计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(9):40−45 [doi: 10.15888/j.cnki.csa.005954]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

魂芯 DSP 上复数类型的支持和优化①

王玉林, 郑启龙, 赵高义

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国科学技术大学 安徽省高性能计算重点实验室, 合肥 230027)

摘　要: 魂芯 DSP 是一款采用 VLIW 和 SIMD 架构的针对高性能计算领域而设计的 32bit 静态标量数字信号处理器. 为了满足数字高性能计算的性能要求, 魂芯 DSP 提供了丰富的复数指令, 而编译器不能直接利用这些复数指令来提升编译性能. 因此针对魂芯 DSP 芯片提供了大量的复数类操作指令的特点, 在传统开源编译器 Open64 的编译框架基础上进行研究, 实现了复数作为编译器基础类型和复数运算操作的支持. 同时, 通过识别特定的复数类操作的模式利用魂芯 DSP 上的复数类指令对程序编译优化. 实验结果表明, 该实现方案在魂芯 DSP 编译器上对复数程序优化后能够取得平均 5.28 的加速比.

关键词: 编译优化; 分簇体系 DSP; 复数指令; Open64 编译器

引用格式: 王玉林,郑启龙,赵高义.魂芯 DSP 上复数类型的支持和优化.计算机系统应用,2017,26(9):40–45. http://www.c-s-a.org.cn/1003-3254/5954.html

Complex Data Type Support and Optimization for BWDSP

WANG Yu-Lin, ZHENG Qi-Long, ZHAO Gao-Yi

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Anhui High Performance Computing Key Laboratory, University of Science and Technology of China, Hefei 230027, China)

Abstract: BWDSP is a 32bit static scalar digital signal processor with VLIW and SIMD features, which is designed for high performance computing. In order to meet the performance requirements of digital high-performance computing, the soul core DSP provides a rich set of complex instructions, and the compiler cannot directly use these complex instructions to improve the compilation performance. Since BWDSP has a wealth of complex type of instructions, and it has high performance demands in the radar digital signal field, the implementation is researched according to the characteristics of BWDSP features based on the traditional open-source Open64 compiler framework to achieve the complex data type and complex operations support operations, and further optimization of complex instruction is realized by identifying a specific type of complex operation of a series of patterns. The experimental results show that the implementation on BWDSP compiler can achieve 5.28 time performance improvement on average.

Key words: compiler optimization; multi-cluster DSP; complex instructions; Open64 compiler

复数分为实部和虚部, 可以用 $n=a+bi$ 来表示一个复数: 其中 a 表示复数的实部, b 表示复数的实部. 一次复数乘法操作 $n1\times n2=(a+bi)\times(c+di)$, 要做 4 次乘法运算、1 次加法运算、1 次减法运算, 在汇编语言层次, 要 6 条指令来完成一次复数乘法的操作.

魂芯 DSP[1]是中国电子科技集团第 38 研究所设计开发的一款高性能 DSP, 适用于雷达信号处理、电子对抗、通信保障等领域. 魂芯 DSP 芯片提供了大量的复数运算类指令, 但是现有的编译器框架是现有编译器框架无法识别出复数操作生成复数类指令, 是通过

① 基金项目: “核高基”重大专项(2012ZX01034-001-001)
收稿时间: 2016-12-28; 采用时间: 2017-01-18

把复数操作分解为实部和虚部单独处理来合成复数类的操作. 复数加法或者减法至少需要 2 条指令, 复数乘法就需要至少 6 条指令, 无法利用魂芯 DSP 上单周期复数指令来对提升程序的性能.

本文重点介绍基于开源 Open64 编译器来完成对复数的支持和优化. 主要是基于对开源编译器 Open64 的前端和后端的修改来完成复数类型的支持, 利用魂芯 DSP 上特有的复数类指令来, 通过定义一些复数类操作的模式, 把本来需要几条指令才能完成的操作, 优化成魂芯 DSP 上对应的一条指令. 这种针对特定高效指令模式识别的方式也适用于其他的平台.

