高技术通讯 2026 年 第36卷 第6期:551 - 563

doi:10.3772/j.issn.1002 - 0470.2026.06.001

处理器专用指令自动化设计方法综述①

李苍源②*** 刘 成③* 王 颖*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 随着嵌入式系统应用需求的增长和第5代精简指令集计算机(reduced instruction set computer V,RISC-V)开源指令集架构的兴起,应用专用指令集处理器(application specific instruction-set processor,ASIP)的设计已成为提升应用性能的关键技术。而为了降低ASIP设计的复杂度和开发成本,其设计自动化方法得到了广泛关注,其中专用指令集的自动设计是ASIP设计自动化流程中最具挑战性的环节之一。本文系统地综述了专用指令自动设计过程的主要流程和代表性工作:首先详细讨论了4种不同类型的专用指令,即单输入单输出指令(single input single output,SISO)、单输入多输出指令(single input multiple output,SIMO)、多输入单输出指令(multiple input single output,MISO)、多输入多输出指令(multiple input multiple output,MIMO)及其特点和应用场景;随后深入探讨了专用指令设计中的代价模型以及各类搜索方法;并分析了专用指令实现优化的2个重要方面,即资源开销优化和搜索性能优化。最后对专用指令自动设计的研究趋势作出预测。

关键词 应用专用指令集处理器; 指令集设计自动化; 代价模型; 设计空间探索

随着嵌入式系统应用需求的不断增长和第5代精简指令集计算机(reduced instruction set computer V,RISC-V)开源指令集架构的兴起[1],对通用处理器进行领域专用化扩展,即应用专用指令集处理器(application specific instruction-set processor,ASIP)的设计方法已成为提升信号处理、人工智能推理、图像处理和数据加密等典型任务执行效率的关键技术之一,在5G通信[2-3]、自动驾驶[4]和物联网[5]等领域被广泛应用。传统通用处理器(general purpose processor,GPP)具有较高的灵活性但效率较低,而专用集成电路(application specific integrated circuit,ASIC)则相反,其往往有着更高的效率而灵活性较差。应用专用指令集处理器通过在特定通用处理器基础指令集架构上扩展专用指令,实现在保持一定灵活性的同时,能够获得接近专用集成电路的性能和效率[6]。

在ASIP的设计过程中,专用指令的设计是最核心也是最具挑战性的环节。这是一个复杂的系统工程,面临着设计空间、性能开销权衡和实现评估3个层面的挑战。首先,随着应用规模增大,潜在的指令组合呈指数级增长,导致设计空间急剧膨胀。专用指令实现时需要在性能提升和硬件开销之间寻求平衡,更复杂的指令虽然可能带来更高的性能收益,但同时也增加了面积和功耗开销。最后,如何在设计早期准确评估不同方案的性能、功耗和面积特征,也是一个关键性问题。面对上述挑战,传统的基于手动的专用指令设计过程需要设计人员反复进行性能分析和架构优化,一方面对设计人员的水平提出了相当高的需求,另一方面设计周期较长。随着应用复杂度的提升,人工设计方法的局限性日益显现。

① 中国科学院战略性先导科技专项(XDB0660000,XDB0660100)资助项目。

② 男,1998年生,博士生;研究方向:专用处理器自动设计;E-mail: licangyuan@ict.ac.cn。

③ 通信作者,E-mail: liucheng@ict.ac.cn。

(收稿日期:2024 - 12 - 02)

因此,专用指令设计自动化变得尤为重要。

1　处理器专用指令设计原理及挑战

随着应用需求的日益多样化,通用处理器难以满足特定领域对性能、功耗等方面的严格要求。根据特定需求扩展专用指令作为提升处理器性能的重要手段,能够针对特定应用特征进行定制化设计,在保持处理器通用性的同时实现性能或能效方面的优化,如图 1 所示。本节将介绍处理器专用指令的设计流程,并分析其自动化设计面临的主要挑战。

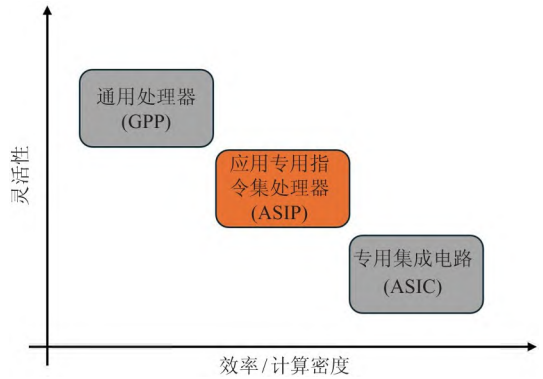

图 1　不同加速实现方法的设计权衡

1.1　专用指令设计

专用指令设计过程如图 2 所示,其输入通常为实现约束(如面积、功耗等)和目标应用的代码,输出为满足当前约束下最优的专用指令的组合,这些组合会在后续架构实现阶段作为参考对基础处理器进行架构层面的扩展和定制。专用指令的设计过程可以分为专用指令识别、专用指令设计 2 个主要阶段,在此过程中也需要考虑使用必要的优化策略提高设计质量或搜索效率。

在指令识别阶段,输入为编译器给出的当前应用的中间表示[7](intermediate representation,IR),其中保留了程序的计算特征和数据依赖关系,此外,性能分析工具对代码进行性能分析的结果也会作为输入。在这一阶段,枚举算法会枚举满足设计约束的候选指令。根据输入输出特征,候选指令可以分为 4 类,它们也有各自的适用场景:即单输入单输出指令(single input single output,SISO)主要用于简单的数据变换操作;单输入多输出指令(single input multiple output,SIMO)适用于数据分解和多路计算;多输入单输出指令(multiple input single output,MISO)常用于数据归约和复杂运算;多输入多输出指令(multiple input multiple output,MIMO)则针对复杂的数据转换和并行计算。

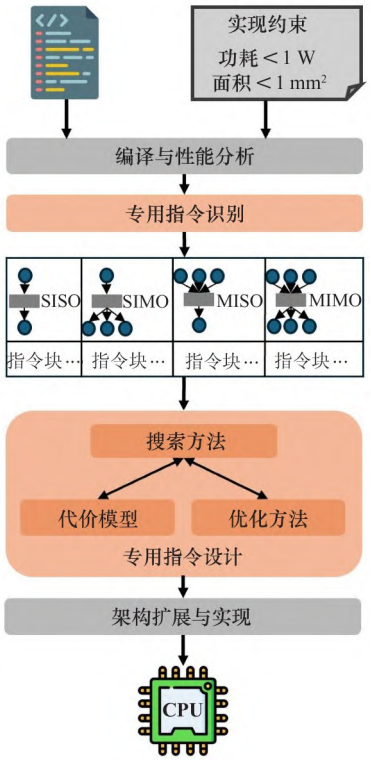

图 2　专用指令设计过程

最后,指令设计和实现阶段旨在从候选指令中选择最优组合。这个过程需要建立准确的代价模型,综合考虑指令的执行延迟、面积、功耗等多个维度。为了评估设计方案的质量,通常采用用户给定领域的典型应用或标准的基准程序(如 MediaBench[8]等)进行测试。

