第32卷 第11期
2009年11月

计 算 机 学 报
CHINESE JOURNAL OF COMPUTERS

Vol. 32 No. 11
Nov. 2009

基于位宽控制提高SIMD架构并行度的优化算法

张为华 朱嘉华 张宏江 臧斌宇

（复旦大学并行处理研究所 上海 200433）

摘 要 随着SIMD功能单元作为多媒体加速部件的广泛应用，如何有效利用这一构架优化应用程序成为编译优化研究的热点.目前典型的SIMD结构为同一操作对不同的数据位宽提供了不同的指令版本，随着操作数位宽的增加，对应的SIMD指令可同时完成的操作个数也随之降低.因此，如何有效识别操作数的有效位宽，对提高优化过程中SIMD指令内操作的并行度将产生至关重要的影响.文中针对SIMD优化面临的并行度问题，提出了一种优化算法，该算法在对操作数的有效位进行分析的基础上，进行溢出控制，从而减少操作数对宽位宽数据类型的依赖.实验数据表明，该算法可以有效提高多媒体程序优化的并行度，对多媒体程序获得较好的加速效果.

关键词 有效位控制；溢出处理；饱和算术；编译优化；并行度

中图法分类号 TP311 **DOI号**：10.3724/SP.J.1016.2009.02168

Optimizing the SIMD Parallelism Through Bitwidth Analysis

ZHANG Wei-Hua ZHU Jia-Hua ZHANG Hong-Jiang ZANG Bin-Yu

（*Parallel Processing Institute*, *Fudan University*, *Shanghai* 200433）

Abstract Although the SIMD units have been widely used in different architecture designs, the automatic optimizations for such architectures are not well developed yet. Since most optimizations for SIMD architectures are transplanted from traditional vectorization techniques, many special features of SIMD architectures, such as packed operations, have not been thoroughly considered. While operands are tightly packed within a register, there is no spare space to indicate overflow. To maintain the accuracy of automatic SIMDized programs, the operands should be unpacked to preserve enough space for interim overflow. However, such a strategy would lead to great overhead. Moreover, the additional instructions for handling overflows can sometimes prevent other optimizations. In this paper, a new technique, BCSA (Bitwidth controlled SIMD arithmetic), is proposed to reduce the negative effects caused by interim overflow handling and eliminate the interference of interim overflows. The algorithm is applied to the multimedia benchmarks of Berkeley. The experimental results show that the algorithm can significantly improve the performance of multimedia applications.

Keywords bitwidth analysis; overflow analysis; saturation operation; compiler optimization; parallelism

1 引 言

随着人们生活水平的提高，各种多媒体应用逐渐成为各种运算平台的主要处理类型.这些应用在极大丰富人们生活的同时，对硬件环境的要求也越来越高.由于多媒体程序自身具有较好的数据并行性，在通用处理器内增加基于单指令流多数据流

收稿日期：2007-05-18；最终修改稿收到日期：2009-06-02.本课题得到国家自然科学基金（60273046）与博士点基金（20050246020）资助.
张为华，博士，主要研究方向为体系结构与编译优化.E-mail：zhangweihua@fudan.edu.cn.**朱嘉华**，博士，主要研究方向为体系结构与编译优化.**张宏江**，硕士，主要研究方向为编译优化.**臧斌宇**，教授，博士生导师，主要研究领域为体系结构与编译优化.

(SIMD)功能单元作为多媒体加速部件成为目前多媒体处理的主要解决方案[1].

随着 SIMD 功能架构的广泛应用,如何有效利用这一架构优化应用程序成为编译优化研究的热点之一.近年来,虽然针对 SIMD 架构的编译优化进行了大量的研究,但这些算法只对一些理想化的核心代码起到了优化作用,而对于真正的应用程序却很难达到预期的加速效果.造成这一性能差距的一个主要原因是 SIMD 优化过程中并行度利用不足.通常 SIMD 结构同一操作对不同的数据位宽提供了不同的指令版本,随着操作数位宽的增加,对应的 SIMD 指令可同时完成的操作个数也随之降低.以 Intel 公司的 SSE2 指令集为例:SSE2 扩展指令集由一组基于 128 位 SIMD 寄存器的向量指令构成.通常每一种向量操作都对应了 3 条 SSE2 指令,这 3 条指令分别可同时完成 4 个标准整型(INT-32 位)的操作,8 个短整型(Short-16 位)的操作或 16 个字符类型(Char-8 位)的操作.利用 SIMD 指令优化相关程序时,在满足计算精度的要求下,选择并行度高的指令能得到更好的优化效果.然而,一方面,由于很多已有的多媒体程序是为通用处理器编写的,在编写过程中由于短数据类型(短整型或字符类型)被定义为标准整型不会对性能造成影响,因此程序员不会深入分析操作数的有效位宽,而把一些短数据类型定义为标准整型;另一方面,作为多媒体程序的主要开发语言,C 语言中 Integer Promotion 规则规定所有短数据类型的整数必须被扩充到标准整型以保留运算过程中的数据溢出部分.这些编程习惯和高级语言的规则极大阻碍了针对 SIMD 架构优化过程中高并行度指令的选择.

本文针对 SIMD 优化面临的并行度问题,提出了一种优化算法,该算法首先对应用程序进行有效位分析,使优化过程中更多的采用并行度高的指令成为可能;然后在此分析的基础上,提出了一种溢出控制算法,控制短数据类型操作过程中可能出现的溢出.实验数据表明,该算法可以有效提高 SIMD 优化过程中的并行度,对多媒体程序可以获得较好的加速效果.

本文第 2 节将介绍与本文相关的基础知识以及问题分析;第 3 节介绍有效位分析算法;第 4 节介绍溢出控制算法;第 5 节给出实验结果及对实验结果的分析;第 6 节给出相关工作;第 7 节对全文进行总结.

