针对多簇架构的软件流水调度框架设计与实现①

冯玉谦, 郑启龙, 陈思灵, 付和萍

(中国科学技术大学 计算机科学与技术学院, 合肥 230027)
(安徽省高性能计算重点实验室, 合肥 230026)

摘 要: 介绍基于编译基础设施 IMPACT 实现针对 BWDSP100 多簇体系结构特点软件流水调度框架. 该调度框架能充分发掘多簇体系架构的硬件资源, 在 DSP 特性应用程序上能有较好性能提升.

关键词: IMPACT; 软件流水; 多簇; DSP

Design and Implementation of the Software Pipelining Scheduling Framework for Multi-cluster Architecture

FENG Yu-Qian, ZHENG Qi-Long, LU Shi-Xian, CHEN Si-Ling, FU He-Ping

(School of Computer Science and Technology, University of Science and Technology of China , Anhui , Hefei,230027, China)
(National High Performance Computing Center(hefei), Anhui , Hefei, 230026, China)

Abstract: This article describes the framework of the software pipelining scheduling for BWDSP100 multi-cluster architecture which is based on compile basic facilities IMPACT. The framewokr can fully exploit the hardware resources of Multi-cluster architecture and get better performance improvement in the application of the DSP features.

Key words: IMPACT; software pipelining; multi-cluster; DSP

1 背景

软件流水[1]是一种用于提高程序中循环部分执行效率的优化技术. 该技术通过发掘循环的不同迭代的不同部分的指令间并行性, 使这些指令并行地执行, 从而提高循环部分的执行效率. 在一些嵌入式的数字信号处理和多媒体应用中有许多很典型的循环, 软件流水在这些应用的性能提升上有很明显的作用.

为了充分利用软件流水发掘出细粒度的指令级并行性(ILP), 处理器中增加大量功能单元(FU). 然而, 传统的集中式体系机构的成本随着 FU 数目的大规模增加而变得非常昂贵, 分簇体系结构应运而生.

在分簇的架构中, 处理器由一系列功能簇构成, 每个功能簇由一定数目的功能单元(FU)和寄存器文件(register file)的连接构成. 在多簇架构中, 簇间传输会带来的调度延迟的增加.

本文在编译基础框架 IMPACT[2]下, 设计并实现了针对 BWDSP100 C 语言编译器 BWCC 的软件流水功能框架. 其余部分组织如下: 第 2 节阐述了 BWDSP100 体系结构. 第 3 节阐述了 BWCC 基础框架.

第 4 节描述了针对多簇体系结构的软件流水模块的设计. 第六部分对全文进行了总结, 分析性能.

2 BWDSP100体系结构

BWDSP100 是一款高性能, 16 发射的 VLIW[3]结构的浮点运算信号处理器, 有 4 个计算簇, 分别是 X 簇, Y 簇, Z 簇, T 簇, 每个计算簇上有 8 个加法器, 4 个特殊功能单元, 4 个乘法器. 计算簇与计算簇之间通过簇间传输总线通信. 有 3 个地址簇即地址生成器, 分别是 U 簇, V 簇, W 簇.

3 BWCC 基础框架

BWCC 是基于可重定位编译器基础设施 IMPACT 研制的. IMPACT 是一款由 UIUC 大学研制的可重定位编译器. BWCC 前端是源程序转换为中间代码. BWCC 后端读入中间代码进行相应的优化和代码生成. BWCC 后端分为 6 大模块, 分别为指令注释、指令分簇、寄存器分配[4]、指令调度[5]和汇编代码生成, 如图 1 所示, 后端的输入是前端生成的与机器无关的

① 收稿时间:2012-07-14;收到修改稿时间:2012-08-17

中间代码；指令注释把中间代码指令注释成与 BWDSPl00 处理器相关的指令；指令分簇是把指令指定在特定的簇上去执行；软件流水对循环进行调度，提高循环部分执行效率；寄存器分配是把虚拟寄存器转换为物理寄存器；指令调度对非循环指令进行调度，提高非循环代码的执行效率；汇编代码生成是生成 BWDSPl00 的目标汇编代码.