在专用指令设计过程中,可以从多个方面考虑进行必要的优化。例如为了提高处理器的面积效率,采用功能单元复用等方式降低硬件开销;为了提高搜索效率,利用早期设计空间剪枝技术排除不可行方案等。

1.2　专用指令设计自动化的挑战

在处理器专用指令的自动化设计过程中,面临以下几个主要挑战。

挑战 1:复杂且庞大的设计空间。

随着应用规模增大,专用指令的设计空间呈指

数级增长,这使得穷举式搜索变得不切实际。在大规模应用中,这个问题表现得尤为突出。首先,数据流图的规模增长导致枚举专用指令的计算复杂度急剧上升;其次,设计空间中存在大量冗余和非最优解,这不仅增加了搜索难度,也提高了筛选成本;最后,应用的动态执行路径带来了行为的不确定性,这进一步扩大了需要考虑的设计空间。传统的启发式算法在处理如此庞大的设计空间时往往表现出效率低下的问题[9],如何在保证搜索算法完备性的同时提高其效率,是一个重要的研究方向。

挑战 2:专用指令的性能与开销权衡。

专用指令设计需要在性能提升和硬件开销之间寻求平衡。一方面,更复杂的指令通常能带来更高的性能提升,如覆盖更大的数据依赖关系图或合并更多基本操作;但另一方面,这类指令会显著增加处理器的面积和功耗开销[10]。例如,增加专用指令的输入输出端口数量可以提高并行度,但会导致寄存器堆端口和互连开销的急剧上升;同样,为专用指令增加流水级数可以提高时钟频率,但也会带来额外的流水寄存器开销和控制逻辑复杂度[11]。因此,如何在有限的硬件资源约束下实现最大的性能收益,是专用指令设计中的关键挑战。

挑战 3:巨大的设计评估开销。

在专用指令设计过程中,准确评估不同方案的性能、功耗和面积特征也是一个挑战。精确的性能、功耗和面积的评估需要完整的硬件实现和详细的时序分析,这在设计空间探索阶段往往带来难以承受的计算开销。虽然快速的分析模型可以提高设计效率,但其准确性往往难以保证,尤其是在预测复杂指令的行为时。此外,不同应用场景下的工作负载特征差异也会影响评估的准确性。如何在保证模型精度的同时提高评估效率,以及如何构建能够准确捕获微架构影响的评估模型,都是亟待解决的问题。这种评估效率与准确性之间的权衡,直接影响着自动化设计流程的实用性。

2 专用指令类型

给定输入应用的一组数据流图,专用指令的设计过程就是在这些图中枚举满足特定约束的模式。根据输入输出特征的不同,专用指令可以分为 4 种主要类型,分别为单输入单输出指令、单输入多输出指令、多输入单输出指令和多输入多输出指令,它们之间的对比如表 1 所示。

表 1 不同专用指令类型及其比较

指令类型	输入/输出特征	典型应用场景	实现复杂度	硬件约束	性能提升潜力	识别算法复杂度	主要优势	主要挑战
SISO	单输入单输出	基本数据变换(如三角函数、滤波)	最低	仅需一个读端口和写端口	有限	最低	实现简单、应用广泛	性能提升有限
SIMO	单输入多输出	数据转换与分解(如颜色信息变换、密钥生成)	中等	需要多个写端口或多周期写回机制	中等	中等	降低数据访问开销、提高计算密度	多输出写回调度、指令编码
MISO	多输入单输出	数据归约和聚合操作(如密码学、信号处理)	中等	需要多个读端口	较高	较高	高计算密度、适合规约运算	关键路径延迟、并行度受限
MIMO	多输入多输出	复杂计算模式(大规模并行运算)	最高	需要多个读写端口或特殊硬件机制	最高	最高	最大并行度、性能提升显著	端口资源约束、识别复杂度高

2.1 单输入单输出指令

SISO 指令是最基本的专用指令类型,它将单个输入数据通过一系列操作转换为单个输出结果。从数据流图的角度看,SISO 模式表现为一条从输入节点到输出节点的计算路径,中间可能包含多个计算节点。这种模式的主要特点是计算路径清晰、数据依赖简单。同时,在实现方面,SISO 指令对处理器架构的要求最低,只需要一个读端口和一个写端口

即可支持。这种简单性使得 SISO 指令特别适合在资源受限的嵌入式处理器中实现。

然而,SISO 指令的性能提升潜力相对有限,因为它们无法利用指令级并行性,也难以封装复杂的计算模式。在实际应用中,SISO 指令常用于实现基本的数据变换操作,如数字信号处理中的三角函数[12]、滤波器[2]、密码学中的数据替换[13]等。这类指令虽然加速效果有限,但实现简单、应用广泛。

基于模板匹配的识别方法是最常用的方法之一[14-16]。在这个过程中,首先需要构建指令模板库,包含预定义的常见计算模式,如基本运算和访存操作序列。识别过程通过图形态学分析,在数据流图中寻找符合 SISO 特征的子图结构,然后将这些候选子图与模板库中的模式进行匹配度评估。

2.2 单输入多输出指令

SIMO 指令能够将单个输入数据转换为多个相关的输出结果,其在数据转换与分解的场景中具有天然的优势。这类指令的特点与其计算特征密切相关。在功能实现上,多输出的特性使其能够有效降低数据访问开销,提高计算密度。通过在一次取数的基础上进行多路计算,可以显著提升指令的计算效率。同时,SIMO 指令的多个输出往往来自对同一输入数据的不同处理,因此通常具有较好的局部性,这种特性有助于减少存储访问和提高缓存利用率。

然而,SIMO 指令的实现也面临着挑战。在微架构层面,需要解决多个输出结果的并行写回问题。寄存器的写端口数量往往是有限的,这就要求在设计时考虑多周期写回或流水线化写回等机制。在指令编码方面,需要在有限的指令字长内编码多个目标寄存器的信息,这增加了指令格式设计的复杂度。此外,指令的多个输出之间可能存在数据依赖关系,这需要在调度时特别注意以保证正确性。

在应用场景方面,这类指令特别适合实现一次读取多次计算的运算模式,例如图像信号处理中的颜色信息变换[17]、密码学应用中从一个种子生成多个密钥[18]等场景。

单输入多输出指令的识别主要关注数据流图中的扇出结构。根据文献[19]的研究,SIMO 模式的识别可以通过分析数据流图中由一个根节点及其所有后继节点构成的子图来实现。这种方法首先识别具有多个输出边的节点作为潜在的根节点,然后分析其后继路径构成的计算模式。

2.3 多输入单输出指令

MISO 指令能够将多个输入数据通过复杂的组合运算转换为单一输出,因此这类指令有较高的计算密度,这种特性使得这类指令特别适合实现复杂的规约运算,能够有效减少中间结果的存储和传输开销。然而在实现层面,MISO 指令对处理器的读端口提出了较高要求,但由于只有单一输出,写回阶段的实现相对简单。

