Matrix 编译器 If 转换算法的实现

刘 飞 陈跃跃 孙海燕 阳 柳
(国防科技大学计算机学院 长沙 410073)

摘 要 指令级并行在提高处理器运行速度方面显得越来越重要,if 转换技术是一种在处理器支持条件执行的前提下,有助于提高指令级并行度的编译优化技术。在详细分析 GCC(GNU Compiler Collection)内部 if 转换技术的实现机制和算法的基础上,针对 matrix 体系结构特点,对 GCC 中现有 if 转换算法进行了移植与改进,实现了 matrix 编译器的 if 转换算法。实验证明,改进后的 if 转换算法能够更有效地移除分支,减少基本块的数量,扩大单个基本块的范围,有助于编译器生成更加优化的代码。

关键词 谓词执行,if 转换,条件执行,指令级并行,Matrix,VLIW
中图法分类号 TP314 **文献标识码** A

Implement of Matrix Compiler's If-convertion Algorithm

LIU Fei CHEN Yue-yue SUN Hai-yan YANG Liu
(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

Abstract The ILP(instruction-level-parallel) becomes more and more important for improving the microprocessor's speed. The if-convertion technology is an effective compiler-optimization method of facilitating exploiting ILP based on the microprocessor which supplies conditional execution. After introducing if-convertion technology and algorithm of GCC and transplanting it into the Matrix compiler successfully, some improvements for if-convertion of Matrix compiler were made. Experiment results indicate that the if-convertion achieved in the paper can remove the branches, reduce the number of basic-blocks and broaden the bound of basic-block. It can also facilitate making more predominant codes for compiler.

Keywords Predicate execution, If-convertion, Conditional execution, ILP, Matrix, VLIW

1 引言

现代处理器中的运算单元大多都是流水线结构,分支预测的错误会造成流水线空转,降低处理器的性能。带有指令级并行[1]特征的处理器一般会有多个运算单元,在一个时钟周期内可以发射多条指令,分支预测的错误会使处理器的性能损失巨大。为了减少分支预测错误带来的性能损失,谓词执行技术[3]被应用到这类处理器的设计当中。

支持谓词执行的处理器可分为支持全谓词和部分谓词两大类。全谓词是指所有的指令都可支持条件执行,部分谓词是指只有一部分指令支持条件执行。Matrix 是我国自主研制的支持全谓词的数字信号处理器,是一款典型的 VLIW[3,6] 体系结构芯片。Matrix 硬件资源丰富,可同时执行多条指令。对指令进行有效的调度,提高指令级并行度,是充分发挥 Matrix 硬件资源关键所在。基于 Matrix 处理器对条件执行[2]的支持,编译器可以有效地移除程序中分支。if 转换技术[5,8]是大多数编译器实现谓词执行技术的一种有效方法。GCC(GNU Compiler Collection)编译器开发谓词执行技术就是通过 if 转换实现的。if 转换能有效地把控制流图转换为数据流图,扩大基本块的大小,增加基本块内调度的范围,能有效地支持指令调度,进而提高指令级并行度。

本文第 2 节介绍了现在比较成熟的 if 转换的定义和算法;第 3 节介绍了 GCC 内部 if 转换算法具体实现过程;第 4 节介绍了 Matrix 编译器 if 转换的实现与改进;第 5 节介绍了 Matrix 编译器 if 转换算法的实现后的实验结果;最后对全文进行了总结。

2 if 转换算法简介

if 转换是把控制流转换为数据流的一种有效的方法,另一种比较有效的方法是循环展开。if 转换消除程序中的条件分支指令,把其转换成带有谓词的线型代码。if 转换不仅可以消除分支,而且也增加了基本块的范围,进而有利于提高代码的并行度,给编译器后续的优化提供了更多的机会。if 转换最初被广泛地用在向量化编译器中。Ferrante 提出了一种有效的 if 转换算法,其用的是 PDG(program dependence graph)[7]。if 转换通过等价变换源程序,将分支语句转换为条件执行语句,执行时通过计算一个与指令相关的逻辑表达式来决定指令是否执行,从而可删除某些控制流,以便于提高指令

