计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(1):201−205 [doi: 10.15888/j.cnki.csa.006146]

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

BWDSP104X 多条件谓词编译优化①

韩东科, 郑启龙, 张仁高

[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)
通讯作者: 韩东科, E-mail: dongke@mail.ustc.edu.cn

摘　要: 目前 BWDSP104X 编译器对程序中条件分支的处理是采用传统的谓词优化方法, 及每条指令和一个谓词相关, 只有当谓词为真时指令才被执行, 但它存在的局限性是当涉及到多条件谓词时, 并不能消除跳转分支, 且多条件谓词之间可能存在控制依赖关系, 不利于指令并行和指令流水. 因此在现有编译器框架下, 针对传统谓词优化方法的不足之处, 本文提出一种基于 BWDSP104X 体系结构下多条件谓词编译优化方法. 实验结果表明, 与传统谓词优化方法相比, 该优化算法在 BWDSP104X 编译器上能够取得平均 5.62 的加速比.

关键词: 条件分支; 谓词优化; 多条件谓词; 编译优化

引用格式: 韩东科,郑启龙,张仁高.BWDSP104X 多条件谓词编译优化.计算机系统应用,2018,27(1):201–205. http://www.c-s-a.org.cn/1003-3254/6146.html

Compilation Optimization of Multi-Condition Predicate on BWDSP104X

HAN Dong-Ke, ZHENG Qi-Long, ZHANG Ren-Gao

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

Abstract: At present, the BWDSP104X compiler deals with the conditional branching in the program by using the traditional predicate optimization method, and each instruction is associated with a predicate. Only when the predicate is true, can the instruction be executed. But its existence is limited when the conditional predicates do not eliminate jump branches, and there may be control dependencies between multiple conditional predicates, which is detrimental to instruction parallelism and instruction flow. Therefore, in the framework of the existing compiler, this paper proposes a multi-condition predicate compiler optimization method based on BWDSP104X architecture in view of the shortcomings of traditional predicate optimization method. The experimental results show that the optimization algorithm can achieve an average speed of 5.62 on the BWDSP104X compiler compared with the traditional predicate optimization method.

Key words: conditional branching; predicate optimization; multi-condition predicates; compilation optimization

程序中的条件分支是影响程序性能的一个重要因素. 其一, 降低调度性能[1]. 条件分支会分割基本块规模, 将一个基本块变成多个基本块, 从而使可调度区域减小, 不利于指令调度. 其二, 减弱甚至中断指令流水. 分支后的指令是否执行需要根据当前分支的执行结果进行判断, 因此当程序中含有大量的条件分支时, 会中断指令流水, 变指令为串行执行.

为了减弱条件分支对程序性能的影响, 传统的方式是采用分支预测, 即在指令调度之前预测哪条分支指令会被执行, 然而当条件分支预测失败时, 将会导致流水线中断, 减弱指令流水的性能. 分支预测并没有对条件分支和多个条件分支之间的控制依赖关系进行处

① 基金项目: “核高基”重大专项 (2012ZX01034-001-001)
收稿时间: 2017-03-29; 修改时间: 2017-04-20; 采用时间: 2017-05-02; csa 在线出版时间: 2017-12-22

理, 谓词执行从根本上消除程序中的跳转分支, 完成从控制相关到数据相关的转化, 改变基本块内部的控制依赖关系, 从而有利于指令流水和指令级并行. 但利用谓词执行技术在处理多条件分支时只是局部的消除程序中的跳转指令, 从全局范围来看多条件分支之间依然存在跳转指令和控制依赖关系.

本文针对传统谓词优化在处理多条件谓词时的局限性, 提出一种基于 BWDSP104X 体系结构下多条件谓词编译优化方法, 以此消除多条件谓词的跳转分支及多谓词之间的控制依赖关系, 从而实现指令流水, 达到指令级并行效果, 提高程序运行效率.

1 概述

有关谓词执行的研究最早是由 Allen 等人提出的 If-conversion 的概念[2], 这一概念在最初的向量机上得到了广泛的应用. 通过逻辑表达式的真假来限定程序中的条件分支是否执行, 这样便删除了程序中相应的分支跳转指令, 有利于代码的向量化. 紧接着, HP 实验室的 Park 和 Schlansker 等人改进了 if-conversion 算法并提出了一个经典的 RK 算法[3], 该算法首先对每个基本块之间的控制依赖关系进行分析并建立谓词控制流图, 然后由谓词控制流图来计算函数 R 和 K, 以此来完成条件分支的谓词化.

