又在 Le 中的数据作为输出，尝试根据该指令的指令模板进行匹配。如果匹配失败，则跳过下一步骤，分析 V 中的下一条指令。

b)根据匹配的模板指令，截断输入数组长度为该指令模板中的循环执行次数 N，同时用 N 值替换循环迭代次数，并修改当前状态，如对于图 4 所示指令模板中的 VPHMINPOSUW 指令，其 N 值为 8，输入典型数据为 (1,2,3,4,5,6,7,8)，对于图 3 实例 for 循环入口处，将迭代数组 a 所指向数据使用典型数据进行赋值，即此时 $\{t_1=1, t_2=2, \cdots, t_8=8\}$，并将循环上界值 $size$ 赋值为 8。然后更新当前符号执行的状态，如对于 min，其本来状态为 $min=t_1$，因为此时 t_1 已经被赋值为 1，因此更新 min 的值为 1。然后同步骤(3)，符号执行该循环体，获得最终状态。如果该符号执行的结果与该指令模板中的预期执行结果相等，则认为该向量指令被该循环体识别，建立该向量指令同该循环体的映射关系保存在 M 中，并退出该循环。

```
VPADDW:[2,1,8] #有符号字加
        [in-1:int,8(1,2,3,4,5,6,7,8)]
        [in-2:int,8(2,4,6,8,10,12,14,16)]
        [out-1:int,8(3,6,7,12,15,18,21,24)]
VPHMINPOSUH:[1,2,16] #无符号半字向量元素取其中最小值
        [in-1:ushort,16(1,2,3,4,5,6,7,8,9,10,11,12,13,14,
        15,16)]
        [out-1:ushort,1(1)]
        [out-2:uchar|ushort|uint|ulong|char|short|int|long,
        1(0)]
VPHMINPOSUW:[1,2,8] #无符号字向量元素取其中最小值
        [in-1:uint,8(1,2,3,4,5,6,7,8)]
        [out-1:uint,1(1)]
        [out-2:uchar|ushort|uint|ulong|char|short|int|long,
        1(0)]
VPHMAXPOSUH:[1,2,16] #无符号半字向量元素取其中最大值
        [in-1:ushort,16(1,2,3,4,5,6,7,8,9,10,11,12,13,14,
        15,16)]
        [out-1:ushort,1(16)]
        [out-2:uchar|ushort|uint|ulong|char|short|int|long,
        1(15)]
VPHMAXPOSUW:[1,2,16] #无符号字向量元素取其中最大值
        [in-1:uint,8(1,2,3,4,5,6,7,8)]
        [out-1:uint,1(8)]
```

[out-2:uchar|ushort|uint|ulong|char|short|int|long,
1(7)]

...

图 4　向量指令匹配模板示例

2.3　基于人机交互的向量化方案确认

由于在采用符号执行技术识别可向量化模块时使用了典型输入输出来缓解符号执行的路径爆炸问题，因此识别的可向量化模块并不十分准确。同时，对于向量化操作的某些特殊需求（如指针分析、数组对齐分析等），编译器的保守分析很难获得理想的向量化效果。因此本方法增加了基于人机交互的向量化方案确认过程，即采用人机交互的方式，利用编程者对程序功能和数据的全面掌控信息（如数据是否对齐、指针是否重叠等）逐步确认约束条件，以选择出最佳的向量化方案。

与目前支持向量化优化的 GCC、ICC 等成熟编译器的人机交互方式不同，本文的交互方式并不需要编程人员在编写程序时就考虑到哪些代码部分需要增加对应的向量化优化指导代码，而是在编译过程中根据编译器识别的代码部分需要编译器优化指导，提供对应的指引信息供程序员选择。该方式一方面提高了人机交互的效率，程序员在编写程序时不需要考虑在何处添加向量化优化指导代码，而是根据编译过程中弹出的人机交互界面进行选择，极大减轻了程序员的编程负担；另一方面，该方法是根据识别的向量化模块可能采取的向量化优化措施需要的条件，通过选择“是”与“否”的简单方式进行交互，因此除非用户（即程序编写者）对所编写的程序不明确，选择了与编程意图完全相反的结果（这对于编程者来说一般是不会发生的），否则不会生成错误代码，从而有效保证了交互的质量，确保交互的内容有利于向量化优化方案的选择。