到稿日期:2012-06-01 返修日期:2012-08-28

刘 飞(1986—),男,硕士生,主要研究方向为微处理器设计,E-mail:liufei8653@163.com;陈跃跃(1960—),男,博士,研究员,主要研究方向为微处理器设计;孙海燕(1976—),女,博士,副研究员,主要研究方向为分布计算与 DSP 调试环境;阳 柳(1979—),女,硕士,助理研究员,主要研究方向为嵌入式操作系统。

级并行度。现在 if 转换面临着两个方面的难题:①如何给基本块分配特定的谓词;②如何正确使用定义谓词的操作[9]。

if 转换为每条指令增加一个操作数(即谓词)作为指令执行的条件,当谓词条件为假时将其转换为空操作处理,为真时执行其操作。图 1(b)所示为 if 转换针对测试用例 count()(如图 1(a)所示)的控制流图,图 2(a)和图 2(b)分别为 count 测试程序在 if 转换前和转换后对应汇编码的控制流图。

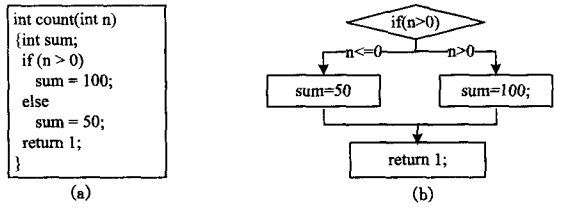

图 1 count 测试程序图和 if 转换前的控制流

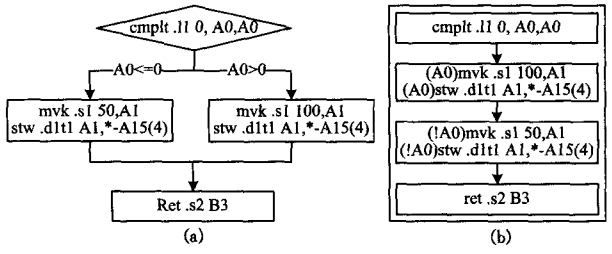

图 2 count()程序 if 转换前和转换后的汇编码控制流图

经过 if 转换后,count()程序的基本块数量由原来的 4 个变成了 1 个,被选定可以 if 转换的基本块组合称为 Hyper-Block[5]。HyperBlock 只有一个入口,亦即它的第一条指令,但可以有多个出口,同一 HyperBlock 内部各基本块之间的所有控制相关都将被消除,编译调度更加充分。通过引入谓词使得编译器有了更大的调度空间,在扩大基本块体积的同时,也扩大了数据和控制前瞻的范围,从而极大地减少了基本块的数量,并且扩大了单个基本块内的指令条数。

3 GCC 内部 if 转换的实现

GCC 编译器的编译过程大致可以分为 3 个部分——前端、中端和后端。前端与体系结构无关,与被解析的语言有关,主要包括词法、语法、语义分析,通过分析将源语言的代码转换成中间表示—tree 结构。中端是将与体系结构无关的或者很少相关的 tree 结构转换成与体系结构关系密切的 rtl 结构的过程。后端是对完全 rtl 结构的优化并且输出相应的汇编代码的过程。3 个部分的划分并没有明显的界限,因本文需要暂且把 tree 结构转换成 rtl 结构之前的部分称作前端,把开始转换到寄存器分配结束的部分称作中端,寄存器分配之后的部分称作后端。

GCC 的内部 if 转换的实现分散在 3 个部分中。其中在前端的调用 if 转换主要针对的是向量化指令的映射,在此不予以介绍。本文主要对标量指令的 if 转换的 GCC 的实现过程进行介绍。GCC 针对标量的 if 转换在中端和前端都有实现,但是针对 HyperBlock 的选择范围和实现方法不同。

