

面向 VLIW 结构的高性能代码生成技术

王红梅^{1,2}, 王 敏^{1,2}, 张铁军², 单 睿², 侯朝焕²

(1 中国科学院 研究生院, 北京 100039; 2 中国科学院 声学研究所, 北京 100190)

摘 要: DSP 处理器通过采用 VLIW 结构获得了高性能, 同时也增加了编译器为其生成汇编代码的难度. 代码生成器作为编译器的代码生成部件, 是 VLIW 结构能够发挥性能的关键. 由此提出并实现了一种基于可重定向编译框架的代码生成器. 该代码生成器充分利用 VLIW 的体系结构特点, 支持 SIMD 指令, 支持谓词执行, 能够生成高度指令级并行的汇编代码, 显著提高应用程序的执行性能.

关键词: VLIW; 代码生成; 指令级并行度; 可重定向编译

中图分类号: TP301.6

文献标识码: A

文章编号: 1000-7180(2010)02-0009-04

A High Performance Code Generator for VLIW Architectures

WANG Hong-mei^{1,2}, WANG Min^{1,2}, ZHANG Tie-jun², SHAN Rui², HOU Chao-huan²

(1 Graduate School of Chinese Academy of Sciences, Beijing 100039, China;

2 Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: VLIW architecture is used to enhance the performance of DSP, but also poses a challenge to compiler to generate efficient code for it. As code generation unit of compiler, code generator plays an important role in exploiting VLIW techniques. A code generator based on retargetable compilation infrastructure is proposed and implemented in this paper. It employs the efficient hardware in VLIW structure to greatest extend, supports both SIMD instructions and predication execution, and is able to generate high instruction-level-parallelism assembly code, which greatly improves performance of applications.

Key words: VLIW; code generation; instruction level parallelism; retargetable compilation

1 引言

随着多媒体和宽带通信技术的应用日益广泛, 基于 VLIW (Very Long Instruction Word) 体系结构的新一代 DSP 被提出并用于高性能数据密集计算. 基于 VLIW 结构的 DSP 处理器具有多个功能单元, 能并行执行多条机器指令, 硬件控制简单, 由编译器检测指令的并行性信息并完成调度. 为了充分利用 VLIW 处理器的高效硬件结构, 就必须在编译期间进行合理的指令调度^[1-2], 因此编译器在开发 VLIW 结构处理器中占有重要地位.

编译器一般分为目标机器无关的前端和目标机器相关的后端. 代码生成器作为编译后端的一个重要部件, 体现目标处理器的体系结构特点. 文中采用

可重定向编译技术^[3-4], 在可重定向编译框架 IMPACT 的前端基础上, 根据目标处理器 SUPERV II 的体系结构特点设计和实现了一个高性能的代码生成器. 该代码生成器支持 SIMD 指令, 支持谓词执行, 生成的代码序列具有高度指令级并行性^[4] (Instruction Level Parallelism, ILP).

2 目标机体系结构介绍

SUPERV II 是我们实验室自主研发的一款定点 VLIW 结构 DSP, 有 7 个功能单元. 体系结构有如下特点: 七级流水线设计, 非对称寄存器堆结构, 支持谓词执行, 同时支持 SIMD 操作. 其内核的体系结构如图 1 所示.

采用 VLIW 结构的 DSP 处理器提高了硬件工

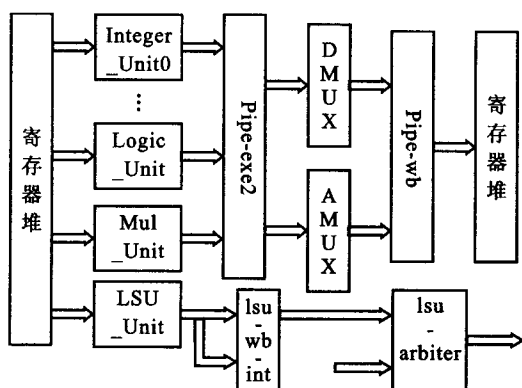


图1 SUPERV II 内核结构图

作效率,对于处理器提升性能有很大的好处,但也对编译器提出新的要求,编译器应当充分分析应用程序的数据依赖关系,发掘程序并行信息,进行有效的指令调度和寄存器分配后,生成具有高度 ILP 的代码序列。

