

异构多核处理器编译期指令乱序调度研究

王笃庆, 满毅

(北京邮电大学电子工程学院, 北京, 100876)

摘要: 随着摩尔定律的日渐终结, 异构及多核成为处理器研究的主要方向。本文针对使用超长指令字架构的 DSP 处理器, 提出了一种编译期进行指令乱序调度的方法, 并根据此方法设计实现指令调度器。该指令调度器主要由汇编语言解析模块、数据冲突分析模块、指令调度模块组成。汇编语言解析模块对原始的汇编语句进行解析及语法成分拆分, 存储为内建的数据结构, 数据冲突分析模块依据此数据结构建立数据依赖有向无环图 (DAG), 指令调度模块依据 DAG 进行指令调度, 在调度过程中消除了数据依赖、资源冲突, 并实现了指令 VLIW 打包与乱序调度。经过实验测试, 该指令调度器功能完善稳定, 乱序调度执行效率高, 能有效地完成原始汇编到处理器上真实可执行汇编的转换。

关键词: 计算机体系结构; 编译器; VLIW; 静态分析

中图分类号: TP3-0

Research on Compiler-Based Instruction Reordering for Heterogeneous Multi-core Processors

WANG Duqing, MAN Yi

(school of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing)

Abstract: As Moore's Law gradually approaches its end, heterogeneity and multi-core architectures have become the main focus of processor research. In this paper, we propose a method for compiler-based instruction scheduling for DSP processors using Very Long Instruction Word (VLIW) architecture, and design and implement an instruction scheduler based on this method. The instruction scheduler consists primarily of an assembly language parsing module, a data dependency analysis module, and an instruction scheduling module. The assembly language parsing module parses and breaks down the original assembly statements into syntactic components, storing them in a built-in data structure. The data dependency analysis module builds a Directed Acyclic Graph (DAG) based on this data structure to represent data dependencies, while the instruction scheduling module schedules instructions based on the DAG. During scheduling, it eliminates data dependencies and resource conflicts, and achieves VLIW instruction packetization and out-of-order scheduling. Experimental tests demonstrate that the instruction scheduler is feature-complete and stable, with efficient out-of-order scheduling performance, effectively converting original assembly code into executable assembly code for the target processor.

Key words: Computer architecture; Compiler; VLIW (Very Long Instruction Word); Static analysis

0 引言

随着半导体技术的发展和集成电路制造工艺的进步, 现代处理器可以在同一芯片上集成多个不同类型的处理器核心, 包括通用处理器核心、图形处理器核心、神经网络处理器核心等, 从而实现异构多核架构^[1]。处理器转向异构多核方向的原因和意义在于应对日益增长的性能需求和多样化的工作负载, 提高处理器的性能、能效和适应性。随着应用程序复杂性的增加, 传统的单核处理器难以满足性能需求, 而采用异构多核架构可以通过增加

作者简介: 王笃庆(1999-), 男, 硕士研究生, 主要研究方向: 计算机体系结构

通信联系人: 满毅 (1974-), 男, 副教授, 硕士生导师, 主要研究方向: 新一代电子信息技术. E-mail: manyi@bupt.edu.cn

处理器核心的数量和类型来提高性能。同时，现代应用程序的工作负载越来越多样化，传统的单核处理器难以同时满足不同类型任务的需求，而采用异构多核架构可以根据任务特性选择合适的处理器核心，提高效率。此外，异构多核架构可以更好地利用硬件资源，提高处理器的利用率和性能。

异构多核体系结构有多种实现形式，超长指令字（Very long Instruction Word，VLIW）就是其中一种，其广泛应用在数字信号处理（Digital Signal Processing，DSP）芯片及人工智能（Artificial Intelligence，AI）芯片中作为其基础架构。在该类型架构中，为了实现大规模的运算单元以及功耗的控制，会将很多传统芯片中硬件负责的功能简化或直接转移到编译器中实现。编译器会变得更加复杂的同时，会有更多机会在软件层面对代码进行更深度的优化。

