第27卷第2期
2006年2月

小型微型计算机系统
MINI-MICRO SYSTEMS

Vol.27 No.2
Feb.2006

分簇结构超长指令字DSP编译器的设计与实现

胡定磊，陈书明，刘春林

（国防科技大学 计算机学院，湖南 长沙 410073）
E-mail：dl_hu@163.com

摘 要：超长指令字(VLIW)是高端DSP普遍采用的体系结构，VLIW DSP在硬件上没有调度和冲突判决的机制，其性能的发挥完全依靠编译器的优化效果.基于可重定向编译基础设施IMPACT，为分簇VLIW DSP YHFT-D4设计与实现了优化编译器.其中着重讨论了可重定向信息的定义、代码注释、SIMD指令的支持、分簇寄存器分配以及指令级并行开发和资源冲突解决等内容.实验结果表明该编译器可以达到较好的优化效果.

关 键 词：VLIW DSP；编译器；分簇寄存器分配；指令级并行
中图分类号：TP311.1　　**文献标识码**：A　　**文 章 编 号**：1000-1220(2006)02-0348-06

Design and Implementation of Clustered VLIW DSP Compiler

HU Ding-lei, CHEN Shu-ming, LIU Chun-lin

(*Computer School, National University of Defence Technology, Changsha* 410073, *China*)

Abstract: Very Long Instruction Word (VLIW) architecture has been adopted by most high performance DSPs. There are no hardware to schedule instructions or to detect conflictions in VLIW DSPs, so the performance of VLIW DSP is dependent on compiler optimization. Based on retargetable compiler infrastructure IMPACT, An optimized compiler has been design and implement for clustered VLIW DSP YHFT-D4. Definition of retargetable information, code annotation, SIMD instructions supporting, clustered register allocation, instruction-level-parallel (ILP) exploring and resource confliction solving are emphatically discussed. Experiments show that the compiler is very effective in optimization.

Key words: VLIW DSP; compiler; clustered register allocation; ILP

1 引 言

数字信号处理器(DSP)是一种用于数字信号处理的嵌入式专用处理器.为了提供强大的运算能力，DSP内部往往采用不同于通用微处理器的体系结构，并支持专用DSP指令以加速常用DSP算法.为了满足嵌入式应用对于时间和空间的要求，可以采用汇编语言编写程序.虽然这样手工编写的汇编代码效率较高，但是耗时、易于出错、维护困难、难于移植.随着市场竞争的加剧，这种方式已经不适应开发周期越来越短的现实.如果有一个优化效果较好的高级语言编译器，则既可以满足DSP应用的时空要求，又可以克服手工编写汇编代码开发方式的缺点.C语言是效率最高的通用高级编程语言，所以目前绝大多数DSP支持的高级语言编译器都是C编译器.

近年来，各种应用(特别是多媒体应用)对DSP的性能要求不断提高.为了提高DSP处理的性能，现在高端的DSP普遍采用了超长指令字(VLIW)体系结构.目前几家主要的DSP厂商都纷纷推出了VLIW体系结构的高端DSP，如TI的C6000系列DSP、Motorola的StarCore系列及ADI的Tiger-SHARC系列、Philips的Tri-Media系列等.面向DSP的应用往往包含较大的数据并行性，VLIW体系结构是挖掘这种数据并行性的一种理想方法.VLIW将指令调度，冲突判决等工作完全交给编译器，一方面可以大大降低芯片中控制单元的复杂度，对于芯片提升频率、降低功耗都有很大的好处；但另一方面也对编译器提出了严峻的挑战：VLIW DSP的高性能是否能够发挥出来，完全取决于其编译器优化效率的高低.

从头开始开发一个编译器的工作量是巨大的.一般情况下编译器的前端是与机器无关的，如果在借用一个现有编译器前端的基础上，针对目标体系结构对编译器的后端进行开发，则可以大大节省工作量.现在有一些开放源代码的可重定向编译基础设施可资利用，如IMPACT、GCC、Trimaran、Zephyr等[1].所谓可重定向是指编译基础设施提供了对目标体系结构的定义机制和利用这些定义来自动构造编译器后端的机制.但就目前而言，这些编译基础设施的重定向机制还很不完善，为目标体系结构生成编译器还需要在后端解决大量技术问题.

