软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007357] [CSTR: 32375.14.jos.007357]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 RISC-V VLIW 架构的混合指令调度算法 *

李奕瑾[1,2,3], 杜绍敏[1,3], 赵家程[1,2,3], 王雪莹[4], 查永权[1,2,3], 崔慧敏[1,2,3]

[1](中国科学院计算技术研究所, 北京 100190)

[2](中国科学院大学, 北京 100190)

[3](处理器芯片全国重点实验室(中国科学院计算技术研究所), 北京 100190)

[4](北京邮电大学, 北京 100088)

通讯作者: 赵家程 E-mail: zhaojiacheng@ict.ac.cn

摘 要: 指令级并行是处理器体系结构研究的经典难题。VLIW 架构是数字信号处理器领域中提升指令级并行的一种常用架构。VLIW 架构的指令发射顺序是由编译器决定的，因此其指令级并行的性能强依赖于编译器的指令调度。为了探索 RISC-V VLIW 架构的扩展潜力，丰富 RISC-V 生态，本文研究 RISC-V VLIW 架构的指令调度算法优化。针对单个调度区域，整数线性规划调度算法能够得到调度最优解但复杂度较高，表调度算法复杂度较低但无法得到调度最优解。为了结合两种调度算法的优点，本文提出了一种 IPC 理论模型指导的混合指令调度算法，即通过 IPC 理论模型定位到表调度未达最优解的调度区域，再对该调度区域进一步实施整数线性规划调度算法。该理论模型基于数据流分析技术协同考虑指令依赖和硬件资源，能够以线性复杂度给出 IPC 的理论上界。混合调度的核心在于 IPC 理论模型的准确性，本文理论模型准确率为 95.74%。在给定的测评基准上，本文提出的理论模型应用于混合指令调度时，能够平均认定 94.62%的调度区域在表调度下已达最优解，因此仅有 5.38%的调度区域需再进行整数线性规划调度。该混合调度算法能够以接近表调度的复杂度达到整数线性规划调度的调度效果。

关键词: RISC-V；VLIW；整数线性规划；表调度；理论模型

中图法分类号: TP314

中文引用格式: 李奕瑾, 杜绍敏, 赵家程, 王雪莹，查永权，崔慧敏. 基于 RISC-V VLIW 架构的混合指令调度算法.软件学报. http://www.jos.org.cn/1000-9825/7357.htm

英文引用格式: Li YJ, Du SM, Zhao JC, Wang XY, Zha YQ, Cui HM. Hybrid Instruction Scheduling Algorithm for RISC-V VLIW Architecture. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7357.htm

Hybrid Instruction Scheduling Algorithm for RISC-V VLIW Architecture

LI Yi-Jin[1,2,3], DU Shao-Min[1,3], ZHAO Jia-Cheng[1,2,3], WANG Xue-Ying[4], ZHA Yong-Quan[1,2,3], CUI Hui-Min[1,2,3]

[1](Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100190, China)

[3](State Key Lab of Processors(Institute of Computing Technology, Chinese Academy of Sciences), Beijing 100190, China)

[4](Beijing University of Posts and Telecommunications, Beijing 100088, China)

Abstract: Instruction level parallelism is a classic problem in the research of processor architecture. VLIW architecture is a common architecture to enhance Instruction level parallelism in the field of digital signal processor. The instruction issue order is determined by the compiler for VLIW architecture, so VLIW's Instruction level parallelism performance strongly depends on the instruction scheduling of compiler. In order to explore the performance poteneial of RISC-V VLIW architecture and enrich the RISC-V ecosystem, this paper

* 基金项目: 新一代人工智能国家科技重大专项(2021ZD0110504); 国家自然科学基金(U23B2020, 62090024, 62302479); 中国科学院计算技术研究所创新课题(E361010, E261110)

收稿时间: 2024-08-23; 修改时间: 2024-10-15, 2024-11-20; 采用时间: 2024-11-26; jos 在线出版时间: 2024-12-10

studies the optimization of instruction scheduling algorithm of RISC-V VLIW architecture. For a single scheduling region, the integer linear programming scheduling can obtain the optimal scheduling solution with high complexity, and the list scheduling, which has low complexity, cannot obtain the optimal scheduling solution. In order to combine the advantages of the two scheduling algorithms, this paper proposes an IPC theoretical model guided hybrid instruction scheduling algorithm. The scheduling region where the list scheduling has not reached the optimal solution can be located with IPC theoretical model, and then the integer linear programming scheduling algorithm further processes the located scheduling region. The theoretical model is based on data flow analysis and considers both instruction dependency and hardware resources, and can give the theoretical upper bound of IPC in linear complexity. The core of hybrid scheduling lies in the accuracy of IPC theoretical model, which is 95.74% in this paper. On the given benchmark, the IPC theoretical model can identify that 94.62% of the scheduling region has reached the optimal solution under list scheduling, so only 5.38% of the scheduling region needs to be further scheduled by integer linear programming. The hybrid scheduling algorithm can achieve the scheduling effect of integer linear programming scheduling with the complexity close to that of list scheduling.

Key words: RISC-V；VLIW；ILP；list schedule；theoretical model

1 绪论

RISC-V[1]是一种开放的、基于精简指令集(RISC)的架构，其设计目标是提供一种灵活、可扩展且高性能的计算机体系结构，适用于各种应用场景。近年来，RISC-V 发展迅速，在通用 CPU[2-5]、开源 GPU[6]等多个领域获得成功的应用。目前，超长指令字(Very Long Instruction Word，VLIW)[7]架构在数字信号处理器(Digital Signal Processor，DSP)领域有着广泛的应用，如高通 Hexagon[8]、德州仪器[9]、魂芯[10] 、Kalray[11]等。其中高通 Hexagon 芯片可应用于图像处理、物联网等领域，Kalray K200 数据加速卡可用于高性能计算。与通用领域相比，DSP 芯片需要较低的功耗和较高的处理速度，VLIW 作为一种无需硬件调度器的多核架构，能够在保持低功耗的同时得到较高的处理速度。因此，研究 RISC-V 的 VLIW 架构能够扩展 RISC-V 的使用场景，促进 RISC-V 在 DSP 领域，如移动设备、物联网设备、通信基站等嵌入式系统的发展，提高 RISC-V 处理器的可用性及覆盖面，促进 RISC-V 的生态建设。然而由于 RISC-V 的研究还在发展中，目前少有使用 RISC-V 指令集进行 VLIW 架构处理器实现的工作。工作[12]虽然提出了基于 RISC-V 的 256 比特 VLIW 架构，但是该工作关注硬件设计且是定长 VLIW，并未对 RISC-V 指令集的编译工具链及指令调度相关问题进行研究。定长 VLIW 需要插入冗余空指令，会造成代码膨胀。代码膨胀会导致内存占用增加，造成存储空间和计算资源的浪费，降低设备的整体效能。同时可能会导致指令缓存的命中率下降，进而影响性能。而变长 VLIW 通过指令编码显式标记指令包的结束，无需插入空指令，能够显著降低代码体积，从而避免存储空间浪费，增加指令缓存命中率，提升整体性能。

VLIW 架构拥有多个执行单元，如何通过指令调度有效利用 VLIW 的指令级并行能力是发挥 VLIW 架构性能的关键。指令调度必须考虑数据依赖性及硬件资源限制，以最小化指令序列执行所需的 cycle。指令调度后，指令执行顺序重新排列，无依赖关系的指令被分组打包在一起发射执行，同一个指令包(也称指令 bundle)内的指令并行执行，指令的发射和执行均以指令包为单位。对于 VLIW 处理器，一般采用表调度方法结合启发式规则的方式进行指令调度。

为了探索 RISC-V 在 VLIW 架构的扩展潜力，本文研究该架构的指令调度算法优化。为了给该研究提供基础，本文提出了基于 RISC-V 的变长 VLIW 架构，并为此架构实现了相应的工具链及模拟器。指令调度的效果与调度区域的扩展和单个调度区域的调度相关，本文专注于研究单个调度区域的指令调度。常用的指令调度算法有整数线性规划调度(简称为规划调度)和表调度。这两种调度算法的优缺点如下：针对单个调度区域，规划调度能够得到最优解但复杂度为 $O(2^n)$，表调度复杂度为 $O(n^2)$ 但无法得到最优解。本文期望结合两种调度算法的优点，设计一种新的调度算法，使得针对单个调度区域，既能确保得到最优解，又能保证复杂度可接受。假设能够判断一个调度区域的表调度结果是否为最优解，对是最优解的调度区域采用表调度结果，

对不是最优解的调度区域再进行规划调度，则可以降低计算最优解的复杂度。由于调度的目标为最大化每周期指令(Instructions Per Cycle，IPC)，本文拟用 IPC 判断调度解是否达最优。因此，为了以较低的复杂度得到单个调度区域的最优解，本文提出了一种 IPC 理论模型指导的混合指令调度算法。该 IPC 理论模型基于数据流分析技术协同考虑指令依赖和硬件资源，能够以 $O(n)$ 复杂度给出 IPC 的理论上界。混合调度首先对调度区域进行 IPC 理论模型分析和表调度，将表调度结果的 IPC 与 IPC 理论模型所得 IPC 不一致的调度区域进一步实施规划调度。混合调度通过 IPC 理论模型的指导，能够以接近表调度的复杂度达到调度最优解。