指令调度就是一个重要的优化阶段，可以重新排序指令的发射顺序，提高指令执行的并行度，降低程序执行所需要的总时长^[2]。相关技术中，采用表调度算法来实现指令调度优化。因此设计指令调度系统完成原始汇编指令到异构多核处理器上真实可执行的汇编序列的转换，该系统在编译期通过插 NOP 的方式消除了指令数据依赖与资源冒险^[3]，在实现指令乱序调度的同时实现了 VLIW 打包。

1 系统架构

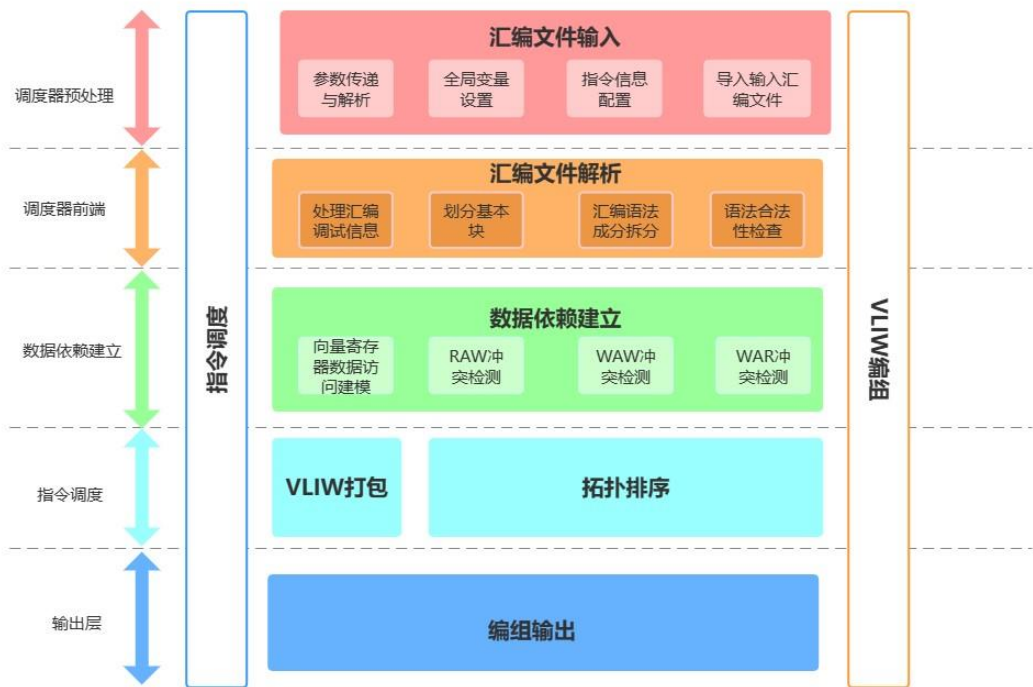


图 1 异构多核处理器的指令编排系统架构

如图 1 所示为异构多核处理器的指令编排系统的整体架构，主要由输入模块、指令解析

模块、指令依赖图建立模块、编排模块和输出模块组成，其中输入模块处理外部文件输入，解析处理外部传入参数，同时对一些在程序完成后需要打印的全局变量进行初始化设置；指令解析模块对汇编指令进行解析，并拆分操作码、操作数等各语法成分，验证其指令合法性。指令依赖图建立模块对解析好的汇编序列分析其数据依赖，建立数据依赖 DAG^[4]。编排模块对数据依赖图进行遍历调度，输出真实芯片可执行的汇编序列。输出模块对编组好的汇编序列进行输出。

2 指令依赖模型

指令之间的依赖关系在计算机体系结构中非常重要，它决定了指令的执行顺序和结果的正确性。根据依赖关系的不同，可以将指令之间的依赖分为以下几种类型：

数据依赖(Data Dependency)：指令之间的数据依赖关系，分为三种类型：读后写(RAW)、写后读(WAR)和写后写(WAW)。具体而言，如果一条指令在执行时需要依赖于另一条指令产生的数据结果，则称为数据依赖。