2 相关研究

自Fisher提出VLIW概念的二十余年来[2]，对于VLIW编译的研究一直是编译领域的一个热点.特别是随着近年来EPIC(显示并行指令计算)结构的Itanium处理器，以及大量

收稿日期：2004-10-10　基金项目：国家"八六三"高技术研究发展计划基金项目(2004AA1Z1040)资助　作者简介：胡定磊，男，1976年生，博士研究生，主要研究方向为超长指令字编译、低功耗编译等；陈书明，男，教授，博士生导师，主要研究方向为计算机体系机构，通用微处理器、DSP及超大规模集成电路设计等；刘春林，男，教授，主要研究方向为程序设计语言与编译等.

采用VLIW体系结构的嵌入式芯片的出现,对于VLIW编译技术的研究日益活跃.Faraboschi等以编译优化所处理的对象代码的范围的不同,对开发指令集并行的主要编译技术分类进行了综述,着重讨论了基于范围的调度技术(Region-based scheduling)和对循环代码的优化调度技术[3].

使用可重定向编译基础设施为目标体系结构,特别是为嵌入式处理器,开发编译器,有很多这样的例子.Kim等使用Zephyr编译基础设施为一个商用的网络处理器Paion PPII构造了编译器[4].Rajagopalan等使用IMPACT编译基础设施为Fujitsu Hiperion定点DSP构造了编译器[5].Shannon等使用IMPACT编译基础设施为VLIW定点DSP核SC140构造了编译器[6].Chakrapani等使用Trimaran编译基础设施为嵌入式芯片StrongARM构造了编译器[7].虽然这些开发者都利用了编译基础设施的可重定向机制,但为了提高ILP在编译后端都加入了针对目标体系结构的优化.

因为VLIW体系结构包含多个功能部件,往往需要多端口的寄存器文件提供支持.但是端口越多,寄存器文件越复杂,频率提升也越困难.为了解决这一问题,现在有些VLIW DSP采用分簇(Clustered)结构,即把寄存器文件分为若干组,每组寄存器对应若干功能单元.TI的C6000系列DSP以及AD的Tiger-SHARC系列DSP就采用了这种结构.编译器必须充分利用这些簇以提高ILP,这就涉及指令和操作数的簇间分配——即将指令映射到合适簇中的功能单元、将操作数映射到合适簇中的寄存器.

Leupers提出了一个簇间指令分派的算法,该算法首选将指令随机的在各个簇间进行分配,然后使用模拟退火算法对分配进行改进[8],但是这种方法会消耗太多的编译时间.Lapinskii也提出了两阶段簇间指令分派算法,第一阶段依据每条指令分配代价的不同,建立初始的分配,第二阶段通过一种边界扰动分析进一步优化分配[9].这两种方法都是从指令的角度来来进行簇间分配,只完成了簇间分配的一部分任务,后面还需要对操作数进行簇间分配才能完成簇间分配的任务.Jang等提出了簇间寄存器分派的算法:簇间寄存器分配时,首先建立寄存器构成图(register component graph),边和节点赋予一定的权重,实施改进的最小费用割集算法,将符号寄存器分配到不同的簇中,在簇中采用传统的着色图法分配寄存器[10].

3 分簇VLIW DSP编译器的总体框架

YHFT-D4是一款VLIW DSP,有8个功能单元,分为对称的两簇.其内核的体系结构的如图1所示.该体系结构有如下特点:内核分为两个对称的簇,每个簇上各有四个功能单元和一个寄存器文件,簇间有交叉读通路;指令可以条件执行,并含有大量DSP专用指令和SIMD指令.YHFT-D4在硬件上没有任何调度和冲突判决机制,指令的执行顺序以及资源的使用都是编译器来安排的.