3.1 GCC 内部中端 if 转换的实现

GCC 在寄存器分配之前的 if 转换,主要是在循环优化之前和指令合并之后。在中端进行 if 转换对符合 if 转换的基本块组合的限制比较多,符合特定控制流要求的基本块组合数量比较少,只是针对几种特定控制流下的基本块组合进行 if 转换。其主要转换的基本块控制流图如图 3 所示,其中虚线表示这个边可以有。

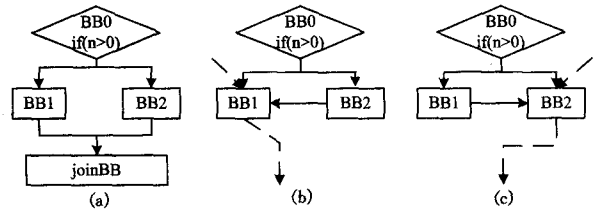

图 3 在中端符合 if 转换的基本块的控制流图

通过图 3 可以看到,GCC 中端的 if 转换最多只是针对 4 个基本块组合,而且对每个基本块的条件限制也比较多,在实验中也发现在中端进行 if 转换的效果不是很明显。

3.2 GCC 内部后端 if 转换的实现

在后端的 if 转换主要是在寄存器分配之后。在寄存器分配之后的 rtl 结构已经是完全的 rtl 结构,也就是说它的表示形式已经可以看作输出汇编代码的另一种表示。寄存器分配完成之后,根据控制流和数据流分析之后的结果,对符合 if 转换控制流要求的基本块组合进行 if 转换。主要对图 4 所示的基本块组合进行 if 转换。从图 4 可以看出,基本块组合的范围明显扩大了,其对基本块的个数没有限制,但是对内部代码分析后可以发现,它对进行 if 转换的所有的基本块中的指令条数的总数是有一定限制的,这个限定值可以根据自己体系结构特点进行调整。

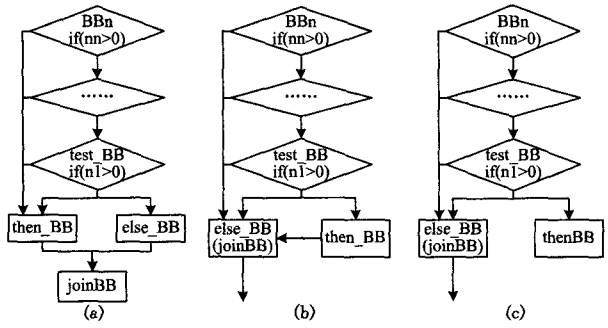

图 4 GCC 后端符合 if 转换的基本块组合控制流图

GCC 除了对图 4 所示的分支指令进行 if 转换之外,还对某些特定的交叉跳转进行 if 转换。符合 if 转换的交叉跳转的控制流图及 if 转换后的控制流图如图 5、图 6 所示。

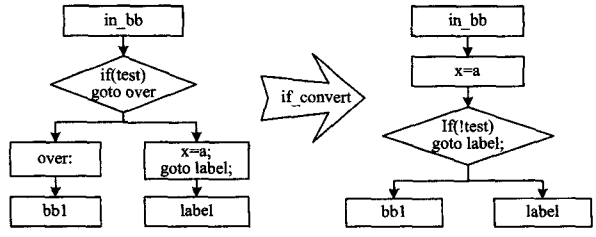

图 5 GCC 后端符合 if 转换的交叉跳转的控制流图

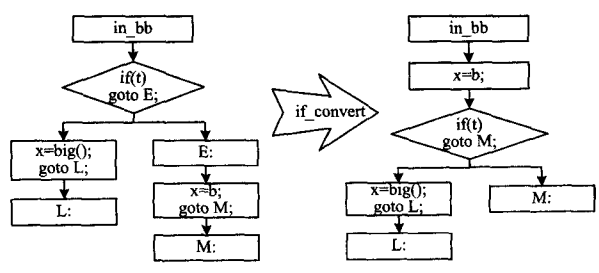

图 6 GCC 后端符合 if 转换的交叉跳转的控制流图

GCC 对交叉跳转的 if 转换只实现了这两种情况，但是交叉跳转的情况还有很多，可以根据应用程序的特点添加能够 if 转换的交叉跳转的情况，详细的添加过程将在后面介绍。