3　应用示例

为验证该方法的可行性，本文将通过一个示例程序说明该方法的实施过程。使用的 c 语言源程序如图 3(a) 所示，以三地址码的形式作为该编译器的中间表示，经过中间代码优化后，其优化后的中间代码如图 3(b) 所示。其使用的目标平台为华睿处理器。华睿处理器是具有自主知识产权并支持 SIMD 的高性能 DSP 处理器。该处理器共有 64 个向量寄存器，每个向量寄存器的长度为 256bit，每条 SIMD 向量指令能够同时对 256 位数进行计算。华睿处理器共设计了 317 条 SIMD 向量指令，以提高系统数字信号处理的能力。

针对优化后的中间代码，首先根据向量指令匹配模板，将华睿处理器指令集中的向量指令填入模板，如图 4 所示（华睿 1 号处理器共有向量指令 317 条，此处仅选取有符号字加指令(VPADDW)、无符号半字向量元素取其中最小值指令(VPHMINPOSUH)、无符号字向量元素取其中最小值(VPHMINPOSUW)，无符号半字向量元素取其中最大值指令(VPHMAXPOSUH)、无符号字向量元素取其中最大值(VPHMAXPOSUW)共 5 条指令作为示例）。

其次根据 2.2 节步骤(1)，识别指令序列中的循环体，获得如图 5(a) 所示的循环体 S。接着根据 2.2 节步骤(2)获得 S 中迭代数组 A、出口点的所有活跃变量 Le 以及 S 中所有定义的变量 D，如图 5(b) 所示，其迭代数组为 a，出口点的所有活跃变量为 $\{result, min, index\}$，循环体内定义的变量为 $\{min, index, i\}$。然后根据 2.2 节步骤(3)，用符号常量初始化函数入口处参数，即 $[result=x_0, *(result+i)=r_i, a=x_1, *(a+i)=t_i]$，其中 $0<i\leqslant 32$。接着，如步 2.2 节步骤(4)所示，依次符号执行每条中间代码，直到循环体 S 的入口处，即中间代码的第 4 条语句，并将符号执行的结果保存到当前程序点，如图 5(c) 所示。

(label,L0,null,null)
(jge,i,size,L1)
([],a,i,tmp1)
(jge,tmp1,min,L2)
(=,tmp1,null,min)
(=,i,null,index)
(label,L2,null,null)
(+,i,1,i)
(jmp,L0)

(a) 识别的循环体 S

A:a[]:uint
Le:min:uint,index:int,result:uint*
D:min:uint,index:int,i:int

(b) 循环体数据分析

初始化:[result=x_0, *(result+i)=r_i, a=x_1, *(a+i)=t_i, size=s]
语句 1:[result=x_0, *(result+i)=ri, a=x_1, *(a+i)=t_i, size=s, min=t_1]
语句 2:[result=x_0, *(result+i)=ri, a=x_1, *(a+i)=t_i, size=s, min=t_1, index=0]
语句 3:[result=x_0, *(result+i)=r_i, a=x_1, *(a+i)=t_i, size=s, min=t_1, index=0, i=1]

(c) 初始化及到 S 入口各步符号执行结果

图 5　向量化识别准备阶段

在进入 S 循环前，依次遍历向量指令集中的每条指令，以查找是否存在匹配指令。不失一般性，此处仅以列出的 5 条华睿指令进行分析。首先分析 VPADDW 指令，由于其输入数据要求有两个有符号整型数组，而实际参与运算的迭代数组只有一个，因此数量不匹配，匹配失败，分析下一条指令 VPHMINPOSUH。而对于 VPHMINPOSUH，虽然其输入数据为一个数组，但其为 ushort 类型，与实际的类型 uint 不匹配，也舍去。接着分析 VPHMINPOSUW 指令，该指令要求的输入数据为 1 个迭代数组，且类型为 uint，迭代数组满足该要求。而其输出为两个数据，一个为整型，一个 uint 类型，满足分析结果要求的数据类型，因此采用该指令作为可匹配指令。