IMPACT是Illinois大学开发的一个编译基础设施,重点在于展示、提高和利用指令集并行技术[11,12].IMPACT使用Hmdes机器描述语言定义目标体系结构和指令集[13],使用Mspec模块的接口函数定义目标编译器的运行时环境(Run-time Environment),可以实现一定程度的重定向.但在

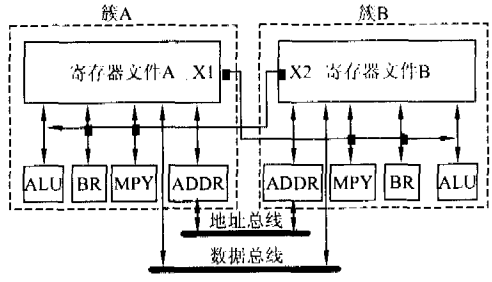


图1 YHFT-D4内核体系结构

后端还需要开发者做许多工作,针对目标体系结构进行处理以保证编译器的正确和高效.图2给出了分簇VLIW DSP编译器设计和实现的总体框架.

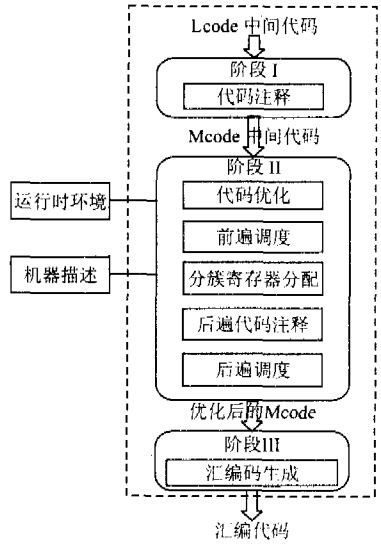


图2 编译器设计和实现的总体框架

这里的输入——Lcode中间代码——是IMPACT的前端将C程序经过若干处理转换而来的机器无关中间代码,最终的输出则是YHFT-D4的汇编代码.编译器后端的工作主要分为三个阶段.第一阶段是代码注释阶段,将Lcode中间代码转换成机器相关的中间代码Mcode;第二阶段主要对Mcode进行调度优化,涉及优化、调度、寄存器分配等,目标机器的描述和Mspec模块主要服务于这一阶段,指令在簇间的分配也在这一阶段完成;第三阶段将优化后的Mcode中间代码转换成目标机器的汇编代码.下一节给出了设计和实现该分簇VLIW DSP编译器中解决的主要技术问题.

4 分簇VLIW DSP编译器设计与实现中解决的主要技术问题

从图2可以看出,我们的分簇VLIW DSP编译器含有多个模块,但其中有一些模块虽然是编译器必不可少的——如将优化的Mcode中间代码转换成目标机器的汇编代码,主要

是一个工程实现问题——但没有很强的理论性，在此我们将不做讨论. 为了保证编译器的正确性，获得较高的 ILP，针对目标 YHFT-D4 VLIW DSP，我们针对以下几个方面进行了重点的研究，

4.1 目标机器可重定向信息的定义(Mdes & Mspec)

使用 IMPACT 编译基础设施为目标机器构造编译器，需要描述两部分可重定向信息：一是描述目标机器的体系结构和指令集信息，二是描述目标编译器所遵循的一些规范(主要是定义编译器的运行时环境).

对于目标机器的描述主要使用 Hmdes 和 Lmdes 两种描述语言来完成. 其中，Hmdes 是面向编译器开发人员使用的一种易读易写的高端机器描述语言，可以在文本编辑器中进行描写. Lmdes 是目标机器描述的内部表示，是一种面向编译程序的低端描述语言. 用 Hmdes 描述完成之后，将描述文件转化为 Lmdes 描述格式，构造机器信息查询接口供编译程序快速访问.