4 Matrix 编译器 if 转换算法的实现

本文基于 GCC 中的 if 转换算法，根据 matrix 的体系结构和应用特点，实现了 Matrix 编译器的 if 转换算法。

GCC 在优化级别为“-O1”、“-O2”、“-O3”和“-Os”时开启 if 转换功能。为了支持编译器开启 if 转换功能，还需要在后端模板“*.md”文件中添加相应的描述。条件执行时用到的谓词寄存器不能和用这个谓词寄存器进行条件执行的指令中的其它寄存器冲突，如果有冲突会使 if 转换失效。为了改善 if 转换的效果，需要尽可能避免这一类的寄存器的冲突。对于一些专门用途的处理器，由于其运行的程序的单一性（如 DSP 主要是用于数字信号处理），可以考虑根据应用程序的特点添加几种适合于 if 转换的 HyperBlock 组合，以提高程序的运行效率。

根据以上分析，Matrix 编译器 if 转换算法的实现主要包括以下步骤：

①对后端模板的“*.md”文件添加有关 if 转换的描述。

②由于 if 转换后需要大量的谓词寄存器，修改寄存器分配算法，以改善 if 转换的效果。

③对 GCC 现有的 if 转换算法进行改进，添加几种符合 if 转换的 HyperBlock 组合。

4.1 对后端模板的“*.md”文件的修改

由于 Matrix 体系结构支持全谓词，因此在后端模板的“*.md”文件中添加相应的谓词指令描述“define_cond_exec”，它会在编译 GCC 时对支持谓词的指令自动进行匹配，生成带有谓词的指令模板的形式。添加的“define_cond_exec”的描述如图 7 所示。

```
(define_cond_exec
[(ne (match_operand:SI 0 "cc_reg_operand" "y")
                (const_int 0))]
""
"[%0]")

(define_cond_exec
[(eq (match_operand:SI 0 "cc_reg_operand" "y")
                (const_int 0))]
""
"[!%0]")
```

图 7 “define_cond_exec”的指令描述

在添加如图 7 所示的指令模板的同时，还要对“*.md”做一些相应的修改和添加，这里就不予以介绍。

4.2 对寄存器分配方法的修改

Matrix 处理器是一款典型的 DSP 处理器，主要是用于信号处理，可同时支持向量和标量的运算。Matrix 处理器共有 82 个相对于用户可用的寄存器，其中标量运算单元含有 28 个通用寄存器（32 位），4 个累加器（40 位）以及一些控制寄存器。条件寄存器为通用寄存器中的 3 个。由此可见，条件寄存器除了数量少外，还并非专用于条件执行，所以为了使 Matrix 编译器的 if 转换效果更好，需要改变一下寄存器分配算法，使条件寄存器专用于条件执行，这样就减少了条件寄存器的资源冲突对 if 转换的影响。

后端的 if 转换虽然相应的限制较少，但是在寄存器分配完成之后，存在谓词寄存器和其它寄存器冲突的情况比较多，特别对像 C6X（TI 公司 DSP 处理器的 C6000 系列的体系结构）这种没有特定的谓词寄存器的体系结构来说，这种现象更加严重。正是由于这种寄存器冲突的存在，使得在后端的 if 转换效果也不是很明显。IA-64（Intel 的安腾-64 处理器体系结构）[2] 是支持全谓词的体系结构，并且有单独的谓词寄存器来支持 if 转换。从比较 GCC 对这两种体系结构的 if 转换的效果来看，C6X 由于没有单独的谓词寄存器，因此其 if 转换效果比 IA-64 的差很多。图 8 所示为某个测试程序的标有基本块信息的控制流图。对这个测试程序分别基于 C6X 和 IA-64 体系结构进行 if 转换的结果如图 9(a)和图 9(b)所示。图 8、图 9 给出的基本块组合，没有考虑 entry 和 exit 两个基本块，其标号分别为 BB1 和 BB0。