3 SUPERV II 代码生成器的总体框架和主要步骤

IMPACT^[5]是伊利诺斯大学面向 VLIW/EPIC 体系结构开发的编译基础框架,主要目的在于发现应用程序中的指令级并行性。IMPACT 使用 Hmdes 机器描述语言定义目标体系结构和指令集,使用 Mspec 模块的接口函数定义目标编译器的运行环境,具有重定向特点,但是与处理器相关的代码生成器需要单独设计。针对 SUPERV II 的体系结构特点,图 2 给出了其代码生成器的设计和实现框架。

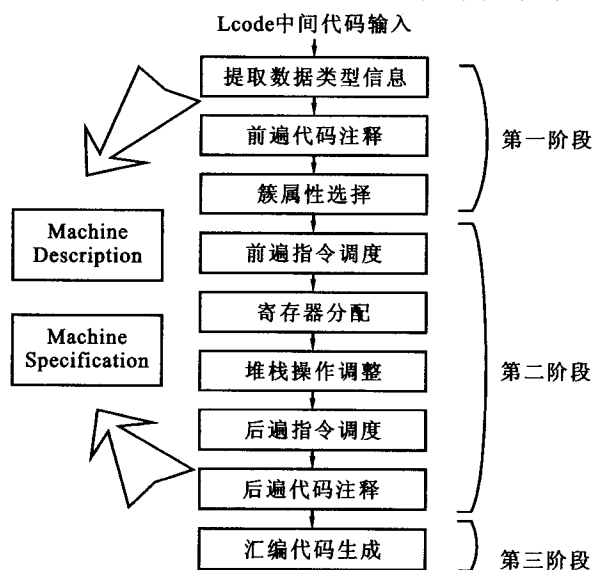


图2 SUPERV II 代码生成器框架

代码生成器的输入是 Lcode(Lcode 是 IMPACT 的前端将 C 程序经过若干处理和优化转换而来的机器无关中间代码),输出则是 SUPERV II 的汇编

代码。由图 2 可以看出,代码生成器包括三个阶段分别是代码注释,代码调度,以及汇编代码输出,而机器定义则为代码生成器提供目标机参数查询系统。为了保证 SUPERV II 代码生成器的正确性,并获得较高的 ILP,我们对这几个阶段进行了详细的研究。

3.1 机器定义

机器定义包括机器描述(machine description)和机器接口规范(machine speciation),它们是代码生成器不可缺少的模块^[6]。机器描述部分使用 Hmdes 机器描述语言,对目标机器资源和指令集进行描述。通过扩展的 HMDDES 编译系统生成机器描述的内部表示,提供一系列标准的查询接口,代码生成器的各模块可以直接和这些接口进行交互获取目标机信息。机器接口规范主要对应用程序部分接口进行了规定,包括数据类型,对齐方式,以及函数调用规则等约定。

3.2 代码注释

代码注释阶段的主要任务是将机器无关的 Lcode 转换成机器相关的 Mcode 中间代码,转换后的 Mcode 每条指令对应一条目标机器指令,其数据结构依然与 Lcode 相似。根据目标机指令集对 Lcode 的支持情况,出现两种注释类型:一对一注释和一对多注释,如图 3 所示。对于 SIMD 指令的注释,文中采用将 C 程序段直接映射为 SUPERV II 指令的内联函数实现方式。这种做法既简化编译器对 SIMD 指令的实现,又可以提高编译器使用者的开发效率。

Lcode op	Mcode op
1. add(r 3 i) < - (r 1 i)(r 2 i)	1. add(r 3 i) < - (r 1 i)(r 2 i)
1. div(r 3 i) < - (r 1 i)(r 2 i)	1. addid(mac \$ P 0 i) < - (r 1 i)(i 0)
	2. addid(mac \$ P 1 i) < - (r 2 i)(i 0)
	3. jsr_div32
	4. addid(r 3 i) < - (mac \$ P 0 i)(i 0)

图3 代码注释举例

3.3 代码调度

代码调度对 Mcode 进行优化和调度,包括指令调度和寄存器分配。考虑到基于 VLIW 结构的 DSP 处理器功能单元较多,而寄存器相对较少,文中采用前遍调度,然后寄存器分配,随后后遍调度的方法,既保证指令调度中较高的 ILP,又保证寄存器分配没有太多溢出代码。

3.3.1 基于超块的列表调度

指令调度的作用是在不改变语义的前提下,合理安排指令执行的顺序,达到提高 ILP,缩短调度长度的目的。文中采用基于超块^[7]的列表调度算法为运行于 SUPERV II 上的应用程序开发 ILP。列表调