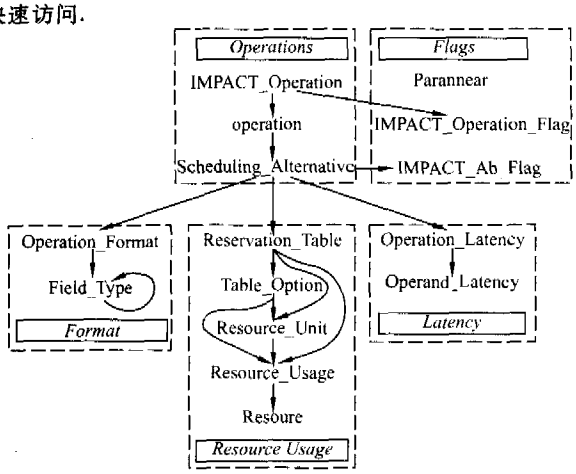


图 3 Hmdes 机器描述结构

Hmdes 对目标机器的描述信息集中在下面 6 部分，其中每部分由不同的段(SECTION)组成，其结构见图 3：

1) 目标机器支持的操作格式信息(Operation_Format 段)；

2) 目标机器的资源信息(Reservation_Table 段)；

3) 延时信息(Operation_Latency 段)；

4) 调度信息(Scheduling_Alternative 段)；

5) 目标机器支持的所有操作信息(IMPACT_Operation 段)；

6) 编译程序特定相关信息(Parameter 段等).

例如，对于指令 XOR 的调度信息是这样定义的：

ALT_XOR (format(OF_A_A_A) resv(RT_L1_S1) latency(OL_Single_cycle))；

这说明，指令 XOR 的格式形如 src1, src2, dst，可以在 L1 和 S1 单元上执行，是一个单周期指令.

为了使编译器的前端能够产生符合目标机器规范的中间代码，必须将编译器的运行时环境通知编译基础设施的前后端. 所有这些信息都是通过 Mspec(即机器规范说明)模块以接口函数的形式传递给编译器的.

4.2 代码注释

代码注释指的是将一种指令格式转换为另一格式的过程. 这一阶段的工作量大小反比于目标机指令集体系结构同 Lcode 的相似程度. 相似程度大，则工作量小，反之，则大.

图 4 是代码注释的结构框图，代码注释主要包括控制块注释(CB Annotation)和指令注释(Oper_Annotation)两部分. 控制块注释将 Lcode 的控制块转换为基本块，以便后续阶段的处理. 指令注释则将 Lcode 指令转换为针对目标机体系结构(YHFT-D4)相关的 Lcode 表示形式 Mcode.

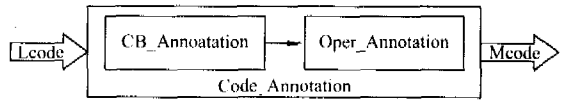


图 4 代码注释结构图

代码生成阶段输入的 Lcode，它的一个控制块可包含多条跳转指令，而此后的指令注释、寄存器分配和调度等阶段，它们每次作用于 Lcode 的指令范围均为一个基本块，因此注释阶段的第一个步骤就是将 Lcode 控制块变换为基本块. 基

```
ALGORITHM CB_Annotation
INPUT    Lcode_fn with control_block
OUTPUT   Lcode_fn with basic_block
{
         Clear_src_flow(Lcode_fn);
         for (each Cb in this Lcode_fn)
              Convert_into_bb(cb);

         Rebuild_src_flow(Lcode_fn);
}
```

图 5 控制块注释算法

本块由一段连续执行的指令序列组成，它有唯一的入口和出口. 控制块 CB 的内部数据结构维持了块间的控制流信息. 控制流分为源流和目的流. 源流连接一个控制块和可以跳转至它的控制块，目的流则连接该控制块和它可以跳转至的控制块. 每条流相应于一条跳转指令. 图 5 是控制块注释算法. 注释前，应清除函数中所有控制块的源流，接着将每一个控制块转换为基本块，最后重建所有基本块的源流.

指令注释的功能主要是将机器无关的 Lcode 中间代码转换成机器相关 Mcode 中间代码，使转换后的 Mcode 中间代码中的每条指令对应一条目标机器的汇编指令，但其格式依然与 Lcode 中间代码相似(因为后续的调度、寄存器分配等是使用这种格式的). 一般来讲会出现这么几种情况. 有一些指令，它们可直接映射为目标机的指令而不必注释；也有一些指令，它们可能需要少量的注释；还有一些指令，目标机不直接支持，要求较复杂的注释. 若从注释前后指令数目的对比考虑，可将注释分为一对一注释(1:1)、一对多注释(1:n)和多对一注释(n:1)，如图 6 所示.