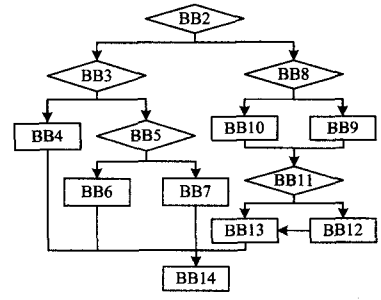

图 8 没有经过 if 转换的控制流图

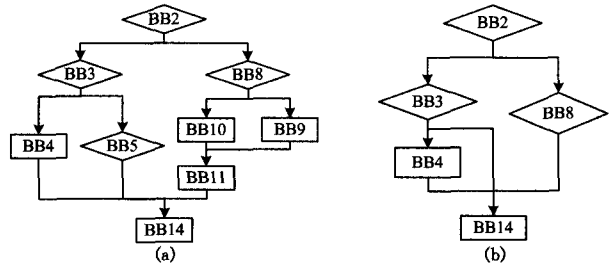

图 9 分别为 C6X 和 IA-64 体系结构经过 if 转换的结果

由以上图的分析可知，原来有 13 个基本块的控制流图，经过 if 转换后，C6X 体系结构的转换后结果为 9 个，IA-64 的只剩下 5 个。存在这方面差距的主要原因就是是否有专门用于谓词的寄存器。

所以要想使 Matrix 编译器的 if 转换效果更加明显，分配特定的寄存器专门用于谓词寄存器是必要的。为了做到这点，本文修改了寄存器分配的算法，实现了专用的谓词寄存器的分配，使所选定的寄存器只能分配给谓词。修改之后，针对图 8 的测试程序，Matrix 编译器在修改之后，图 8 所示的测试程序经过 if 转换后，其控制流图如图 9(b)所示。

4.3 if 转换算法的改进

Matrix 编译器的 if 转换是基于 GCC 实现的，在此基础上，需要针对 Matrix 的特点和应用，对 Matrix 编译器中的 if 转换算法进行改进，可以添加几种符合 if 转换的 HyperBlock 组合。本文介绍的主要是对包含交叉短跳转的 HyperBlock 组合的 if 转换的实现。

GCC 的 if 转换算法，在中前端的转换比较少，对符合 if 转换的控制流图的基本块的组合限制较多，所以 GCC 在中前端 if 转换的效果不是很明显。而在后端 if 转换由于是在寄存器分配完成之后，数据依赖关系、寄存器冲突等关系都已经确定，在此基础上进行 if 转换可以有充分的信息，因此可以在后端添加几种适合于 if 转换的 HyperBlock 组合。

经以上分析可以知道，针对 GCC 符合 if 转换的控制流图的限制比较多的情况，考虑添加几种新的符合 if 转换的 HyperBlock 组合。GCC 对 if 转换只限于分支和几种有限的交叉跳转的情况。但是在某些应用程序中，某些交叉跳转的情况比较多。所以增加对某些交叉跳转的 if 转换功能是很有必要的。

但是交叉跳转的情况很多，也很复杂，想要把所有的交叉跳转都考虑到是不太现实的，只能折中地添加几种针对特定应用程序相对比较多的情况。大致可以把交叉跳转分为长跳转和短跳转。长跳转跨越基本块的数量比较多，如果对长跳转进行 if 转换会带来寄存器压力过大等诸多问题，所以只考虑对短跳转进行 if 转换。本文只介绍了一种符合 if 转换的 HyperBlock 组合添加的情况，对如图 10 左图所示的情况进行转换，转换结果如图 10 右图。

