可扩展处理器的自定义指令自动识别综述

肖成龙[1,2],王珊珊[1,2],王心霖[2],林 军[2],王晶玥[2]
(1. 汕头大学工学院,广东汕头 515063; 2. 辽宁工程技术大学软件学院,辽宁葫芦岛 125105)

摘 要: 近年来,可扩展处理器越来越多地应用于嵌入式系统当中. 在可扩展处理器周围使用自定义指令能够保证一定的灵活性,同时也能很好地满足嵌入式应用对高性能和低功耗的需求. 自定义指令自动识别是可扩展处理器设计中的关键问题之一. 针对可扩展处理器的应用领域和发展趋势,介绍近年来自定义指令自动识别的研究进展;在此基础上,对于自定义指令识别涉及的关键步骤:中间表示生成、自定义指令枚举、自定义指令选择和代码转换,分别进行总结和归纳,分析不同方法的优点和难点;按照不同应用领域,对可扩展处理器的应用进行了总结和分析;最后展望了自定义指令自动识别的未来发展趋势和研究方向.

关键词: 可扩展处理器; 自定义指令识别; 自定义指令枚举; 自定义指令选择
中图分类号: TP368.1 **文献标识码:** A **文章编号:** 0372-2112(2020)08-1655-10
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2020.08.027

A Survey of Automatic Identification of Custom Instructions for Extensible Processors

XIAO Cheng-long[1,2], WANG Shan-shan[1,2], WANG Xin-lin[2], LIN Jun[2], WANG Jing-yue[2]
(1. *College of Engineering, Shantou University, Shantou, Guangdong* 515063, *China*;
2. *School of Software, Liaoning Technical University, Huludao, Liaoning* 125105, *China*)

Abstract: In recent years, extensible processors have been increasingly applied in various embedded systems. The use of custom instructions around the extensible processors can guarantee flexibility while still meeting the high performance and low power requirements of embedded applications. Automatic custom instruction identification is one of the key issues involved in the design of extensible processors. In view of the application fields and development trends of extensible processors, this survey introduces the research progress of automatic identification of custom instructions in recent years. On this basis, there are four key issues involved in the automatic identification of custom instructions: intermediate representation generation, custom instructions enumeration, custom instructions selection and code transformation. This survey analyzes the advantages and difficulties of previous methods, and presents the summary of the applications of extensible processor in terms of different application fields. Finally, the trends and possible research directions of automatic identification of custom instructions is presented.

Key words: extensible processor; custom instruction identification; custom instruction enumeration; custom instruction selection

1 引言

近年来,电子设备的市场渗透率正稳步提升. 从桌面到嵌入式计算的过渡使得在非常短的时间窗口内设计高性能,低成本的嵌入式软件/硬件系统变得至关重要. 通用处理器在其应用领域内虽然具有较强的通用性,但是通用处理器通常采用的是基于冯·诺依曼体系结构的计算模型,这种体系结构需花费较长的时间和较多的能耗执行每个指令,从这个意义上来说,通用处理器是以牺牲能耗和性能为代价的最灵活的硬件[1]. 在特定应用的计算环境中,专用集成电路(Application Specific Integrated Circuit, ASIC)则采用合适的架构来满足特定的约束条件和目标应用. ASIC 的整个应用程序都是硬连线控制的,因此在 ASIC 上运行的程序具有较高的运行效率并能够显著地降低能耗. 然而,ASIC 降低能耗和提升性能是以低灵活性和较差的可编

收稿日期:2019-10-10;修回日期:2020-04-23 ;责任编辑:马兰英
基金项目:国家自然科学基金(No. 61404069);辽宁省教育厅青年基金(No. LJ2017QL033)

程性为代价的,对于每个新的功能或应用,需要重新设计硬件[2].因此设计和制造 ASIC 具有设计周期长和测试成本高的缺点.

可扩展处理器是一种架构与指令集优化设计的处理器,其通过扩展指令集,使得目标应用程序的部分代码在基准处理器执行,其它计算密集的代码在自定义指令的硬件实现-自定义功能单元(Custom Function Unit,CFU)中执行[3-5].相对于通用处理器,可扩展处理器通过使用自定义指令封装一系列基本操作指令(例如,加法,减法,乘法以及逻辑操作),使得这些基本操作指令之间根据数据依赖关系自动链化,没有数据依赖关系的基本指令并行化,从而很大程度地提高运算的速度.此外,由于多个基本指令封装到一个自定义指令中,使得取指令次数和数据在寄存器与处理器之间传输的次数减少,进而可扩展处理器的功耗显著低于通用处理器.

