

基于 LLVM 的指令并行调度与实现

屈秋雯, 梁利平

(中国科学院 微电子研究所, 北京 100029)

摘 要: IME-Diamond 处理器是一款 VLIW 结构的多核处理器, 具有多个通道, 可并行执行多条指令. 为了充分利用多通道并行执行的特性, IME-Diamond 处理器需要编译器提供并行度信息来实现指令并行. 开源的 LLVM 编译器架构为编译器的移植和优化提供了一个便利的平台. 在此平台上可以配置和插入相应的 Pass 模块对指令进行分析与静态调度, 并根据分析结果插入 Paralink 指令来显式的指示出指令的并行度, 指导硬件并行发射. 实验结果表明, 通过并行调度后, 生成代码的性能提升了 16%~30%.

关键词: LLVM; 编译器; 指令并行; 指令调度

中图分类号: TP314

文献标识码: A

文章编号: 1000-7180(2013)11-0060-04

Instruction Parallel Scheduling and Implementation Based on LLVM

QU Qiu-wen, LIANG Li-ping

(Institute of Microelectronics, Chinese Academy of Sciences, Beijing 100029, China)

Abstract: IME-Diamond is multi-core processor based on VLIW (Very Long Instruction Word) and has multichannel to support instruction parallelism. To make full use of the character of IME-Diamond, compiler needed to provide information of parallelism. Low Level Virtual Machine (LLVM) is an open source compiler infrastructure which provides a convenient platform to transplant and optimize compiler. The Pass module can be inserted to implement instruction analysis and static scheduling. Paralink is a particular instruction in IME-Diamond that can be generated according to the result of analysis. Compared to the code before scheduling, the performances have developed 16%~30%.

Key words: Low Level Virtual Machine (LLVM); compiler; instruction parallel; instruction scheduling

1 引言

超长指令字 (Very Long Instruction Word, VLIW) 结构相对于超标量而言, 芯片设计较为简单, 开发周期较短, 硬件复杂度较低, 被广泛用于数字信号处理 (Digital Signal Processing, DSP) 芯片的设计. VLIW 结构硬件上具有多个逻辑处理单元, 支持多条指令同时执行. 但该结构硬件上不支持指令的动态调度, 需要软件来完成调度工作. 然而一般编译器生成的代码都不带有指令并行的信息, 不能充分利用硬件资源. 为此, IME-Diamond 使用了一条名为 Paralink 的指令来显式的指示指令的并行度, 硬件根据 Paralink 所标识出的并行度来发射指令. 编译器需要完成指令分析调度后, 插入 Paralink 指

令来辅助硬件实现指令并行.

基于数据流的指令调度器能够完成对汇编指令分析并插入 Paralink 指令^[1], 实现指令并行. 这种处理方式对于随机生成指令效果较好^[1], 而对一般编译器生成的指令而言, 由于基本块较小, 基本块内数据相关性较大, 每次都要重新建立数据结构来存储和分析指令的相关信息, 复杂度较高, 且易于出错.

本文基于 LLVM 编译器架构, 充分利用架构所提供的指令相关信息, 根据处理器的特性对程序进行分析与调度, 使编译器可直接生成高效的并行指令.

2 IME-Diamond 处理器系统

2.1 IME-Diamond 处理器

IME-Diamond 处理器是中国科学院微电子研

究所自主研发的一款基于 VLIW 结构的高性能多核 DSP. 该处理器每个核均有八个通道,最多支持八条指令并行执行. 该处理器不仅支持一整套自定义的高效的 DSP 指令集,同时还兼容 MIPS32 4KC 的指令集. 因此该处理器不仅能够作为通用的 CPU,同时在媒体和通信处理上也有很大的优势.

IME-Diamond 处理器硬件上不支持动态调度,需要软件辅助完成指令并行的工作. 在 IME-Diamond 的指令集中设有专门用来指示指令并行度的 Paralink 指令. Paralink 指令指示出了它后面的八个周期的指令并行度,硬件根据 Paralink 所标示的并行度来发射指令,从而实现多指令并行. 考虑到硬件复杂度和执行效率,处理器对可并行执行的指令有一定的约束,需编译器结合实际情况分析出指令的并行度,保证程序能够正确无误的运行.