度算法基于全局的数据流程图继续处理,优化效果好且复杂度不大.具体算法如下:

```

Algorithm hyperblock-based list_scheduling
Input: unscheduled control block
Output: scheduled control block
{ /* 构造控制块的数据依赖图 */
  NewDDG(cb);
  /* 计算块内操作的优先级 */
  Compute_Oper_Priority(cb);
  /* 建立相关性已满足的就绪指令队列 */
  Build_Ready_Queue();
  While (queue != NULL)
  {
    Reset_tested_flag(oper);
    While(((oper_nex_ready()) != Null)
    & ! tested & ! sched)
    {
      If (resource_ok(oper))
        Fill_resource_table(oper, time);
        Schedule_Oper(oper, time);
        Set_sched_flag(oper, scheduled);
      Else
        Set_tested_flag(oper, Tested);
    }
  }
  Update_Ready();
  Time ++;
}

```

其中, NewDDG(cb) 是依据机器模型中所描述的延时信息建立的基本块依赖关系图; compute oper_priority(cb) 计算每条指令调度优先级, 使用节点深度作为启发式方法, 关键路径优先保证调度长度最短. Build_Ready_Queue() 建立就绪指令队列, 所谓就绪指令是指在 DDG 图中所有父母节点均已被调度的指令或者是没有父母节点的指令. oper_nex_ready() 在每个时钟周期依次获取当前时刻就绪指令集合中的所有指令. resource_ok(oper) 通过查询机器描述 MDES 获得每条指令的延时和功能单元状态, 如果机器资源可用, 则填充 MDES 资源表格 (Fill_resource_table(oper, time)) 和设置该指令的调度时间, 并将该指令加入已调度队列 (Schedule_Oper(oper, time)) 并标记指令为已调度; 否则标记该指令为已测试 (Set_tested_flag(oper, Tested)). 所有已调度和已测试指令在下一个指令槽填充时可以直接跳过. 当所有就绪指令已调度或已测试, 则跳到下一个指令周期并更新就绪指令集合

(Update_Ready()), 进行新一轮的调度, 直到所有指令都被调度.

3.3.2 寄存器分配

前遍指令调度完成后, Mcode 指令的操作数都是虚拟寄存器, 寄存器分配阶段负责将所有的虚拟寄存器和 SUPERV II 的物理寄存器对应起来, 并保证溢出操作最少. 文中的寄存器分配算法是传统图着色寄存器分配算法上的改进, 它支持不同寄存器簇的分配, 支持不同保存方式的寄存器选择 (调用者保存和被调用者保存), 算法流程如图 4 所示.

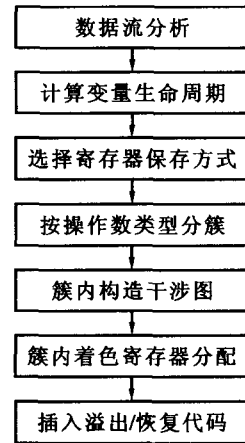


图4 寄存器分配流程

在为寄存器选择保存方式时, 为了减少溢出代码, 需要尽量将生命周期中含函数调用较少的虚拟寄存器放入调用者保存类型的物理寄存器中, 而将生命周期中含函数调用较多的虚拟寄存器放入被调用者保存类型的物理寄存器中, 这样可以在子函数真正用到该虚拟寄存器时才需要将其值进行保存, 从而减少不必要的溢出/恢复操作. 文中采用以下代价函数判断虚拟寄存器分配到两种类型的寄存器的收益, 进而为虚拟寄存器选择合适的物理寄存器. 式(1)给出虚拟寄存器溢出的代价, 式(2)和式(3)则计算将虚拟寄存器分配给调用者保存和被调用者保存的收益.

$$\text{spill_cost}(\text{vreg}) = \text{def_instr}(\text{vreg}) \times \text{store_cost} + \text{use_instr}(\text{vreg}) \times \text{load_cost} \quad (1)$$

$$\text{caller_benefit}(\text{vreg}) = \text{spill_cost}(\text{vreg}) - \text{jsr_weight}(\text{vreg}) \times (\text{store_cost} + \text{load_cost}) \quad (2)$$

$$\text{callee_benefit}(\text{vreg}) = \text{spill_cost}(\text{vreg}) - \text{func_weight}(\text{vreg}) \times (\text{store_cost} + \text{load_cost}) \quad (3)$$

其中式(1)中的 def_instr 和 use_instr 分别代表对虚拟寄存器 vreg 进行赋值和引用的指令节点数, store_cost 和 load_cost 表示将一个 vreg 从内存中加载或向内存中存储所耗费的指令周期代价. 式

(2)中的 jsr_weight 表示 vreg 生命周期中发生函数调用的频率,而式(3)中的 func_weight 则表示该函数被调用的频率^[8].