控制依赖(Control Dependency)：指令之间的控制依赖关系，表示某些指令的执行取决于条件分支语句的执行结果。在条件分支语句中，如果条件成立，则执行某些指令序列；如果条件不成立，则执行另外一些指令序列。在这种情况下，被条件分支影响的指令就存在控制依赖关系。

资源依赖(Resource Dependency)：指令之间的资源依赖关系，表示某些指令需要访问相同的资源，但是这些资源不能同时被多条指令访问，需要通过互斥机制来保证资源的正确使用。

这些依赖关系对于程序的正确性、性能和并行执行都具有重要的影响，因此在设计指令集架构和优化编译器时需要考虑和处理这些依赖关系。

2.1 数据依赖分类

数据依赖是指一个指令在执行时需要依赖于另一个指令产生的数据结果。在程序中，如果某条指令需要使用另一条指令产生的数据作为输入，那么这两条指令之间就存在数据依赖关系。数据依赖分为三种类型：读后写(Read After Write, RAW)、写后读(Write After Read, WAR)和写后写(Write After Write, WAW)。

读后写(RAW)依赖：在程序中，如果某条指令读取了另一条指令尚未写入的数据，则称为读后写依赖。也就是说，第一条指令读取了第二条指令尚未写入的数据，而第二条指令的写操作对第一条指令的读操作产生了依赖。

写后读(WAR)依赖：在程序中，如果某条指令写入了一个数据，而另一条指令在这之后读取了这个数据，则称为写后读依赖。也就是说，第一条指令的写操作对第二条指令的读操作产生了依赖。

写后写(WAW)依赖：在程序中，如果两条指令都对同一位置的数据进行写操作，并且这两条指令之间存在顺序关系，则称为写后写依赖^[5]。也就是说，第二条指令的写操作对第一条指令的写操作产生了依赖。

数据依赖是程序并行化和优化的重要考虑因素之一。在并行执行时，需要考虑依赖关系，

以避免不正确的结果。

2.2 基本块划分

105 建立数据依赖是一个复杂的过程，涉及到程序分析、图论和优化[6]等多个领域。首先，建立数据依赖图需要先划分基本块。基本块是程序中的一段代码，它满足以下条件：

- 1) 有一个入口（首指令）和一个出口（尾指令）。
- 2) 从入口到出口的路径是线性的，即没有分支和跳转。
- 3) 入口只有一个，出口只有一个。

110 基本块的划分可以通过以下三种方式进行：

1) 控制流图（CFG）： 构建控制流图是基本块划分的常见方法。在控制流图中，每个节点代表一个基本块，每条边代表控制流的方向。基本块的入口节点就是控制流图中的首节点，出口节点就是尾节点。控制流图可以通过对程序进行遍历，找到分支和跳转语句，从而划分基本块。

115 2) 语法分析： 在语法分析阶段，编译器通常会生成语法树（AST）。通过遍历语法树，可以确定基本块的起始和结束位置。语法树的叶子节点通常对应于基本块的起始和结束。

3) 代码扫描： 编译器可以通过扫描源代码来划分基本块。在这种情况下，基本块的划分通常以控制流语句（如分支和跳转）为标志。

120 本指令编排系统采用第三种代码扫描的方式来实现基本块的划分。

2.3 建立数据依赖图

在基本块划分的基础上，编译器可以在基本块内建立数据依赖图。这个图的节点表示程序中的操作，边表示数据的依赖关系。数据依赖图中的边通常有以下类型：

125 1.真依赖边： 如果一个操作的输出是另一个操作的输入，并且它们在程序中的执行顺序是相邻的，就存在真依赖。

2.反依赖边： 如果一个操作在程序中的执行顺序在后，但在数据流上依赖于另一个操作的输出，就存在反依赖。

3.输出依赖边： 如果两个操作试图写入相同的变量，就存在输出依赖。

130 建立数据依赖图是为了更好地理解程序中的数据流动，并为后续的指令调度编排提供基础。这一过程需要解决诸多复杂的问题，包括对程序结构的准确建模和对不同数据流关系的精确分析。