在找到匹配的向量指令后，使用该向量指令循环迭代次数 8 限定输入数据的长度，此处即限定数组 a 的长度为 8，并确定循环迭代次数 s 值为 8。然后使用该向量指令的典型输入(1,2,3,4,5,6,7,8)为输入数组赋值，即令 $(t_1=1, t_2=2, \cdots, t_8=8)$，更新符号执行中各变量的值，如图 6 第 2 行所示，获得进入循环体时各变量的初始值：$[result=x_0, *(result+i)=r_i, a=x_1, a[0]=1, a[1]=2, \cdots, a[7]=8, size=8, min=t_1, index=0, i=1]$。在获得初始数据后，利用初始数据依次符号执行每条语句，根据每条语句的语义修改对应的变量值，获得如图 6 所示的执行列表。在跳出循环体，即进入图 3(b) 中语句 13 时，根据 2.2 节步骤(2)求得的 $Le\{result, min, index\}$ 和 $D\{min, index, i\}$，确定当前应该获取的输出结果为两者的交集 $\{min, index\}$。由于此时 min 和 $index$ 的结果分别

为 $min=1, index=0$，同该指令的预期结果一致$(1,0)$匹配，因此将其映射关系加入 M 中。

使用典型输入替换后的状态：

[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=t1，index=0，i=1]

循环 1：

语句 4：[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=t1，index=0，i=1]

语句 5：[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=t1，index=0，i=1]

语句 6：[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=t1，index=0，i=1]

语句 7：[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=t1，index=0，i=1]

语句 11：[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=t1，index=0，i=2]

语句 12：[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=t1，index=0，i=2]

循环 2 结果：

[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=1，index=0，i=3]

循环 3 结果：

[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=1，index=0，i=4]

循环 4 结果：

[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=1，index=0，i=5]

循环 5 结果：

[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=1，index=0，i=6]

循环 6 结果：

[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=1，index=0，i=7]

循环 7 结果：

[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=1，index=0，i=8]

Out：

[result=x_0，*(result+i)=ri，a=x_1，a[0]=1，a[1]=2，…，a[7]=8，size=8，min=1，index=0，i=8]

图 6　代入典型输入值后的执行过程

同理，继续分析该函数的循环体。由于该函数只有一个循环体，因此退出循环，然后将 M 中记录的映射关系通过人机交互接口显示给程序员确定。为提高人机交互的效率，该人机交互界面并不是以向量化优化后的汇编代码显示给程序员，而是将其转换为易于理解的自然语言来显示，其示例显示结果如图 7 所示。其中，界面最左边显示的为原始的高级语言代码，右边分层次地显示识别的优化以及要求的约束条件。程序员如果发现识别的优化并非预期的功能，则直接选择“不优化”即可；否则，选择相应的优化指令以及其数据满足的约束条件。如在此实例中，显然其满足“数组 a 无符号字向量元素取其中最小值”的功能要求，则在选择完“数组 a 无符号字向量元素取其中最小值”后可根据数组 a 的首地址是否对齐 16 字节、$size$ 是否被 8 整除等获得最佳的优化方案。