4.3 SIMD 指令的支持

DSP 的主要应用之一是媒体处理，媒体处理的一大特点是具有高度的数据并行性，为此，DSP 使用向量指令或类似向量指令的 SIMD 指令来加速媒体处理. YHFT-D4 DSP 也

加入了许多 SIMD 指令,如 add4、sub4 等.应当说就目前而言,如何从普通C程序中识别出SIMD指令,仍然是一个前沿

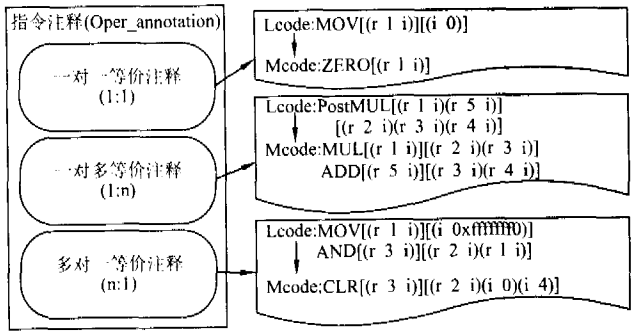


图 6　代码注释情况示意图

性的研究课题.在本编译器中,我们是通过目前业界普遍采用的内联函数(intrinsics)机制实现对 SIMD 指令支持的,这样做既可以简化编译器支持 SIMD 指令的实现,又可以提高编译器使用者的开发效率.除了 SIMD 指令外,一些特殊的 DSP 指令,如饱和加法等,也是用内联函数实现的.使用内联函数可快速优化 C 代码.内联函数是直接映射为内联的 YHFT-D4 指令的特殊函数,使用时同调用函数一样调用它.

为了便于对内联函数的增减,我们决定借用 IMPACT 的重定向机制来实现编译器对内联函数的支持.具体工作分为两大步骤:

1) 通过 Hmdes 描述语言对所要支持的 SIMD 指令进行描述,建立内联函数的数据库.

内联函数的 Hmdes 描述文件,主要包含各个节的定义,以及对每条内联函数的描述,如图 7 所示.

```
CREATE SECTION Intrinsics
 REQUIRED function_call (STRING);    /* intrinsics 的函数名 */
 OPTIONAL opcode (STRING);    /* 替代后的汇编指令的名字 */
 OPTIONAL dest_type (LINK(OperandTypes)); /* 目的操作数类型 */
 OPTIONAL src_type (LINK(OperandTypes) * ); /* 源操作数类型 */
 REQUIRED enabled (INT);    /* 是否允许进行 intrinsics 替代 */
{
add4 (function_call(_add4)
     dest_type(Integer)
     src_type(Integer Integer)
     enabled(1));
...
}
```

图 7　内联函数的 Hmdes 描述

2) 在 IMPACT 编译基础设施中从中间代码 Hcode 到 Lcode 的转换过程中,将对应于内联函数的函数调用转换成相应的汇编指令.算法如图 8 所示.

```
ALGORITHM Intrinsify
INPUT Lcode_fn with intrinsics
OUTPUT Lcode_fn with intrinsics substituted
{
  for (each fn_call in Lcode_fn)
  {
    Finding fn_call name in intrinsics database;
    if(unfinded)
      Break;
    if(finded) /* this function is an intrinsic */
    {
      Do some type checking;
      Substitute the intrinsic with an instruction;
      Delete redundant context code in original function call;
    }
  }
}
```

图 8　内联函数转换算法

4.4　分簇寄存器分配

YHFT-D4 VLIW DSP 是一种分簇结构,必须将程序代码尽量分布到两簇上,才能获得较高的 ILP.我们经过对 YHFT-D4 指令的剖析,发现其指令绝大多数是如下形式:

(opcode. unit xsrc1, src2, dst)or (opcode. unit src1, xsrc2, dst)