对数据依赖图进行建模，使其不仅表示了数据依赖关系，同时还表示了依赖的最小间隔时间，对后续更精细的指令调度提供了可能。

135 对此数据依赖图进行建模时，由于在在编译期未作别名分析（即便有别名分析也不能完全解决）的情况下，无法了解访存指令的寄存器间接寻址所使用的具体内存地址，因此对访存指令均认为存在数据依赖。

数据依赖图的设计如下：

1.表明数据依赖的部分：RAW（真依赖）、WAR（反依赖）、WAW（输出依赖）分别存储

2.数据依赖的间隔时间部分：按照指令信息提供的信息，每条指令都会提供：读寄存器时钟周期偏移；写寄存器时钟周期偏移；读内存时钟周期偏移；写内存时钟周期偏移。根据依赖类型和寄存器或存储器的区别，用前后指令的上述偏移可计算出依赖的最小间隔时间。

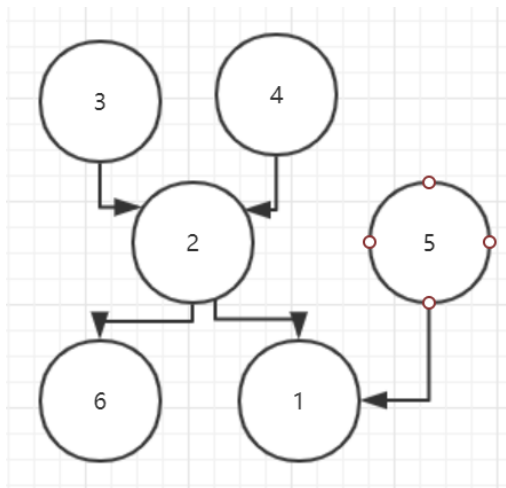


图2 数据依赖有向无环图

3 指令编组算法

指令编排是 VLIW 打包调度器的核心功能，要实现从用户产生的逻辑上顺序执行的汇编指令序列到实际处理器上可正确执行的 VLIW 指令序列的转换。由于访存指令只有在运行其才能知道其实际访存位置，因此访存指令不能并发运行，按照存在数据依赖，按原有顺序并保持最小间隔。VLIW 指令共有 3 个 slot，编组方式为：LSE||SPE||VPE。一个 VLIW 指令中的子指令之间不应有数据依赖^[6]。

由上述的依赖建立过程，每条语句都可能依赖于其余语句，也可能被其余语句依赖。

对于某条语句，我们把别的语句依赖于它，称为它的入度，如果它依赖于其余语句，我们称它具有出度。0 入度 0 出度的语句即可在任何时间输出，基本块入口一定存在 n 入 0 出的语句，基本块出口一定存在 0 入 n 出的语句。生成 VLIW 语句的过程即为对入度出度进行处理的过程。同时考虑到 VLIW 的三个槽位（LSE||VPE||SPE）及流水线级数、旁路及指令 stall 资源冲突等问题。

指令编组算法是列表调度（List Scheduling）算法的变体，将待调度的指令放置在一个待调度指令列表中，并根据一定的启发式规则来确定每条指令的调度顺序。算法包括以下几个步骤：