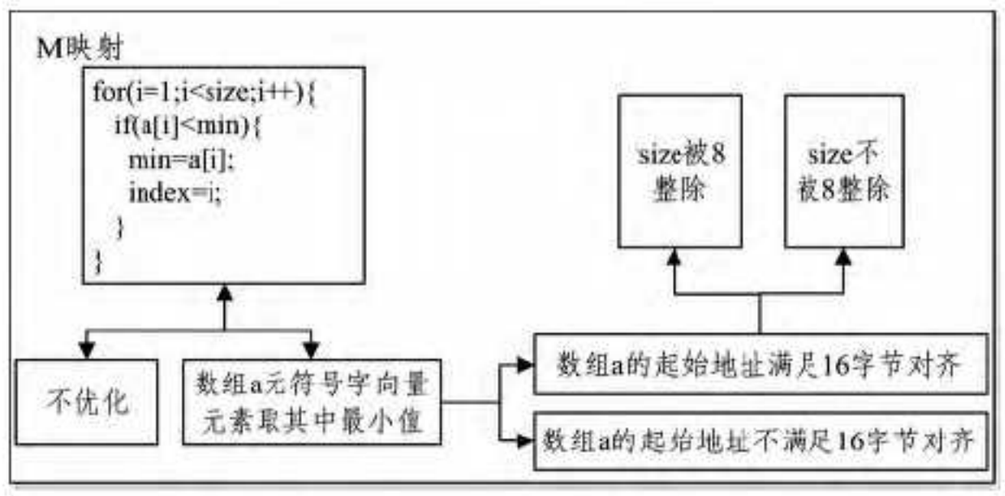

图 7　人机交互示例

当程序员给出数组 a 是首地址对齐且 $size$ 被 8 整除的选项后，优化前后汇编代码的对比如图 8 所示。在优化后，由于执行了向量化操作，每次操作的数据长度变为原来的 8 倍，因此随着处理的数据量的增加，系统的性能将获得显著提升。

```
        globl       Min
Min:
        lw          $ 2,0($ 5)
        addiu       $ 7,$ 0,1
        slt         $ 1,$ 7,$ 6
        beq         $ 1,$ 0,.L2
        move        $ 3,$ 0
        addiu       $ 5,$ 5,4
.L9:
        lw          $ 8,0($ 5)
        sltu        $ 1,$ 8,$ 2
        beq         $ 1,$ 0,.L4
        nop
        move        $ 2,$ 8
        move        $ 3,$ 7
.L4:
        addiu       $ 7,$ 7,1
        slt         $ 1,$ 7,$ 6
        bne         $ 1,$ 0,.L9
        addiu       $ 5,$ 5,4
.L2:
        sw          $ 2,0($ 4)
        jr          $ 31
        move        $ 2,$ 3
        .type       Min,@function
        .size       Min,.-Min
        .end        Min
```

(a) 优化前汇编

```
        .globl      Min
Min:
        move $ 8, 0
        move $ 9, 0
.L1
        vldql       $ z32,0($ 5)
        vldql       $ z33,1($ 5)
        vpermute    $ z32,$ z32,$ z33,1
        vphminposuw $ z33,$ z32
        vzero       $ z32
        vpinsrw     $ z32,$ z32,$ z33,0
        vmovdv2r    $ 10,$ z32
        sltu        $ 1,$ 10,$ 9
        beq         $ 1,$ 0,.L2
        addi        $ 5,$ 5,32
        move        $ 8,$ 10
        vpinsrw     $ z32,$ z32,$ z33,1
        vmovdv2r    $ 10, $ z32
        move        $ 9, $ 10
.L2     bne         $ 6,$ 0, L1
        subi        $ 6,$ 6,8
        sw          $ 8,0($ 4)
```

```
jr          $31
move        $2,$9
.type       Min,@function
.size       Min,.-Min
.end        Min
```

(b)优化后汇编

图8 向量化优化前后的对比

4 实验与结果分析

为验证该方法的有效性,本文以GCC 5.3为基础实验平台,以9个典型的数字信号处理算法为测试用例,对该方法进行了实验。实验结果如图9所示,它显示了本文提出的自动向量化方法相对于GCC的自动向量化方法的性能提升比(即$rate=t_{gcc}-t_{sym}/t_{gcc}$;其中$t_{gcc}$表示GCC优化后程序的执行时间,$t_{sym}$表示本文向量化优化后程序的执行时间)。

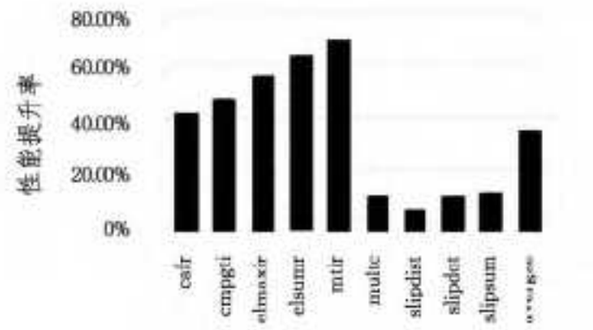

图9 优化后性能提升率

由该结果可以看出,本文向量化算法能够使程序获得较好的性能提升,最高的性能提升率可达70%左右,最低性能提升率也可达6%左右,平均性能提升率约为36%,这对提升数据处理密集度高的信号处理的性能是极为重要的。