3.4 汇编代码输出

Mcode 经过代码调度已经获得了较高的 ILP,但是由于仍使用和 Lcode 一样的数据结构,还需要经过汇编输出的处理才能完全符合 SUPERV II 的规范输出格式.该阶段的工作是根据 Mcode 输入类型分别输出操作码,操作数和指令包信息.

4 试验结果及讨论

在为 SUPERV II 设计并实现了代码生成器后,选取一些 DSP 常用算法对生成代码的性能进行测试,这些算法包括矩阵乘积 mm_int.c,有限冲击响应滤波器 fir_int.c 和汉娜塔 hana.c 等.图 5 给出了测试程序的执行时间加速比,图 6 给出了平均 IPC(Instruction per cycle)和峰值 IPC 测试结果.

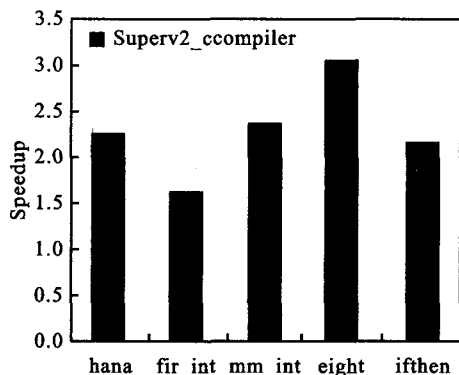


图 5 测试程序的执行时间加速比

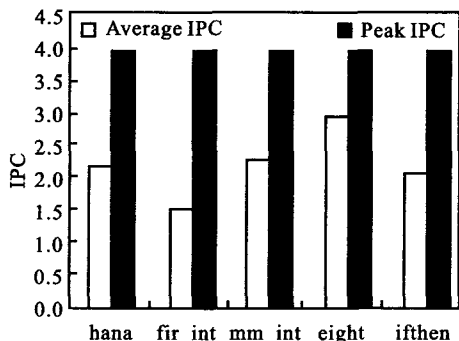


图 6 测试程序的 IPC 比较

从图 5 和图 6 的测试结果可以看出,测试程序的最高加速比为 3(以 GCC^[8]执行时间为参照),最低加速比为 1.6,受硬件发射宽度制约峰值处的 IPC 为 4.这说明代码生成器生成的代码在识别程序的并行性上,取得了比较理想的效果.

5 结束语

文中在可重定向编译基础框架 IMPACT 的基础上,为 SUPERV II 处理器设计和实现了一个高性能的代码生成器.实验数据表明该代码生成器能够识别程序中的并行部分,并能生成具有较高 ILP 的指令序列,使应用程序的执行性能获得改善.

参考文献:

- [1] Muchnich S. Advanced compiler design and implementation - n[M]. San Francisco; Morgan Kaufmann, 2001.
- [2] 胡定磊. VLIW 编译器设计及性能与功耗的优化研究[D].长沙:国防科学技术大学,2006.
- [3] 戴桂兰.编译基础设施中多目标编译技术探讨[J].计算机研究与发展,2003,40(2): 312-317.
- [4] 周志雄. ASIP 处理器的可重定位编译器的设计[J].微电子学与计算机,2007,24(1): 38-41.
- [5] Chang P P, Mahlke S A. Impact: an architectural framework for multiple-instruction processors[C]// Proceeding of the 18th Annual International Symposium on Computer Architecture. Chicago: Illinois University, 1991.
- [6] Bringman R A. A template for code generator development using the IMPACT-1 C compiler[D]. USA, Alabama: University of South Alabama, 1983.
- [7] Hwu W W, Mahlke S A, Chen W Y, et al. The superblock: an effective technique for VLIW and superscalar compilation[J]. The Journal of Supercomputing, 1993(7):229-248.
- [8] 赵贤鹏.一种基于 GCC 的 VLIW 编译器指令调度算法[J].微电子学与计算机,2004,21(1): 62-64.

作者简介:

王红梅 女,(1984-),博士研究生.研究方向为高性能编译技术和体系结构.