2　问题分析

2.1　背景知识

2.1.1　SIMD 架构

目前,SIMD 功能单元作为多媒体加速部件越来越广泛地应用到各种处理器中.在典型的 SIMD 架构中,同一操作对不同的数据位宽提供了不同的指令版本.随着操作数位宽的增加,对应的 SIMD 指令可同时完成的操作个数也随之降低.以 Intel 公司集成于奔腾处理器芯片中的 SSE2 指令集为例:SSE2 扩展指令集由一组基于 128 位 SIMD 寄存器的向量指令构成.通常每一种向量操作都对应 3 条 SSE2 指令,这 3 条指令分别可同时完成 4 个标准整型(INT-32 位)的操作,8 个短整型(Short-16 位)的操作或 16 个字符类型(Char-8 位)的操作. 同时,在 SIMD 架构中,每一个 SIMD 算术指令都有两种模式:标准模式和饱和模式.在标准模式下,当计算结果发生溢出时处理器会自动去掉溢出部分,计算结果取与该数据类型相应的低位.在饱和模式下,当计算结果发生溢出(上溢或下溢)时,处理器会自动去掉溢出的部分,使计算结果取该数据类型表示数值的上限值(如果上溢)或下限值(如果下溢).比如,对于一个值为 255 的字符类型变量,将其值加 1,在标准模式下,相加结果为 0(去掉进位);在饱和模式下,结果为 255.饱和模式用类似的方法来处理下溢出,比如对于一个字节数据类型的数在饱和模式下,1 减 2 的结果为 0(而不是-1).

2.1.2　Integer Promotion 规则

在 C 语言规范 C99 中,对整数计算有如下要求:当一个较短的有符号或无符号整型运算中可能出现溢出时,计算应该被扩展到标准整型.也就是说,在较短的整型计算中,如果出现数据溢出,这些溢出部分将被保留下来并参加以后的运算.当使用普通标量指令实现这些运算时,操作数将自动被扩展到标准整型,运算也完全按标准整型来进行,因此这一规范得到严格的遵守.这一规则被称作 Integer Promotion,在下文中我们简称这个规则为 IP 规则.

2.2　问题分析

由于 SIMD 指令集合提供一系列并行度各异的向量指令,因此利用 SIMD 指令优化相关程序时,在满足计算精度的要求下,选择并行度高的指令能得到更好的优化效果.然而一些传统的编程习惯和高级语言的规则却对高并行度指令的选择造成了很大

的限制.

一方面,由于传统的多媒体程序主要运行在通用处理器上,短数据类型被定义为长数据类型并不会对性能造成影响.因此,传统的多媒体程序员并不会深入分析各个操作数所需要的有效位宽,而是把不确定宽度的整型数据类型直接定义为标准整型,这种情况造成的直接后果是在一些多媒体程序的许多计算过程中,有一些中间结果的部分数据位并不参与之后的计算,或是部分数据位虽然参与后续的计算,但它们的取值不影响最终的计算结果.我们将这样的数据位称为无效位,而将其它数据位称作有效位.虽然这种情况不会影响这些应用程序在传统通用处理器上的执行效率,但是却给针对 SIMD 架构的编译优化造成了极大的障碍.

另一方面,基于寄存器的 SIMD 构架将若干个字符类型或者短整型数据紧密排列在一起进行统一的运算,这种构架充分利用了寄存器的宽度,但是并没为 IP 规则预留足够的空间.按照 IP 规则,所有短数据类型数据都应该被扩展到标准整型进行计算,在计算完毕后再根据目标变量长度进行剪裁.在 SIMD 构架上进行这样的操作会引入大量的额外开销,扩展数据用的指令和裁剪数据用的指令都需要较多的 CPU 周期才能完成;如果数据是带符号的整数,还需要通过一系列操作来获取每一个操作数的符号位(0 或者 -1).这些额外开销,往往会抵消 SIMD 优化带来的性能提高,甚至导致负的加速效果.

对于标准模式,由于存在短类型基于 IP 规则到 32 位的扩展,因此只要在计算中不出现右移,是否严格遵循 IP 规则对计算结果不会造成任何影响.然而,当标准模式中出现右移或处于饱和模式时,高位数据将在计算中发挥作用,是否严格遵循 IP 规则将影响最终结果.如果严格遵循 IP 规则,短数据类型的右移操作实际上都是标准整型的右移操作.也就是说,对于字符类型的运算来说,运算过程中的第 9～32 位数据将影响到最终的计算结果,而对短整型运算第 17～32 位数据将影响到最终的计算结果.但是,由于这些数据属于计算的溢出部分,在标准模式下,这些数据都会被处理器自动去掉从而导致最终结果的错误.为此,如果在计算中出现右移操作,不得不引入整数扩展,并且忍受随之而来额外开销.而在 IP 规则下,参与饱和算术运算的不仅仅是低位的数据,也包括了高位的数据.而高位数据及运算的高位结果对饱和运算的最终结果也起到了决定性的作用,在进行饱和计算之前就把溢出部分处理掉,不管使用的是标准模式还是饱和模式都会导致最终计算结果的错误.因此,为了得到正确的计算结果,必须保留数据和运算结果的高位部分.然而,通过整数扩展来保留数据和运算结果的高位部分,往往会使得饱和算术指令的操作数位数大于计算结果以及饱和算术本身的位数,而目前的 SIMD 架构并不支持这样的饱和算术指令.也就是说,即使进行了整数扩展,仍然无法从饱和算术指令中获得应有的性能提高.

因此,在 SIMD 优化过程中,必须首先识别代码中变量的有效位,并在此基础上通过溢出控制算法来确保降低额外的开销和保证程序的正确性.

3 有效位分析算法

在多媒体程序中,通常有两种途径来对操作数的位宽进行压缩:

(1)高位压缩.由于多媒体应用所处理的数据多为 8 位或 16 位整数,因此在计算过程中高位部分往往对最终计算结果没有影响.