Mahlke 等率先提出了 Hyperblock 编译优化技术来支持谓词执行[4], Hyperblock 是通过采用 IF 转换多个条件分支而形成的一个单入口多出口的谓词化基本块的集合, 但却并不是对程序中所有的条件分支都进行条件转换, 而是仅仅对那些有利于程序性能提升的条件分支进行转化.

在国内对谓词编译优化也作了大量的研究, 中国科学院计算技术研究所芦运照等[5]对 IA-64 体系结构下的谓词优化进行了深入的研究, 提出了谓词敏感的数据流分析技术, 使得谓词间关系得到了更精确的描述. 国防科学技术大学田祖伟等[6]对谓词执行的 IF 转换技术进行了详细的研究, 包括 IF 转换分支的选择、IF 转换时机的选择对程序性能的影响.

1.1 BWDSP104X 谓词指令

BWDSP104X 体系下谓词指令主要包含谓词定义指令和谓词执行指令生成. 谓词定义指令生成是把中间代码的比较指令转化为谓词指令, 其指令形式为{*x*, *y*, *z*, *t*}CPred[*k*]=Rm>Rn, 其中, *x*、*y*、*z*、*t* 为 BWDSP104X 多簇运算宏编号, Rm, Rn 为通用寄存器. 指令含义为根据 Rm、Rn 寄存器中的比较结果设置 CPred 寄存器中的相应位, 即若判断结果为真, CPred 谓词寄存器第 *k* 位为 1, 否则为 0[7]. 谓词执行指令的生成是消除中间代码中的分支跳转指令, 将条件分支分割的多个基本块合并为一个基本块, 其指令形式为 if CPred[K1]==C1 do *n*, 即通过判断 CPred 对应 K1 为 1 位置的数据是否等于常数 C1 对应 K1 为 1 的位置的数据, 若判断结果为真, 则执行当前指令行中该指令后的前 *n* 条指令[8].

1.2 BWDSP104X 编译器谓词优化流程

BWDSP104X 编译器是基于开源 Open64[9]编译器作为基本框架, 其中 BWDSP104X 的谓词模块在后端代码生成 (CG) 阶段, 在 CG 阶段开始后, 以中间语言 WHIRL[10]作为输入, 经过 CG expansion、生成 region tree 信息后开始以区域[11]为单元执行谓词优化. 具体优化阶段如图 1 所示, 从图可以看出谓词执行技术优化 if-conversion/parallel cmp 在生成 region formation 之后, 在软流水 (swp) 阶段之前, 谓词分析技术 (predicate analysis) 在软流水之后.

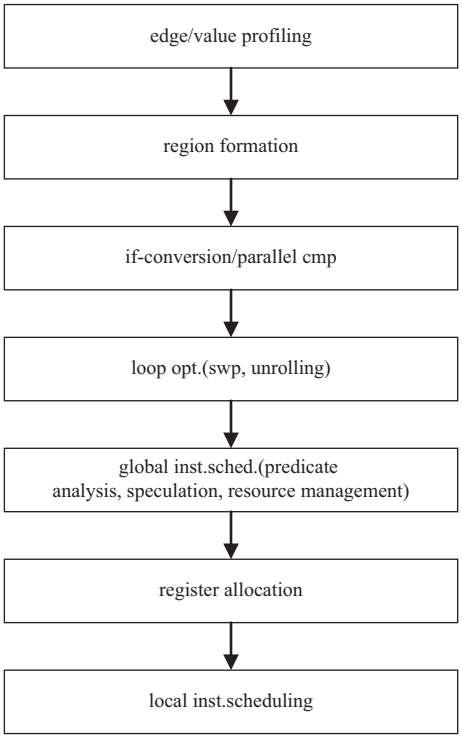

图 1 谓词执行流程

2 BWDSP104X 多条件谓词编译优化实现

2.1 传统谓词优化针对多条件谓词的局限性

传统谓词的优化形式为:

p1, p2=cond1

(p1) branch1

(p2) branch2

条件分支指令被转换为定义的谓词 p1 和 p2 指令, 当比较条件 cond1 结果为真时, p1 为 1, p2 为 0, 反之, p1 为 0, p2 为 1. 然后根据条件谓词的值来决定其分支是否执行, 若 p1 为 1, p2 为 0, 则执行 branch1, 否则执行 branch2, 如此程序中的分支跳转被消除, 控制关系转换为数据关系, 达到谓词优化的目的. 但这种形式的优化并不能消除多条件谓词的跳转语句和谓词之间的控制依赖关系.