2.2 IME-Diamond 工具集

为了方便用户高效开发适用于处理器的各种应用程序,IME-Diamond 提供了一套完整从编译、链接到仿真的工具链. 整套工具集包括编译器、汇编器、连接器、二进制反汇编工具、归档管理器以及二进制转换器等众多工具,提供 C 语言、汇编等多级别的编译、静态检查(流水线冲突检测)、优化、仿真等全面支持. 用户可选择使用 IME-Diamond DSP 汇编语言、C 语言来完成程序的开发、调试和仿真.

3 LLVM 编译器架构

LLVM 是美国伊利诺斯大学开发的一款开源的编译器架构^[2]. 它为编译器提供了一系列的高层信息,旨在为程序提供透明的,贯穿整个程序生命过程的分析与转换^[3]. 如图 1 所示,整个编译架构根据编译的三个阶段划分为三个部分:前端、代码优化器和后端^[2]. 前端主要负责程序的语法分析,词法分析以及语义分析等工作,将高级语言编译生成 LLVM 中间代码,同时也做一部分优化工作^[4]. 目前 LLVM 前端已支持 C、C++ 和 Fortran 等高级语言. 后端的主要任务是根据对目标处理器的描述,将 LLVM 中间代码翻译成对应目标机的汇编代码,同时还针对目标机的特性实施一系列优化. 目前后端已能支持常用的处理器汇编指令的生成,例如: X86、ARM、MIPS、Sparc、PowerPC 等. 代码优化器是一个可选部分,它的主要任务是针对中间代码实施各种优化与转换,得到更优的代码. 由于不同的程序和处理器所适用的优化算法各有差异,所以用户可在代码优化器中配置选择合适的优化方案. 同时

允许编译器开发者在此阶段实施自己的优化算法,便于对比调试.

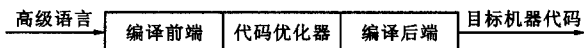


图 1 LLVM 架构编译三阶段

LLVM 最大的特点是整个架构模块清晰,分工明确,为程序处理的各个阶段的优化和分析提供了便利,为编译器的移植与开发减少了复杂度和工作量^[5]. 在架构中所有的处理对象都被抽象成为类,并提供了很多有用的接口来访问相关的信息. 为了最大限度地支持编译架构的可扩展性,LLVM 将程序的处理都分配到多个 Pass 模块中实现. 每个 Pass 模块可以完成一次对程序的分析,也可以完成一次对程序的转换. 模块与模块之间相对独立,同时前面模块的处理信息可以被后面的模块使用. 这样的结构设计,使得整个编译架构层次分明,便于调试和扩展. 如果开发者想针对程序的某个部分进行一些处理,则可在合适的位置插入自己编写的 Pass 模块,编译器会根据用户配置来是选择合适的 Pass 模块对程序进行处理

4 指令并行调度与实现

4.1 问题分析

指令调度是在满足指令间的数据依赖和控制依赖的前提下,调整指令发射顺序,减少指令执行时的等待时间,在尽可能短的时间内执行完程序^[6]. 而本文的指令并行调度在保证并行发射的指令间不存在数据依赖关系的同时,还要考虑到处理器的处理规则,排除某些不能并行执行的情况. 想要编译器生成的代码能够充分利用 IME-Diamond 处理器的特性,就需要编译器对指令做分析与调度,保证程序能够正确的利用处理器并行执行的特性.

解决指令并行调度涉及两个方面:首先需要选择合适的方式来实现指令并行的分析与调度;其次需要设计合适的数据结构来存储数据约束,以便完成指令的并行调度.

在实现方式的选择上,本文采用了在 LLVM 架构中指令发射前插入 Pass 子模块来完成指令的并行调度. 开源的 LLVM 编译器架构具有结构清晰、易于移植、易于扩展等特点,LLVM 中所提供的 Pass 模块更是为指令并行调度提供了便利的条件. 新生成的 Pass 模块不仅可以利用架构中已有的数据结构对指令进行分析,同时还可以利用先前 Pass 模块分析程序所产生的信息辅助指令的并行调度.