为了实现混合调度，本文设计了该架构适配的规划调度和表调度算法。现有的规划调度算法无法直接用于该架构的原因在于 VLIW 架构多用于 DSP 领域，其执行单元多为功能受限的，即单个执行单元只能执行特定类型的指令，而不能执行所有类型指令。现有的表调度算法的启发式规则仅考虑了指令依赖关系(如入度、出度等)，而没有考虑硬件资源限制。本文首先提出了 RISC-V VLIW 架构的整数线性规划调度算法(简称为规划调度算法)和表调度算法。在规划调度方面，本文为该架构设计了新的约束求解条件和最优化目标。在表调度方面，本文从该架构的硬件资源限制出发，提出了硬件资源适配的启发式规则，提升表调度算法的性能。

本文的主要贡献如下：

(一)设计了基于 RISC-V 的变长 VLIW 扩展，实现了相应的指令编码、工具链和模拟器支持。

(二)针对 RISC-V VLIW 架构提出了规划调度算法，并基于表调度算法提出了该 VLIW 架构适配的启发式规则。

(三)针对规划调度复杂度过高和表调度无法得到最优解的问题，提出了 IPC 理论模型指导的混合指令调度算法。混合调度通过 IPC 理论模型的指导，能够以接近表调度的复杂度达到规划调度的调度效果。

本文结构如下：本章为绪论，对本文的研究内容进行概述；第二章对相关工作介绍；第三章对基于 RISC-V 的 VLIW 架构进行介绍，包括整体架构设计和指令集 VLIW 扩展；第四章介绍规划调度算法及表调度算法的启发式规则优化；第五章介绍 IPC 理论模型指导的混合指令调度算法；第六章介绍实验环境及分析实验结果。

2 相关工作

2.1 VLIW指令调度

指令调度问题本身是一个复杂的问题，已经证明在任意约束下自动找到代码序列的最优调度结果是一个 NP 完全问题[13]，要确定最佳调度方案，必须检查所有合法的调度。只有在确定 1 个 cycle 的指令延迟时，问题才能在小于指数的时间中得到最优解[14]。指令调度分为编译时调度和运行时动态调度。编译时调度偏重于静态分析和优化，而运行时调度则通过动态决策来适应实际执行情况。一般来说，运行时调度能够获取当前的执行环境和资源现状，如数据的实时可用性等，因此相比于编译时能够进行更优的调度。但运行时调度需要硬件调度器的支持。因此，VLIW 架构只进行编译时调度。对于 VLIW 处理器，由于指令调度问题本身的复杂性，在编译器中一般采用表调度[15]方法进行简化求解，并添加启发式规则对调度进行优化。目前指令调度方式根据调度方法是否跨越基本块边界，可分为局部调度和全局调度两类；根据是否在寄存器分配之前进行调度，可分为前遍调度和后遍调度两类[16]。对于算法决策应用的启发式规则，一般需要考虑关键路径、依赖关系。除使用表调度方法外，也可采用数学建模和动态规划[17]等方法，但是由于其复杂度太高无法直接用于编译器中。

局部调度多以基本块作为调度单元，但由于分支指令的存在基本块往往较小，难以实现足够的指令级并行性来充分利用 VLIW 处理器的多个执行单元。因此全局调度尝试将多个执行效率高的基本块合并成一个更大的调度单元或进行跨区域调度，例如轨迹调度[18,19]、超块(Superblock[20]/Hyperblock[21])调度等。该类算法可以扩大指令调度空间，但是会使控制流更加复杂，特别是当存在复杂的控制流结构时会增加分支预测错误的风险，增加编译器的处理时间和资源消耗。由于全局调度专注于通过某些编译技术跨越或合并调度区域，对于单个调度区域的调度算法仍然是局部调度算法。因此，本文专注于局部调度研究，其与全局调度的跨区域调度技术是正交的。

指令调度多以表调度算法作为基础算法，包括表调度算法的各种变体，变体的核心在于不同的启发式规则设计。启发式规则包括静态启发式和动态启发式两类，静态启发式即根据程序依赖特征和硬件架构特征预先定义启发式规则，动态启发式是在调度过程中调整启发式规则。常用的动态启发式包括基于遗传算法[22]、进化算法[23]、图神经网络[24]的调度等。基于遗传算法的调度使用遗传算法来调整表调度中的启发式规则，允许算法根据特定输入程序进行动态适应，从而优化调度效果。基于进化算法的调度综合考虑代码合并、指令调度和寄存器分配，根据输出程序和目标硬件配置进行动态启发式调整。基于图神经网络的调度重点考虑寄存器压力下的指令调度，训练了一个图神经网络在指令调度过程中降低寄存器压力，避免寄存器溢出。同时也有一些工作[25-28]专注于指令调度算法的形式化验证。

不同的调度算法各有其适用的场景。在编译时间敏感的场景下，多用表调度算法以满足编译时间的限制。在程序规模不大的情况下，如果追求对程序指令级并行的极致优化，则可以使用规划调度。规划调度的编译时间开销非常高昂，远非一个产品级编译器所能接受。当需要综合考虑多个优化目标，如寄存器分配和指令调度时，可以运用遗传算法、图神经网络等人工智能的方法来解决。但是人工智能的方法属于黑盒，未必能达到全局最优，且需要收集样本数据提前训练。并且人工智能的方法属于指令集敏感的，对于新的硬件架构需要重新训练。与表调度和各种迭代算法相比，本研究能够确保达到单调度区域内的最优调度。并且本研究可以扩展到复杂的调度算法，如迭代调度，图神经网络调度等。

为了提高处理器的并行处理能力，同时简化编译器的优化任务，出现了一类簇结构 VLIW 处理器，该类处理器将多个执行单元划分成若干簇，一个簇由多个执行单元组成，每个簇都有自己的指令存储器和数据存储器。每个簇可以独立地执行一组指令，并且可以在一个时钟周期内执行多个簇。编译器只需要将指令打包成簇，而不需要考虑执行单元级别的细节。针对簇结构 VLIW 架构的调度算法有二维力量引导算法[29]、PCC 算法[30]、CAeSaR 算法[31]。二维力量引导算法以时钟周期和簇作为二维向量，采取先分簇后调度的方式，但仍然存在无法全局考虑簇间相互作用的问题；PCC 算法可以考虑簇间相互作用来提升整体调度效果，但是 PCC 算法无法准确找出时间代价函数迅速下降的方向；CAeSaR 算法需要综合考虑调度问题与簇分配和簇间通信问题，优化求解困难。

此外，部分工作还通过调整指令集和硬件架构来优化指令调度的整体性能，如谓词执行[32]、硬件循环[33]等。添加谓词可以对基本块进行合并，增大调度空间，但是会增加编译器中数据流关系的复杂性。硬件循环通过添加硬件循环指令和控制状态寄存器，将部分普通循环转化为硬件循环，减少循环判断和分支转移来提升 IPC 从而优化调度效果，但是却增加了编译器生成代码的难度。

2.2 RISC-V指令集、工具链及评测基准

RISC-V 指令集具有模块化和易扩展的特点，按照指令功能的不同划分为多个扩展模块，包括基础指令集 RV32I、乘除指令集扩展 M、浮点运算扩展 F 和 D 等。RV32I(也称基础整数指令集)是 RISC-V 指令集中最基础的指令集，它可以单独在核心上运行完整的软件栈。M 在基础指令集上增加了整数乘、除和余数指令。F 和 D 是浮点型扩展，两者分别增加了单精度和双精度浮点数类型的支持。

与现有的 VLIW 架构相比，RISC-V 的 VLIW 具有指令集开源且通用、工具链完善、模块化和易于扩展升级等优点。例如，高通 Hexagon 芯片是 VLIW 架构，是闭源且专用于 DSP 领域的指令集，不具有通用性，需要单独为其做编译器后端。RSIC-V 的 VLIW 扩展基于开源 RISC-V 指令集，具有通用性，能够复用已有的 RISC-V 相关的软件基础设施，如编译器中指令选择、后端优化等，极大降低硬件扩展难度和软件工具链开发难度，从而降低芯片设计周期和成本。

目前主流的 RISC-V 工具链有两个，分别是 RISC-V-GNU-toolchian[34]和 LLVM[35]。RISC-V-GNU-toolchain 是一个由 RISC-V 国际开放组织协作社区维护的项目，旨在为 RISC-V 处理器提供一套完整的开发环境，它基于 GNU 工具集，并针对 RISC-V 指令集进行了优化，支持多种操作系统平台。该项目提供了构建、编译和调试 RISC-V 架构处理器上的软件所需的全套工具，包括 GCC 编译器、GDB 调试器以及 Binutils 等，但是 RISC-V-GNU-toolchain 本身并无对 VLIW 功能的支持。LLVM 是一个用于构造编译器的基础框架，编程人员

可以利用该基础框架，构建一个包括编译时、链接时、运行时等功能的编译工具链。LLVM 对多种后端提供支持，包括 x86、ARM、RISC-V、Hexagon 等，且 LLVM 提供了对 VLIW 架构的支持，如高通的 Hexagon VLIW 后端，这也为在 LLVM RISC-V 后端增加 VLIW 架构的支持提供了参考。