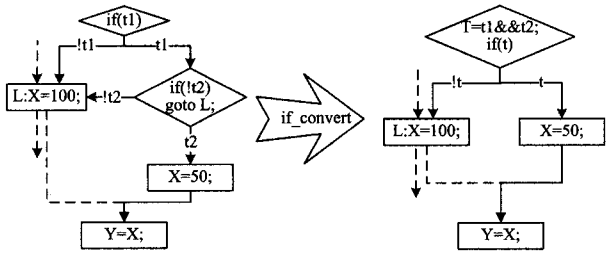

图 10　增加的交叉跳转的 if 转换控制流图

在对被编译程序进行数据流和控制流分析的基础上，介绍了符合图 10 左图的控制流的结构 if 转换，转换过程大致分为 3 个阶段：①选择基本块加入到 HyperBlock 中；②对 HyperBlock 中的基本块合并操作并将基本块中的相应指令转换成条件执行的指令；③更新基本块和指令的信息，更新控制流图和数据流图。图 11 是转换函数的伪代码。在这里只介绍了一种 HyperBlock 组合添加情况，其它情况类似。

```
int find_if_case_5(test_bb,then_edge,else_edge)
{
  /*判断所选基本块是否符合特定的要求*/
  if (!find_if_fit_for_case(test_bb,then_edge,else_edge))
            return false;
  /*检测then_bb里的求条件的语句是否与test_bb里求条件的语句
  存在寄存器冲突，如果存在寄存器冲突则转换失败，否则将
  then_bb里的求条件的语句放到test_bb的跳转语句之前*/
  if (!if_condition_suit_for_conversion (test_bb,then_bb))
            return false;
  /*把then_bb里的条件与test_bb里的条件相与后的结果作为
  test_bb新的分支条件,如果存在寄存器冲突则转换失败*/
  if (!put_the_condition_ahead_of_the_test(test_bb,then_bb))
            return false;
  /*删除then_bb到else_bb的分支边*/
  remove_edge(find_edge (then_bb,else_bb));
  /*重新定向test_bb到then_bb的分支边到then_bb的后继块*/
  redirect_edge_succ (find_edge (test_bb,then_bb),join_bb);
  /*删除基本块then_bb*/
  delete_basic_bb (then_bb);
  /*更新基本块和基本块中的指令的信息，例如分支指令的分支
  方向概率等信息*/
  update_info_of_bb ()
  return ture;
}
```

图 11　转换函数的大致过程

5　实验结果与比较

5.1　实验结果

GCC 内部的 if 转换是迭代进行的，亦即各个实现 if 转换的函数会循环运行多次，直到没有符合 if 转换的 HyperBlock 基本块组合为止。经过以上的 if 转换，图 10 左侧的控制流图被转换成图 10 右侧的控制流图后，会被其它转换函数选择并进一步 if 转换。经过对如图 12(a)所示的测试用例 main. c 进行测试可得到对 if 转换函数 find_if_case_5()的测试结果。为了阅读方便，所有测试结果的汇编代码中删除了一些代码。

图 12(b)所示为在添加转换函数 find_if_case_5()之前，测试用例 main. c 经过 if 转换后的汇编代码。

```
int main()                Main:
{volatile int a,b;                  ldw     .d2t1   *+B15(12), A3
                                    cmpgt   .l1     0, A3, A0
                          [!A0]     b       .s2     .L2
if (a < 0)                .L3:
L:b = 10;                           mvk     .d1     10, A6
                                    stw     .d2t1   A6, *+B15(8)
                                    b       .s2     .L4
else                      .L2:
{if(a ==1)                          ldw     .d2t1   *+B15(12), A4
                                    cmpeq   .l1     1, A4, A1
goto L;                   [A1]      b       .s2     .L3
                                    mvk     .s1     100, A5
b = 100;}                           stw     .d2t1   A5, *+B15(8)
                          .L4:
return 1;}                          ret     .s2     B3
(a)                       (b)
```

图 12　main. c 测试用例和添加 find_if_case_5()函数之前的汇编代码

图 13 所示为在添加转换函数 find_if_case_5()之后，测试用例 main. c 经过此函数转换后的汇编代码。由图 13 可以看到交叉跳转已经被删除，这还不是最终的结果，在经过迭代的 if 转换之后其最终的结果如图 14 所示，其中“||”表示本条指令与上一条指令并行。

```
main:
                    ldw     .d2t1   *+B15(12), A3
                    ldw     .d2t1   *+B15(12), A4
                    cmpgt   .l1     0, A3, A0
                    cmpeq   .l1     1, A4, A1
                    or      .b1     A0,A1, A0
          [!A0]     b       .s2     .L2
                    mvk     .d1     10, A6
                    stw     .d2t1   A6, *+B15(8)
                    b       .s2     .L4
.L2:
                    mvk     .s1     100, A5
                    stw     .d2t1   A5, *+B15(8)
.L4:
                    ret     .s2     B3
```