利用 LLVM 中的 Pass 模块来实现指令并行调度是较为便捷的方式。

在数据结构的选择上,本文采用 DAG 图(带权有向图),DAG 图不仅能够有效地表示指令之间的依赖关系,还能表示被约束的指令需要等待的时间。判断多条指令是否可并行不仅需要考察指令间是否存在数据相关,同时还需要考虑 IME-Diamond 处理器对并行指令执行的相关规定。因此与一般的指令发射不同,编译器在并行调度指令时还需要检测指令间是否满足 IME-Diamond 处理器对并行指令的要求。由于每次可并行发射的指令随着指令的发射顺序变化而不同,因此不易在 DAG 图上表示出 IME-Diamond 处理器对并行指令的限制。这就需要在指令发射前做检测和处理,若符合并行执行条件,则并行发射。否则分析存在限制的指令,选择合适的指令分次发射,争取获得最大的并行度。

4.2 指令并行调度过程

Paralink 指令指示出了接下来的八个周期的指令并行度,如果中途发生跳转,Paralink 失效。因此 Paralink 只能指示基本块内部的指令并行度。本文调度也是基于基本块内部的调度,即局部调度。且调度发生在代码发射之前,此时寄存器分配已经完成,因此建立 DAG 图时中仅需要考虑数据相关性。根据 DAG 图分析指令,记录每个周期的并行度,最终在合适的位置插入 Paralink 指令。

执行一段指令序列所需的最短时间由它所对应的 DAG 图的关键路径所决定。若每次优先发射与后续节点时延较长的节点,则可以减少指令执行时的等待时间。在 DAG 图中没有前序节点表示它们不依赖任何指令,可以直接发射。因此每个发射周期指令并行度分析的对象为 DAG 图中所有无前序的节点,记为集合 U 。集合 V 用于记录本周期可并行发射的所有指令。主要的步骤如下:

(1) 选出 U 中,与后续节点时延最大的节点 k ,将 k 加入集合 V 中;

(2) 从 $U - V$ 中取出一个节点 s ,若该节点与 V 中的所有元素可并行,则将 s 计入 V 中,否则从 U 中删除节点 s 。

重复(2),直到 $U - V$ 为空;此时 V 中记录了所有可以发射的指令, V 中元素个数即为并行度。发射 V 中的指令,并将 V 中所有节点从 DAG 图中删除,更新 DAG 图,此时便完成一个周期的指令调度。整个调度过程的流程图如图 2 所示。

由于一条 Paralink 只能指示八个周期的指令

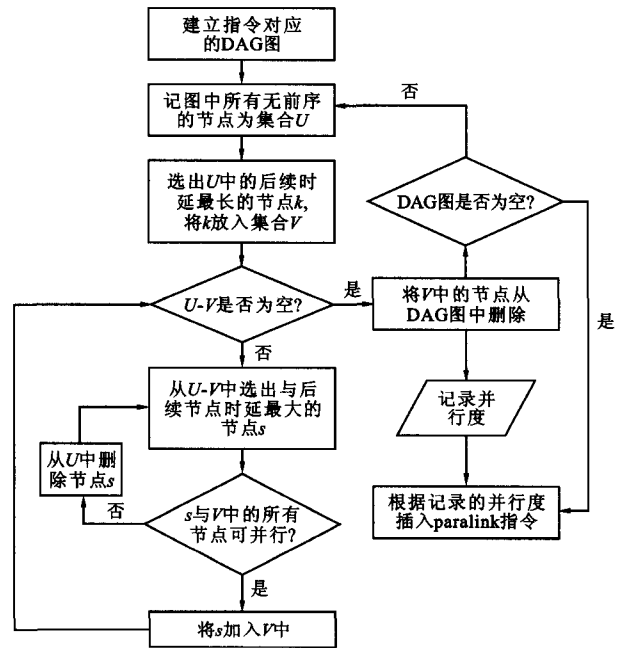


图 2 指令并行调度流程图

并行度,若基本块内的指令在八周期内未发射完毕,则需要插入新的 Paralink 指令来指示剩余指令的并行度。插入的 Paralink 指令会导致程序代码量增大,对于并行度较低的程序可以选择不插入 Paralink 指令,从而减少程序代码。