Spike[36]是 RISC-V 官方提供的一款开源的基于 C/C++开发的 RISC-V ISA 模拟器，实现了 RISC-V 完整的功能模型。相比于其他 RISC-V 模拟器(如 QEMU[37]和 Gem5[38]等)，Spike 的模拟速度和易扩展性最好，方便进行快速的原型迭代开发。本文专注于 VLIW 指令调度的研究，对于内存等全系统的模拟关注较少，同时，现有的模拟器都不支持 VLIW 的性能模拟，需要进行进一步的开发。因此，Spike 更适合作为本文模拟器，并进一步拓展对 VLIW 功能的支持。

Coremark[39]是由 EEMBC 于 2009 年提出的一项基准测试程序，其代码使用 C 语言编写，包含以下的算法实现：列表处理、矩阵操作、状态机和 CRC 校验等。Coremark 在嵌入式 CPU 行业中作为普遍公认的性能测试指标，拥有体积小、方便移植、易于理解、免费并且显示单个数字基准分数的优点，同时 Coremark 具有特定的运行和报告规则，从而可以避免由于所使用的编译库不同而导致的测试结果难以比较的情况。

3 基于 RISC-V 的 VLIW 架构设计及实现

根据每个指令包的并行指令数是固定还是可变的，可以将 VLIW 分为定长 VLIW 和变长 VLIW。在定长 VLIW 架构下，每个指令包都存储了可并行指令条数的最大值，当指令包中实际指令数小于最大值时，需要插入 NOP(空操作)指令来填充指令包。由于定长 VLIW 插入过多的 NOP 指令会带来指令数膨胀的问题，因此本文设计了基于 RISC-V 的可变长 VLIW 架构，并基于可变长 VLIW 设计了表示指令包结束的指令编码，兼容现有 RISC-V32 和 RISC-V64 指令编码。

3.1 整体硬件架构设计

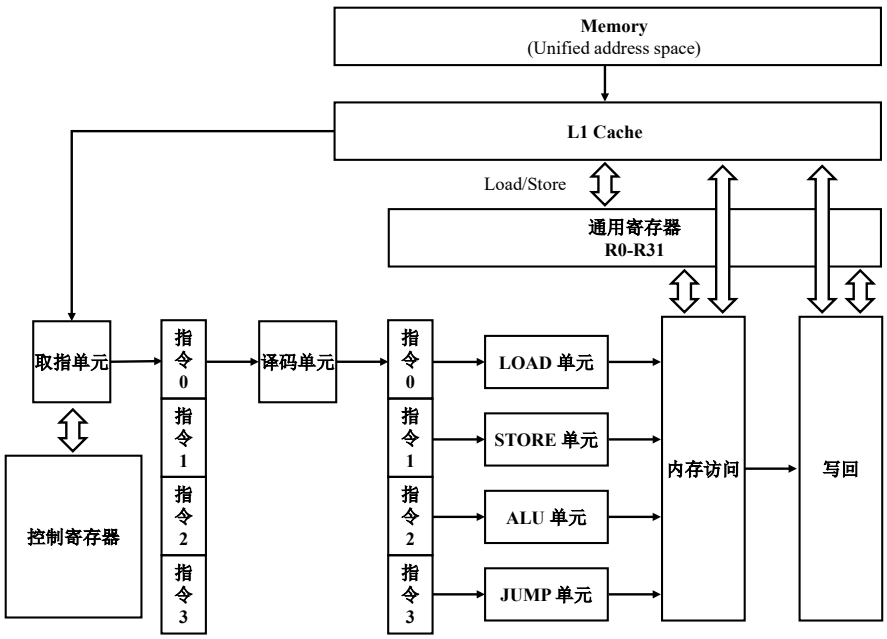


图 1 本文硬件整体架构图

由于动态 VLIW 在 DSP 中较为普遍，因此本研究的 VLIW 架构是基于动态 VLIW 的，即硬件负责阻塞，软件负责指令分组和执行单元分配。执行单元数量和每个执行单元的类型是根据应用场景需要配置的[40]。假

设执行单元数量为 N，则每个指令包的大小为 $n\times32$ 比特, 其中 $1\le n\le N$ 。一个指令包经过译码单元处理后得到多条 RISC-V 指令，并分配给各自的执行单元执行。本文参照 Hexagon 的执行单元，设计了 4 种执行单元，分别为 LOAD、STORE、ALU 和 JUMP 单元，图 1 以 N=4，执行单元配置为{LOAD, STORE, ALU, JUMP}为例给出了 VLIW 架构图，表 1 给出了各个执行单元支持的指令类型。

表 1 不同执行单元支持的指令类型

执行单元类型	支持指令类型
LOAD	LOAD、ALU
STORE	STORE、ALU
ALU	ALU
JUMP	JUMP

3.2 指令集VLIW扩展及编码

本文硬件架构基于 RISC-V 指令进行实现，选用 RV32 作为基础指令集，并在其上进行 VLIW 扩展。由于 RV32 指令的 opcode 均在编码的 0-6 位，且 opcode 的 0-1 位均为 1，因此本文使用编码的 0-1 位表示 VLIW 信息。该方法在 32 位指令编码如常用的 RV32、RV64 上是可以直接实现的。但是对于 16 位指令编码的指令，由于其编码 0-1 位不全为 1，会影响当前设计下指令包结束标志的判定，因此不能直接用于 16 位指令编码。但这一点仅影响变长 VLIW 的实现，与本文的混合指令调度算法是正交关系。

本文使用指令编码第 0 位来表示当前指令是否为指令包结尾，即一个指令包中的最后一条指令。该位默认值 1 表示是包结尾，取 0 时表示不是指令包结尾。图 2 给出了本文的编码示意图。

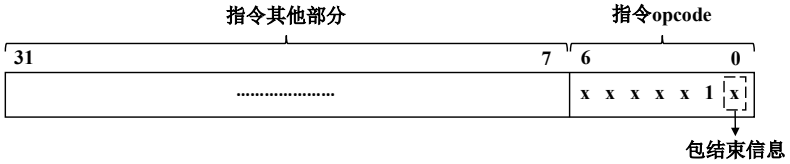


图 2 指令编码结构示意图

该 VLIW 扩展方式与 RV32 具有良好的兼容性，扩展的一位 VLIW 编码仅标识当前指令是否为包结尾，能够兼容指令本身的功能和含义，以 RV32I 中 add 指令为例进行说明(编码如表 2 所示)，add 指令实现的功能为 rd=rs1+rs2，即两个源寄存器 rs1 和 rs2 的值相加并将结果写入到目的寄存器 rd。

表 2 add 指令 VLIW 扩展示例

编码位	31-25	24-20	19-15	14-12	11-7	6-0
指令名称	funct7	rs2	rs1	funct3	rd	opcode
add	0000000	rs2	rs1	000	rd	0110011
add_ne	0000000	rs2	rs1	000	rd	0110010

其中 add_ne 指的是在指令包内而非指令包结束的 add 指令。上述两种情况下指令均会被识别为 add 指令，执行的基本操作也均为 add 操作，只不过执行时硬件会根据其 opcode 第 0 位的值在不同的模式下执行。

4 整数线性规划调度算法及表调度算法

指令调度中如何扩展调度区域与单个调度区域的指令调度的研究是正交关系。本文专注于研究单个调度区域的指令调度，针对单个调度区域的调度评价指标为该调度区域的 IPC，计算公式为 IPC=指令数/cycle 数。

4.1 整数线性规划调度算法

对于一个调度区域 R，有调度策略 $s \in S$，S 是所有调度策略的集合。IPC(R, s)指的是调度区域 R 在调度策略 s 下的 IPC，cycle(R, s)指的是调度区域 R 在调度策略 s 下的 cycle 数。则指令调度的优化目标为求一个调度策略 $s \in S$，使得 IPC(R, s)最大。由于一个调度区域 R 中的指令是固定的，因此 max IPC(R, s)等价于 min cycle(R, s)。虽然已有将整数线性规划用于指令调度算法的工作，但是现有方法并不适用于本文的 VLIW 架构，原因在于 VLIW 架构多用在 DSP 领域，其执行单元多为功能受限的，即单个执行单元只能执行特定类型的指令，而不能执行所有类型指令，如表 1 中 LOAD 执行单元仅能执行 LOAD 类型和 ALU 类型的指令。

执行完一个调度区域 R 中指令所需的 cycle 受源程序、编译器以及硬件资源 3 个方面的影响，具体来说受指令类型、指令间依赖关系和硬件执行单元的限制。每条指令有 Issue Cycle、Execute Unit、Execute Cycle、Available Units 四个属性。Issue Cycle 为该指令的发射 cycle；Execute Unit 指的是当前指令分配的执行单元；Execute Cycle 是每条指令执行所需的 cycle 数，该属性对同一条指令而言是确定的常量；Available Units 指当前指令能够在哪些执行单元上执行，其值为一个整数集合。例如一条 ALU 指令的 Execute Units 为{0, 1, 2}，表示该指令可以在 0, 1, 2 三个执行单元上执行。当硬件架构固定时，Avaliable Units 对每类指令也是固定的。