1 研究背景

当应用程序中存在大量的复数乘法操作时, 程序执行时间相当程度上依赖于完成复数乘法操作的时间, 快速傅里叶变换[2](Fast FourierTransform, FFT)程序中就存在很多循环的复数加法和乘法操作.

C 语言的 C99 语法中规定了对复数操作的语法定义, 对于复数的定义如下:

$$z = a + b * i; a, b \in R \tag{1}$$

那么复数的乘法和除法定义为如下的操作, 可以看出来复数操作要实数 4 次乘法、一此加法、一次减法, 复数除法操作需要实数 6 次乘法、三次加法、两次除法、一次减法.

$$\begin{aligned}(a+ib)\times(c+id) &= (ac-bd)+i(ad+bc)\\ \frac{(a+ib)}{(c+id)} &= \frac{(ac+bd)}{(c^2+d^2)}+\frac{(bc-ad)}{(c^2+d^2)}i\end{aligned} \tag{2}$$

GCC 编译器[3]前端从 4.2 版本以后就支持 C99 语法, 也就是支持在 C 源程序支持复数的语义. 但是 GCC 后端对复数操作的指令实现, 是通过一组指令组合操作来得到复数操作的结果, 没有对应的高效的复数指令.

BWDSP 编译器采用开源 Open64[4]编译器基础设施作为编译器研究框架, 而 Open64 编译器是用的 GCC 的前端, 所以原有的编译器框架在移植到魂芯 DSP 上后对复数操作的编译结果和原有的 GCC 中实现基本是一样的. Open64 是一款 GPL 协议的工业级开源编译基础设施, 以中后端提供的强大的分析优化能力著称. 主要架构如图 1 所示.

Open64 开源编译器前端采取的是 GCC4.2 前端, 中间语言为抽象语法树 ASTtree. 高级语言经过前端时, 以语法 tree 的形式存在, 经由 gspin(一种从 gcc 的 tree 到 WHIRL 转换的中间表示)的媒介, 转换成为对应的 WHIRL[5]中间语言. Open64 的前端将源程序转化为中间表示 WHIRL, 后端读入 WHIRL, 经过翻译生成代码生成阶段(Code Generation, CG)的中间表示 CGIR, 在经过一系列优化, 最终 CGIR 经过代码输出生成汇编程序.

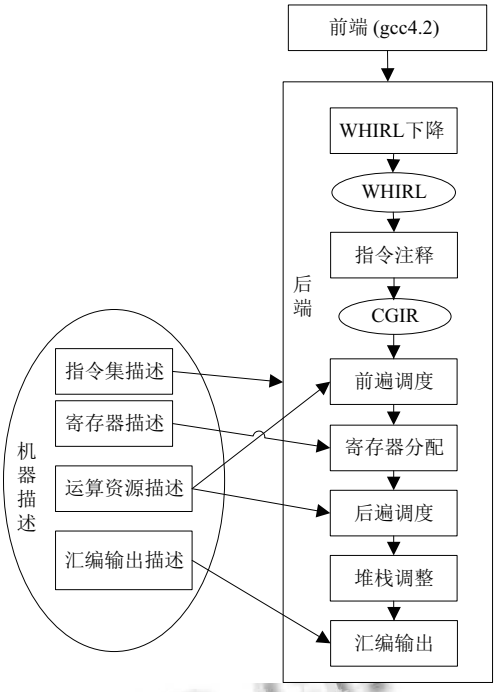

图 1 Open64 编译器架构

2 问题与解决方案

复数类型支持的基本的思路是把复数作为一种新的基础数据模型, 为这个新的数据类型建立一套 ADT (abstract data type)并利用 Open64 编译器框架实现和复数有关的数据操作, 而不是把复数作为复合数据类型来处理. 但是目前 Open64 编译器框架中并不把复数操作作为基础的数据类型, 通过修改编译器的前端来禁止编译器把复数操作展开为实部和虚部. 通过修改编译器后端的机器描述, 以及在编译器 WHIRL 下降阶段通过识别出复数操作来处理针对复数运算的逻辑, 利用魂芯 DSP 单周期复数指令, 生成高效的汇编程序.