MISO 指令也面临一些固有的限制。最主要的是单一输出的约束可能无法充分利用现代处理器的并行处理能力。在实际应用中,很多计算密集型操作会产生多个相关的输出结果,这种场景下 MISO 指令的加速效果就会受到影响。此外,当输入数量较多时,可能会出现关键路径延迟增加的问题,这需要在指令设计时特别注意时序约束的处理。相比 SISO 和 SIMO 指令,MISO 指令的设计空间更加复杂。输入数量的增加不仅带来了组合关系的指数增长,也增加了硬件实现的复杂度。为了控制设计复杂度,研究人员提出了多种优化策略。例如,通过分析输入数据的相关性来优化数据通路[10],或者利用流水线技术[20]来平衡延迟和吞吐率。

MISO 指令多输入单输出的特征特别适合实现数据归约和聚合操作,在密码学[21]、信号处理[22]等领域有广泛应用。

关于 MISO 指令设计识别的研究已经相当深入。近期的工作主要集中在 MaxMISO 模式的识别上,这类模式的特点如图 3 所示,这种不能被其他 MISO 完全包含的 MISO 模式具有独特的优势:确保指令能够捕获到最完整的计算模式,从而最大程度地发挥硬件加速的潜力。此外,MaxMISO 之间不能部分重叠的特性也简化了识别算法的实现,使其能够在线性时间内完成模式识别。Pozzi 等人[23]提出的算法能在线性时间内枚举最大 MISO 模式,通过利用数据流图的拓扑排序特性,自底向上构建最大规约模式,很大程度上提高了识别效率。随后,Galuzzi 等人[24]提出的变长 MISO 构建技术进一步

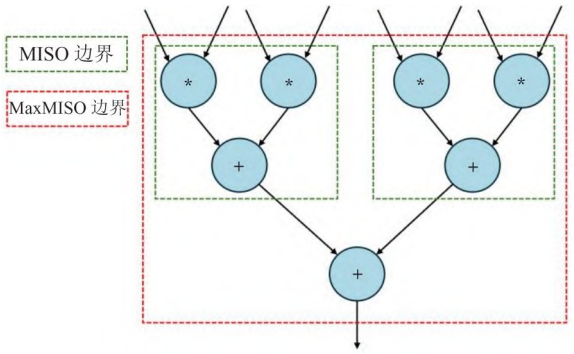

图 3　MaxMISO 与 MISO 的关系

扩展了这一思路,通过引入启发式方法,在保持识别效率的同时提供了更灵活的模式构建能力。

2.4　多输入多输出指令

MIMO 指令是专用指令中最复杂但也最具潜力的类型,这类专用指令能够同时处理多个输入数据并产生多个输出结果,并通过封装更复杂的计算模式和更高的并行度,如矩阵计算[25],提供更显著的性能提升。

然而,MIMO 指令的实现层面面临着挑战。最主要的问题是输入输出端口数量的实现约束,尤其在精简指令集处理器中表现得尤为突出。为此,研究人员提出了多种创新性解决方案:如采用流水寄存器对 I/O 串行化等方法来提高端口利用效率、使用状态寄存器来缓存中间结果[10,26]等。

MIMO指令的识别也面临巨大挑战。对于包含 $|V|$ 个节点的数据流图,在考虑输入输出约束的情况下,可能的模式数量仍然高达 $|V|^{In_{\max}+Out_{\max}}$,其中 $In_{\max}$ 和 $Out_{\max}$ 分别代表专用指令的扇入和扇出的实现约束。为了应对这种设计空间的爆炸性增长,研究人员提出了多种高效的枚举算法。Atasu 等人[27]提出了一种在凸性和输入输出约束下枚举所有模式的算法,通过反向拓扑排序和二叉树搜索来探索设计空间。该算法利用输出数量的单调性特征进行剪枝,从而显著提高了搜索效率。后续研究引入了基于永久输入的剪枝准则[28],进一步缩小了搜索空间。

MaxMIMO 在 MIMO 的基础上,通过放松输入和输出约束来获得更好的性能提升。Pothineni 等人[29]的研究表明,与受限的 MIMO 指令相比,MaxMIMO 能够提供多达 50% 的额外性能提升。他们将最大凸子图枚举问题转化为最大独立集问题,提出了一种两步策略来高效识别所有最大凸子图。虽然该问题已被证明为非确定性多项式时间(nondeterministic polynomial time,NP)完全问题,但通过引入群组不相容图等技术,该方法在实际应用中表现出良好的效果。

近期的研究更加关注 MIMO 指令的实用性优化。Yu 等人[19]提出了基于锥的模式识别方法,通过合并相邻的上锥和下锥来构建连通模式。Chen 等人[6]则开发了一种策略性选择机制,通过评分系统来确定节点的优先级,并使用融合和拆分函数来维护凸性约束。这些方法在保证识别效率的同时,也考虑了操作模式的实现可行性。

基于整数线性规划(integer linear programming,ILP)的方法为 MIMO 指令识别提供了新的思路。这类方法将问题形式化为优化问题,目标函数通常包括执行时间的减少、硬件开销的控制等多个方面。特别是在考虑通信开销的情况下,ILP 方法能够更准确地评估不同模式的实际价值[30]。这种基于数学优化的方法虽然计算复杂度较高,但能够获得更优的解决方案。

3　专用指令设计

专用指令设计的核心目标是从候选指令中选择一个子集,在满足硬件资源或功耗约束的前提下实现最大的性能提升。这个过程可以形式化描述为:给定一组图 $G = \{G_1, G_2, \cdots, G_n\}$ 表示给定应用的基本块的数据流图,以及在上一阶段识别出的一组专用指令候选 $C = \{C_1, C_2, \cdots, C_m\}$,选择 C 的一个子集 S,使得 $P(S)$ 最大化,其中 $P()$ 为优化目标函数,如性能,同时满足非重叠性、无环性和成本约束 $Cost(S) \leqslant C_{\max}$,其中,$Cost(S)$ 表示所选专用指令子集 (S) 的总实现成本,$C_{\max}$ 表示系统允许的最大成本预算,例如硬件面积、功耗或实现复杂度上限。为了解决这个复杂的优化问题,需要建立准确的代价模型并采用高效的搜索方法。

3.1　代价模型

代价模型是专用指令设计中的关键组成部分,

它直接影响着指令选择的质量和最终的实现效果。

3.1.1 面积模型

常见的面积评估模型主要有 4 类方法,分别为基于门级网表的精确评估模型、基于操作粒度的估算模型、基于模板匹配的预测评估模型和基于机器学习的预测评估模型,它们之间的对比如表 2 所示。