指令调度的最优化目标为 $\min\limits_{s\in S}\ \max\limits_{x\in I}\text{IssueCycle}(x)$，即求得一个调度策略，使得调度区域 R 内所有指令 Issue Cycle 的最大值最小，其中 I 表示调度区域 R 内所有指令的集合。下面给出该最优化问题需要满足的约束条件：

(一)指令间依赖关系约束

指令间的依赖关系分为 4 种：数据依赖、反依赖、输出依赖和控制依赖。定义指令 x 和指令 y 之间的依赖距离为 $\overrightarrow{d_{x,y}}$，当指令 x 和指令 y 之间为数据依赖时 $\overrightarrow{d_{x,y}}=1$，当指令 x 和指令 y 之间为反依赖、输出依赖或控制依赖时 $\overrightarrow{d_{x,y}}=0$。不同依赖类型导致指令间执行顺序需要满足的约束也有所区别：如果指令 x 和指令 y 之间存在数据依赖，则要求满足 $\text{IssueCycle}(x)<\text{IssueCycle}(y)$；如果指令 i 和指令 j 之间存在输出依赖、反依赖或者控制依赖，则要求满足 $\text{IssueCycle}(x)\le\text{IssueCycle}(y)$。

(二)硬件资源约束

VLIW 架构的硬件资源约束源自同一个 cycle 中最多能够执行的不同类型指令的数量，这受限于硬件的执行单元数量及类型，对不同的 VLIW 架构可以得到不同的约束。以图 1 中的硬件架构配置为例，其对应的约束如下：

$$\forall x,y\in I_t,\text{ExecuteUnit}(x)\neq\text{ExecuteUnit}(y) \tag{1}$$

其中 I_t 表示指令集合 I 中任意 t 时刻发射的指令，即对于任意在 t 时刻发射的指令 i 和指令 j，其分配的执行单元不同。

结合上述两个约束及本文的最优化目标，本文的最优化问题可以总结为公式(2-3)：

最优化目标：

$$\min_{s\in S}\ \max_{x\in I}\text{IssueCycle}(x) \tag{2}$$

约束条件：

$$\begin{cases}\forall x,y\in I,\ \text{IssueCycle}(x)+\overrightarrow{d_{x,y}}\le\ \text{IssueCycle}(y)\\ \forall x,y\in I_t,\ \text{ExecuteUnit}(x)\neq\text{ExecuteUnit}(y)\end{cases} \tag{3}$$

4.2 表调度算法的启发式规则

4.1 节的规划调度算法由于遍历了所有满足约束的情况，因此能得到调度单元内的最优解，但复杂度较高。因此实际编译器中常使用表调度算法进行指令调度。但是现有表调度的启发式规则仅考虑指令依赖关系，如入度、出度等，而没有考虑硬件资源限制，本文从 RISC-V VLIW 架构的硬件资源限制出发，提出了硬件资源适配的启发式规则，从而提升表调度算法的性能。

当前通用的启发式规则中只考虑了关键路径的深度、出度等概念，并没有考虑硬件资源相关的优先级。结合硬件执行单元的实际情况进行考虑，LOAD 单元能够执行 LOAD 指令和 ALU 指令，但是反过来不成立，即 ALU 单元仅能执行 ALU 指令。这说明 LOAD 指令执行受到的限制更多，因此在同等条件下，应先调度受限制更多的 LOAD 指令。该条规则适用于满足“LOAD/STORE 计算资源比 ALU 紧张”特征的硬件架构，但该规则不一定对任何调度区域都有效果，只有在调度区域中 LOAD/STORE 类指令会互相抢占资源的情况下才可能会有优化作用，因此启发式规则一般看整体的统计效果。

因此对于本文架构，根据硬件架构特点为表调度算法添加启发式规则如下：

- 规则 1：优先调度 LOAD、STORE 类指令。
- 规则 2：优先调度依出度大的指令。
- 规则 3：优先调度依赖关键路径上的指令。

以图 3 为例展示该规则的效果。从图 3 的指令 DAG 图可以看出，在计算调度优先级时，由于指令 0-4 的出度和深度都相同，在原始启发式规则下优先级相同，因此若仍按照原始指令顺序调度，图 3 中的 6 条指令将打包为 4 个指令包，分为 4 个 cycle 执行，如图 3(a)所示。在增加了“优先调度 LOAD、STORE 类指令”的启发式规则后，由于指令 0 到 4 的出度和深度都相同，而指令 3、4 为 LOAD 指令，指令 0、1、2 为 ALU 指令，若优先考虑 LOAD 和 STORE 类型指令，则调度顺序变为{3,4,0,1,2}，图 3 中的 6 条指令打包为 3 个指令包，分为 3 个 cycle 执行，如图 3(b)所示。可以看出，在增加了新的启发式规则后，执行所需 cycle 从 4 减少为 3，IPC 从 1.5 提升为 2.0。

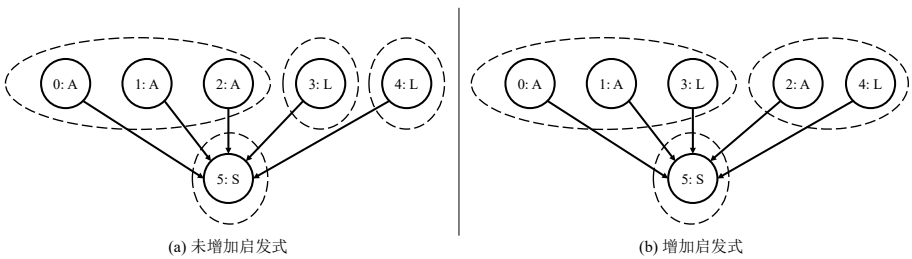


图 3 指令调度结果

5 IPC 理论模型指导的混合指令调度算法

第 4 章给出的规划调度能够在一个调度区域内求得调度最优解，但复杂度较高；表调度算法虽然复杂度较低，但无法确保得到最优解。为了能够以较低的复杂度得到最优解，本章提出了 IPC 理论模型指导的混合指令调度算法。该混合指令调度算法的流程图如图 4 所示。

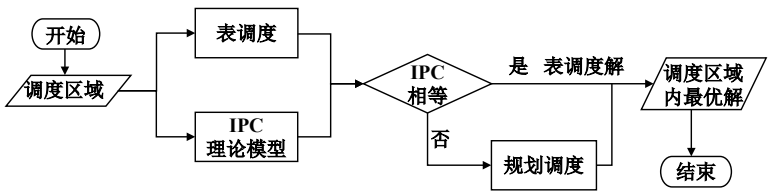


图 4 混合指令调度算法流程图

这里需要指出的是，理论模型得到的 IPC 上界并非上确界。也就是说，对于一个调度区域而言，如果表调度的解对应的 IPC 等于理论模型给出的 IPC，则说明表调度在该调度区域上得到了最优解；但反之如果不相等，并不意味着表调度得到的解一定不是最优解。也就是说，混合调度中需要再次进行规划调度的调度区域有可能已经得到了最优解。

5.1 基于数据流的IPC理论模型

该混合调度的核心在于 IPC 理论模型。本文提出的 IPC 理论模型从指令依赖约束和硬件资源约束出发，基于编译技术中经典的数据流分析，可以以 O(n)复杂度给出一个调度区域 R 内的 IPC 上界。由第 4 章可知，由于一个调度区域内的指令是固定的，求一个调度区域 R 内 IPC 的最大值等价于求调度区域的 cycle 的最小值，即要对调度区域 R 的 cycle 计算下界。

我们用 A[x]表示指令 x 的所有依赖距离 $\overrightarrow{d_{x,y}}>0$ 的必经指令集合，用 B[x]表示指令 x 的所有依赖距离 $\overrightarrow{d_{x,y}}\geq 0$ 的必经指令集合，pred[x]表示指令 x 所有前驱指令的集合。则根据数据流的顺序求得 A[x]和 B[x]的公式为：

$$\mathrm{A}[x]=\bigcup_{p\in \mathrm{pred}[x]}(\mathrm{B}[p]\cup \overrightarrow{d_{p,x}}>0?\{p\}:\varnothing) \tag{4}$$

$$\mathrm{B}[x]=\mathrm{A}[x]\cup\{x\}\cup\bigcup_{p\in \mathrm{pred}[x]}\{p\} \tag{5}$$

一个调度区域 R 中的指令可以表示为集合(i,j,k,l)，分别表示 R 中 ALU、LOAD、STORE、JUMP 四种类型指令的数量。由于不同的执行单元支持的指令类型不同，因此在进行指令调度时首先需要对资源约束进行考虑。若只考虑指令的数量和种类而不考虑指令间依赖关系，则具体到本文架构(图 1)完成调度区域 S 中指令所需的最小 cycle 数 $\varphi(i,j,k,l)$ 的计算公式为：

$$\varphi(i,j,k,l)=\max(\frac{i+j+k}{3},j,k,l) \tag{6}$$

定义指令节点 x 的最小发射 cycle，简记为 MIC[x]，表示指令 x 的最小发射 cycle，即该指令最早能够发射的 cycle。则根据必经指令集合 A[x]和 B[x]可以计算出 MIC[x]。

$$\mathrm{MIC}[x]=\max_{p\in \mathrm{pred}[x]}(\mathrm{MIC}[p]+\overrightarrow{d_{p,x}}\ ,\ \varphi(\mathrm{A}[x])+1\ ,\ \varphi(\mathrm{B}[x])) \tag{7}$$