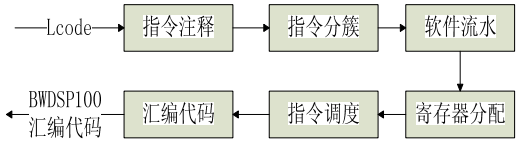

图 1 BWCC 基础框架

4 BWCC软件流水调度框架设计

软件流水技术的关键思想是找到核心代码，这样当重复执行这段核心代码时，就可以在前一次迭代尚未完成的时候初始化新一次的迭代. 在一个软件流水循环中，对一次迭代的调度被分为几个阶段，这样相继的迭代的不同阶段就可以相互覆盖并行地执行. 一次迭代中阶段的数目称为 “阶段数”，相继的迭代的初始化之间的周期数称为 “启动间隔”. 图 2 是一个 3 个迭代的例子，它分 3 个阶段，启动间隔为 1, Prologue 是流水装入阶段, Kernel 是流水的核心阶段，是由第一个迭代的 3 号指令，第二个迭代的 2 号指令和第三个迭代的 1 号指令一起执行, Epilogue 是流水的排空阶段.

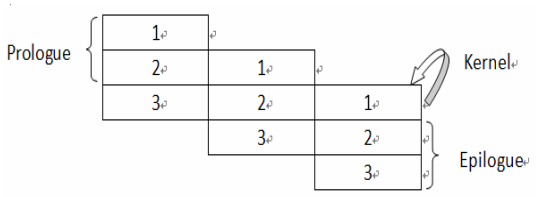

图 2 3 条指令的软件流水例子

在针对多簇体系结构的软件流水调度模块设计时，需考虑分簇信息，可行的方法有先分簇再进行流水调度的方法，分簇与流水调度同时进行的方法，先流水调度再进行分簇的方法. 在 BWCC 的软件流水模块设计中，是先进行分簇，再在簇内上和簇间进行软件流水.

4.1 分簇算法

BWCC 采用的改进的基于寄存器压力的分簇算法[6]. 在进行寄存器压力分簇前，先根据资源约束和依赖关系约束对核心代码进行启发式模调度[7].

(1) 若适合软件流水，则对根据模调度算法确定的迭代间隔 I，展开次数 N，对核心代码进行循环展开，并根据迭代次数将不同迭代划分到不同计算簇上. 对剩下的尚未分簇的指令进行正常的基于寄存器压力分簇.

对适合软件流水核心代码分簇原则: 根据迭代次数进行分簇，设置每个簇的最小迭代次数阈值，当低于该阈值时尽量将不同迭代分到较少簇上，以便充分发挥簇内并行性，当大于阈值时，尽量进行均匀划分，将不同迭代分到不同簇上，以充分利用不同簇的硬件资源.

(2) 若不合适，则不进行流水，直接进行正常的基于寄存器压力的分簇.

改进的基于寄存器压力的分簇算法如图 3 所示.

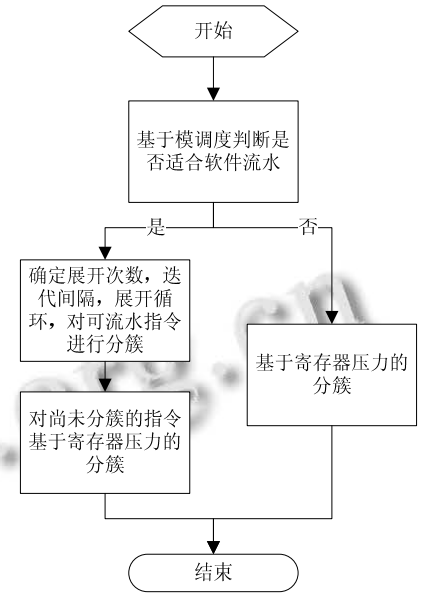

图 3 改进的基于寄存器压力的分簇算法流程

4.2 相关冗余代码消除

下图 4 使用一个 C 语言代码示例. 程序实现的是数组 b 和数组 c 的累加并赋值给数组 a.