初始化：将所有待调度的指令放入一个待调度指令列表中，并初始化一些数据结构，如

调度图、时间表等。

选择：从待调度指令列表中选择一条可以调度的指令，通常选择满足调度条件且具有最高优先级的指令。

调度：将选定的指令插入到时间表中的合适位置^[7]，考虑到硬件资源的约束和依赖关系，使得指令之间不存在资源冲突和数据依赖。

更新：根据调度后的情况更新待调度指令列表和相关数据结构，如更新指令的调度状态、更新资源占用情况等。

终止条件：重复步骤 2 至步骤 4，直到所有指令都被调度^[8]完成。

4 系统验证

4.1 正确性验证

对此汇编指令重排器设计系统进行验证，首先验证其正确性，通过测试手段保证。

1) 指令级功能验证。

编排器用于支持指令的功能性验证，通过执行单条指令后插入足够多的 NOP，后检查寄存器及内存状态，可用于支持硬件设计的功能性验证。

2) 随机指令测试

用随机生成器随机生成无意义指令序列，以测试程序健壮性，测试级别达到百万指令级。随机指令仅测试 SPE 与 VPE，load/store 指令无法进行随机测试。

4.2 性能验证

同一程序在不同优化级别下的最终执行时间，是衡量优化性能的最主要因素。

1) 编写典型数据冲突下的标准输出模板，进行编译数据文件与标准文件的时钟数目对比。可验证在程序功能正确的前提下，是否存在冗余时钟。

2) 通过对比同一程序是否在更高级别的优化中时钟数减少，可验证高级优化级别的性能。

以上验证均通过 python 或 shell 脚本调用仿真器自动化验证完成，仿真器与处理器硬件设计的一致性由其余验证工具保证。

性能验证结果：

性能验证结果分两部分，第一部分为传统的通信常用算法，如快速傅里叶变换、LU 分解、QR 分解、svd 分解等等。

对这些算法所对应的汇编文件进行指令重排与调度，在目前应用场景下统计其性能优化情况。

表 1 通信算法在指令编排系统时钟数

算法	汇编指令条数	指令序列时钟	指令编排器时钟	性能优化比例
det2-complex	4	16	16	0.00%
det4-real	126	498	423	9.44%
fft16	48	221	89	52.49%
givens2x2	71	264	97	59.85%
givens4x4dyn	309	1285	873	23.35%
householder2x2	126	493	420	11.76%
householder4x4dyn	466	2122	1193	40.43%
ifft16	50	231	101	52.81%
LU-single-2x2	43	159	131	9.43%
schmidt2x2	59	231	186	10.82%
schmidt2x4dyn	56	257	169	33.07%
svd_2x2_mul	169	819	453	40.90%

5 结论

本文给出了一种在编译期进行指令乱序调度的方法，并对此方法的调度结果进行大量测试，充分验证指令调度器的功能与性能。对于指令进行充分验证，仿真运行百万条指令、历经千万级 cycle。调度器保持了程序调度的运行正确性，同时乱序执行带来的效果提升统计平均值约为 20%。

[参考文献] (References)

[1] YanGui-hai,Lu Wen-yan,Li Xiao-Wei,etel. Comparative study of the domain speicific processors [J]. Scientia SinicaInformations, 2022, 52(2):358-375

[2] GiesemannFlorian,GerlachLukas,PayaVayaGuillermo.Evolutionary algorithms for instruction sceeduling, operatuin merging, and register allocation in VLIW compilers [j]. Journal of Signal Processing Systems, 2020 ,92(7):655-678.

[3] Schirmer,Oskar.NOP - A Simple Experimental Processor for Parallel Deployment[J].2016.

[4]Xiao Z. Optimizing pipeline for a RISC processor with multimedia extension ISA[J]. Journal of Zhejiang University Science A,2006,7(2):269-274.

[4] Architecture Design of a Variable Length Instruction Set VLIW DSP[J]. 沈钲;何虎;杨旭;贾迪;孙义和.Tsinghua Science and Technology,2009(05)

[5] Rainer Leupers, "Instruction Scheduling for Clustered VLIW DSPs", International Conference on Parallel Architectures and Compilation Techniques Proceedings, 2000.

[6] C. Lee, J. Lee and T. Hwang, "Compiler Optimization on Instruction Scheduling for Low Power", Proceedings of the 13th international Symposium on System Synthesis, pp. 55-60, 2000.

[7] O. Wolfe and J. Bier, "TigerSharc Sinks Teeth Into VLIW", Microprocessor Report, vol. 12, no. 16, Dec. 1998.

[8] A. Aleta, J. Codina, J. Sanchez and A. Gonzalez, "Graph-Partitioning based Instruction Scheduling for Clustered Processors", 34th Annual International Symposium on Microarchitecture (MICRO'01), pp. 150, 2001.