定义一个调度区域 R 的最小发射 cycle 为该调度区域内各指令节点最小发射 cycle 的最大值，即

$$\mathrm{MIC}[R]=\max_{x\in I}\mathrm{MIC}[x] \tag{8}$$

基于必经指令集合公式(4-5)和 MIC 公式(6-8)，本文提出了一种线性复杂度的 IPC 理论模型，该算法可以给出单个调度区域内的 cycle 下界，即 IPC 上界。注意这里的下(上)界非下(上)确界。

5.2 IPC理论模型计算示例

下面通过图 5 的 3 个例子给出根据必经指令集合(4-5)及资源约束公式(6-8)计算 cycle 下界的示例。

(1)资源约束计算

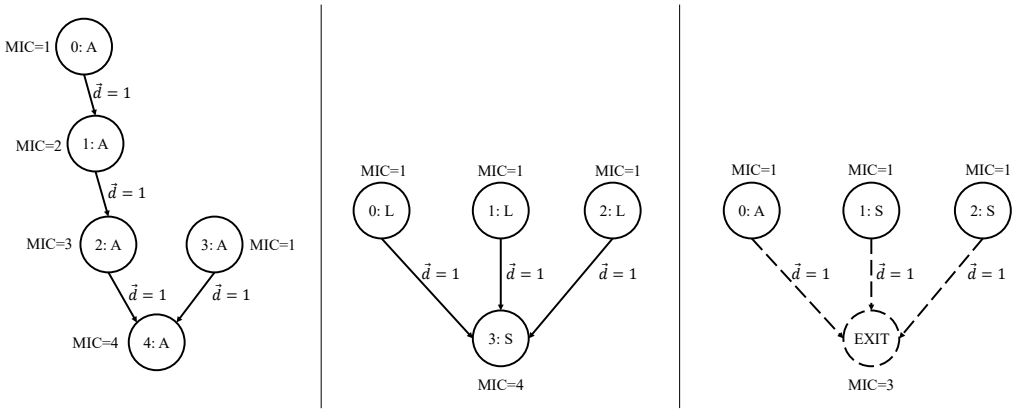
图 5(a)中节点 4 的 MIC 值取决于其直接前驱节点 2 和节点 3 的 MIC 值，因此节点 4 的 MIC 为 4。

(2)前驱节点约束计算

图 5(b)中节点 3 的 MIC 值取决于所有前驱指令类型。因为节点 0-2 为 3 条 LOAD 指令，因此至少需要 3 条指令完成 0-2，因此节点 3 的 MIC 为 4。

(3)虚拟退出节点的计算

图 5(c)中由于不存在统一的退出节点，无法根据节点的 MIC 值给出 DAG 图的 MIC 值。因此通过引入虚拟退出节点这一编译器的常用技术来得到 DAG 图的 MIC 值。根据节点 0-2，可得虚拟退出节点的 MIC 值为 3。



(a) 资源约束计算示例 (b) 前驱节点约束计算示例 (c) 虚拟退出节点计算示例

图 5 指令节点 MIC 值计算示例

6 实验

6.1 实验基础设置

本文基于 LLVM 16.0 实现了混合指令调度算法及编译器，使用 RISC-V GNU toolchain 中的链接器，基于 RISC-V 模拟器 Spike 扩展了 VLIW 功能，选用嵌入式系统的性能评测基准 Coremark 对工具链进行评测，整体执行流程图如图 6 所示。为了验证该系统在 DSP 领域的表现，我们进一步采用 DSP 领域的真实应用 AlexNet[41]对工具链进行测评。

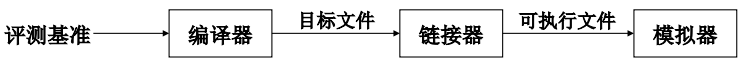


图 6 工具链整体执行流程图

6.2 变长VLIW的代码体积

由于本文设计了可变长的 VLIW 编码，即编码中携带的信息可以在指令解码阶段用于分割各指令包，因此指令打包时不需要像定长 VLIW 一样插入额外的 NOP 指令，这大大减小了生成的二进制文件的体积。Coremark 中各函数代码体积在变长编码和定长编码占用空间对比如图 7 所示。变长编码生成的二进制共包含 2,855 条指令，代码体积为 11.15KB。定长 VLIW 编码生成的二进制中共包含 6,340 条指令，代码体积为 24.77KB。可以看出，与定长 VLIW 架构相比，本文提出的基于 RV32 指令编码特性设计的可变长 VLIW 架构在保证兼容性的前提下，将代码体积平均减小了 54.97%。

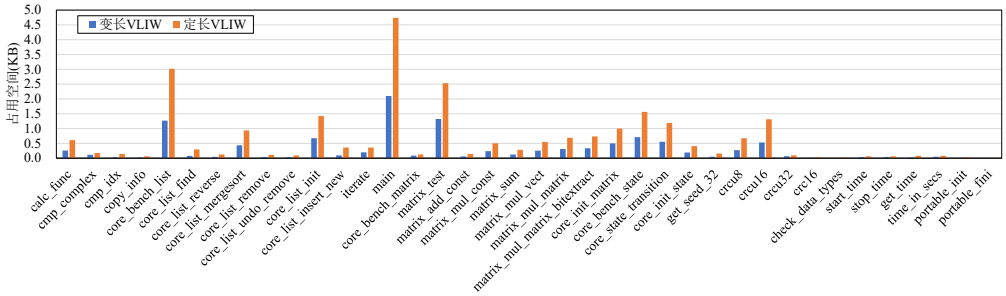


图 7 Coremark 中各函数的代码体积

6.3 表调度的启发式规则优化

我们将混合调度与表调度及其各种变体，即自顶向下表调度、自底向上表调度、和自顶向下与自底向上相结合的表调度进行了对比，并探究了我们在 4.2 节中提出的 3 个启发式规则的效果。从图 8 可以看出，混合调度，与自顶向下表调度相比提升了 4.02%，与自底向上表调度相比提升了 4.62%，与自顶向下和自底向上相结合表调度相比提升了 4.02%。图 8 同时也给出了 4.2 节中 3 个启发式规则的效果。可以看出，规则 1 的 LOAD/STORE 启发式能够提升 3.45%，规则 2 的关键路径启发式能够提升 1.72%，规则 3 的出度启发式能够提升 0.57%，可以看出，规则 1 的 LOAD/STORE 启发式能够获得更为有效的提升。

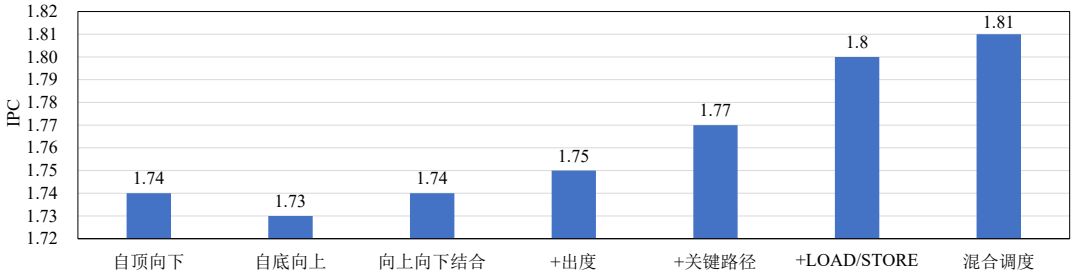


图 8 各种调度算法在 Coremark 中的 IPC

为了更为详细地展示启发式规则对表调度性能的影响，图 9 给出了 Coremark 中各函数在是否引入 4.2 节规则 1-3 的启发式规则时的 IPC。通过引入启发式规则，IPC 平均从 1.74 提升到 1.80，提升了 3.09%。

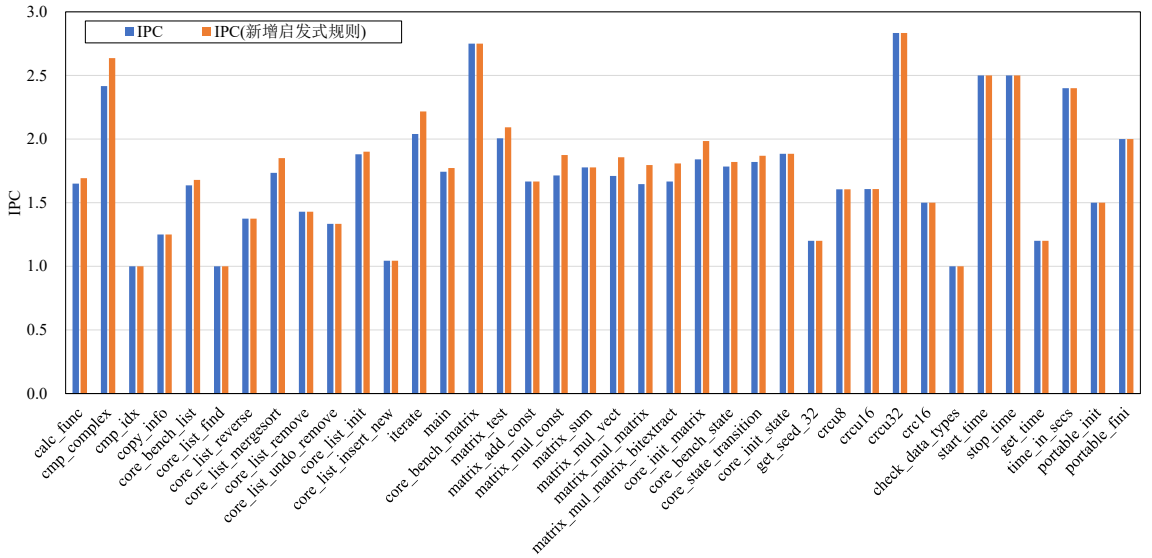


图 9 Coremark 中各函数在表调度下的 IPC

6.4 IPC理论模型的性能评估

图 10 给出了 Coremark 中调度区域内指令数的分布统计。可以看出，Coremark 中共有 446 个调度区域，其中 59.19%的调度区域内指令数小于等于 4, 88.12%的调度区域内指令数小于等于 10。这使我们对调度区域的规模有了大致的了解。