(2)低位压缩.多媒体应用有一个显著的特点,即对计算精度的要求较低.这一特点在代码中常常表现为低位计算往往对最终结果没有影响.

因此在编译时通过有效位分析计算过程中高位/低位的作用,压缩不影响最终计算结果的高位/低位以提高 SIMD 计算的并行度.为了达到这样的目的,可以将变量与计算中间结果的数据位划分成高端无效位、有效位和低端无效位三个部分.为了便于在分析过程中描述,我们用有效区间来描述变量或计算中间结果的有效数据部分.有效区间下限为有效位中的最低位,而区间的上限为有效位中的最高位.图 1 给出了代码 1 中各变量与计算中间结果的数据位划分.

有效位分析与传统位宽分析一样,均属于双向的数据流分析.但除了能够像传统位宽分析那样预测正确实现代码所必须的计算精度外,还必须提供变量以及计算中间结果的各个数据位可能的取值范围以及对计算结果的影响.因此,有效位分析需要编译器在中间代码分析过程中传播变量以及中间结果的有效区间,基于文献[6]中的位宽分析算法,有效位分析方法可以通过以下 4 个步骤完成:

(1)将代码中的复合表达式分解成为一系列一元/二元表达式,并引入临时变量来存储复合表达式

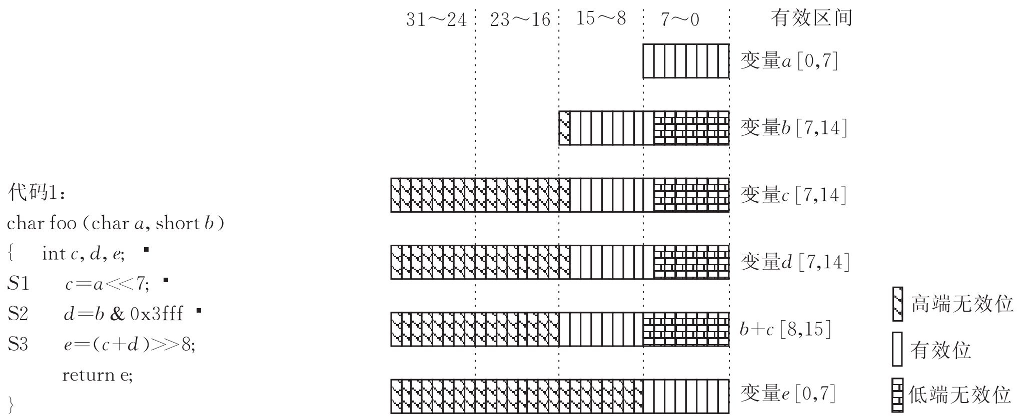

图 1　代码 1 中各变量数据位的分类

计算的中间结果. 根据 C 语言标准的要求，这些临时变量将被申明成为标准整型.

(2) 在开始传播有效位区间之前，为变量、常数和常数数组设定有效位区间的初始值. 变量的有效位区间初值可以根据变量申明的数据类型来进行设定，表 1 给出了不同数据类型的有效区间的初始值. 常数的有效位区间可以通过常数数值获取. 与变量和常数不同，常数数组在代码中以数组变量的形式出现，然而这一数组的每一个元素都是一个确定的常数. 在有效位区间传播过程中，我们将这样的常数数组当作特殊的常数，其有效位区间的上界为所有元素有效位区间上界的最大值，而下界为所有元素有效位区间下界的最小值.

表 1　数据类型与有效位区间初值对照表

数据类型	有效位区间初值
Char	[0,7]
Short	[0,15]
Int	[0,31]

(3) 设定变量有效位区间初值后，前向遍历静态单赋值语法树，在每一个赋值节点上依照文献[6]中的正向传播规则传播变量的有效位区间.

(4) 完成有效位区间的前向传播后，逆向遍历静态单赋值语法树. 并在每一个赋值节点上依照文献[6]中的逆向传播规则传播变量的有效位区间.

图 2 给出了代码 1 广义有效位区间在这段代码的静态单赋值语法树上的传播过程. 在语法树的左边是前向传播的过程，而右边则是逆向传播的过程.

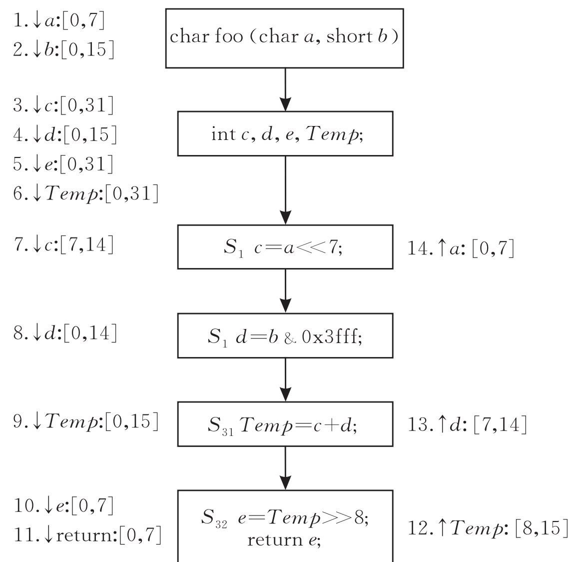

图 2　有效位区间的双向传播示例

4　溢出控制算法

经过有效位分析后，可以获得更多的短数据类型操作，在此基础上，我们进行溢出控制，来降低短数据类型对更高宽度类型的依赖以消去 IP 规则对优化过程中并行度的影响.