符号执行过程会使得编译器本身性能受到一定的损失,但由于本文采用了典型值作为输入,极大地减少了路径爆炸问题,使得向量化识别阶段近似为一个线性匹配过程,系统总的时间复杂度不会超过$O(n*m)$,其中n是体系结构中向量指令的条数,m为识别的可向量化模块数。增加了本文算法后,GCC的编译效率损失率如图10所示,从中也可以看出,性能损失最高只有1.2%。

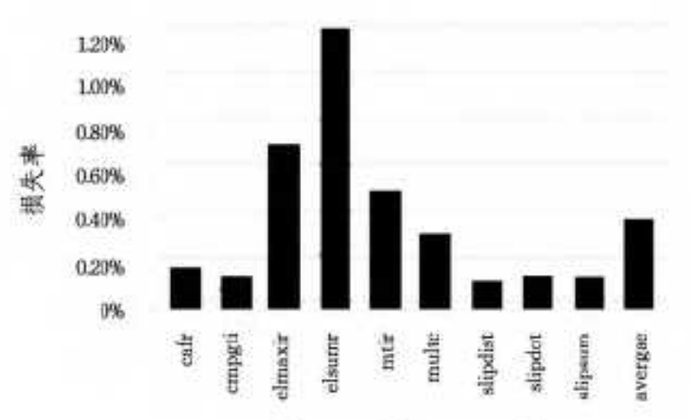

图10 编译器性能损失率

由以上实验结果可以看出,可向量化的代码越多,性能提升越明显,但由于识别可向量化部分需要增加对应的计算,从而导致编译时间也增多。如对于elsumr、elmaxir等测试用例,其优化后程序性能有显著提升,但同时编译时的时间开销也相对较多。而对于slipdist、slipdot等测试用例,由于其可向量化部分较少,需要识别的模式也较少,因此其对编译的开销也相对较少。

结束语 本文针对现有的自动向量化优化方法中存在的不足,结合符号执行和人机交互技术,设计了一种新的自动向量化优化方法。应用示例及实验结果表明:

1)该方法具有较强的可扩展性。通过一套灵活的携带部分语义的指令匹配模板,可以方便地增减新的向量指令。

2)该方法具有较强的普适性。该方法借助于指令匹配模板中表明的指令语义,首先指导符号执行技术提取出程序中可能存在的可向量化模块,然后通过人机交互技术将提取的结果展现给用户,接着编程者根据其编程意图为可向量化模块的识别以及向量化方案提供指引,从而确定最终的向量化方案。它消除了传统自动向量化优化方法在确定可向量化模块时依赖于对应中间语言的限制,有效增强了系统的普适性。

3)该方法具有较强的自动向量化优化能力,可以应用于复杂向量指令的识别。

然而该方法仍然存在一些不足,如在符号执行阶段,如果典型输入不够合理,将导致较大的误识别率,增加后续人机交互的负担。此外,本系统有专门的向量寄存器供向量化优化使用,在寄存器分配时不会对通用寄存器分配产生影响。当向量寄存器同通用寄存器有相关关联时,虽然可以通过增加寄存器约束的方法保证其寄存器分配的正确性,但此时对系统性能的影响还有待进一步分析。今后将对这些问题进行深入探讨,以求在降低可向量化误识别率的基础上获得更好的用户体验,进一步提升系统的普适性。

参 考 文 献

[1] Lee Y, Avizienis R, Bishara A, et al. Exploring the Tradeoffs between Programmability and Efficiency in Data-Parallel Accelerators[C]// Proceedings of Annual International Symposium on Computer Achitecture, 2011: 129-140

[2] Pujara C, Modi A, Sandeep G, et al. VC-1 video decoder optimization on ARM Cortex-A8 with NEON[C]// 2010 National Conference on Communications (NCC). IEEE, 2010: 1-5

[3] Lacassagne L, Etiemble D, Zahraee H, et al. High Level Transforms for SIMD and low-level computer vision algorithms[C]// Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing. ACM, 2014

[4] Trifunovic K, Nuzman D, Cohen A, et al. Polyhedral-Model Guided Loop-Nest Auto-Vectorization. [C] // Proceedings of PACT'09. Washington DC, USA, 2009: 327-337