通过研究认为复数数据作为基础类型, 具有如表 1 所示的原子操作定义.

复数类型作为编译器中的基础类型, 表 1 中列出的操作在编译器中需要识别并实现为单条操作. 对于

编译器预期的结果则是汇编代码中除了复数其他操作对应于一条单周期的指令就可以完成对应的语义.

表 1 复数数据类型的 ADT

抽象符号表示	含义
complex	复数类型的定义
float complex_imag(compex rhs)	求复数的虚部
float complex_real(complex rhs)	求复数的实部
complex complex_neg(complex rhs)	求复数的负数
complex complex_conj(complex rhs)	求复数的共轭数
bool complex_equal(complex rhs)	判断复数是否相等
complex complex_add(complex rhs)	复数加法
complex complex_sub(complex rhs)	复数减法
complex complex_mul(complex rhs)	复数乘法
complex complex_div(complex rhs)	复数除法

2.1 复数类型的支持

在魂芯 DSP 处理器提供了丰富的单周期复数指令. 包括复数加减法、复数乘法、共轭复数运算[6]等. 其中 Macro 表示该指令的分簇情况, 由指令可以看出可以仅仅利用一条指令来解决一种复数运算操作. 然而通常编译器对于复数的运算都会转化到实数范围内对复数的实部和虚部分别进行多次的实数运算之后再将实部和虚部分别写回, 这样的处理方法使得程序并不能利用到 DSP 自身提供的各种复数运算指令.

{Macro}CFRs+1:s = CFRm+1:m + CFRn+1:n

{Macro}CFRs+1:s = CFRm+1:m - CFRn+1:n

{Macro}CFRs+1:s = CFRm+1:m * CFRn+1:n

{Macro}CFRs+1:s = conj CFRn+1:n

研究发现 Open64 开源编译器在编译优化的后期为了通用性考虑, 把复数操作展开为实部和虚部的单独运算操作[7]. 所以重点工作在编译器框架的中端和后端部分.

Open64 中经过前端词法和语法的分析会生成中间语法树 WHIRL. WHIRL 树中定义了一些列的操作算子 opcode[8], 其中每一个 opcode 就代表了程序的一个算子, 可以通过识别复数操作的算子在 WHIRL 下降到 CGIR 的过程中增加针对魂芯 DSP 目标体系结构的处理逻辑, 通过指令注释生成对应的汇编代码.

对于复数类型操作支持的解决方案流程如图 2 所示. 主要分为三步: (1)前端禁止编译器把复数展开为实部和虚部; (2)修改编译器中端 WHIRL 节点定义增加复数操作的 opcode 码; (3) 在编译器机器描述文件中增加复数操作的指令描述, 修改编译器后端 CGIR 框架, 增加对应 WHIRL 树种新增的复数的 opcode 的处理逻辑.

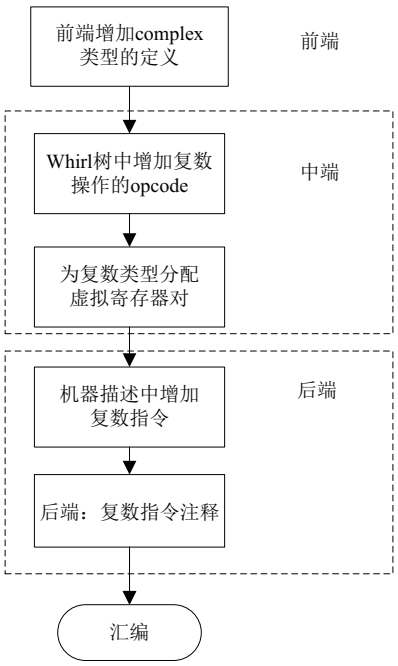

图 2 复数类型支持流程

2.2 复数类型优化

魂芯 DSP 芯片除了提供复数基本的加减乘单周期指令外还提供了许多和复数操作相关的高效指令. 指令形式如下:

{Macro}CFRs+1:s = CFRm+1:m + jCFRn+1:n

{Macro}CFRs+1:s = (CFRm+1:m + CFRn+1:n)/2

{Macro}CFRs+1:s = (CFRm+1:m + jCFRn+1:n)/2

{Macro}CFACCs+1:s += CFRn+1:n (conc, con=Rm)

第一个指令是一个复数同另一个复数乘上 j 后的浮点加法(等价于一个复数先乘上一个实数, 然后运算的结果会存储为一个中间结果, 然后中间结果再加上另外一个复数得到最后结果), 第二条和第三条指令分别是复数相加除 2 指令和第一条指令运算结果再除 2 指令. 最后一条指令是复数累加指令, 累加的结果存在累加寄存器 ACC 中. 因为编译器到后端一般是三地址代码, 这些指令在 Open64 的中间 WHIRL 树中对应的并不是一条单一的 opcode 操作码, 无法通过简单的代码翻译就可以利用到对应的高效指令, 要通过合并几条特有的操作算子来翻译到魂芯 DSP 上高效的指令.

WHILR 中节点是树的结构组织的, 那么可以从树的根节点, 也就是一个程序的 region 入口处开始遍历树的节点, 当发现某一个子节点中的孩子节点有符合我们要优化的模式时就去匹配定义好的模式处理逻辑,

把本来要好几条处理指令才能完成的逻辑用魂芯 DSP 上一条指令就可以完成, 提高程序的指令效率. 所以需要定义我们的遍历树节点的算法, 对应的模式串和匹配到模式串后处理的算法. 对 WHIRL 树节点的编译算法如图 3 所示.

```
algorithm Complex_Opt_Trace
input WHIRL tree, Rules rules;
outout:
var
   WHIRL node, opt_node;
   bool has_complex_op;
   bool Match_flag;
   Rule r;
begin
   node = tree;
   while node is not null do
       has_complex_op = Has_Complex_OP(node);
       if has_complex_op == true
       then
           for r in rules do
               Match_flag = Match_Rule(node, rule);
               if Match_flag == true
               then
                  opt_node = GEt_Match_Rule(node, rule);
                  Do_Opt_REduce(opt_node, r);
               end
           end for
       end if
    node = tree->next_child;
    end while
return;
end algorithm
```

图 3 复数高效指令优化流程

算法首先判断 WHIRL 树节点中是否含有和复数操作有关的节点, 如果包含复数操作的节点, 那么就按已经确定好的优化模式串来对树的节点进行遍历, 如果 Match_Rule 返回结果是真, 那么就说明此树种有节点符合我们优化的模式. 然后针对此节点进行优化处理, 处理的逻辑在 Do_Opt_Redule, 这个子函数是用来对 WHIRL 中它包含的符合优化处理的一系列指令到 CGIR 的下降处理剪切我们预先定义的模式处理流程, 然后在经过指令注释生成 DSP 上的高效的指令.

对于一个节点如何满足某个模式优化串, 也就是 Mach_Rule 的流程, 我们定义如下的模式串流程, 如果 WHIRL 树种节点符合这种规则, 则优化逻辑就对次节点进行优化, 以上面列出的高效指令的第一个指令为例, 其中模式串的规则定义如图 4 所示. 首先判断 tree 节点中是否为复数加法操作, 如果是然后在判断是否第二个操作数的孩子节点类型是否是复数和浮点数类型, 如果条件都满足, 那么就说明符合预定义的优化规则.

但是这种优化算法会带来编译器的编译速度的降低, 对于 DSP 嵌入式设备来说, 代码效率和代码大小是很重要的考量, 这种优化逻辑既可以减少程序的执行周期又可以减少 DSP 上代码的大小.

```
algorithm opt_Rule
input WHIRL tree
outout: true/false
var match_flag
begin
   if tree->opcode == OPC_C4ADD
   then
      if tree->kids[1]->opcode == OPC_C4MUl
      and type(tree->kids[1]->opnds[0]) == MTYPE_C4
      and type(tree->kids[1]->opnds[1]) == MTYPE_F4
         match_flag = true;
      else
         math_flag = false;
       end if
    end if
return match_flag;
end algorithm
```

图 4 复数加乘优化 Rule 规则

3 实现中的核心问题

3.1 WHITL 中增加复数操作码

Open64 在从 WHIRL 树转换到 CGIR 的过程会进行一系列的优化, VHO, IPA, LNO, WOPT, CG[9]. 在最后一个优化阶段 CG 中增加对魂芯 DSP 目标平台的宏定义, 禁止编译器框架把复数展开为复合操作, 令 WHIRL 转换到 CGIR 模块则可以识别出复数为基础数据类型.