4.1　标准模式的溢出控制算法

对于标准模式，由于存在短类型(8 位或 16 位)基于 IP 规则到 32 位的扩展，只要在计算中不出现右移，是否严格遵循 IP 规则对计算结果不会造成任何影响. 因此标准模式下的溢出控制算法主要是通过相对提前右移运算，延迟溢出的产生，使得在右移运算之前不会有溢出产生，从而保证溢出不会影响到右移操作的结果. 在本节中将从最简单的表达式开始介绍基本的变换，而复杂的表达式可以通过递归使用基本变换实现溢出控制. 此外，变换的递归使用会造成表达式的复杂化，本节也将给出对应的表达式简化方法. 由于只有短整型经过扩展后才会引入右移溢出问题，因此在本节以后部分中，如不特殊说明，一个变量均为经过 IP 规则从短整型变量扩展的标准整型变量. 而由于移位操作仍针对这些短整

型数据类型的有效位，而左移/右移宽度大于变量有效宽度时则无实际意义，因此也假设后继讨论中左移或右移操作的宽度均不大于对应数据有效位的宽度.

4.1.1 基本表达式变换

由于右移对其操作数的溢出敏感，因此，考虑表达式 $SubExp \gg N$，其中子表达式 $SubExp$ 有溢出的可能. 我们讨论以下 7 种基本表达式(其中 E_1，E_2 可能是变量、数组或者表达式，m，n 是正整数)：

1) $(E_1 \& E_2) \gg n$.

2) $(E_1 | E_2) \gg n$.

3) $(E_1 \gg m) \gg n$.

4) $(E_2 \ll m) \gg n$.

5) $(E_1 + E_2) \gg n$.

6) $(E_1 - E_2) \gg n$.

7) $(E_1 * E_2) \gg n$.

表达式 1)，2)可以利用式(1)，(2)将右移操作前提.

$$(E_1 \& E_2) \gg n = (E_1 \gg n) \& (E_2 \gg n) \tag{1}$$

$$(E_1 \mid E_2) \gg n = (E_1 \gg n) \mid (E_2 \gg n) \tag{2}$$

表达式 3)，4)则可利用式(3)，(4)将左右位移操作合并起来

$$(E_1 \gg m) \gg n = E_1 \gg (m+n) \tag{3}$$

$$(E \ll m) \gg n = \begin{cases} E \gg (n-m) (\text{if } n-m > 0) \\ E \ll (m-n) (\text{if } n-m < 0) \\ E \ (\text{if } n-m = 0) \end{cases} \tag{4}$$

对于表达式 5)，6)，7)，则可应用式(5)，(6)，(7)进行变换

$$(E_1 + E_2) \gg n = (E_1 \gg n) + (E_2 \gg n) + (((E_1 \& (2^n - 1)) + (E_2 \& (2^n - 1))) \gg n) \tag{5}$$

$$(E_1 - E_2) \gg n = (E_1 \gg n) - (E_2 \gg n) + (((E_1 \& (2^n - 1)) - (E_2 \& (2^n - 1))) \gg n) \tag{6}$$

$$(E_1 * E_2) \gg n = (E_1 \gg n) * E_2 + ((E_1 \& (2^n - 1)) * E_2 \gg n) \tag{7}$$

以上 7 个公式中式(1)～式(4)是显然成立的，而式(5)～(7)则是建立在分配律和交换律的基础之上. 由于右移 n 位与除以 2^n 在实数域上是等价的，因此可以把除法的运算法则应用于右移计算. 然而分配律和交换律仅仅成立在实数域上，在整数域上应用分配律和交换律会导致精度损失，因此除了结合律和交换律的变换以外，还需要额外计算可能损失的精度以保证计算结果的正确性.

在式(5)～(7)的等式右侧，有用于计算低位进位的子表达式. 这些表达式都包含右移操作，如果这些右移操作的子表达式可能产生溢出，那么变换无法避免溢出对右移操作的影响. 由于这里的子表达式中进行的是低位运算，因此溢出的可能性概率很低. 然而为了保证变换的有效性，必须检查低位运算是否会产生溢出. 如果表达式 8)～10)的计算中间结果没有溢出产生，则变换是有效的.

8) $((E_1 \& (2^n - 1)) + (E_2 \& (2^n - 1))) \gg n$.

9) $((E_1 \& (2^n - 1)) - (E_2 \& (2^n - 1))) \gg n$.

10) $(E_1 \& (2^n - 1)) * E_2 \gg n$.

4.1.2 表达式化简

通过递归使用式(1)～(7)，将所有的右移调整到临界点(临界点是表达式中可能导致溢出的运算，且这个运算之前的子表达式不会溢出)之前，就可使右移操作不受溢出的影响，从而在随后的操作中不为 IP 规则保留空间. 但是，递归使用这些公式，会使整个表达式的计算急剧膨胀. 下面是这种情况的一个例子：如果对表达式 $(a+b+c) \gg 4$ 递归应用前面的公式，这个表达式将变形成为表达式 11). 显然即使是被向量化，如此复杂的计算势必占用大量计算时间.

11) $a \gg 4 + b \gg 4 + c \gg 4 + ((a \& 0xf) + (b \& 0xf)) \gg 4 + (((a+b) \& 0xf) + (c \& 0xf)) \gg 4$.

为了降低递归优化带来的额外开销，必须简化这些优化后的复杂表达式. 由于绝大部分复杂表达式都可以看作是 12)～14)嵌套所得，因此，只要递归应用针对这 3 个表达式的简化方法，我们就可以简化相应的复杂表达式.

12) $((E_1 + E_2) + E_3) \gg n$.

13) $((E_1 + E_2) * E_3) \gg n$.

14) $((E_1 * E_2) + E_3) \gg n$.

经过递归变换，表达式 12)将被变形为表达式 15).

15) $E_1 \gg n + E_2 \gg n + E_3 \gg n + (E_1 \& (2^n - 1) + E_2 \& (2^n - 1)) \gg n + ((E_1 + E_2) \& (2^n - 1) + E_3 \& (2^n - 1)) \gg n$.