5 测试结果

为了测试并行调度效果,本文使用了 IME-Diamond 处理器的一套软件工具链。其中主要使用了 IME-Diamond 汇编器、链接器和仿真器。IME-Diamond 汇编器和链接器能够完成对汇编指令的汇编和链接,生成可直接运行于 IME-Diamond DSP 上的可执行文件。IME-Diamond 仿真器完整的模拟了 IME-Diamond DSP 的硬件执行过程,并能够统计出程序执行的跳转次数、指令 cache 和数据 cache 的命中率、程序执行的总周期数等有效信息。

本文采用的测试程序包括常用的离散傅里叶变换(FFT)、字符串搜索算法(StringSearch)、最短路径算法(Dijkstra)和视频解码程序(Decoder)。将程序经过实现了并行调度的 LLVM 编译器进行编译,并用 IME-Diamond 工具链进行汇编、链接和仿真。编译器对程序进行并行调度后,一方面由于使指令并行执行,会带来程序执行效率提升,测试结果如图 3 所示;另一方面,调度过程会插入 Paralink 指令,会导致代码量的增加,测试结果如图 4 所示。

由于本文的调度算法属于局部调度,因此调度效果很大程度上取决于基本块内指令之间的数据相

关程度.从图3可以看出,对程序进行并行调度后,其执行总周期数明显下降,程序的执行效率有16%~30%的提升.图4显示出来的指令总数的变化幅度较小,大约5%左右.总体而言,并行调度能够较好的利用到 IME-Diamond DSP 并行执行的特点,提升程序的执行效率.

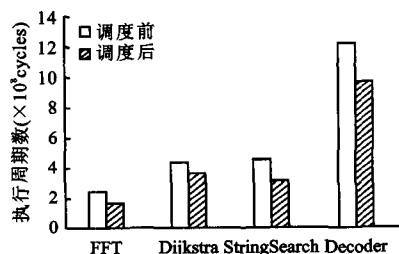


图3 并行调度前后程序执行周期数对比

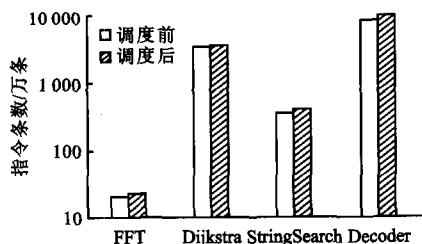


图4 并行调度前后指令数对比

6 结束语

本文基于开源 LLVM 编译器架构,结合 IME-Diamond 处理器的特性,实现了编译器对指令的静态并行调度,为 IME-Diamond DSP 生成可并行的高效代码.测试结果表明这种显式指示并行度的静

态调度方式会带来小幅的指令膨胀,同时程序的执行效率也得到了大幅提升.

参考文献:

- [1] 董锐,王志君,梁利平.基于数据流的指令调度器的设计与实现[J].微电子与计算机,2011,28(11):148-151.
- [2] Chris Lattner. LLVM Home[EB/OL]. [2012-06-20]. <http://www.llvm.org>.
- [3] Chris Lattner, Vikram Adve. LLVM: a compilation framework for lifelong program analysis and transformation[C]//Proceedings of the 2004 International Symposium on Code Generation and Optimization. California, 2004: 20-24.
- [4] Computer Science Department of University of Illinois Urbana Champaign. Clang: AC language family frontend for LLVM[EB/OL]. [2012-06-20]. <http://clang.llvm.org>.
- [5] 董峰. LLVM 编译系统结构分析与后端移植[D]. 上海:上海交通大学,2007.
- [6] John L Hennessy, David A Patterson. Computer architecture a quantitative approach[M]. 4th ed. Beijing: China Machine Press, 2010.

作者简介:

屈秋雯 女,(1988-),硕士研究生.研究方向为编译器的移植与开发.

梁利平 男,(1969-),博士,研究员.研究方向为高性能数字信号处理器、微处理器和高速混合信号集成电路设计技术.

(上接第59页)

作者简介:



耿焕同 男,(1973-),教授,博士生导师.研究方向为计算智能与约束多目标优化、气象资料预处理与资料同化.

冯慧菁 女,(1987-),硕士研究生.研究方向为进化计算、GIS气象应用.

贾婷婷 女,(1989-),硕士研究生.研究方向为进化计算.