传统的谓词优化技术对多条件分支的优化提供了一种可借鉴的实现思路, 但存在以下几点不足之处:

(1) 传统的谓词优化技术仅对单条件分支有很好的优化效果, 对多条件分支的优化不能很好的支持.

(2) 传统的谓词优化技术是在一个局部的基本块内实现的, 对多条件分支的全局谓词关系不能很好的描述.

(3) 传统谓词优化技术并不能很好的处理多条件分支间的跳转和控制依赖关系.

图 2 实例中, 多条件优化后生成的汇编格式图 3, 可以看出基本块之间仍然含有跳转语句分支, 图 4 实例经过多条件优化后生成的汇编格式图 5, 其中多个谓词之间含有控制依赖关系, 如 p3 和 p4 依赖谓词 p2. 这两种多条件谓词采用传统谓词优化后都不利于指令流水和指令并行, 为此提出一种基于 BWDSP104X 体系下多条件优化编译框架.

2.2 多条件谓词编译优化实现

传统的谓词编译框架主要包三个步骤: 将区域初始化生成条件转换候选结构的必备信息, 寻找合适的候选区域, 对候选区域进行转换生成谓词化代码. 具体流程如图 6 所示.

上述谓词优化方法对单条件谓词优化有很好的支持, 但对于多条件谓词并不能达到很好的优化效果. 其主要原因是, 多条件谓词之间往往含有控制依赖关系, 而上述优化方法在谓词转换时只是局部的进行代码谓词化, 并没有全局化分析多条件谓词之间的控制关系. 因此在转换生成谓词代码之前进行全局的谓词关系分析是很有必要的. 为此通过改进传统的谓词编译框架, 提出一种新的谓词编译框架, 其不仅适用单条件谓词优化, 对多条件谓词也有很好的支持, 具有更广的适用范围. 改进后的谓词编译优化框架如图 7 所示.

```
int mulcond1 (int a, int b, int c)
{
        int re;
        if (a>b&&a>c||b>c)
                re=1;
        else
                re=2;
        return re;
}
```

图 2 多条件谓词示例 1

```
BB1:
        p1, p2=a>b
        (p1) b BB2
        (p2) b BB3
BB2:
        p3, p4=a>c
        (p3) b BB4
        (p4) b BB3
BB3:
        p5, p6=b>c
        (p5) b BB4
        (p6) b BB5
BB4:
        re=1
BB5:
        re=2
```

图 3 示例 1 传统谓词优化生成汇编

```
int mulcond2 (int a)
{
        int re;
        if (a<0)
                re=1;
        else if (a<10)
                re=2;
        else if (a<20)
                re=3;
        else
                re=4;
        return re;
}
```

图 4 多条件谓词示例 2

下面具体阐述算法改进部分 Cal_Predicate_Path 的实现过程, 首先由起始基本块出发, 沿着控制流方向, 不断迭代, 直至找到每个控制块的全部控制谓词路径. 算法实现的流程如图 8 所示.

```
p1, p2=a<0
(p1) re=1
(p2) p3, p4=a<10
(p3) re=2
(p4) p5, p6=a<20
(p5) re=3
(p6) re=4
```