由于可扩展处理器能够在设计周期、灵活性、性能以及功耗等方面提供良好的折中[6],近年来,可扩展处理器大量的在嵌入式系统和电子设备中使用.很多商业化的可扩展处理器相继出现,比如:Tensilica Xtensa,ARCtangent,Xilinx MicroBalze 和 Altera Nios II 等.这些可扩展处理器经常应用于信号处理和图像处理等领域.

扩展指令集的自动生成是可扩展处理器设计实现的关键.本文针对自定义指令识别涉及的关键问题(重点分析自定义指令枚举和自定义指令选择)、可扩展处理器的应用进行总结和分析,并给出了自定义指令识别的未来发展趋势和研究方向.

2 自定义指令识别相关方法

经典的自定义指令自动识别流程如图 1 所示,流程一般包括以下四个阶段,其中子图枚举:

(1)源代码的中间表示生成:该方法的输入是应用程序的源代码,应用程序通常为 C 或者 C ++ 代码.使用前端编译器 GECOS 将源代码转换为控制数据流图.

(2)自定义指令枚举(子图枚举):以图论相关理论为基础,基于控制数据流图内的数据流图,使用子图枚举算法从数据流图中,枚举出所有满足约束条件的子图.

(3)自定义指令选择(子图选择):在枚举出满足约束条件的子图的基础上,依据设计目的及逻辑综合工具所生成的自定义指令的面积和时延信息,利用子图选择算法从枚举出的子图当中选择部分最佳子图作为最终的自定义指令(关键步骤).

(4)代码转换:将源代码转换为包含所选自定义指令的新代码,新代码与源代码功能等价.

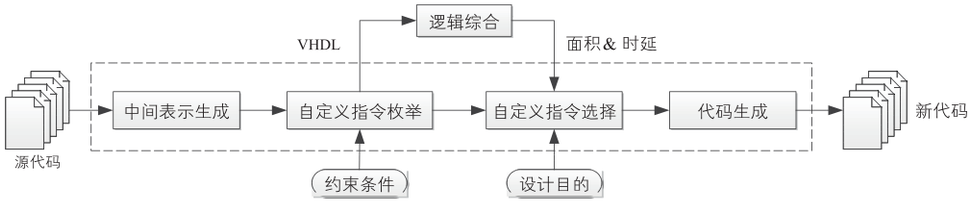

图1 自定义指令识别流程

2.1 中间表示生成

近年来,随着应用的规模不断扩大,处理器的设计也日益复杂化,所以可扩展处理器设计周期早期阶段的建模问题十分重要.中间表示生成作为处理源程序的第一步,通常将应用程序中的数学表达或者运算指令转换为抽象的图形表达方式,以此将应用程序转换为图论领域内的问题并加以研究.图形作为处理行为和算法的高抽象级别的建模方式,在研究可扩展处理器中频繁执行的应用代码段方面上有着显著的优势.较为常用的中间表示生成方式有泰勒展开图和控制数据流图两种方式.

泰勒展开图(Taylor Expansion Diagram,TED)是一种简洁、规范的基于图的表示法,其抽象能力适用于行为和算法层面上的数据流设计.泰勒展开图经常作为多变量多项式的表达形式[7].TED 具有紧凑和规范的数据结构,能够准确地表达应用程序中的数据运算关系,但是只能表示为多元多项式.对于应用程序的中间表示具有一定的局限性.

控制数据流图可以准确地表示数据依赖关系以及控制关系,并且控制数据流图更加接近程序的表达方式,使研究人员容易理解数据之间的传递关系[8].当前关于自定义指令自动识别的大部分研究工作均采用控制数据流图作为中间表示,并基于控制数据流图进行自定义指令枚举和选择.