表 2 不同面积评估模型对比

评估方法	原理	精度	效率	主要优势	主要局限性	适用阶段
基于门级网表的精确评估模型	通过完整的逻辑综合流程获得门级网表和面积指标	最高	最低	结果最接近实际实现;可获得详细时序和功耗信息	评估时间长;依赖具体工艺库;需要完整 RTL 描述	后期验证阶段
基于操作粒度的估算模型	将指令分解为基本操作,通过预标定模型计算	中等	最高	评估效率高;支持操作共享优化;适合早期筛选	可能忽略优化机会;难以反映时序相关性	早期设计阶段
基于模板匹配的评估模型	通过与已有设计案例对比来估算面积	较高	高	快速获得较准确结果;特别适合常见操作	依赖模板库质量;难以处理新结构	设计探索阶段
基于机器学习的预测评估模型	利用深度学习模型预测面积	高	较高	预测精度高;可考虑复杂特征关系	需要大量训练数据;可解释性差;泛化能力受限	设计探索阶段

基于门级网表的精确评估模型[31]。该方法首先使用寄存器传输级(register transfer level,RTL)硬件描述语言(如 Verilog 等)实现带有专用指令功能的处理器,然后通过完整的逻辑综合流程并获得门级网表,最终得到准确的面积数据。典型的工作如指令集体系结构描述语言(language for instruction-set architecture,LISA)通过商业综合工具进行评估[32]。该方法的优势在于可以获得最接近实际实现的面积估算,同时还能得到详细的时序信息和功耗数据。然而,这种方法也存在明显的局限性:首先,完整的综合流程通常需要数分钟到数小时不等,这在需要评估大量候选指令的场景下是难以接受的;其次,综合结果高度依赖于具体的工艺库和综合工具的优化策略,使得评估结果难以跨平台复用;此外,在早期设计阶段,往往缺乏完整的 RTL 描述,这限制了该方法的适用范围。

基于操作粒度的估算模型。为了克服门级评估的效率问题,研究者提出了基于操作复杂度的快速估算方法。这类方法的核心思想是将复杂的专用指令分解为基本操作(如加法、乘法、移位等),然后通过预先标定的面积模型计算总体开销。Witte 等人[33]提出的快速估算方法建立了详细的操作面积数据库。此外,该方法引入了操作共享机制,通过识别指令中可重用的功能单元来优化面积估算。例如,当多个操作可以复用同一个加法器时,总面积会相应减少。这种方法显著提高了评估效率,使得在早期设计阶段实现快速筛选成为可能。然而,该方法也面临着精度挑战:首先,实际电路优化往往能够发现更多的资源共享机会,导致估算结果偏保守;其次,操作之间的时序相关性可能影响实际实现的面积,这在简化模型中难以准确反映。

基于模板匹配的评估模型[34]。这种方法试图通过与已有设计案例的对比来预测新指令的面积开销。具体来说,在评估专用指令的面积时,首先将其分解为基本块,然后通过模式匹配找到库中最相似的结构,并根据参数差异进行调整[14,16]。这种方法的优势在于可以快速获得较为准确的估算结果,特别是对于常见的数字信号处理操作。为了进一步提高评估的准确性,可进一步采用优化方法,如:(1)图同构算法,用于识别结构相似性;(2)参数插值方法,处理位宽等可调参数的影响;(3)组合估算策略,处理复杂指令的拆分和重组。然而,该方法的效果强烈依赖于模板库的质量和完备性。对于全新的专用指令,可能找不到合适的参考模板,导致预测精度下降。此外,模板库的维护和更新也是一个持续的挑战,需要随着工艺演进和设计方法的发展不断扩充。

基于机器学习的预测评估模型。Wang 等人[35]提出了一种基于深度神经网络的面积预测模型。该模型首先将专用指令表示为特征向量,包括操作类型分布、数据依赖关系、关键路径长度等特征,然后通过多层感知机进行预测。为了提高模型的鲁棒

性,作者采用了大量综合实验数据进行训练,并引入了正则化技术来防止过拟合。实验表明,该方法在各种基准测试集上都取得了优于传统估算方法的预测精度。Brumar 等人[36]利用高层综合工具和周期精确仿真获取训练数据,然后构建预测模型来指导早期设计空间探索。近年来,有研究基于图神经网络针对网表进行面积预测,并取得了一定的效果[37]。

然而,机器学习方法也面临一些挑战。首先是训练数据的获取难度,需要大量的实际综合结果才能构建可靠的模型。其次是模型的可解释性问题,难以直观理解预测结果的依据。此外,当应用场景与训练数据差异较大时,模型的泛化能力也面临考验。

3.1.2 功耗建模方法

功耗建模方法主要包括基于 RTL 级仿真的精确模型、基于指令级能耗特征的估算模型和基于微架构级别的模拟模型 3 类,它们之间的对比如表 3 所示。

表 3 不同功耗评估模型对比

评估方法	原理	精度	效率	主要优势	主要局限性	适用场景
基于 RTL 级仿真的精确模型	通过完整的 RTL 仿真和功耗分析流程,结合工艺库数据计算	高	低	能准确评估动态和静态功耗;可获得瞬时功耗信息;结果最接近实际实现	需要完整 RTL 设计;需要详细工艺信息;仿真过程耗时长	后期验证阶段
基于指令级能耗特征的估算模型	分析指令执行过程中的动态特性,建立能耗特征模型	低	高	实现相对简单;适合早期评估;能反映指令级行为特征	采用较多简化假设;难以反映微架构事件;精度受数据相关性影响	指令设计探索阶段
基于微架构级别的模拟模型	在架构级建立处理器各部件的解析模型	中	中	能评估架构决策影响;评估速度较快;考虑微架构特征	依赖工艺参数标定;模型精度有限;难以反映细粒度优化	架构设计阶段

基于 RTL 级仿真的精确模型。这是获取最准确功耗数据的方法。在 RTL 级别,设计的结构和行为都有了明确的描述,通过综合工具可以得到更接近实际的门级网表。主流电子设计自动化(electronic design automation,EDA)工具如 Synopsys 的 PrimeTime PX 和 Cadence 的 Joules RTL 等提供了完整的 RTL 功耗分析流程:首先进行逻辑综合得到网表,然后通过时序仿真获取详细的信号切换信息,最后结合工艺库中的功耗特征数据计算精确的功耗值。这种方法不仅能准确评估动态功耗,还能分析静态功耗和瞬时功耗。但由于需要完整的 RTL 设计和详细的工艺信息,且仿真过程较为耗时,因此该方法主要用于后期验证阶段而不适合早期评估。

基于指令级能耗特征的估计模型。这种方法从指令执行的微观过程入手分析功耗特征。Lee 等人[38]提出了一种能量效率评估方法,通过分析专用指令在执行过程中的动态特性来预测功耗。Lin 和 Fei[39]研究了考虑流水线资源共享场景下的功耗估算问题。进一步地,研究者提出了一种层次化的建模方法[40],将指令的能耗分解为基础能耗和状态转换能耗两部分。其中基础能耗由指令类型和操作数特征决定,而状态转换能耗则与指令序列和数据相关性有关。通过建立指令能耗特征库,结合实际程序的指令统计信息,可以估算应用程序的总体功耗。这种方法实现相对简单,特别适合早期设计空间探索。但由于采用了较多简化假设,难以准确反映流水线停顿、缓存缺失等微架构级事件的功耗影响。