图 5　示例 2 传统谓词优化生成汇编

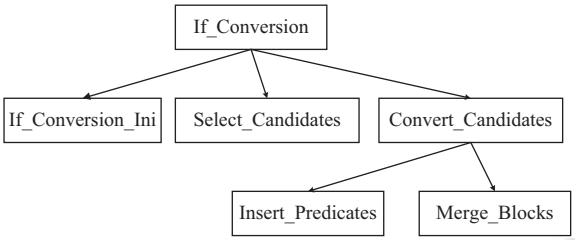

图 6　传统谓词编译优化框架

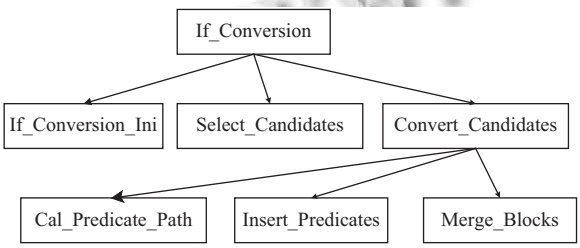

图 7　改进谓词编译优化框架

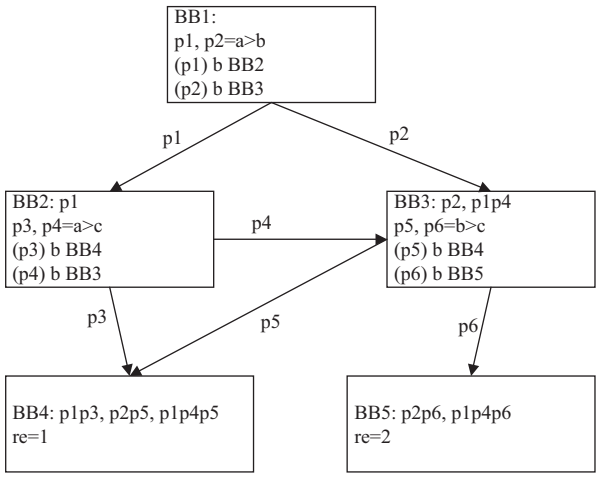

图 8　全局谓词控制路径图

由上述的谓词控制流分析, 由于多个谓词之间含有控制依赖关系, 一个基本块可能含有多个控制谓词路径, 如基本块 BB3 的控制谓词路径为 p2 或 p1p4, 其意思为控制谓词路径 p2 或 p1p4 有一个成立是, 基本块 BB3 便执行. 在计算出多条件谓词全局控制路径后, 其执行语句所对应的区块的每条控制路径都要插入一条谓词执行指令, 如基本块 BB4 含有 p1p3, p2p5, p1p4p5 三条控制路径, 故要插入三条谓词执行指令与其对应, (p1p3)re=1, (p2p5)re=1, (p1p4p5)re=1. 在基本块合并过程中主要完成: 删除无用分支指令, 即候选区域中的跳转指令, 将代码按拓扑排序排列, 并将其合并到一个基本块中. 最终生成的谓词化指令如图 9 所示.

```
p1, p2=a>b
p3, p4=a>c
p4, p5=b>c
(p1p3) re=1
(p2p5) re=1
(p1p4p5) re=1
(p2p6) re=2
(p1p4p6) re=2
```

图 9　谓词化指令

对比图 3 可以看出, 改进后的谓词优化消除了跳转分支, 将多个基本块合并为一个基本块. 同样, 对于图 5 进行全局谓词关系分析后, 可以消除多谓词之间的控制依赖, 从而解决传统谓词在优化多条件谓词的不足, 提高程序性能.

2.3　BWDSP104X 体系下汇编指令生成

对于上述谓词化指令按照 BWDSP104X 指令格式进行谓词定义指令和谓词执行指令汇编输出. 由于 BWDSP 体系下是使用谓词寄存器中的一位来保存条件判断的比较结果, 比如第一个条件分支 p1, p2=a>b, 对应的谓词定义指令为 CPred[0]=a>b. BWDSP 谓词执行指令汇编格式为 if CPred[K]==C do *n*. 其中 K 值取决于条件分支个数 *m*, K=2^*m*–1, 然后由谓词化指令中控制路径的加权值计算 C, 比如控制路径为 p1p4p5, C=p1*2^0+p4*2^1+p5*2^(*m*–1). 生成的谓词执行指令为 if CPred[7]==5 do 1 || re=1. 综上, 上述谓词化指令转化为 BWDSP 下汇编形式如图 10 所示.

```
CPred [0] =a>b
CPred [1] =a>c
CPred [2] =b>c
if CPred [7] ==3 do 1 || re=1
if CPred [7] ==4 do 1 || re=1
if CPred [7] ==5 do 1 || re=1
if CPred [7] ==0 do 1 || re=2
if CPred [7] ==1 do 1 || re=2
```

图 10　BWDSP104X 体系汇编代码

3 实验结果和分析

本文在 BWDSP104X 上实现了多条件谓词编译优化算法, 为了验证编译优化的的效果, 我们选取 DSPstone 以及 DSP 广泛应用的图形图像处理领域的实例进行测试. 为了更好的说明实验结果, 表 1 给出了测试用例中的核心代码.