通过在编译器后端增加复数类型运算的操作码 (opcode), 来支持复数操作, 在操作码文件 opcode.h 中增加复数运算操作码. 例如增加复数加操作, OPC_C4ADD 操作码表示是一条加法操作, 输入的源操作的类型是 32 位的复数类型. 通过增加转换条件来支持转换 WHIRL 转化为 CGIR 的中间表示, 转化后的 CGIR 中的每条指令与魂芯 DSP 目标处理器指令集中的汇编指令格式一一对应.

MTYPE_C4=17 /* for 32-bit complex */

OPC_C4ADD=OPR_ADD+RTYPE(MTYPE_C4)+DESC(MTYPE_V)

在 WHIRL 转换到 CGIR 过程中通过识别出 OPR_ADD 操作, 继而再判断操作数是复数类型, 转到复数操作数的处理流程.

3.2 复数寄存器对分配

由于复数类型包含实部和虚部, 所以对复数类型要分配两个连续编号的寄存器, 即寄存器对. 实部存放奇序号寄存器, 虚部存放偶序号寄存器. 对于连续编号

寄存器对[10]的保证可以通过 TN_Pair 在寄存器分配中申请, TN_Pair 表示两个连续编号的寄存器对, 且较小编号为偶数.

TN_Pair 的实现需要修改原有的框架中寄存器分配的算法. 魂芯 DSP 编译器使用启发式的图着色算法进行全局的寄存器分配. 基本的算法流程是先为函数中的寄存器建立寄存器干涉图[11], 然后统计虚拟寄存器的使用频率, 依照使用频率的大小依次进行着色. 针对复数操作对寄存器的特殊要求, 对寄存器分配算法修改. 将复数指令上的所有寄存器看成是一个整体分配相应的连续寄存器. 具体的实现是在寄存器分派阶段, 为复数指令操作数分配寄存器时, 将寄存器属性标记为 R_CONTAINS_COMPLEX, 并将这些寄存器添加到 vreg_complex_list 中, 表示寄存器对为复数操作所需的寄存器.

3.3 复数类指令注释

在利用目标机器提供了有关复数运算的指令时, 首先要对相关的目标或机器指令进行描述, 以便在后端的分簇、指令调度等相关阶段使用. Open64 编译器框架采用了二次编译的方式设计机器描述文件的架构. 在编译器后端的寄存器描述文件中增加复数运算的指令, 对指令进行描述和注释. 最终生成对应格式的汇编代码. 魂芯 DSP 机器描述主要通过描述目标机器的指令格式信息、资源使用信息、延迟信息和指令信息来将目标机器的指令集[12]信息提供给编译器. 机器描述的抽象表示主要由以下五个部分组成:

(1) SECTION opcode_desc 描述了机制指令操作码;

(2) SECTION Operation_Format 描述了源操作数和目的操作数的数目以及类型;

(3) SECTION Resource Usage 描述资源使用信息;

(4) SECTION Table_Option 描述资源选项;

(5) SECTION Scheduling_Alternative 描述指令调度信息.