基于微架构级别的模拟模型。这类方法在较高抽象层次对处理器功耗进行建模和评估。McPAT 工具[41]提供了一个集成的建模框架,能够同时评估功耗、面积和时序特征。它将处理器划分为计算核心、缓存层次和片上互连等主要部件,针对每个部件建立了考虑工艺参数的解析模型。通过配置不同的微架构参数,设计者可以快速评估架构决策对功耗的影响。类似地,Wattch[42]提供了另一个架构级功耗分析框架,其特别关注了动态功耗和时钟网络功耗的建模。这类方法的优势在于能够在保持一定精度的同时提供较快的评估速度,支持早期架构探索。但模

型的准确性很大程度上依赖于工艺参数的标定。

3.1.3 性能建模方法

性能建模方法[43]主要包括基于 RTL 仿真的精确模型、基于指令周期统计的简单模型、考虑架构实现细节的性能模型和基于统计采样的性能预测模型 4 类,它们之间的对比如表 4 所示。

表 4 不同性能建模方法对比

建模方法	原理	精度	效率	主要优势	主要局限性	适用场景
基于 RTL 仿真的精确模型	在 RTL 层面仿真目标应用,获得精确性能指标	最高	最低	周期精确;最接近最终实现结果	速度慢;需要得到完整 RTL 设计	高度精确的性能建模、验证
基于指令周期统计的简单模型	将执行时间分解为基本操作延迟的组合	低	最高	实现简单;评估速度快;易于集成	难以反映现代处理器特性;忽略微架构影响;精度有限	快速原型评估
考虑架构实现细节的性能模型	精确模拟处理器微架构组件行为	高	低	高精度性能预测;支持复杂架构特性;可获得详细行为信息	模拟开销大;配置复杂;需要详细参数	详细性能分析和验证
基于统计采样的性能预测模型	通过采样代表性程序片段预测整体性能	中高	高	评估效率高;可平衡精度和效率;支持大规模评估	依赖采样策略;可能遗漏关键行为;需要合理的采样配置	大规模应用/复杂架构的快速性能建模

基于 RTL 仿真的精确模型。基于 RTL 仿真的精确模型通过完整的寄存器传输级描述,实现了对处理器行为和性能的精确建模。这种方法在 RTL 级别对处理器的微架构行为进行了详细描述,包括流水线状态、资源竞争、数据相关等关键特性。通过对指令执行过程的周期精确仿真,可以准确捕获处理器的动态行为,如分支预测、缓存访问、指令重排等复杂特征。虽然这种方法能够提供最接近实际的性能评估结果,但由于需要完整的 RTL 实现和大量的仿真时间,因此其主要用于后期的性能验证阶段,而不适合早期的设计空间探索。

基于指令周期统计的简单模型。基于指令周期统计的简单模型主要代表是 SimpleScalar 中的 sim-fast 模拟器。这类方法的核心思想是将应用执行时间分解为基本操作延迟的组合。这种方法在 20 世纪 90 年代得到广泛应用,例如 IMPACT 编译器项目[44]就采用类似的性能估计方法来指导优化。这种方法虽然效率高,但这种简化模型难以准确刻画现代复杂处理器的行为,例如多发射、乱序执行、缓存和分支行为等。

考虑架构实现细节的性能模型。这类方法以 Gem5[45]为代表。这类模型通过精确模拟处理器微架构来评估性能,包括流水线行为、缓存系统、分支预测等核心部件。在流水线模拟方面,它详细模拟了取指、译码、执行等各个阶段,能够准确反映资源竞争和数据相关的影响。在内存层次方面,它集成了多级缓存模型和内存访问模型。为了提高模拟效率,研究者开发了多种优化技术:PTLsim[46]引入事件驱动机制减少不必要的计算;Multi2Sim[47]提供并行模拟框架提升性能。这类方法虽然能提供高精度的性能预测,但模拟开销大,配置和调优较为复杂。

基于统计采样的性能预测模型。SMARTS[48]通过对程序执行过程进行系统采样来预测整体性能,从而实现通过少量的架构级模拟或 RTL 级模拟就能估计整个应用的性能。SimPoint[49-50]提出了基于程序阶段的采样方法,显著提高了效率。为了确保采样的代表性,研究者开发了多种采样策略:系统采样保证覆盖率,分层采样提高效率,自适应采样能够根据程序行为动态调整采样密度。这种方法特别适合早期设计空间探索,但样本的选择对预测准确性有重要影响。

3.2 搜索方法

专用指令选择的搜索方法可以分为 3 类:基于数学规划的精确方法、基于启发式搜索方法和基于机器学习的搜索方法,每类方法都有其特定的应用场景和优势,它们之间的对比如表 5 所示。

表 5　不同搜索方法对比

搜索方法	原理	主要算法	优势	局限性	适用场景
基于数学规划的精确方法	将问题形式化为数学优化模型,通过严格求解获得全局最优解	ILP;约束规划;分支定界	保证全局最优解;可处理复杂约束;结果可解释性强	求解时间随问题规模指数增长;难以处理非线性约束;内存开销大	小规模问题;对最优性要求高的场景
基于启发式的搜索方法	通过模拟自然进化或物理过程来探索解空间	遗传算法;蚁群算法;模拟退火算法	可处理大规模问题;计算效率高;易于实现和调整	无法保证最优解;结果依赖参数配置;可能陷入局部最优	大规模问题;对求解时间敏感的场景
基于强化学习的搜索方法	通过与环境交互学习最优决策策略	Q-learning;SARSA	具有良好泛化能力;避免局部最优;可持续优化	需要较长训练时间;状态空间设计复杂;依赖训练数据质量	复杂决策问题;需要自适应优化的场景

基于数学规划的精确方法。数学规划方法将专用指令选择问题形式化为优化问题,通过严格的数学推导寻找全局最优解。整数线性规划是这类方法中应用最广泛的技术。Atasu 等人[51]提出将指令选择建模为 ILP 问题,通过设置二进制决策变量表示模式的选择状态,并将面积约束、非重叠约束等转化为线性约束条件。这种方法的优势在于能够保证解的最优性,并且可以灵活地处理各种约束条件。

基于启发式搜索方法。启发式方法通过模拟自然过程来探索解空间。遗传算法在这类方法中应用最为广泛,它通过编码、交叉、变异等操作来迭代解的种群。Pozzi 等人[28]设计的多目标遗传算法能够同时优化性能提升和硬件开销,在大规模问题中表现出色。蚁群算法也被用于专用指令搜索,Wang 等人[52]的研究表明,蚁群算法在解决指令选择问题时,能够在较短时间内找到接近最优的解。也有研究采用模拟退火算法,其通过控制“温度”参数来平衡探索与利用[52]。这类方法的共同特点是能够处理大规模问题,并在计算时间和解质量之间取得良好平衡。