即执行该条指令的计算单元必须和目的操作数在同一簇年,而源操作数中一般有一个可以从另一簇中读取.只要操作数的所在簇确定了,则执行单元的簇随之确定,因此我们决定在寄存器分配的同时,进行寄存器的分簇分配.基于传统的着色图法寄存器分配方法[14],我们提出了符号寄存器干涉图(interference graph)先进行两簇预着色,然后在簇内进行着色的寄存器分配方法.算法如图 9 所示.

```
ALGORITHM Clustered_Register_Allocation
INPUT lcode function with virtual registers
OUTPUT lcode function with clustered physical registers
{
 R_compute_dataflow_info ();/* 数据流分析 */
 R_compute_live_ranges ();/* 计算寄存器的生命周期 */
 R_register_saving_convention_selection ();/* 选择寄存器的保存方式 */
 R_construct_interference_graph ();/* 构造干涉图 */
 R_cluster_assign ();/* 分簇预着色 */
 R_graph_coloring ();/* 簇内着色 */
 R_virtual_to_machine_conversion ();  /* 转换成物理寄存器 */
 R_insert_spill_fill_code ();/* 插入溢出及加载代码 */
}
```

图 9　分簇寄存器分配算法

对寄存器的干涉图进行两簇预着色,其基本思想是根据其相邻节点的簇的属性,决定本节点的簇属性(指明该寄存器应当分配的哪一簇寄存器文件).基本原则是将干涉图中的节点平均分配到两簇寄存器文件上.然后在查找空闲的物理寄存器时,依据簇属性从相应的寄存器文件中查找.算法描述如下.

```
ALGORITHM R_cluster_assign
INPUT virtual register interference graph
OUTPUT clustered virtual register interference graph
{
  for (each node in interference grapth )
  {
```

```
    oper = Find_def_oper(node);
    if (oper can only be processed in certain unit)
      node.cluster is determined;
    else if (the dst must be the same cluster as the srcx)
      node.cluster = srcx.cluster;
      else if (the dst can be put into either cluster)
      {
       Compute_neighbour_of(node);
       if (cluster A neighbours > cluster B neighbours)
         node.cluster = B;
       else
         node.cluster = A;
      }
  }
  Insert_compensation_code();
}
```

图 10 干涉图两簇预着色算法

上述分簇寄存器分配完毕之后，有可能会有一些指令的源操作数不在正确的簇中，因此寄存器分配之后需要插入补偿代码，校正不正确的簇间分配. 这主要是通过插入簇间的 copy 指令(MV)实现的：在两个寄存器文件中，各保留一个寄存器作为插入的 MV 指令的目的寄存器. 然后依据指令的寄存器的分配情况，就可以设定其功能单元，从而完成了两路分配的问题. 寄存器相对于函数调用而言有 Caller-saved、Callee-saved 两个种类，Caller-saved 寄存器由主叫函数负责保存，而 Callee-saved 寄存器由被调用函数负责保存. 为了保证函数调用的正确性，在寄存器分配完毕之后，需要插入保存 Callee-saved 寄存器的代码. 需要实现三个功能：

1) 在函数的入口和出口处插入对堆栈指针进行增减的指令，以正确得为函数分配堆栈空间；

2) 计算该函数用到的 Callee-saved 寄存器；

3) 函数入口处插入将 Callee-saved 寄存器存入堆栈的指令(Store)，函数出口处插入将 Caller-saved 寄存器从堆栈中恢复的指令(Load).

4.5 指令级并行开发和资源冲突解决

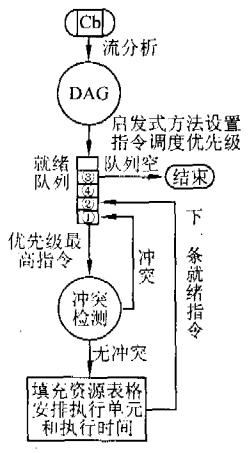
指令级并行开发和资源冲突解决都是在调度阶段完成的(包括前遍、后遍调度). 所谓调度，无非是在不改变程序语义的前提下，合理安排指令执行的顺序，达到提高 ILP，缩短调度长度的目的. 常见的调度方法可分为两大类：线性调度方法(如 ASAP，ALAP)和基于图的调度方法(如 List 调度方法). 线性调度方法简单，执行效率高，但缺乏对全局信息的响应，优化效果不好；而基于图的调度算法是对全局的数据流图进行处理，虽然算法复杂，但优化效果好. 因此选取了 List 调度方法来为 YHFT-D4 开发 II.P. List 调度方法如图 11 所示.