例如在机器描述文件中 opcodes.knb 文件中增加复数加法指令, opcode 表示增加了复数加法的 c4add 机器描述, 需要占用浮点 ALU 算术单元, 这条指令的操作数描述码是 112. 操作数描述中, 负号表示输入操作数, 正号表示输出操作数, 复数加法的指令表示有 4 个 32 位的浮点操作数, 输出的是 2 个 32 位的浮点数.

opcode+="230, c4add, fuALU, intel1, 112, NULL";

opndsgrp+="112, ?pr/OU_predicate, -fp32/OU_opnd1, -fp32/OU_opnd2, -fp32/OU_opnd3, -fp32/OU_opnd4, +fp32, +fp32";

复数的加减和乘法都可以从过从 CGIR 在到指令注释的过程直接一一对应到魂芯 DSP 上的一条复数操作指令, 但是魂芯 DSP 上并没有针对复数除法的单周期指令, 针对复数除法必须利用一组指令的结合运算来得到最后的除法结果. 根据第 1 节对复数除法的定义可以知道, 要先算出除数的结果, 利用魂芯 DSP 上特有的指令, 我们知道这个平方和的结果就等于一个复数和他的共轭复数的乘积, 利用魂芯 DSP 上特有的指令可以把复数的除法指令由原来的 12 条指令减少到只需要 7 条就可以完成.

3.4 Intrinsic 技术识别实部和虚部

要在源代码级别识别复数的实部和虚部, 可以通过 intrinsic[13]技术, 利用编译器把函数转换为汇编指令. intrinsic 函数又称为 built_in 内建函数, 是一种通过使用 C 函数方式来封装 BWDSP 底层系统结构的特殊汇编指令的编译器功能模块. 编程人员可以通过使用 C 函数, 编译器能够在内部将该 C 函数直接转换为指令系统所支持的一条或若干条高效的特殊汇编指令. 通过将 creal 和 cimag 定义为 intrinsic 操作来在源码级获得复数类型的实部和虚部. 复数虚部的实现结果如图 5 所示, 把求复数虚部操作_image_f 定义为一个 F4INTRINSIC_OP 操作. 通过后端的指令注释可以直接得到复数的虚部, 其实就是取代表复数寄存器对的较大序号的寄存器中的值.

4 实验结果

在完成了复数类型的支持和优化后, 为了测试效果, 选取了 DSP 经典的测试集[14]来测试编译器性能, 测试是和以前编译器未针对复数类型进行优化的对比. 在 ECS(Effective Coding Studio)统计程序执行的周期数, 程序优化前后执行的时钟周期数如表 2 所示.

涉及大量复数运算的是 FFT 和复数求点积算法, 并行度比较高可以看出其加速比效果是原来的 5~6 倍, 而复数求点和和规约求和因为本身数据并行度不大, 优化效果是原来的 2~3 倍. 至此, 基于开源编译器框架 Open64 完成了针对魂芯 DSP 上新增复数类型为基础类型的开发.

优化复数支持后程序的加速比效果如图 6 所示, 从图中可以看出, 在完成编译器对复数类型的基本支持后, 对复数程序的编译优化取得了较好的结果. 相关算法的平均加速比达到了 5.28 倍.

```
#include <complex.h>

int main()
{
    float complex a = 1.0 + 2.0*I;
    float b = __imag_f(a);
    return 0;
}
```

WHIRL转换

```
BLOCK {line: 1/22}
 PRAGMA 0 119 <null-st> 0 (0x0)
  C4CONST <1,52,____1.000000,_____2.000000>
 C4STID 0 <2,1,a> T<17,.predef_C4,4>
   C4C4LDID 0 <2,1,a> T<17,.predef_C4,4>
   C4PARM 2 T<17,.predef_C4,4> #  by_value
  F4INTRINSIC_OP 1 <830,IMAG_F> 0
 F4STID 0 <2,2,b> T<10,.predef_F4,1>
  I4INTCONST 0 (0x0)
 I4RETURN_VAL
 END_BLOCK
```

图 5 复数虚部 intrinsic 转换

表 2 优化前后的时钟周期数(cycles)

测试基准	优化前	优化后	加速比
FFT_radix2	114390	22338	5.12
FFT_radix4	298130	50417	5.91
FIR_cplx	21292	3196	6.67
IIR	2890	483	5.98
ImsFIR	3364	890	3.78
Matrix_100	7989541	1736856	4.60
Complex_dotadd	13688	2442	5.60
Complex_dct	2460	5447	4.42
Complex_dotprod	24561	4759	5.16