这里，我们试图重组表达式 $(E_1 \& (2^n - 1) + E_2 \& (2^n - 1)) \gg n + ((E_1 + E_2) \& (2^n - 1) + E_3 \& (2^n - 1)) \gg n$. 其中 $(E_1 \& (2^n - 1) + E_2 \& (2^n - 1)) \gg n$ 是 $E_1 \& (2^n - 1)$ 加上 $E_2 \& (2^n - 1)$ 的第 $n+1$ 位，而 $(E_1 + E_2) \& (2^n - 1)$ 则可被看作 $E_1 \& (2^n - 1)$ 加上 $E_2 \& (2^n - 1)$ 的低 n 位. 经过表达式化简，可以得到等式(8). 类似的，基于表达式 13)和 14)我们还可以

演绎出等式(9)和(10).

$$(E_1 \& (2^n-1)+E_2 \& (2^n-1)) \gg n+((E_1+E_2) \& (2^n-1)+E_3 \& (2^n-1)) \gg n = (E_1 \& (2^n-1)+E_2 \& (2^n-1)+E_3 \& (2^n-1)) \gg n \tag{8}$$

$$((E_1 \& (2^n-1)+E_2 \& (2^n-1)) \gg n) * E_3 + (((E_1+E_2) \& (2^n-1)) * E_3) \gg n = E_3 * ((E_1 \& (2^n-1))+(E_2 \& (2^n-1))) \gg n \tag{9}$$

$$((E_1 \& (2^n-1)) * E_2) \gg n+((E_1 * E_2) \& (2^n-1)+E_3 \& (2^n-1)) \gg n = ((E_1 \& (2^n-1)) * (E_2 \& (2^n-1))+E_3 \& (2^n-1)) \gg n \tag{10}$$

对于表达式 12)、13)和 14)我们首先根据式(1)~(7)进行溢出控制变形,然后利用式(8)~(10)化简变形后复杂的表达式,最终可以得到表达式 16)、17)和 18).对于其它复杂的表达式,我们也可以递归利用等式(8)~(10)以及等式(1)~(7)进行化简,使得经过溢出控制变形的表达式变得更高效.

16) $E_1 \gg n+E_2 \gg n+E_3 \gg n+(E_1 \& (2^n-1)+E_2 \& (2^n-1)+E_3 \& (2^n-1)) \gg n$.

17) $(E_1 \gg n+E_2 \gg n) * E_3+((E_1 \& (2^n-1)+E_2 \& (2^n-1)) * E_3) \gg n$.

18) $(E_1 \gg n) * E_2+E_3 \gg n+((E_1 \& (2^n-1)) * E_2+E_3 \& (2^n-1)) \gg n$.

4.2 饱和模式下的溢出控制方法

本节我们将给出在操作数溢出时利用饱和算术指令进行编译优化的方法.作为溢出方式控制的基础,我们将首先介绍饱和算术的结合律——条件结合律.然后在这一定律的基础上,给出针对饱和模式的溢出控制方法.

4.2.1 条件结合律

在有些情况下,饱和运算并不满足结合率.下面是这种情况的一个例子:

表达式 $a \oplus_{16} b \oplus_{16} c$.其中变量 a,b,c 是有符号的 16 位整数,符号"$\oplus_{16}$"表示 16 位的饱和加法.当 $a=$ 0x7FF5,$b=$0x14 和 $c=$0xFFEC 时,$(a \oplus_{16} b) \oplus_{16} c=$ 0x7FEB 而 $a \oplus_{16}(b \oplus_{16} c)=$0x7FFF.

虽然,普通的算术结合律不能应用在饱和算术上,但只要增加一些限制条件,结合律在饱和算术上仍然成立.我们称这种针对饱和运算在限定条件下的结合率为条件结合律.

条件结合律:如果所有加数/减数 $E_i(i=2,\cdots,n)$的符号一致,或者为 0,那么有等式(11)、(12)成立(其中$\oplus_m$表示 m 位的饱和加法,$\ominus_m$表示 m 位的饱和减法).

$$E_1 \oplus_m E_2 \oplus_m E_3 \oplus_m \oplus \cdots \oplus_m E_n == E_1 \oplus_m (E_2+\cdots+E_n) \tag{11}$$

$$E_1 \ominus_m E_2 \ominus_m E_3 \ominus_m \ominus \cdots \ominus_m E_n == E_1 \ominus_m (E_2+\cdots+E_n) \tag{12}$$

4.2.2 条件结合律的表达式变形

由于操作数的溢出部分的保留与否,会直接影响到饱和算术结果的正确性.因此一旦在计算过程中存在溢出的可能,就必须扩展操作数单元的宽度为可能发生的溢出留出足够的空间.然而扩展操作数单元的宽度会造成操作数单元的宽度与饱和算术的宽度不一致,从而无法利用饱和 SIMD 指令进行编译优化.然而通过分析,可以将这种无法用饱和 SIMD 指令进行优化的情况归纳为:$E_1 \oplus_m E_2$,是 m 位的饱和加/减法,其中表达式 E_1 的结果能够用 m 位整数表示,而 E_2 的结果则需要 $m+p$ 位整数才能完整表示.

如果能将表达式 E_2 拆分成若干个可以用 n 个 m 位表示的整数 $E_{21} \cdots E_{2n}$ 之和,而且这若干个数有着相同的符号或者为 0,那么表达式 $E_1 \oplus_m E_2$ 可以被写成表达式 19).再根据饱和算术的条件结合律,我们可以将表达式 19)进一步变形为表达式 20).而在表达式 20)的计算过程中,不会有溢出产生,因此可以利用 m 位的饱和算术 SIMD 指令实现该表达式的计算.

19) $E_1 \oplus_m (E_{21}+\cdots+E_{2n})$ ($E_{2i}(i=1,\cdots,n)$都有相同的符号或者为 0).

20) $E_1 \oplus_m E_{21} \oplus_m \oplus \cdots \oplus_m E_{2n}$.