对 List 调度的性能影响最大的是指令调度优先级的设置，因为它是按照优先级对指令进行调度. 对于 DAG 中的节点(代表指令)，我们计算优先权主要采用一些启发式方法，具体讲包括下面一些：

1. 节点的最晚可能调度时间(ALAP). 在 DAG 中，节点的 ALAP 表示节点代表的指令必须在 ALAP 之前调度，否则将影响整个 DAG 的调度效率.

2. 节点到退出节点(叶节点)的距离(Dis). Dis 越大表示节点的优先级越高，这体现了关键路径优先的原则.

3. 执行指令的功能单元数目(Func_num). 能执行节点代表的指令的功能单元数越多，则指令的调度优先权越低.



(a) ListSchedule 过程示意图

```
ALGORITHM  ListSchedule
INPUT  unscheded BB
OUTPUT  scheded BB
{
  /*Create a dep DAG of bb;*/
  BuildDepDAG();

  /*Compute sched_priority of each node;*/
  Compute_sched_priority();

  /*Ready=nodes with no predecessors;*/
  BuildReady();

  /*Loop until ready is empty schedule each
   *node without conflicting;and then update
   *the ready queue;
   */
  Loop
     CheckConflicting();
        ScheduleOper();
        UpdateReady();
    Until Empty(Ready);
}
```

(b) ListSchedule 算法

图 11 List 调度过程和算法

4. 后继节点个数(Succ). 后继节点个数越多，该节点优先权越高. 这种计算方法体现了关键节点优先的原则.

我们 List 调度程序综合考虑上述因素得到下面公式所表示的指令调度优先权. 通过适当的改变这些微调系数，可以灵活的调节各种启发式算法在调度优先级计算中所在的比重. 这对于体验各种启发式算法的效果具有十分重要的意义. 其中 n∈DAG，A，B，C，D 分别为微调系数.

$$Sched\_priority(n) = ALAP(n) * A + Dis(n) * B + Func\_num(n) * C + S$$

资源 \ 时钟	0	1	2	3	4
L1	①	③			
S1			②		
D1					
Crosspath1			②		
LongReadPath1		③			

图 12 资源表格示意图

调度除了进行优化外，还要保证正确性，一是保证程序正确的依赖关系，再一个是保证不会产生资源冲突的情况. 这就

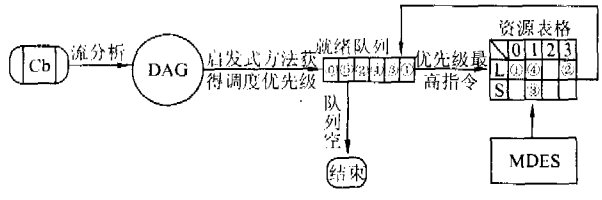


图 13 调度中资源冲突的解决

需要调度过程中能够检测出潜在的资源冲突. 我们使用了资源表格技术来检测资源冲突. 资源表格采用二维结构，它描述

目标机的资源以及各种资源在指令执行过程的占用情况.图12是资源表格的示意图.

指令调度过程中对于每条指令,通过查询机器描述MDES获得每条指令的执行延时和执行功能单元来填充资源表格,可以十分容易的解决指令在执行过程中的避免使用相同资源的限制.图13是使用资源表格解决限制的过程示意图.

5 实验结果及评价

我们选取了一些DSP常用算法核对实现的分簇VLIW DSP编译器的优化效果进行了测试,这些算法包括点积运算prod、滤波器fir、块搬移move、矩阵乘法mm、快速傅立叶变换fft.这些测试程序中,prod、fir和move是较小的测试程序,而mm和fft是较大的测试程序.这里主要就各个测试程序的平均IPC和峰值IPC进行了测试,结果如下.