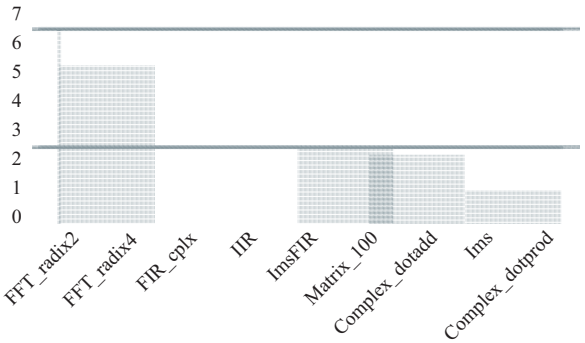

图 6 编译器优化加速比

5 结语

本文针对魂芯 DSP 高性能处理芯片, 利用其提供的复数运算指令, 提出了 32 位复数数据类型作为编译器基础数据类型的模型, 抽象出了复数作为编程基础数据类型具有的原子操作语义. 基于开源的 Open64 编译器基础上, 利用魂芯 DSP 特有的复数指令对复数操作进行了优化. 通过对魂芯 DSP 芯片现有的高效的复数指令, 提出了基于模式匹配的优化算法, 此算法也可以通过扩展对其他的匹配模式进行优化. 为了验证优化的效果, 测试了编译器优化前后程序执行的时钟周期数, 实验结果表明平均加速比为 5.28. 所提出的复数类型作为基础数据模型的一种通用增加数据模型的方法, 以及基于模式匹配的优化算法也可以抽象推广到其他需要优化指令的平台.

参考文献

1 CETC38TM. BWDSP100 硬件用户手册. 合肥: 中国电子科技集团公司第三十八研究所, 2011: 1–2.

2 李欣, 刘峰, 龙腾. 定点 FFT 在 TS201 上的高效实现. 北京理工大学学报, 2010, 30(1): 88–91.

3 王国栋, 侯朝焕. GCC 在高性能微处理器 DSP 和 CPU 上的移植. 计算机工程与设计, 2005, 26(4): 891–920.

4 Sui YL. Open64 introduction. http://www.cse.unsw.edu.au/~ysui/saber/open64.pdf. [2015-03-17].

5 Open64. Open64 compiler whirl intermediate representation. http://www.mcs.anl. gov/OpenAD/open64A.pdf. [2015-03-17].

6 CETC38TM. BWDSP100 软件用户手册. 合肥: 中国电子科技集团公司第三十八研究所, 2011: 181–191.

7 丁陈飞, 郑启龙, 徐华叶, 等. 多簇超长指令字 DSP 复数运算的编译优化. 计算机应用与软件, 2015, 32(2): 14–17.

8 Cheong G, Lam MS. An optimizer for multimedia instruction sets. Proc. of the 2nd SUIF Compiler Workshop. Stanford, USA. 1997.

9 黄胜兵, 郑启龙, 郭连伟. 分簇 VLIW DSP 上支持单双字模式选择的 SIMD 编译优化. 计算机应用, 2015, 35(8): 2371–2374. [doi: 10.11772/j.issn.1001-9081.2015.08.2371]

10 张军超. 相连多寄存器组体系结构上的寄存器分配技术[博士学位论文]. 北京: 中国科学院计算技术研究所, 2005: 92–94.

11 姜军, 王超, 尉红梅. 一种局部寄存器分配的优化策略. 计算机应用与软件, 2013, 30(12): 215–217, 254. [doi: 10.3969/j.issn.1000-386x.2013.12.056]

12 王向前, 洪一, 王昊, 等. 魂芯 DSP 的编译器设计与优化. 电子学报, 2015, 43(8): 1656–1661.

13 Batten D, Jinturkar S, Glossner J, *et al*. Interaction between optimizations and a new type of dsp intrinsic function. Proc. of International Conference on Signal Processing Applications and Technology (ICSPAT'99). Orlando, Florida, USA. 1999.

14 Živojnović V, Velarde JM, Schläger C, *et al*. DSPstone: A DSP-oriented benchmarking methodology. Proc. of International Conference on Signal Processing Applications and Technology (ICSPAT 1994). Dallas, USA. 1994.