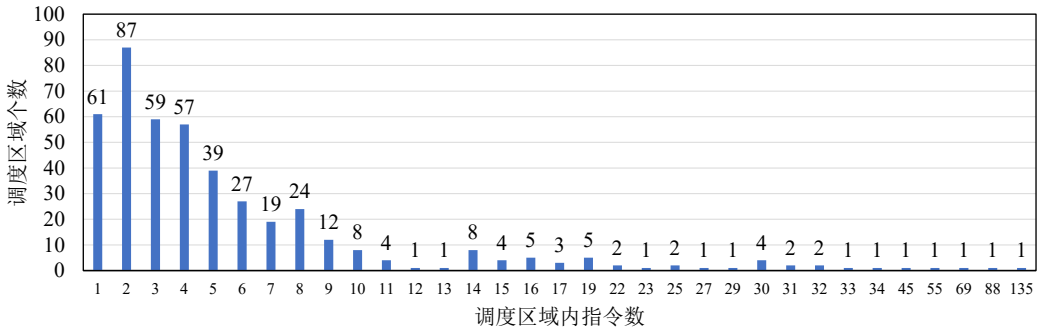
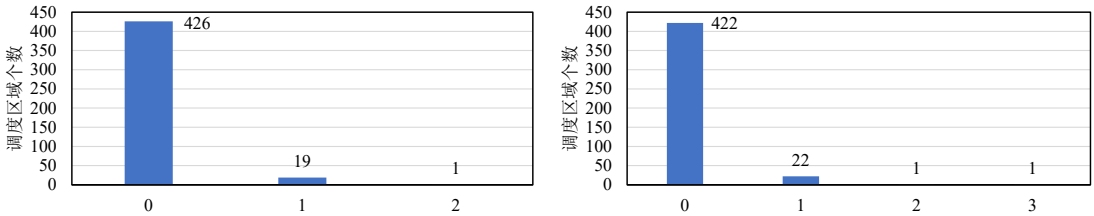


图 10 调度区域内指令数分布统计图

本文使用规划调度求得的最优解来评估 IPC 理论模型。图 11(a)给出了 IPC 理论模型与规划调度算法的最优解对应的 IPC 的差值分布，可以看出在 446 个调度区域中，有 95.52%的调度区域理论模型给出的 IPC 就是最优解的 IPC，仅有 4.48%的调度区域理论模型的 IPC 与最优解有小于等于 2 个 cycle 的差距。

为了说明混合调度算法中理论模型降低了多少复杂度，即有多少调度区域仅需表调度即可达到最优解，图 11(b)给出了理论模型和表调度的 IPC 的差值分布，可以看出，94.62%的调度区域理论模型与表调度的 IPC 相同，5.38 %的调度区域 cycle 数差值小于等于 3。即在理论模型的指导下，94.62%的调度区域在表调度下即可认定达到最优解，仅有 5.38%的调度区域需再次进行规划调度。



(a) 理论模型与规划调度结果的cycle差值 (b) 理论模型与表调度结果的cycle差值

图 11 不同 cycle 数差值调度区域个数统计

表 3 IPC 理论模型分类结果

	实际正例 (表调度实际达到最优解)	实际负例 (表调度实际未达最优解)
预测正例 (预测表调度达到最优解)	真正 TP=422	假正 FP=0
预测负例 (预测表调度未达最优解)	假负 FN=19	真负 TN=5

表 4 IPC 理论模型评价指标

评价指标	数值
准确率	95.74%
精确率	100.00%
召回率	95.69%
F1 值	97.79%

由于理论模型在混合调度中实际被用于分类(即判断表调度是否在一个调度区域内达到了最优解)，因此可以用衡量分类模型性能的指标来评估 IPC 理论模型。表 3 给出了真正 TP、假正 FP、假负 FN、真负 TN 的值，用于计算表 4 中的评价指标，即准确率、精确度、召回率和 F1 值。

准确率用于衡量模型分类的准确性，95.74%的准确率说明理论模型分类性能较好。精确率用于衡量模型识别为正例的结果中实际正例的比例，精确率为 100.00%印证了理论模型认定为最优解的情况一定是最优解。理论模型召回率为 95.69%，这意味着 95.69%的最优解都能够被理论模型定位出来，即仅有 4.31%经过表调度得到的最优解会被认定为非最优解，需要再额外进行规划调度。此时由于表调度已经得到了最优解，再次使用规划调度算法得到的解与表调度的解应是相同的。F1 值是准确率和召回率的加权平均，为 97.79%。

图 12 给出了 Coremark 中各个函数分别使用本文提出的表调度、规划调度与理论模型三种方法计算所得的 IPC。由第 4、5 章的内容可知，IPC(表调度)<=IPC(规划调度)<=IPC(理论模型)，其中规划调度的结果可以作为最优解来评价表调度和理论模型。从图 12 可以看出，在表调度下有 89.47%的函数能够达到规划调度解。由于调度算法是针对单个调度区域的，因此函数级别的统计结果不如图 11 区域级别的结果更有统计意义。

为了探索 IPC 理论模型与指令规模的关系，我们也研究了在不同的指令规模下，IPC 理论模型的准确性的变化。从表 5 中可以看出，在调度区域内指令数小于等于 5 时， IPC 理论模型在 Coremark 和 AlexNet 上的准确率达到了 100%和 99.57%。这是因为指令数较少时，指令依赖和资源依赖的相互干扰较少，理论模型能够得到准确结果。在指令数为 6 到 15 之间，准确率稍有下降，但仍高达 97.22%(Cormark)和 92.68%(AlexNet)。当指令数大于等于 16 时，准确率持续下降。但是从表 5 中可以看出，指令数小于等于 15 的调度区域占比为 92.16%(Coremark)和 93.85%(AlexNet)。因此，理论模型在指令数小于等于 15 时的高准确率是有意义的。为了更为详细的看出调度区域内指令数对理论模型的准确率的影响，图 13(a)和图 13(b)分别给出了当 Cormark 和 AlexNet 中的调度区域内的指令数变化时，理论模型给出的 IPC 和规划调度给出的 IPC。可以看出当指令

数小于等于 15 时，理论模型与规划调度的两条线是高度重合的。

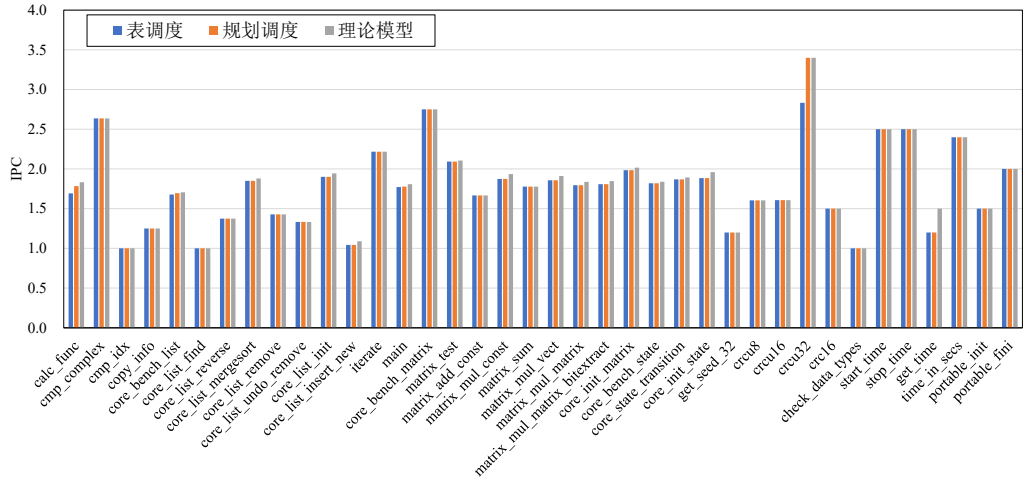
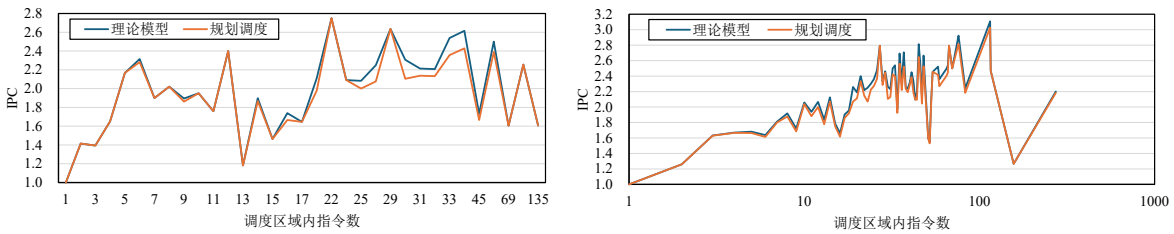