图 13　main. c 测试用例在经过转换函数 find_if_case_5()的汇编代码

```
main:
                    ldw     .d2t1   *+B15(12), A3
                    ldw     .d2t1   *+B15(12), A4
                    cmpgt   .l1     0, A3, A0
                    cmpeq   .l1     1, A4, A1
                    or      .b1     A0,A1, A0
          [A0]      mvk     .d1     10, A6
          [!A0]     mvk     .s1     100, A5
          || [A0]   stw     .d2t1   A6, *+B15(8)
          [!A0]     stw     .d2t1   A5, *+B15(8)
                    ret     .s2     B3
```

图 14　经过迭代 if 转换后的最终结果

5.2　结果分析

可以看出经过添加转换函数 find_if_case_5()之后，测试用例的所有基本块组合都符合了 GCC 内部的 if 转换条件，由原来的 5 个基本块合并为 1 个基本块，已经移除了所有的分支，并且从表 1 和表 2 的实验结果数据分析可得，if 转换算法改进后，main. c 的运行时间缩短了约 65%，执行包的个数也减少到原来的约 45%，比较充分地挖掘了代码的并行度。但是值得注意的是关键路径大大增加，由原来的 8 条指令增加到了 9 条指令。所以对于那种分支偏重于某一个方向的应用程序而言，if 转换的作用可能会被关键路径的加长所抵消，

（下转第 77 页）

参 考 文 献

[1] 赵忠华,皇甫伟,孙利民,等. 无线传感器网络管理技术[J]. 计算机科学,2011,38(1):8-14

[2] Xiang Qiao, Zhang Hong-wei, Xu Jin-hong, et al. When in-networks processing meets time: Complexity and effects of joint optimization in wireless sensor networks[J]. IEEE transactions on mobile computing, 2011, 10(10): 1488-1502

[3] Shenker S, Ratnasamy S, Karp B, et al. Data-centric storage in sensornets[J]. SIGCOMM Computer Communication Review, 2003, 33(1): 137-142

[4] Ratnasamy S, Karp B, Shenker S, et al. Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table[J]. Mobile Networks and Applications, 2003, 8(4): 427-442

[5] Karp B, Kung K. GPSR: greedy perimeter stateless routing for wireless networks[C]// Proc. of the 6th annual international conference on mobile computing and networking. ACM: New York, USA, 2000: 243-254

[6] Michele A, Stefano C, Francesco N, et al. Dealing with non uniformity in data centric storage for wireless sensor networks[J]. IEEE Transactions on Parallel and Distributed System, 2011, 22(8): 1398-1406

[7] Liao Wen-hwa, Wu Wan-chi. Effective hotspot storage management schemes in wireless sensor networks[J]. Computer Communications, 2008, 31: 2131-2141

[8] Yu Zhao-chun, Xiao Bin, Zhou Shui-geng. Achieving optimal data storage position in wireless sensor networks[J]. Computer Communications, 2010, 33: 92-102

[9] Joung Y-J, Huang S-H, Lin Shi-hang. Making data-centric storage adaptive and cost-optimal[J]. Computer Networks, 2012, 56(1): 213-230

[10] Das S K, Wang Jing, Ghpsh R K, et al. Theoretical aspects of distributed computing in sensor networks[M]. Springer, 2011: 257-291

[11] Weisstein E W. Square line picking[OL]. http://mathworld.wolfram.com/SquareLinePicking.html

[12] Bratislav M, Miroslaw M. Dropped edges and faces' size in Gabriel and relative neighborhood graphs[C]// Proc. of IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS). 2006: 407-416

(上接第 58 页)