表 1 测试用例及核心代码

测试用例	核心代码
vector_dot	sum += x[*i*] * y[*i*];
vector_max	if(x[*i*] > max) max = x[*i*];
sign	if(x[*i*]>0) y[*i*]=1; else if (x[*i*]==0) y[*i*]=0; else y[*i*]=-1;
vector_sum	if(cond1&&cond2||cond3&&cond4||…) sum+=x[*i*];

在进行测试时, 选取向量的规模大小为 N=2048, 并在 BWDSP 调试器 ECS(Effective Coding Studio) 中来测试谓词编译优化生成汇编指令的时钟周期, 具体结果如表 2 所示.

表 2 优化前后时钟周期数

运算(*N*=2048)	优化前	优化后	加速比
vector_dot	12298	12298	1.00
vector_max	9631	9631	1.00
sign	13605	2538	5.36
vector_sum	16050	2734	5.87

从表 2 可以看出当程序中不含有跳转语句 (vector_dot) 或者仅含有单条件跳转分支 (vector_max) 时, 优化前后的两种谓词模式持平, 实际上在此种情形下, 两种谓词模式产生相同的谓词控制指令. 但对于程序中含有多条件分支 (sign, vector_sum) 时, 相对于传统谓词优化模式, 新的谓词优化算法在汇编指令周期上有很大的性能提升, 这是因为改进的谓词优化算法消除了程序中多条件分支间的控制依赖和分支跳转指令.

4 结语

本文针对 BWDSP104X 体系结构, 提出一种多条件谓词编译优化方案. 通过全局的谓词关系分析, 得到每个控制块的谓词控制路径, 进而消除多条件谓词之间的控制依赖关系和每个基本块中的跳转分支, 将多个基本块合并为一个基本块, 从而实现指令流水, 达到指令级并行效果. 实验结果表明, 该方案在多条件谓词编译优化方面能够取得平均 5.62 倍的加速比. 今后, 还要对多谓词之间的等价、互斥、互补等关系进行分析, 通过建立谓词关系数据库 (PRDB), 对编译器下一优化阶段的谓词指令调度, 谓词寄存器分配以及谓词数据流分析提供更好的谓词模式支持.

参考文献

1 Ferrante J, Ottenstein KJ, Warren JD. The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems, 1987, 9(3): 319–349. [doi: 10.1145/24039.24041]

2 Allen JR, Kennedy K, Porterfield C, *et al*. Conversion of control dependence to data dependence. Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. Austin, TX, USA. 1983. 177–189.

3 Park JCH, Schlansker MS. On predicated execution. Technical Report HPL-91-58, Palo Alto, CA: Hewlett-Packard Laboratories, 1991.

4 Mahlke SA, Lin DC, Chen WY, *et al*. Effective compiler support for predicated execution using the hyperblock. ACM SIGMICRO Newsletter, 1992, 23(1-2): 45 –54. [doi: 10.1145/144965]

5 芦运照. 谓词相关编译技术和深层代码优化[博士学位论文]. 北京: 中国科学院研究生院 (计算技术研究所), 2004.

6 田祖伟. 基于 IA-64 谓词执行的 IF 转换技术研究[硕士学位论文]. 哈尔滨: 国防科学技术大学, 2005.

7 China Electronics Technology Group Corporation. CETC38. BWDSP100 hardware user manual. Heifei: China Electronics Technology Group Corporation Research Institute, 2011: 178–179.

8 China Electronics Technology Group Corporation. CETC38. BWDSP100 hardware user manual. Heifei: China Electronics Technology Group Corporation Research Institute, 2011: 180–181.

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

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

11 Aho AV, Lam MS, Sethi R, *et al*. Compilers: Principles, Techniques, and Tools. 2nd ed. Wilmington, DE, USA: Addison Wesley, 2006: 428–436.