[5] Wang Y, Pan L, Shao Z, et al. Loop Transforming for Reducing Data Alignment on Multi-Core SIMD Processors[J]. Journal of Signal Processing Systems, 2014, 74(2): 137-150

[6] Haine C, Aumage O, Enguerrand P. et al. Exploring and Evaluating Array Layout Restructuration for SIMDization[C]// The 27th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2014). Hillsboro, USA, 2014

[7] Ramanarayanan R, Gupta M, Chakraborty S S, et al. Harnessing partial vectorization in Open64 compiler[C]// 2014 IEEE International on Advance Computing Conference (IACC). IEEE, 2014: 813-824

[8] Guelton S, Falcou J, Brunet P. Exploring the vectorization of Python constructs using Pythran and Boost SIMD[C]// Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing. ACM, 2014: 79-86

[9] 徐金龙,赵荣彩,丁锐. 面向循环的混合向量化方法研究[J]. 小型微型计算机系统, 2014, 35(12): 2764-2769

(下转第492页)

定位原理如图 6 所示。

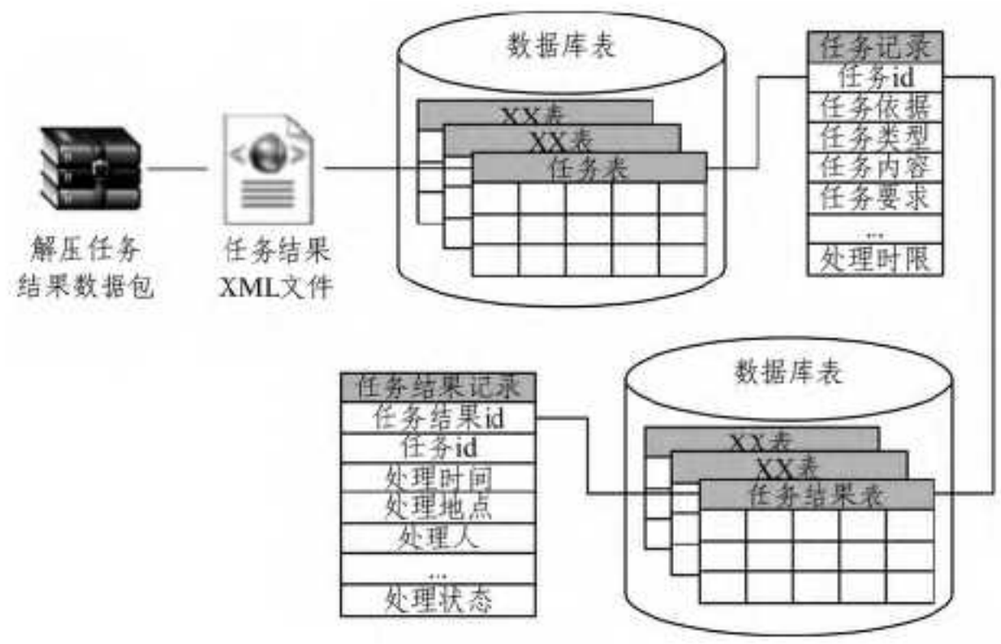

图 6　数据寻址与定位原理图

6.3　数据导入与同步

在数据导入的过程中，必须考虑版本的问题，同一个数据结果包有可能被多次导入，数据包中的内容可能不尽相同，但数据结果包只要符合既定的格式规范，就应该被成功导入，但是不允许进行覆盖操作，必须保留每一次导入的详细信息，即允许符合数据规范的数据结果包被多次重复导入，只是需要在每次导入时添加导入时间戳以示区分。

处理数据同步的过程就是对多个局部数据库同时进行添加、修改、删除操作，以实现异构数据库的数据同步[10]。XML 序列化是将文件转换成字符串的过程，其目的有两点：一是压缩文件，减少冗余数据；二是预处理，方便以字符流进行操作。序列化的主要思想是用元素标签(Tag)来表达 XML 数据，包括 XML 文件中的元素(Element)、属性(Attribute)、文本(Text)[2]。本文在具体执行数据导入时，首先定位待导入的 XML 文件，从 XML 文件中读取结果集并将其转化为数据流，并存储在列表 List 中；同时通过对象拷贝操作，将任务结果信息拷贝到任务结果记录表中，并更新任务记录的任务状态信息(如将任务状态信息由“二级单位上报”改为“一级单位接收”)。通过任务 id 和任务表建立主子表关联关系，从而实现了数据的同步工作。