4.2.3 溢出控制

在上述表达式变形的基础上,我们首先给出当 $p=1$ 时,也就是 E_2 只溢出一位时的溢出控制方法.根据上节的分析可知,如果能将 E_2 拆分成若干个能用 m 位表示的整数之和,那么 $E_1 \oplus_m E_2$ 就可以变形为表达式 19).当 E_2 需要用 $m+1$ 位来表示时,我们可以将 E_2 拆分成 $E_2 \gg 1$,$E_2 \gg 1$ 和 $E_2 \& 1$ 之和.这里 $E_2 \gg 1$ 用 m 位整数可以表示而 $E_2 \& 1$ 只需要 1 位来表示.因此,如果 $E_2 \gg 1$ 与 $E_2 \& 1$ 的符号相同或者其中有一个为 0,那么我们就可以把 $E_1 \oplus m E_2$ 变形为表达式 21).

21) $E_1 \oplus_m (E_2 \gg 1) \oplus_m (E_2 \gg 1) \oplus_m (E_2 \& 1)$,

即

$$E_1 \oplus_m E_2 = E_1 \oplus_m (E_2 \gg 1) \oplus_m (E_2 \gg 1) \oplus_m (E_2 \& 1) \tag{13}$$

然而可惜的是 $E_2 \& 1$ 只能等于 0 或者 1,而当 E_2 小于 0 时,$E_2 \gg 1$ 必定是一个负数.也就是说当

E_2是一个小于 0 的奇数时，等式(13)是不成立的. 为了解决这个问题，我们将等式(13)改造成等式(14)：

$$E_1 \oplus_m E_2 = E_1 \oplus_m (E_2 \gg 1) \oplus_m ((E_2 \gg 1)+(E_2 \& 1)) \tag{14}$$

表 2 给出了 E_2取不同值的时候，等式(14)的子表达式 $E_2 \gg 1$ 与 $(E_2 \gg 1)+(E_2 \& 1)$ 可能的取值范围，这张表说明，无论 E_2 取什么值，$E_2 \gg 1$ 与 $(E_2 \gg 1)+(E_2 \& 1)$ 都不会是不同符号的.

表 2 表达式 $E_2 \gg 1$ 与 $(E_2 \gg 1)+(E_2 \& 1)$ 的取值范围

E_2	$E_2 \& 1$	$E_2 \gg 1$	$(E_2 \gg 1)+(E_2 \& 1)$
≥ 0	≥ 0	≥ 0	≥ 0
-1	1	-1	0
-2	0	-1	-1
<-2	0 或 1	<-1	<0

因此无论 E_2的取值范围如何，等式(14)是一定成立的，而且饱和算术的操作数也不会溢出. 进一步将这个优化方法扩展到 $p=n$ 的情况，即可以得到等式(15). 利用等式(15)我们可以将有一个操作数溢出 n 位的饱和算术，替换成为若干个没有溢出的饱和算术.

$$\begin{aligned}
E_1 \oplus_m E_2 = E_1 \\
&\oplus_m [(E_2 \gg n)+(E_2 \& 2^n \gg (n-1))] \cdots (2^n \text{个}) \cdots \\
&\oplus_m [(E_2 \gg n)+(E_2 \& 2^n \gg n-1)] \\
&\oplus_m [(E_2 \gg n)+(E_2 \& 2^{n-1} \gg (n-2))] \cdots (2^{n-1} \text{个}) \\
&\quad \cdots \\
&\oplus_m [(E_2 \gg n)+(E_2 \& 2^{n-1} \gg n-1)] \\
&\oplus_m \cdots \oplus_m [(E_2 \gg n)+(E_2 \& 1)] \\
&\oplus_m (E_2 \gg n)
\end{aligned} \tag{15}$$

4.2.4 值域检查

当 $E_2=2^{m+1}-1$ 时，$E_2/2+(E_2 \& 1)=2^m$，而 2^m 需要用 $m+1$ 位来表示. 类似的，当 $E_2>2^{m+n}-2^n+1$ 时，$(E2 \gg n)+(E_2 \& 2^i \gg (i-1))$，$i=0,\cdots,n-1$，也需要用 $m+1$ 位整数才能表示. 在这种情况下，只要利用 E_2需要 $m+n+1$ 位整数来表示的等式进行变换就可以避免这个问题了. 因此，在进行优化变换之前，除了要分析表示 E_2所需要的位数以外，还需要分析 E_2的取值范围，如果 E_2可以用 $m+n$ 位整数来表示而且 E_2的最大值不会大于 $2^{m+n}-2^n+1$，那么可以按照 $p=n$ 的等式进行优化. 而如果 E_2的最大值有可能大于 $2^{m+n}-2^n+1$，则需要按照 $p=n+1$ 的等式进行优化.

5 实验结果

为了说明位宽控制优化算法(BCSA)的优化效果①，我们以 Intel 公司的 ICC 编译器作为比较的基础. 之所以选择 ICC，主要是由于 ICC 是 Intel 公司开发的提供了向量化功能的编译器[7-8]，同时 ICC 也是目前针对 SIMD 优化性能最好的商用编译器. 然而由于 ICC 是商业编译器，无法拿到其源代码，因此只能通过手工优化嵌入内嵌函数，再通过 ICC 来生成代码.

实验的硬件平台：CPU 为 Intel Pentium 2GHz，内存为 512MB. 操作系统为 redhat9. 我们选择了 BMW(Berkeley Multimedia Workload)[12] 中的 ADPCM-Decoder、DJVU-Encoder、MESA-Reflect 3 个程序来测试优化效果. 这 3 个测试程序是比较典型的多媒体测试程序，除 BMW 将其作为典型多媒体测试程序收录外，另一典型多媒体测试程序集 Mediabench 中也包含这 3 个测试程序，因此使用这 3 个作为测试程序具有一定的代表性.