下图 5 是 C 语言示例对应的中间代码. 其中 op13, op14 是从寄存器中取出 b[i],op15,op16 是从寄存器中取出 c[i], op17 是实现 b[i]*c[i], op18,op19 是将 c[i] 存入从寄存器中，op20,op21 完成下标 i 自增操作，op23,op22 是下标边界判断.

```
int a[32],b[32],c[32];int i ;
int main (){
for(i = 0; i<32 ; i++)
{a[i] = b[i]*c[i];}
}
```

图 4　C 语言代码示例

当确定迭代次数 N 后，需要对核心循环代码进行展开，由此带来代码膨胀，但是其中存在一定的冗余代码.

每个循环均有针对循环下标的判断操作指令，如图 5 中 op23,op22，但是进行展开后仅需要一遍判断指令即可. 可以消除其他循环的下标判断指令.

```
(cb 5 0.000000 [(flow 1 5 0.000000)(flow 0 10 0.000000)] <(L_CB_SOFTPIPE)>)
    (op 13 mov [(r 14 pnt)] [(l_g_gp _b)] <(popc (i 1525)(i 0))>)
    (op 14 ld_i <L>[(r 3 i)] [(r 14 pnt)(r 11 pnt)(i 0)] <(label (l_g_abs _b))(popc (i 1054)(i 0))>)
    (op 15 mov [(r 15 pnt)] [(l_g_gp _c)] <(popc (i 1525)(i 0))>)
    (op 16 ld_i <L>[(r 5 i)] [(r 15 pnt)(r 11 pnt)(i 0)] <(label (l_g_abs _c))(popc (i 1054)(i 0))>)
    (op 17 mul [(r 6 i)] [(r 3 i)(r 5 i)] <(popc (i 1022)(i 0))>)
    (op 18 mov [(r 16 pnt)] [(l_g_gp _a)] <(popc (i 1525)(i 0))>)
    (op 19 st_i <L>[] [(r 16 pnt)(r 11 pnt)(i 0)(r 6 i)] <(label (l_g_abs _a))(popc (i 1064)(i 0))>)
    (op 20 add [(r 11 pnt)] [(r 11 pnt)(i 1)] <(popc (i 1017)(i 0))>)
    (op 21 mov [(r 17 i)] [(r 11 pnt)] <(popc (i 1550)(i 0))>)
    (op 23 mov [(r 18 i)] [(i 32)] <(loop_nest (i 1))(LB_inner)(LE_inner)(popc (i 1002)(i 0))>)
    (op 22 br i lt [] [(r 17 i)(r 18 i)(cb 5)] <(loop_nest (i 1))(LB_inner)(LE_inner)(popc (i 1082)(i 0))>)
```

图 5　C 语言示例对应的中间代码

在每次循环中装载数据操作需要 2 个指令操作，如装载 a[i]需要 op13,op14. 其实每次数据装载数据时 oper< ld_i>都相同的，只不过把地址存入不同的地址寄存器，我们可以在每个簇上共用一个地址寄存器，这样就可以删除相同簇上剩余的 oper< ld_i>. 同时对相应的地址寄存器进行调整.

对核心代码进行循环展开分簇，并进行冗余代码消除. 其中展开次数为 4 次，指令都分在 X 簇，地址运算簇分在 U 簇. 考虑到展开后代码规模膨胀，便省略具体代码，以代码的结构示意图表示，如图 6 所示.

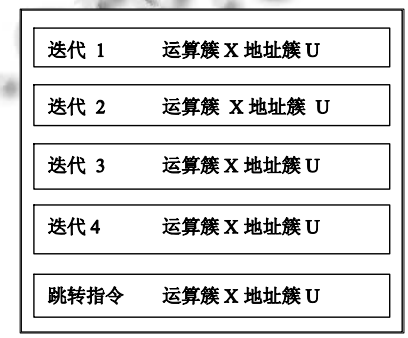

图 6 带有分簇信息消除冗余代码后的中间代码结构图

其中每个迭代的指令基本类似，包含取数，运算，存数操作指令，迭代 1 指令数目 8 条. 迭代 2，迭代 3，迭代 4 经过冗余代码优化后指令数目为 5 条. 跳转指令包含下标自增(展开 4 次对应的自增变量为 4)，判断是否循环结束等操作指令，指令数目为 4 条. 所以展开分簇并进行冗余代码优化后对应控制块的指令数目为 27 条.