但对于开发指令集并行度而言,基本块数量的减少、单个基本块范围的扩大,有助于指令调度开发指令级并行。

表 1 if 转换算法改进前 main.c 的运行情况

分支方向(a=?)	-5	1	5
执行周期(拍)	6582	7877	7865
执行包(个)	46	46	41

表 2 if 转换算法改进后 main.c 的运行情况

分支方向(a=?)	-5	1	5
执行周期(拍)	1978	1978	1978
执行包(个)	21	21	21

结束语 编译器对计算机性能的开发起着至关重要的作用,特别是有指令集并行特征的处理器对编译器的要求更高。为了提高编译器性能,充分利用硬件资源,编译器中嵌入了很多优化算法,而指令调度又是关键所在。谓词执行技术的应用能合并基本块,消除控制流中的分支,从而为指令调度级的优化提供更加广阔的调度空间,能充分挖掘指令级并行度。研究表明,有效地消除分支可以使指令级并行度提高到原来的 3 倍,而谓词执行技术能够平均消除 56% 的分支错误和 27% 的分支[10,11]。本文以 GCC 内部的 if 转换算法为基础,移植并改进了 Matrix 编译器中的 if 转换功能和算法,成功地实现了 Matrix 编译器的 if 转换算法。实验结果表明,本文设计实现的 if 转换有效地消除了测试程序中的分支,增加了调度的基本块的范围,简化了控制流图,有助于编译器生成更加优化的代码。

但是在选择符合 if 转换的 HyperBlock 组合时,并没有考虑基本块的执行频率、分支可能性、基本块大小等信息,这样一些执行频率低、分支可能性小的基本块在 if 转换后反而会使代码的执行效率变低。所以本文的下一步工作是,根据 GCC 内部自带的 profile 信息[4]、执行频率和基本块大小等信息选择符合 if 转换的 HyperBlock 组合。

参 考 文 献

[1] 张晨曦,王志英,张春元,等. 计算机体系结构(第二版)[M]. 北京:高等教育出版社,2005: 79-80

[2] 田祖伟,赵克佳,汪小飞. GCC 基于 IA-64 谓词执行的 if 转换技术研究[J]. 微电子学与计算机,2005,22(6): 188-196

[3] Strätling A. Optimizing the GCC Suite for a VLIW Architecture [OL]. http://www.qucosa.de, 2013

[4] Stallman R M. GNU Compiler Collection (GCC) Internals[OL]. http://gcc.gnu.org, 2013

[5] Kumar R, Saxena A K, Singh P K. A Novel Heuristic for Selection of HyperBlock in If-Conversion[J]. Electronics Computer Technology, 2001, 6: 232-235

[6] Jacome M F, de Veciana G, Pillai S. Clustered VLIW Architectures with Predicated Switching[A]// DAC '01 Proceedings of the 38th annual Design Automation Conference, 2001[C]. New York: Association for Computing Machinery, 2001: 696-701

[7] Park J C H, Schlansker M. On Predicated Execution. Software and Systems Laboratory[OL]. https://www.hpl.hp.com, 2013

[8] Zimmerman E, Zilles C. On the Energy Effectiveness of If-conversion in Superscalar Microprocessors[OL]. http://www.cs.illinois.edu, 2013

[9] Chuang Wei-haw, Calder B, Ferrante J. Phi-Predication for Light-Weight if-Conversion[A]// Proceedings of the International Symposium on Code Generation and Optimization: Code Generation and Optimization, CGO 2003. International Symposium on, 2003[C]. California, 2003: 179-190

[10] Monica S L, Robert P W. Limits of control flow on parallelism [A]// ISCA '92 Proceedings of the 19th annual international symposium on Computer architecture, 1992[C]. New York: Association for Computing Machinery, 1992: 46-57

[11] Mahlke S A, Lin D C, Chen W Y, et al. Effective compiler support for predicated execution using the HyperBlock[A]// MICRO 25 Proceedings of the 25th Annual International Symposium on Microarchitecture, 1992[C]. New York: Association for Computing Machinery, 1992: 45-54