实验数据如表 3 所示，其中，ICC 表示 ICC 关闭自动向量化功能编译得到的测试结果；AVI 表示 ICC 自动向量化的测试结果；TV 表示利用传统向量化方法[9-10]，手工向量化的测试结果；BCSA 表示手工应用位宽优化算法后的测试结果；TV vs.BCSA 表示与仅使用传统向量化方法相比，位宽优化方法取得的加速比.

通过表 3 的数据可知，ICC 针对 SIMD 构架的编译优化，不能使这 3 个程序的性能得到任何的提高. 而我们利用传统的自动向量化方法，能够使 ADPCM-Decoder 的性能有所提高，但是会使 DJVU-Encoder 和 MESA-Reflect 的性能下降. 而使用了 BCSA 后，相比串行程序 ADPCM-Decoder 的性能提高了 46%，DJVU-Encoder 的性能降低了 1%，Mesa-Reflect 的性能提高了 13%. 而相比没有 BCSA

表 3 手工优化数据

基准算法	ADPCM-Decoder		DJVU-Encoder		MESA-Reflect	
	时间/s	加速比	时间/s	加速比	时间/s	加速比
ICC	0.295	1x	10.73	1x	14.48	1x
AVI	0.295	1x	10.73	1x	14.48	1x
TV	0.24	1.22x	11.48	0.93x	14.58	0.99x
BCSA	0.201	1.46x	10.76	0.99x	12.71	1.13x
TV vs. BCSA		1.19x		1.06x		1.14x

① 我们利用了文献[8]中的算法来识别 C 代码中的饱和算术部分.

的向量化程序，ADPCM-Decoder 的性能提高了19%，DJVU-Encoder 的性能提高了 6%，MESA-reflect 的性能提高了 14%.

为了进一步验证本文算法的效果，我们在 GCC 3.4 中实现了本文中的算法，并分别与未向量化、手工优化、不包含溢出控制的向量化算法的性能进行了比较，具体数据如表 4 所示，其中：

GCC 3.4 为使用标准 GCC3.4 编译，选用-O2 优化选项获得的数据；

GCC3.4-M 表示使用 GCC 3.4 编译手工优化代码；

GCC 3.4-V 表示使用修改过的 GCC3.4 进行自动量化但不使用溢出控制；

GCC3.4-BCSA：使用修改过的 GCC3.4 进行自动量化并使用位宽优化算法.

表 4 的数据表明本文所介绍的位宽优化方法不仅适用于手工优化，同时也适用于实现在自动编译优化，并达到了与手工优化接近的效果.

表 4　编译优化数据

基准算法	ADPCM-Decoder		DJVU-Encoder		MESA-Reflect	
	时间/s	加速比	时间/s	加速比	时间/s	加速比
Gcc 3.4	0.2901	1x	13.0334	1x	11.0105	1x
Gcc 3.4-M	0.2048	1.42x	12.9896	1x	9.8215	1.12x
Gcc 3.4-V	0.2194	1.32x	13.5036	0.95x	10.8620	1.01x
Gcc 3.4-BCSA	0.2082	1.39x	13.0382	1x	9.8215	1.12x
BCSA-加速比		1.05x		1.05x		1.11x

6　相关工作

尽管传统的针对向量机的编译优化[9-10]对于 SIMD 编译优化具有很高的参考价值，很多研究成果可以直接应用到 SIMD 编译优化，但由于多媒体扩展指令集和传统向量机之间存在很多差异，所以仍然有很多大学和公司就此展开了大量研究. 这方面的研究可以分为以下几类：

(1) SIMD 编译优化特点的研究. SIMD 编译优化和传统向量化针对的是不同的硬件平台和应用领域，所以并不能机械地照搬已有的研究结果. 文献[11]从理想硬件的角度探讨了在实际程序中可能获得的 SIMD 并行性. 文献[13]对多媒体程序进行 SIMD 优化后引进的问题进行了分析，其分析认为造成性能不足的主要原因是 SIMD 优化引入的额外操作，并针对该问题给出了一种解决方案. 文献[6]提供了一种位宽分析方法，然而由于缺少有效溢出控制分析，因此无法应用到针对 SIMD 功能单元的优化中.

(2) 可向量化操作的识别. 为了从多媒体程序代码中识别可向量化的多媒体操作，很多方法被提出过. 例如，简单的模式匹配算法，如文献[14]要求编译器设计人员枚举出多媒体典型操作的所有种类和形式. 然后，编译器根据这些模式机械的和源代码进行匹配以判定某个代码块是否是多媒体典型操作. 但这种方法太过死板. 它有如下的问题：(a) 编译器设计人员无法枚举出所有的可能形式，模式库太过庞大. (b) 无法处理像操作嵌套这样的情况. (c) 无法识别必须用多条语句表达的可向量化操作. 文献[8]提出了一系列的代码规范化方法，有效地对多媒体程序中的典型代码形式进行规范化，从而保证识别的质量. 文献[15]解决了 SIMD 优化过程中相关性分析的问题并对 SIMD 编译优化中的相关性问题及其与传统向量化的差异进行了详细探讨.

(3) 内存的优化. 目前，针对 SIMD 内存的优化主要集中在数据对齐问题和改进内存访问的连续性问题两个方面：

(a) 数据对齐问题. 文献[16]主要针对多媒体程序中广泛使用的指针，讨论了指针情况下的对齐问题；文献[17]讨论在进行 SIMD 编译优化的过程中，如何进行对齐向量数据的读取以提高整个处理的效率.

(b) 内存访问连续性问题. 文献[18]针对一些多媒体程序编译优化中频繁出现的 permutation 操作，提出了一系列理论和算法，通过合并 permutation 操作来减少 permutation 操作的数目，提高程序的效果. 文献[19]针对多媒体优化中数据不连续时，需要进行数据收集的问题，提出了一种优化算法，当数据向量中各个元素的原始地址距离为 2 的幂次时，该算法可以有效地进行数据抽取，提高生成代码的执行效率.