分簇时是按照迭代次数进行分簇，包括分配运算簇和地址簇. 其中运算簇是基于迭代次数进行分配，地址簇是基于簇内可用寄存器压力进行分簇. 在上图 6 中，迭代 1—4 都分在运算簇 X 上，地址簇是 U.

4.3 指令调度以及生成多发射指令

指令调度基本原则:

(1) 对每个运算簇内指令进行流水，根据簇内的资源情况，以及同簇内指令数据流信息就行流水.

(2) 将不同簇的指令进行流水合成，考虑到不同簇直接可能有访存读写冲突等一些约束，进行全局调整.

(3) 形成展开后迭代的流水代码.

最后根据指令调度的相关信息通过指令注释形成含多发射的指令的汇编代码. 基本原则将可以在同一

个发射周期进行调度的指令同时发射，在汇编代码中以||表示同时发射．如下图 7 所示．

```
_main_5:
u5=u14 + 0
u13=_b||u6=u14 + 1
xr24=[u13+u5,0]||u7=u14 + 2
u12=_c||xr19=[u13+u6,0]||u0=u14 + 3
xr23=[u12+u5,0]||xr18=[u13+u7,0]
xr22=r24*r23||xr16=[u12+u6,0]||xr17=[u13+u0,0]
u11=_a||xr13=r19*r16||xr15=[u12+u7,0]
[u11+u5,0]=xr22||xr12=r18*r15||xr14=[u12+u0,0]
[u11+u6,0] = xr13||xr10=r17*r14
[u11+u7,0] = xr12
[u11+u0,0] = xr10
u14=u14 + 8
xr11=u14
xr10=32
```

图 7　含有多发射指令的汇编代码

5　总结

软件流水是目前主流编译器常见的优化策略．在基于多簇体系结构的编译器中实现软件流水框架需要考量分簇和流水之前的次序．

本文主要介绍了基于 BWDSP100 的编译器 BWCC 中软件流水框架的设计与实现．将分簇与流水进行统一考量．针对 DSP 的常见应用程序的特点即数值计算多，迭代间依赖较少来实现的．在 DSP 应用领域有较好的性能提升．

在针对本文所示的 C 语言示例(循环 32 次)，在流水前汇编指令条数 32×11=352；软件流水后(循环展开次数 4 次)汇编指令(包括多发射指令)条数 8×14=112．可以得出在核心循环代码处经过流水后性能提升 68%．

参考文献

1 Lam MS. Software pipelining: An effective scheduling technique for VLIW machines. Proc. of the SIGPLAN88 Conference on Programming Language Design and Implementation, June 1988. 318–328.

2 Chang PP, Mahkle SA, Chen WY, Water NJ, HWU WW. IMPACT: An architectural framework for multpile-instruction-issue prcoessors. 18th Annual International Symposium on Computer Architecture, Bacrelona: ACM Press, 1998: 408–417.

3 Fisher JA.Very Long Instruction Word Architectures and the ELI- 512;1983,11(3): 140–150.

4 Chow F, Hennessy J. Register allocation by priority-based coloring. ACM SIGPLAN Notices, 1984,19(6):222–232.

5 Philip B, Steven PS. Gibbons Efficient instruction scheduling for a pipelined architecture. ACM SIGPLAN Notices, 1986,21(7):11–16.

6 雷一鸣,洪一,徐云,姜海涛.一种基于寄存器压力的 VLIW DSP 分簇算法.计算机应用,2010,30(1):274–276.

7 Rau1 BR. Iterative modulo scheduling: An algorithm for software pipelining loops1. Proc. of the 27th Annual Int'l Symp on Microarchitecture1 New York: ACM Press, 1994: 63–71.

(上接第 56 页)

机结构静力试验计算机辅助设计系统对象体系开发的 ASSTDS 系统已得到初步应用，在今后的应用中系统对象体系与软件系统还会不断地完善和发展．

参考文献

1 Norman RJ. Object-Oriented Systems Analysis And Design. 北京:清华大学出版社,1997.

2 Buhr RJA, Cassselman RS. Use Case Maps for Object-Oriented Systems.北京:清华大学出版社,1997.

3 侯同济.面向对象的飞机结构试验设计软件分析与开发.结构强度研究.2004,4.

4 侯同济.飞机结构静力试验计算机仿真系统的构架与组成.结构强度研究.2005,2.