基于强化学习的搜索方法。强化学习作为机器学习的重要分支,近年来在专用指令选择问题上展现出独特优势。与传统启发式方法相比,强化学习能够通过持续与环境交互来学习最优策略,这种自适应学习机制使其在复杂决策问题中表现出色。

专用指令选择问题可以自然地建模为马尔可夫决策过程。在这个框架下,状态空间表示当前已选择的专用指令集合,动作空间对应对候选指令进行选择或拒绝的决策,状态转移函数根据选择的指令更新当前指令集合,而奖励函数则量化所选指令带来的性能提升[53-54],基于强化学习方法实现的专用指令搜索过程如图 4 所示。在实际求解过程中,强化学习智能体通过多轮训练来优化其决策策略。每轮训练中,智能体基于当前状态选择动作,获得即时奖励并转移到新状态,直到完成整个数据流图的覆盖。这种迭代学习机制使得智能体能够不断改进其决策策略,避免陷入局部最优解。

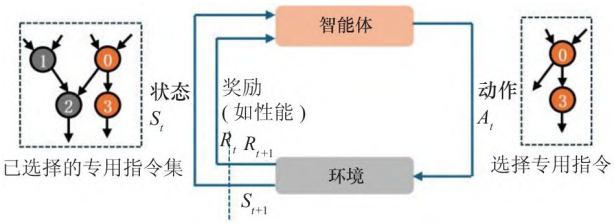

图 4　基于强化学习方法实现的专用指令搜索过程

研究表明,多种强化学习算法在专用指令选择问题上都取得了良好效果[54],如 Q-learning、状态-动作-奖励-状态-动作(state-action-reward-state-action,SARSA)等。与模拟退火算法等传统启发式方法相比,强化学习方法在指令选择的性能提升方面展现出明显优势。这主要得益于其能够从历史决策中学习并持续优化策略。然而,由于需要较长的训练过程,强化学习方法在运行时间上相对传统方法略显劣势。

强化学习方法的另一个重要特点是其良好的泛化能力。通过合理设计状态表示和奖励机制,训练好的模型可以应用于不同规模和结构的数据流图,这为专用指令选择问题提供了一个可扩展的解决

方案。

4 专用指令实现优化

4.1 资源开销优化

专用指令的资源优化主要致力于降低硬件开销和静态功耗。研究表明,虽然专用指令能够有效降低执行时间和动态能耗,但其引入的静态功耗可能会抵消部分节能效果。此外,在多应用场景下,为每个应用开发独立的专用指令会导致硬件成本过高。因此,资源开销优化的核心在于挖掘专用指令中操作的复用、减小硬件实现的冗余,而同构性检测[36,55]是实现这一目标的重要技术。

Ahn 和 Choi [56-57]提出了基于图规范化的方法,通过构建规范形式来替代传统的同构性检测方法,将算法复杂度从 $O(n2)$ 降低到 $O(n)$。Clark 等人[58]采用分支定界算法来选择最具前景的重复指令集。Bonzini 和 Pozzi 提出的多阶段策略则将问题分解为枚举、同构检测和图覆盖 3 个子任务[59],通过专门的算法分别求解。然而,同构性检测也面临一些挑战。首先,指令同构性检测本身是一个 NP 完全问题[60],计算开销较大。其次,在许多应用中,完全同构的计算模式较少,限制了这种方法的适用范围。因此,需要探索更灵活的资源优化方案。

4.2 搜索性能优化

提高自动设计过程并行度。为了提高搜索速度,部分研究考虑对专用指令自动化设计过程的某些阶段进行并行优化,例如在进行专用指令识别时,Xiao 等人[61]提出的主从式并行模型,该方法指定一台计算机作为主节点,负责任务分配和结果收集,而从节点负责具体的计算任务。通过将应用图 G 划分为 n 个复杂度相当的子分区 $G_1, G_2, \cdots, G_n$, 每个计算节点 i 负责识别其分配子分区 G_i 中的子图。这种方法在实验中实现了接近 3.7 倍的线性加速比。然而,随着计算节点数量增加,该方法面临负载不均衡的问题。为此,Wang 等人[35]提出了多层次图分割策略,引入性能估计模型来模拟计算节点的运行时间,当检测到负载不均衡时,将高估计运行时间的子问题进一步划分,直到计算节点运行时间差异低于指定阈值。实验表明,这种多层次分割方法显著改善了负载均衡性。

设计空间剪枝[36]。除了提高搜索算法的并行度,通过剪枝方法提前排除无效的专用指令选项或给出早期的设计建议同样可以提高专用指令自动设计过程的效率。例如早期评估剪枝方法可借助机器学习方法实现:使用多层感知器(multi-layer perceptrons,MLP)等机器学习模型,对候选加速器进行面积和性能的早期评估。通过静态分析代码特征,如低级虚拟机(low level virtual machine,LLVM)中间表示指令的类型和数量以及动态分析信息(如执行频率)作为模型输入,对于预测不能带来面积收益的合并方案提前剪枝,可以将评估时间从 22 h 降低到 10 s 以内;此外,在考虑专用指令在应用内复用时通过对操作序列使用序列对齐算法(如 Needleman-Wunsch 算法)计算目标数据流图的相似度,对相似度低于阈值(如 5%)的函数对直接剪枝,避免无效的指令专用化尝试。实验表明这种方法可以有效减少约 60% 候选方案的探索。

5 结 论

专用指令设计经历了从手工到自动化再到自动化的演进过程。早期主要依赖设计人员的经验进行手工设计,随着应用需求的增长和设计复杂度的提升,自动化设计方法逐渐成为主流。特别是随着 RISC-V 等开源指令集架构的兴起,专用指令集处理器有了更广泛的应用场景。

在专用指令设计过程中,关键挑战主要体现在 3 个方面:首先是设计空间的急剧膨胀,即使采用各种约束条件,可行解的数量仍然庞大,这对自动化设计方法提出了很高的要求;其次是多维度约束的权衡,需要在性能提升、硬件开销、指令粒度等多个维度之间寻找平衡点;最后是设计评估的复杂性,如何在早期设计阶段准确评估不同方案的性能、功耗和面积特征,是确保设计质量的关键问题。这些挑战的存在推动了相关研究的深入开展,促进了新方法和新技术的不断涌现。

为了应对上述挑战,研究者们提出了一系列解决方案。针对设计空间膨胀的问题,通过引入机器学习辅助的设计空间剪枝和多层次的并行化搜索策略,显著提升了专用指令识别和选择的效率。在权

衡性能提升和硬件开销方面,有 SISO 到 MIMO 4 类具有不同性能、开销特性的指令类型可供选择。为了解决设计评估的复杂性,研究者构建了包含面积、功耗和性能的综合评估体系,并通过机器学习等方法提高了早期评估的准确性,为设计决策提供了可靠的依据。

展望未来,专用指令自动设计呈现出 3 个重要的发展趋势:智能化、跨层次优化和注重可重构与灵活性。在智能化方面,机器学习等方法的引入不仅将提高设计空间探索的效率,还能帮助代价模型更好地在准确性和效率之间取得平衡,其语义理解能力有望在未来为专用指令识别自动化方面帮助发现更优的设计方案。在跨层次优化方面,未来的设计方法将更加注重编译器、微架构和电路实现的协同设计,通过建立端到端的自动化设计流程,实现各个抽象层次之间的优化协同。在可重构与灵活性方面,专用指令将具备更强的动态适应能力,能够根据不同应用场景灵活调整,同时提高指令级的复用效率,从而在保持性能优势的同时提升设计的适用范围。

参考文献

[1] 刘畅,武延军,吴敬征,等. RISC-V 指令集架构研究综述[J]. 软件学报,2021,32(12):3992 - 4024.