图 12 Coremark 中各函数在表调度、规划调度及理论模型下的 IPC

表 5 IPC 理论模型评价指标

调度区域内指令数	Coremark			AlexNet		
	调度区域数	调度区域百分比	理论模型准确率	调度区域数	调度区域百分比	理论模型准确率
[1,5]	303	67.94%	100.00%	4144	72.37%	99.57%
[6,15]	108	24.22%	97.22%	1230	21.48%	92.68%
[16,50]	31	6.95%	48.38%	333	5.82%	65.46%
>50	3	0.67%	75.00%	19	0.33%	47.37%



(a) Coremark中不同指令数的调度区域的IPC　　(b) AlexNet中不同指令数的调度区域的IPC(横坐标为对数刻度)

图 13 Coremark 和 AlexNet 中不同指令数的调度区域的 IPC

6.5 编译时间分析

由于存在着不同的调度算法，其复杂度和调度效果都各有区别。本文对比了表调度、规划调度和混合调度的编译时间，如表 6 所示。可以看出，混合调度的编译时间只有规划调度的 15.67%。虽然混合调度只判定了 24 个调度单元需要进行规划调度，但是由于这 24 个调度单元的平均指令数目为 26，大于 coremark 所有调度单元的平均指令数 6.4，所以只有 5.38%的调度单元需要进行规划调度，但是却需要 15.67%的规划调度的编译时间。并且，混合调度可以进一步优化为编译时间自适应的，即可以根据应用场景对编译时间的限制去自适应选择规划调度的调度区域，从而满足任意的编译时间限制。

表 6 三种调度算法的编译时间对比

调度算法	表调度	规划调度	混合调度
编译时间(秒)	0.58	78.01	12.23

6.6 IPC理论模型的可扩展性分析

在设计硬件参数配置时，我们参考了业界主流的 VLIW 架构功能单元设计理念设计了 3 种硬件配置，根据极限情况设计了硬件 1 配置，参考 Hexagon 的 4 个功能单元设计了硬件 2 配置(即前文所用配置)，参考 Kalray 的 6 个功能单元设计了硬件 3 配置，如表 7 所示。我们在该硬件配置下实施理论模型和混合调度，以便验证本文理论模型的可扩展性。

表 7 三种功能单元配置下的硬件

功能单元	硬件 1	硬件 2	硬件 3
FU0	ALU、LOAD	ALU	ALU
FU1	ALU、STORE、JUMP	ALU、LOAD	ALU
FU2	/	ALU、SOTRE	ALU
FU3	/	JUMP	ALU、LOAD
FU4	/	/	ALU、STORE
FU5	/	/	JUMP

表 8 给出了 Coremark 在表 7 的三种硬件下对应的准确率，精确率，召回率和 F1 值。可以看出，当硬件变化时，本文的 IPC 理论模型仍然能得到较高的准确率。这验证了我们的 IPC 理论模型和混合调度的可扩展性。

表 8 三种硬件配置下的 IPC 理论模型评价指标

指标	硬件 1	硬件 2	硬件 3
准确率	89.91%	98.46%	95.51%
精确率	100.00%	100.00%	100.00%
召回率	89.66%	96.63%	95.46%
F1 值	94.73%	98.34%	97.68%

6.7 AlexNet的实验结果

本文选用 DSP 领域中 AlexNet 这一经典神经网络对本文提出的 IPC 理论模型的准确率进行评测。图 14 给出了表调度、规划调度和理论模型在 AlexNet 上的表现。可以看出，在 AlexNet 共 5726 个调度区域中，有 95.93%的调度区域理论模型给出的 IPC 等于规划调度的 IPC，有 3.62%的调度区域理论模型与规划调度仅有 1 个 cycle 的差距。为了说明混合调度算法中理论模型降低了多少复杂度，即有多少调度区域仅需表调度即可达到最优解，图 14 中也给出了理论模型和表调度的 IPC 的差值分布。可以看出，AlexNet 中有 94.72%的调度区域理论模型与表调度的 IPC 相同，因此仅有 2.58%的调度区域需再次进行规划调度。

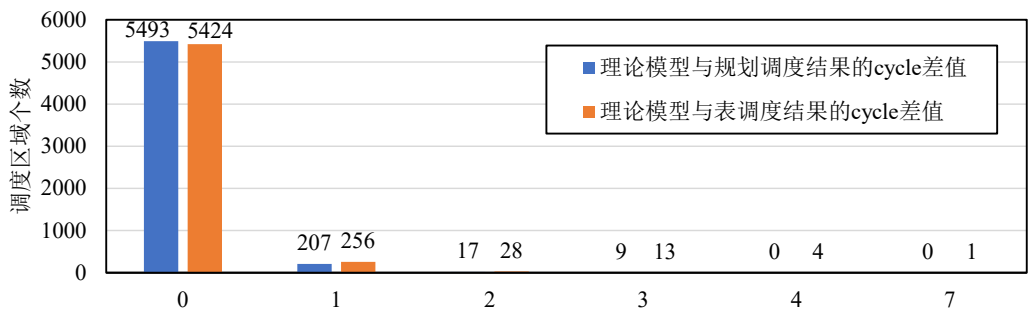
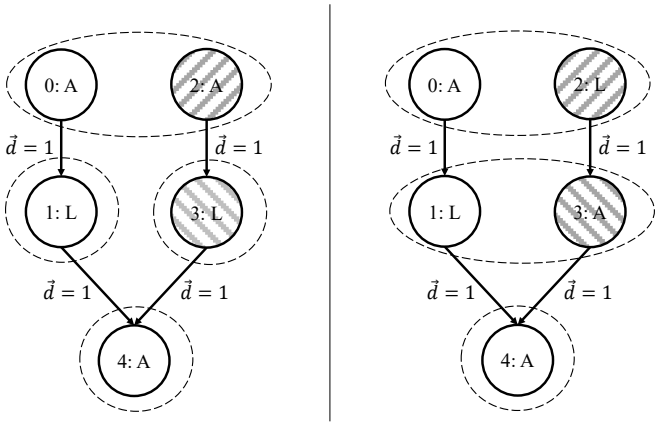


图 14 AlexNet 中不同 cycle 数差值调度区域个数统计

6.8 个例分析



(a) 理论模型未达上确界　　(b) 理论模型可达上确界

图 15 理论模型未达/可达 IPC 上确界示例

本文的理论模型得到的是 IPC 上界(cycle 下界)，而非上(下)确界。图 15 举例说明了理论模型无法得到 IPC 上确界(cycle 下确界)的场景和原因。图 15(a)和(b)分别为不同调度区域的依赖图，能够在同一个 cycle 执行的指令用虚线圆圈一起。可以看出，图 15(a)最优 cycle 为 4, 但理论模型给出的 cycle 为 3。图 15(b) cycle 为 3，I 理论模型给出的 cycle 为 3。易知图 15(a)论模型给出的 cycle 不是下确界。原因为：指令 1 和指令 3 均为 LOAD 指令，而由于硬件资源限制(只有 1 个 LOAD 执行单元)，指令 1 和指令 3 不能在同一个 cycle 执行，必须分在 2 个 cycle 先后执行。而在数据流分析中，指令 4 作为交汇点，无法精确感知两条前驱路径精确的资源冲突。即在本文的 IPC 理论模型的数据流分析中，无法区分图 15(a)和图 15(b)两种情况。因此面对图 15(a)前驱路径相互存在资源依赖的情况，理论模型无法得到 IPC 上确界。

7 总结

本文设计了基于 RISC-V 指令集的可变长 VLIW 架构，并提出了针对单个调度区域的 IPC 理论模型指导的混合指令调度算法。混合调度通过 IPC 理论模型定位表调度可能没有得到调度最优解的调度区域，再对该调度区域进一步实施规划调度。混合调度的核心在于 IPC 理论模型的准确性，本文的 IPC 理论模型准确率为 95.74%，F1 值为 97.79%。本文提出的 IPC 理论模型能够认定 94.62%的调度区域在表调度下已达最优解，因此仅有 5.38%的调度区域需再进行规划调度。因此该混合指令调度算法能够以接近表调度的复杂度达到规划调度的调度解。

References:

[1] Waterman A, Asanovic K. The RISC-V Instruction Set Manual Volume I: Unprivileged IAS. Document Version 20191213[EB/OL]. https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf. 2024-08-09.

[2] Bao Y G, Sun N H. Opportunities and challenges of building CPU ecosystem with open-source mode. Bulletin of Chinese Academy of Sciences, 2022, 37(1): 24-29. (in Chinese)

[3] Celio C, Patterson D A, Asanovic K. The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor[J]. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167, 2015.