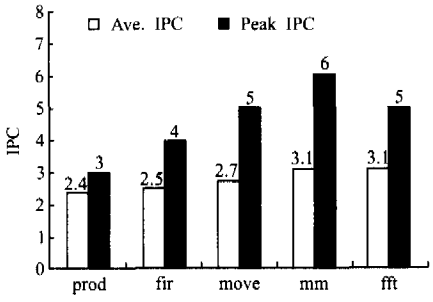


图14 测试程序的IPC

从测试结果中可以看出峰值IPC最高为6(mm),最低为3(prod);平均IPC最高为3.1(fft),最低为2.4(prod).说明该编译器在识别程序中的并行性上,取得了比较理想的效果.另外我们从图x中可以看出对于较大的测试程序,其IPC也较大.这是因为编译器的优化空间变大,有利于在更大范围内开发程序的并行性.

6 结束语

编译器总是为一定的体系结构服务的.对于VLIW体系结构,相对于超标量体系结构,其性能的发挥更加依赖编译器.我们在可重定向编译基础设施IMPACT的基础上,为分簇VLIW DSP设计和实现了编译器,达到了较好的优化效果.应当看到这是在没有进行DSP相关优化的基础上取得的,因此编译器优化的潜力还非常大.目前,性能和功耗成为衡量处理器,特别是嵌入式处理器的两个主要指标,因此将来我们也将从这两方面着手进一步优化VLIW DSP编译器:一是研究DSP相关优化,提高编译器的性能;二是研究软件低功耗编译技术,降低被编译程序的功耗.

References:

[1] Dai G et al. A study of compiler techniques for multiple targets in compiler infrastructure[J]. Journal of Computer Research and Development, 2003, 40(2): 312-317.

[2] Fisher J. Very long instruction word architectures and the ELI-512[C]. Proceedings of the Tenth Annual International Symposium on Computer Architecture, Stockholm, Sweden, 1983, 140-150.

[3] Faraboschi P, Fisher J, Young C. Instruction scheduling for instruction level parallel processors [C]. Proceedings of the IEEE, 2001, 89(11): 1638-1659.

[4] Kim J et al. Experience with a retargetable compiler for a commercial network processor[C]. Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, Grenoble, France, 2002, 178-187.

[5] S Rajagopalan et al. A retargetable VLIW compiler framework for DSPs with instruction level parallelism[J]. IEEE Trans. on Computer-Aided Design, 2001, 20(11): 1319-1328.

[6] Shannon C J. The IMPACT SC140 code generator[D]. MS Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana IL, 2002.

[7] Chakrapani L N et al. Triceps: enhancing the trimaran compiler infrastructure for strongARM code generation [R]. CREST Technical Report: CREST-TR-01-01.

[8] Leupers R. Instruction scheduling for clustered VLIW DSPs [C]. IEEE PACT 2000, 291-300.

[9] Lapinskii V S et al. Cluster assignment for high-performance embedded VLIW processors[J]. ACM Transactions on Design Automation of Electronic Systems, 2002, 7(3): 430-454.

[10] Jang S et al. A code generation framework for VLIW architectures with partitioned register banks[C]. In:Proceedings of the Third International Conference on Massively Parallel Computing Systems (MPCS), 1998, 61-69.

[11] Chang P P et al. IMPACT: an architectural framework for multiple-instruction-issue processors[C]. ISCA 1991, 266-275.

[12] Bringmann R. A template for code generator development using the IMPACT-I C Compile[D]. MS thesis, Department of Computer Science, University of Illinois, Urbana IL, 1992.

[13] Gyllenhaal J C, Rau B R, Hwu W W. Hmdes version 2.0 specification[R]. Technical Report: IMPACT-96-3, The IMPACT Research Group, University of Illinois, Urbana, IL, 1996.

[14] Hank R E. Machine independent register allocation for the IMPACT-I C compiler[D]. MS thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana IL, 1993.

附中文参考文献:

[1] 戴桂兰 等. 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.