[2] Shahabuddin S, Mämmelä A, Juntti M, et al. ASIP for 5G and beyond: opportunities and vision [J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2021,68(3):851 - 857.

[3] Paulin G, Andri R, Conti F, et al. RNN-based radio resource management on multicore RISC-V accelerator architectures[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2021,29(9):1624 - 1637.

[4] Valente L, Nadalini A, Veeran A H C, et al. A heterogeneous RISC-V based SoC for secure nano-UAV navigation[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2024,71(5):2266 - 2279.

[5] Bouwens F, Huisken J, De Groot H, et al. A dual-core system solution for wearable health monitors[C]// Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI. New York, USA: ACM, 2011:379 - 382.

[6] Chen X, Maskell D L, Sun Y. Fast identification of custom instructions for extensible processors [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2007,26(2):359 - 368.

[7] Lattner C, Adve V. LLVM: a compilation framework for lifelong program analysis and transformation[C]// International Symposium on Code Generation and Optimization. San Jose, USA: IEEE, 2004:75 - 86.

[8] Lee C, Potkonjak M, Mangione-Smith W H. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems[C]// Proceedings of 30th Annual International Symposium on Microarchitecture. Research Triangle Park, USA: IEEE, 1997:330 - 335.

[9] Cong J, Fan Y, Han G, et al. Application-specific instruction generation for configurable processor architectures[C]// Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays. New York, USA: ACM, 2004:183 - 189.

[10] Cong J, Fan Y, Han G, et al. Instruction set extension with shadow registers for configurable processors[C]// Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays. New York, USA:ACM, 2005:99 - 106.

[11] Yu P, Mitra T. Scalable custom instructions identification for instruction-set extensible processors[C]// Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. New York, USA: ACM, 2004:69 - 78.

[12] 李飞,郭绍忠,郝江伟,等. 面向 RISC-V 的基础数学库实现[J]. 电子学报,2024,52(5):1633 - 1647.

[13] Pan L, Tu G, Liu S, et al. A lightweight AES coprocessor based on RISC-V custom instructions[J]. Security and Communication Networks, 2021(1): 9355123.

[14] Han M, Liu L, Wei S. A graph covering method for template based system partition[C]// 2008 International Conference on Communications, Circuits and Systems. Xiamen, China: IEEE, 2008:1374 - 1377.

[15] Cheung N, Henkel J, Parameswaran S. Rapid configuration and instruction selection for an ASIP: a case study[C]// Automation and Test in Europe Conference and Exhibition 2003 Design. Munich, Germany: IEEE, 2003:802 - 807.

[16] Liem C, May T, Paulin P. Instruction-set matching and selection for DSP and ASIP code generation[C]// Proceedings of European Design and Test Conference EDAC-ETC-EUROASIC. Paris, France: IEEE, 1994:31 - 37.

[17] Fanucci L, Cassiano M, Saponara S, et al. ASIP design and synthesis for non linear filtering in image processing[C]// Proceedings of the Design Automation and Test in Europe Conference. Munich, Germany: IEEE, 2006:1 - 6.

[18] Tsekoura I, Selimis G, Hulzink J, et al. Exploration of cryptographic ASIP designs for wireless sensor nodes[C]// 2010 17th IEEE International Conference on Electronics, Circuits and Systems. Athens, Greece: IEEE, 2010:827 - 830.

[19] Yu P, Mitra T. Disjoint pattern enumeration for custom instructions identification[C]// 2007 International Conference on Field Programmable Logic and Applications. Amsterdam, Netherlands: IEEE, 2007:273 - 278.

[20] Jayaseelan R, Liu H, Mitra T. Exploiting forwarding to improve data bandwidth of instruction-set extensions[C]// Proceedings of the 43rd Annual Design Automation Conference. New York, USA: ACM, 2006:43 - 48.