7　结　论

本文提出了一种针对 SIMD 架构的位宽优化算法. 通过该算法，可以有效提高 SIMD 指令内部的并行度. 在手工优化实验中，位宽优化的 SIMD 算法对提高 ADPCM-Decoders、DJVU-Encoder 和 MESA-Reflects 这 3 个多媒体测试程序的性能起到了显著的积极的作用. 另一方面，由于算法本身在实现上没有困难，因此，这一算法可以很容易被移植到编译优

化中去.我们已经在开放源代码编译器 GCC-3.4 中实现了这一优化方法,并取得了相应的效果.

参 考 文 献

[1] Diefendorff K, Dubey P K. How multimedia workloads will change processor design. IEEE Computer, 1997, 30(9): 43-45

[2] Krall A, Lelait S. Compilation techniques for multimedia processor. International Journal of Parallel Programming, 2000, 18(4): 347-361

[3] INTERNATIONAL STANDARD © ISO/IEC ISO/IEC 9899: 1999

[4] IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture.

[5] IA-32 Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference.

[6] Stephenson M, Babb J, Amarasinghe S. Bitwidth analysis with application to silicon compilation//Proceedings of the ACM SIGPLAN conference on Programming Language Design and Implementation0 Vancouver, British Columbia, 2000: 108-120

[7] Bik A J C, Girkae M, Grey P M, Tian Xin-Min. Automatic intra-register vectorization for intel Architecture. International Journal of Parallel Programming, 2002, 30(2): 65-98

[8] Jiang Wei-Hua, Zhu Jia-Hua, Zang Bin-Yu, Zhu Chuan-Qi. Boosting the performance of multimedia applications by using SIMD instructions//Proceedings of the 14th International Conference on Compiler Construction (CC). Edinburgh: Springer-Verlag, 2005: 59-75

[9] Padua D, Wolfe M. Advanced compiler optimizations for supercomputers. Communications of the ACM, 1986, 29(12): 1184-1201

[10] Bacon D F, Graham S L, Sharp O J. Compiler transformations for high-performance computing. ACM Computing Surveys, 1994, 26(4): 345-420

[11] Scott K, Davidson J. Exploring the limits of sub-word level parallelism//Proceedings of the 2000 International Conference on Parallel Architectures and Compilation Techniques. Philadelphia, PA, USA, 2000: 81

[12] Slingerland N, Smith A J. Design and characterization of the Berkeley multimedia workload. Multimedia Systems, 2002, 8(4): 315-327

[13] Talla D, John L K, Burger D. Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements. IEEE Transactions on Computers, 2003, 52(8): 1015-1031

[14] Boekhold M, Karkowski I, Corporaal H. Transforming and parallelizing ANSI C programs using pattern recognition. Lecture Notes in Computer Science, 1999: 673-682

[15] Bulic P, Gustin V. Data dependence analysis for intra-register vectorization//Proceedings of the 2nd International Symposium on Parallel and Distributed Computing. Ljubljana, Slovenia 2003: 50-56

[16] Pryanishnikov I, Krall A, Horspool N. Pointer alignment analysis for processors with SIMD instructions//Proceedings of the 5th Workshop on Media and Stream Processors. Seoul, Korea. 2003: 148-153

[17] Eichenberger A E, Wu P, O'Brien K. Vectorization for SIMD architectures with alignment constraints//Proceedings of the 2004 Conference on Programming Language Design and Implementation. Washington, DC, 2004: 82-93

[18] Ren Gang, Wu Peng, Padua D. Optimizing data permutations for SIMD devices//Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation. Ottawa, Canada, 2006: 118-131

[19] Nuzman D, Rosen I, Zaks A. Auto-vectorization of interleaved data for SIMD devices//Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation. Ottawa, Canada, 2006: 132-143

ZHANG Wei-Hua, born in 1974, Ph. D., assistant professor. His main research interests are parallel computing and compiling optimization.

ZHU Jia-Hua, born in 1977, Ph. D.. His main research interests are parallel compiling and optimization.

ZHANG Hong-Jiang, born in 1981, master. His main research interests are parallel compiling and optimization.

ZANG Bin-Yu, born in 1965, Ph. D., professor. His current research interests include parallel compiler and computer architecture.

Background

Since multimedia has become a dominating computing field, to meet such a trend, almost all general purposed processor (GPP) venders have integrated multimedia extensions (MME) into their processors. Due to the potential parallelism and the low calculative precision requirement of multimedia applications most MME are implemented with Single In-

struction Multi Data (SIMD) instruction sets.

Currently, programmers are mainly restricted to utilize these SIMD instructions through in-lining assembly codes or intrinsic functions. With these methods, the development become extremely inefficient and the code would be hard to be transplanted between different platforms. An alternative way is to make compiler automatically generate SIMD instructions from the code of standard high level programming languages. Although SIMD optimization is a part of vectorization, the traditional vectorization technique could not be simply transplanted to SIMD optimization due to the differences between vector processor and SIMD architecture. Currently, there is only few compilers could speedup some individual multimedia applications.

With the support of Specialized Research Fund for the Doctoral Program of Chinese Higher Education under Grant No. 20050246020; the National Nature Science Foundation of China under Grand No. 60273046; Shanghai Science and Technology Committee of China Key Project Funding (02JC14013), the authors carried on a series research to develop efficient SIMD optimization techniques. Based on the deep study to the SIMD architecture and widely analyzing to the multimedia workload, they find out some useful techniques in this area, such as how to perform highly accurately data bit width analysis, how to develop potential parallelism in saturation arithmetic mode and how to automatically transform C programs into SIMD instructions based on Iburg. Meanwhile, the authors implemented these techniques with open source compiler Gcc3.5 and parallelization research platform Aggassiz as well. Experimental results show those methods are effective.