[4] Li XL, Han M, Hao K, Xue HY, Lu SJ, Zhang KM, Qi N, Niu XM, Xiao LM, Hao QF. Design of RISC-V CPU for 100 Gbps Network Application[J]. Journal of Computer-Aided Design & Computer Graphics, 2021, 33(6): 956-962. DOI: 10.3724/SP.J.1089.2021.18538 (in Chinese with English abstract).

[5] Hu ZB. Teach You to Design CPU by hand. RISC-V Processor [M]. Posts and Telecommunications Press, 2018.

[6] Tine B, Yalamarthy K P, Elsabbagh F, Hyesoon K. Vortex: Extending the RISC-V ISA for GPGPU and 3D-graphics[C]//MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture. 2021: 754-766.

[7] Fisher J A. Very long instruction word architectures and the ELI-512[C]//Proceedings of the 10th annual international symposium on Computer architecture. 1983: 140-150.

[8] Qualcomm. The Qualcomm Hexagon SDK[EB/OL]. https://www.qualcomm.com/developer/software/hexagon-npu-sdk. 2024-08-09.

[9] Texas Instruments. TMS320C64x Technical Overview. https://www.ti.com/lit/ug/spru395b/spru395b.pdf. 2024-08-09.

[10] WANG XQ, HONG Y, WANG H, ZHENG QL. Compiler Design and Optimization for BWDSP[J]. Acta Electronica Sinica, 2015, 43(8): 1656-1661. https://doi.org/10.3969/j.issn.0372-2112.2015.08.028 (in Chinese with English abstract).

[11] Michael Larabel. Kalray VLIW processor family (kvx) [EB/OL]. https://www.phoronix.com/news/Kalray-KVX-Linux-Port . 2024-10-29.

[12] Qui N M, Lin C H, Chen P. Design and implementation of a 256-bit RISC-V-based dynamically scheduled very long instruction word on FPGA[J]. IEEE Access, 2020, 8: 172996-173007.

[13] Hennessy J L, Gross T. Postpass Code Optimization of Pipeline Constraints[J]. Acm Transactions on Programming Languages & Systems, 1983. DOI:https://doi.org /10.1145/2166.357217

[14] Bernstein D, Gertner I. Scheduling expressions on a pipelined processor with a maximal delay of one cycle[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1989, 11(1): 57-66.

[15] Auyeung A, Gondra I, Dai H K. Multi-heuristic list scheduling genetic algorithm for task scheduling[C]//Proceedings of the 2003 ACM symposium on applied computing. 2003: 721-724.

[16] Huang L, Feng XB. Survey on techniques of integrated instruction scheduling and register allocation[J]. Application Research of Computers. 2008, 25(4): 979-982. (in Chinese with English abstract).

[17] Deng C, Chen Z, Shi Y, Ma Y, Wen M, Luo L. Optimizing VLIW Instruction Scheduling via a Two-Dimensional Constrained Dynamic Programming[J]. ACM Transactions on Design Automation of Electronic Systems, 2024.

[18] Fisher J A. Trace scheduling: A technique for global microcode compaction[J]. IEEE transactions on computers, 1981, 30(07): 478-490.

[19] Colwell R P, Nix R P, O'Donnell J J, Papworth D B, Rodman P K. A VLIW architecture for a trace scheduling compiler[J]. ACM SIGARCH Computer Architecture News, 1987, 15(5): 180-192.

[20] Hwu, W. M. W., Mahlke, S. A., Chen, W. Y., Chang, P. P., Warter, N. J., Bringmann, R. A., ... & Lavery, D. M. The superblock: An effective technique for VLIW and superscalar compilation[M]//Instruction-Level Parallelism: A Special Issue of The Journal of Supercomputing. Boston, MA: Springer US, 2011: 229-248.

[21] Mahlke S A, Lin D C, Chen W Y, Hank R E, Bringmann R A. Effective compiler support for predicated execution using the hyperblock[J]. ACM SIGMICRO Newsletter, 1992, 23(1-2): 45-54.

[22] Giesemann F, Payá-Vayá G, Gerlach L, Blume H, Pflug F, von Voigt G. Using a genetic algorithm approach to reduce register file pressure during instruction scheduling[C]//2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS). IEEE, 2017: 179-187.

[23] Giesemann F, Gerlach L, Paya-Vaya G. Evolutionary algorithms for instruction scheduling, operation merging, and register allocation in VLIW compilers[J]. Journal of Signal Processing Systems, 2020, 92(7): 655-678.

[24] Stuckmann F, Payá–Vayá G. A Graph Neural Network Approach to Improve List Scheduling Heuristics Under Register-Pressure[C]//2024 13th International Conference on Modern Circuits and Systems Technologies (MOCAST). IEEE, 2024: 01-06.

[25] Six C, Boulmé S, Monniaux D. Certified and efficient instruction scheduling: application to interlocked VLIW processors[J]. Proceedings of the ACM on Programming Languages, 2020, 4(OOPSLA): 1-29.

[26] Six C, Gourdin L, Boulmé S, Monniaux D, Fasse J, Nardino N. Formally verified superblock scheduling[C]//Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs. 2022: 40-54.

[27] Yang Z, Shirako J, Sarkar V. Fully Verified Instruction Scheduling[J]. Proceedings of the ACM on Programming Languages, 2024, 8(OOPSLA2): 791-816.

[28] Herklotz Y, Wickerson J. Hyperblock Scheduling for Verified High-Level Synthesis[J]. Proceedings of the ACM on Programming Languages, 2024, 8(PLDI): 1929-1953.

[29] Zhou ZX, He H, Zhang YJ, Yang X, Sun YH. Two-dimensional force-directed cluster scheduling algorithm for the clustered VLIW architecture[J]. Journal of Tsinghua University (Science and Technology) 2008, 48(10): 1647-1650. (in Chinese with English abstract).

[30] Desoli G. Instruction assignment for clustered VLIW DSP compilers: A new approach[M]. Hewlett Packard Laboratories, 1998.

[31] Porpodas V, Cintra M. CAeSaR: Unified cluster-assignment scheduling and communication reuse for clustered VLIW processors[C]//2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES). IEEE, 2013: 1-10.

[32] Park J C H, Schlansker M. On predicated execution[M]. Palo Alto, California: Hewlett-Packard Laboratories, 1991.

[33] Traber A, Zaruba F, Stucki S, Pullini A, Haugou G, Flamand E, Gürkaynak F K, Benini L. PULPino: A small single-core RISC-V SoC[C]//3rd RISC-V Workshop. 2016: 15.

[34] riscv-collab. RISCV-GNU-Toolchain[EB/OL]. https://github.com/riscv-collab/riscv-gnu-toolchain. 2024-08-09.

[35] llvm. The LLVM Compiler Infrastructure[EB/OL]. https://github.com/llvm/llvm-project. 2024-08-09.

[36] riscv-software-src. Spike RISC-V ISA Simulator[EB/OL]. https://github.com/riscv-software-src/riscv-isa-sim. 2024-08-09.

[37] Bellard F. QEMU, a fast and portable dynamic translator[C]//USENIX annual technical conference, FREENIX Track. 2005, 41(46): 10-5555.

[38] Binkert N, Beckmann B, Black G, Reinhardt S K, Saidi A, Basu A, ... & Wood D A. The gem5 simulator[J]. ACM SIGARCH computer architecture news, 2011, 39(2): 1-7.

[39] eembc. CoreMark is an industry-standard benchmark that measures the performance of central processing units (CPU) and embedded microcrontrollers (MCU) [EB/OL]. https://github.com/eembc/coremark. 2024-08-09.

[40] Zha YQ. Research on Software-Hardware Co-Optimization Based on VLIW Architecture[D]. University of Chinese Academy of Science, 2024. (in Chinese with English abstract).

[41] Dynmi. Implementation of AlexNet with C[EB/OL]. https://github.com/Dynmi/AlexNet. 2024-10-29.

附中文参考文献:

[2] 包云岗, 孙凝晖. 开源芯片生态技术体系构建面临的机遇与挑战. 中国科学院院刊, 2022, 37(1): 24-29.

[4] 李晓霖, 韩萌, 郝凯, 薛海韵, 卢圣健, 张昆明, 祁楠, 牛星茂, 肖利民, 郝沁汾. 面向 100 Gbps 网络应用的 RISC-V CPU 设计与实现[J]. 计算机辅助设计与图形学学报, 2021, 33(6): 956-962.

[5] 胡振波. 手把手教你设计 CPU. RISC-V 处理器篇[M]. 人民邮电出版社, 2018.

[10] 王向前, 洪一, 王昊, 郑启龙. 魂芯 DSP 的编译器设计与优化[J]. 电子学报, 2015, 43(8): 1656-1661.

[16] 黄磊, 冯晓兵. 结合的指令调度与寄存器分配技术[J]. 计算机应用研究, 2008(979-982).

[29] 周志雄, 何虎, 张延军, 杨旭, 孙义和. 一种用于分簇 VLIW 结构的二维力量引导簇调度算法[J]. 清华大学学报: 自然科学版, 2008, 48(10): 1647-1650.

[40] 查永权. 基于 VLIW 架构的软硬件协同优化研究[D]. 中国科学院大学, 2024.