[21] Hu J, Dai W, Yao L, et al. An application specific instruction set processor (ASIP) for the niederreiter cryptosystem[C]//2018 6th International Symposium on Digital Forensic and Security. Antalya, Turkey:IEEE,2018:1-6.
[22] Shen Z, He H, Zhang Y, et al. A video specific instruction set architecture for ASIP design[J]. VLSI Design, 2007(1): 058431.
[23] Pozzi L, Vuletic M, Ienne P. Automatic topology-based identification of instruction-set extensions for embedded processors[C]//Automation and Test in Europe Conference and Exhibition Proceedings 2002 Design. Paris, France: IEEE, 2002:1138-1144.
[24] Galuzzi C, BertelS K, Vassiliadis S. A linear complexity algorithm for the generation of multiple input single output instructions of variable size [M]. Berlin: Springer, 2007:283-293.
[25] 郭玉石,黄骏,周晓军,等.基于 RISC-V 架构的矩阵卷积计算方法,接口,协处理器及系统:201910125953[P]. 2019-06-07.
[26] 王明登,严迎建,郭朋飞,等. 基于 RISC-V 指令扩展方式的国密算法 SM2,SM3 和 SM4 的高效实现[J]. 电子学报, 2024,52(8):2850-2865.
[27] Atasu K, Mencer O, Luk W, et al. Fast custom instruction identification by convex subgraph enumeration[C]// 2008 International Conference on Application-Specific Systems, Architectures and Processors. Leuven, Belgium: IEEE, 2008:1-6.
[28] Pozzi L, Atasu K, Ienne P. Exact and approximate algorithms for the extension of embedded processor instruction sets[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,2006,25(7):1209-1229.
[29] Pothineni N,Kumar A,Paul K. Application specific datapath extension with distributed I/O functional units[C]// The 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems. Bangalore, India: IEEE, 2007:551-558.
[30] Atasu K, Dimond R G, Mencer O, et al. Optimizing instruction-set extensible processors under data bandwidth constraints[C]//Automation and Test in Europe Conference and Exhibition 2007 Design. Nice, France: IEEE, 2007:1-6.
[31] Atasu K, Ozturan C, Dundar G, et al. CHIPS: custom hardware instruction processor synthesis[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2008,27(3):528-541.
[32] Pees S, Hoffmann A, Zivojnovic V, et al. LISA—machine description language for cycle-accurate models of programmable DSP architectures[C]//Proceedings of the 36th Annual ACM/IEEE Design Automation Conference. New Orleans, USA: ACM, 1999:933-938.
[33] Witte E M, Chattopadhyay A, Schliebusch O, et al. Applying resource sharing algorithms to ADL-driven automatic ASIP implementation[C]//2005 International Conference on Computer Design. San Jose,USA: IEEE, 2005: 193-199.
[34] Martinez A F, Kuchcinski K. Graph matching constraints for synthesis with complex components[C]// The 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools. Lubeck, Germany: IEEE, 2007:288-295.
[35] Wang S, Xiao C, Liu W. Parallel enumeration of custom instructions based on multidepth graph partitioning[J]. IEEE Embedded Systems Letters, 2019,11(1):1-4.
[36] Brumar I, Zacharopoulos G, Yao Y, et al. Early DSE and automatic generation of coarse-grained merged accelerators[J]. ACM Transactions on Embedded Computing Systems, 2023,22(2):1-29.
[37] 田春生, 陈雷, 王源, 等. 基于图神经网络的电子设计自动化技术研究进展[J]. 电子与信息学报, 2023, 45(9):3069-3082.
[38] Lee J, Choi K, Dutt N D. Energy-efficient instruction set synthesis for application-specific processors[C]// Proceedings of the 2003 International Symposium on Low Power Electronics and Design. New York,USA: ACM, 2003:330-333.
[39] Lin H, Fei Y. Resource sharing of pipelined custom hardware extension for energy-efficient application-specific instruction set processor design[J]. ACM Transactions on Design Automation of Electronic Systems, 2012, 17(4):1-20.
[40] Brock B, Rajamani K. Dynamic power management for embedded systems[C]//IEEE International [Systems-on-Chip] SOC Conference. Portland, USA: IEEE, 2003: 416-419.
[41] Li S, Ahn J H, Strong R D, et al. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures[C]// Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. New York, USA:ACM, 2009:469-480.
[42] Brooks D,Tiwari V,Martonosi M. Wattch:a framework for architectural-level power analysis and optimizations[J]. Proceedings of 27th International Symposium on Computer Architecture. Vancouver, Canada: IEEE, 2000,28(2): 83-94.
[43] 史惠康,王泽胜,张士宗,等. 通用 CPU 性能基准测试研究综述[J].电子学报, 2023,51(1):246-256.
[44] Chang P P, Mahlke S A, Chen W Y, et al. IMPACT: an architectural framework for multiple-instruction-issue processors [J]. ACM SIGARCH Computer Architecture News, 1991,19(3):266-275.
[45] Binkert N, Beckmann B, Black G, et al. The gem5 simulator[J]. ACM SIGARCH Computer Architecture News, 2011,39(2):1-7.
[46] Yourst M T. PTLsim: a cycle accurate full system x86-64 microarchitectural simulator[C]//2007 IEEE International Symposium on Performance Analysis of Systems and Software. San Jose, USA: IEEE, 2007:23-34.
[47] Ubal R, Jang B, Mistry P, et al. Multi2Sim: a simulation framework for CPU-GPU computing[C]//Proceedings of the 21st International Conference on Parallel Ar-

chitectures and Compilation Techniques. New York, USA: ACM, 2012:335 - 344.
[48] Wunderlich R E, Wenisch T F, Falsafi B, et al. SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling[C]// Proceedings of the 30th Annual International Symposium on Computer Architecture. San Diego, USA: ACM, 2003:84.
[49] Perelman E, HamerlY G, Van Biesbrouck M, et al. Using SimPoint for accurate and efficient simulation[J]. ACM SIGMETRICS Performance Evaluation Review, 2003,31(1):318 - 319.
[50] 徐易难, 余子濠, 王凯帆, 等. 处理器敏捷开发背景下的功能验证案例: 流程整合[J]. 计算机科学技术学报, 2023,38(4):737 - 753.
[51] Atasu K, Dündar G, Özturan C. An integer linear programming approach for identifying instruction-set extensions[C]// Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. Jersey City, USA: ACM, 2005: 172 - 177.
[52] Wang S, Xiao C, Liu W, et al. Selecting most profitable instruction-set extensions using ant colony heuristic[C]// 2015 Conference on Design and Architectures for Signal and Image Processing. Krakow, Poland: IEEE, 2015: 1 - 7.
[53] Kaelbling L P, Littman M L, Moore A W. Reinforcement learning: a survey[J]. Journal of Artificial Intelligence Research, 1996,4:237 - 285.
[54] Wang S,Xiao C. Reinforcement learning for selecting custom instructions under area constraint[J]. IEEE Transactions on Artificial Intelligence, 2024,5(4):1882 - 1894.
[55] Huang H, Kim T, Hoskote Y. Edit distance based instruction merging technique to improve flexibility of custom instructions toward flexible accelerator design[C]// 2014 19th Asia and South Pacific Design Automation Conference. Singapore: IEEE, 2014:219 - 224.
[56] Ahn J, Choi K. Isomorphism-aware identification of custom instructions with I/O serialization[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems: IEEE, 2013,32(1):34 - 46.
[57] Brisk P, Kaplan A, Sarrafzadeh M. Area-efficient instruction set synthesis for reconfigurable system-on-chip designs[C]// Proceedings of the 41st Annual Design Automation Conference. New York, USA: ACM, 2004: 395 - 400.
[58] Clark N, HormaTI A, Mahlke S, et al. Scalable subgraph mapping for acyclic computation accelerators[C]// Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems. Seoul, Korea: ACM, 2006:147 - 157.
[59] Bonzini P, Pozzi L. Recurrence-aware instruction set selection for extensible embedded processors[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2008,16(10):1259 - 1267.
[60] Garey M R, Johnson D S. Computers and intractability: a guide to the theory of NP-completeness[M]. New York: Freeman, 2009.
[61] Xiao C, Wang S, Liu W, et al. Parallel custom instruction identification for extensible processors[J]. Journal of Systems Architecture, 2017,76:149 - 159.

A survey on custom instruction design automation for processors

Li Cangyuan[*,**], Liu Cheng[*], Wang Ying[*]
([*]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
([**]University of Chinese Academy of Sciences, Beijing 100049)

Abstract

With the growing demands of embedded system applications and the rise of the reduced instruction set computer V(RISC-V) open instruction set architecture, application-specific instruction-set processor (ASIP) has become a key technology for improving application performance. To reduce the complexity and development costs of ASIPs, automated design methods have gained widespread attention, with custom instruction set synthesis being one of the most challenging aspects in the ASIP design automation process. This paper systematically reviews the main processes and representative work in automated custom instruction design: First, it discusses in detail four different types of custom instructions, i.e. single input single output (SISO), single input multiple output (SIMO), multiple input single output (MISO), and multiple input multiple output (MIMO) and their characteristics and application scenarios; then explores the cost models and various design exploration methods in custom instruction design process; analyzes two important aspects of custom instruction implementation optimization, namely resource overhead optimization and search performance optimization. Finally, predictions are made regarding the research trends in automated custom instruction design.

Key words: application-specific instruction-set processor, automated instruction set design, cost model, design space exploration