6.4　数据结果展示

数据导入最终的目的在于数据的同步及结果展示。在数据导入成功后，通过数据结果展示页面直观地显示任务数据的流转及执行情况。由于允许数据包被多次重复导入，在任务数据结果展示时，默认只展示最后一次导入的信息。同时系统提供了浏览历史版本信息的功能，通过导入时间戳信息，即可浏览该时间戳对应的导入信息。

结束语　本文以任务为驱动，基于 XML 语言和数据交互场景，给出了一种数据传输与同步方法，通过制定数据模板，避免了数据识别和出错问题，通过数据导出及导入接口，数据光盘传递，使得工作在彼此隔离的局域网环境中的信息系统实现了犹如在互联的广域网环境中的效果，解决了跨地域的信息沟通交流问题。实践证明，该方法能更好地适应特定的行业现状，在网络不连通的环境下，也可进行业务数据的传输和处理，既提高了信息系统的安全保密水平，也降低了对通信线路的连通要求。

参 考 文 献

[1] 希赛教育软考学院. 系统架构设计师教程(第 3 版)[M]. 北京：电子工业出版社，2014

[2] 侯陈达，李栋，邱杰凡，等. EasiDEF：一种水平化轻量级物联网数据交换协议[J]. 计算机学报，2015，38(3)：602-613

[3] 周军锋，孟小峰. XML 关键字查询处理研究[J]. 计算机学报，2012，35(12)：2459-2478

[4] 陈华城，杜学绘，陈性元，等. 基于 D-S 证据理论的 XML 文档潜在信息获取算法[J]. 计算机应用研究，2013，30(4)：1187-1190

[5] 郑扬飞，金辉，张勇，等. 消防一体化环境下的信息交换平台关键技术研究[J]. 计算机工程与应用，2012，48(7)：219-223

[6] 曹兰英，严义，邬惠峰. 基于模式匹配的 XML 自动转换技术[J]. 计算机工程与应用，2012，48(25)：72-76

[7] 方跃坚，余枝强，翟磊，等. 一种混合并行 XML 解析方法[J]. 软件学报，2013，24(6)：1196-1206

[8] 鉴保瑞，宋余庆，陈健美，等. 一种基于关系的 XML 文档模型映射方法[J]. 计算机应用研究，2011，28(12)：4621-4624

[9] 侯莹，李凤岐，牛纪桢，等. 关系模式到模块化的 XML Schema 的模型映射方法[J]. 计算机工程与应用，2011，47(12)：122-125

[10] 申利民，李卫东. 面向协同系统集成的数据同步模型[J]. 计算机应用研究，2012，29(4)：1384-1386

(上接第 466 页)

[10] Zhu Q S. Improving Program Performance via Auto-Vectorization of Loops with Conditional Statements with GCC Compiler Setting[J]. Applied Mechanics and Materials, 2014, 433: 1410-1414

[11] Ojha D K, Sikka G. A Study on Vectorization Methods for Multicore SIMD Architecture Provided by Compilers[C]// ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India-Vol I. Springer International Publishing, 2014: 723-728

[12] 李春江，黄娟娟，徐颖，等. 典型编译器自动向量化效果评估与分析[J]. 计算机科学，2013，40(4)：41-46

[13] King J. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394

[14] Bardin S, Kosmatov N, Cheynier F. Efficient Leveraging of Symbolic Execution to Advanced Coverage Criteria[C]// 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation (ICST). IEEE Computer Society, 2014: 173-182

[15] Ibing A. SMT-Constrained Symbolic Execution for Edipse CDT / Codan [M] // Software Engineering and Formal Methods. Springer. International Publishing. 2013: 113-124

[16] Reus B, Charlton N, Horsfall B. Symbolic Execution Proofs for Higher Order Store Programs[J]. Journal of Automated Reasoning, 2015, 54(3): 199-284

[17] Chen T, Zhang X S, Ji X L, et al. Test Generation for Embedded Executables via Concolic Execution in a Real Environment[J]. IEEE Transactions on Reliability, 2015, 64(1): 284-296