2.2 自定义指令枚举

自定义指令枚举问题是从应用程序对应的数据流图中枚举出所有满足一定设计约束或者用户自定义约束的凸子图作为候选自定义指令.一般来说,数据流图中的每个基本指令都可以包含在潜在子图实例中或者不包含在其中,从而一个给定的数据流图中可以产生 2^n 个候选自定义指令[9],其中 n 代表数据流图中节点的数目.由此可以看出自定义指令枚举是算法复杂度非常高的问题.为了降低复杂度,部分学者根据处理器的结构,引入微体系结构的约束或者是人为加入一些

约束条件.根据约束条件的不同,此前的研究可如下进行分类和分析.

输入、输出约束　根据可扩展处理器体系结构特点,其指令编码长度的不同和寄存器的读写端口数目有限.子图枚举期间,自定义指令的输入数量、输出数量受到限制.以下按照不同的输入输出条件进行分析并给出其优缺点:

(1)树形子图(Tree Subgraph,TS):为了降低枚举的复杂度,早期的研究主要集中于枚举所有树形子图.然而,只枚举树形子图作为自定义指令,而未加入输入、输出约束条件时,只能非常有限地提升性能或者降低功耗[10].

(2)多输入单输出子图(Multiple Inputs Single Output,MISO):多输入单输出约束条件简化可扩展处理器的结构设计,实现了一个写入端口设计,从而避免了写入冲突.相比于枚举所有树形子图,枚举多输入单输出的子图可以很大程度上降低问题的复杂度.此外,输入数目不同,枚举效率差距较大.文献[10]在相同的面积、单输出的条件下,枚举2个输入的子图的速度是3个输入的4.5倍.而枚举2个输入的模式的速度是4个输入的7.5倍.原因是将输入约束条件从2放松到4时,所选择的子图数目增多,所需枚举时间更长.就枚举多输入单输出子图的时间复杂度而言,其时间复杂度仍为指数级.此外,枚举多输入单输出子图作为自定义指令带来的性能提升或者功耗降低还是非常有限[11].

(3)最大多输入单输出子图(Maximum Multiple Inputs Single Output,MAXMISO):在多输入单输出子图的基础上,节点最多数目的子图被称为最大多输入单输出子图.最大多输入单输出模式枚举的复杂度上是线性的.文献[12]采用迭代的方法枚举最大多输入单输出子图,并将该方法所识别的子图可用于可重构的超长指令字处理器之中,可减少处理器整体的运行时间.文献[13]将每个最大多输入单输出子图折合成一个节点,将每个节点重新构造多输入单输出子图.文献[14]在文献[13]基础上,将聚合后的节点构造成多输入单输出的凸子图,并将新划分的子图进行并行化处理,提升了处理器的处理性能.

(4)多输入多输出子图(Multiple Inputs Multiple Outputs,MIMO):由于寄存器的读写端口数目有限,如NISO II中的输入输出端口数(I/O)一般为2/1,近期的一些研究主要是枚举满足I/O限制的子图.其中Pozzi等人[15]提出了基于二元决策树的枚举算法,该算法利用数据流图当中按拓扑顺序形成子图的输出数目的单调性来削减搜索空间.文献[16]在利用数据流图的拓扑特性的同时,通过预过滤方法来进一步削减搜索空间.文献[17]首次证明了满足I/O条件的子图数目的上限为$n^{(\text{in}+\text{out})}$,其中$n$为数据流图中节点的个数,in为输入数目上限、out为输出数目上限,同时也提出了一个能够通过参数设置来分别枚举出连通子图和分离子图的算法.然而,此算法在枚举所有子图(包括连通子图和分离子图)上的运行时间仅与文献[15]提出的算法运行时间相当.文献[1]提出一个能够根据用户要求灵活地枚举出连通子图或分离子图的算法,实验表明,该算法在枚举所有子图上的运行时间要比文献[15]提出的算法快一到两个数量级.

(5)凸子图(Convex Subgraph,CS):除了以上类型的自定义指令枚举的研究,也有部分学者提出相关的算法来枚举出所有的凸子图[18].文献[18]首次证明了数据流图中的凸子图的数目的上限为$2^n+n+1-d_n$,其中n为数据流图中的结点数目,如果n为偶数$d_n=2*2^{n/2}$,如果n为奇数$d_n=3*2^{(n-1)/2}$.文献[19]提出能够枚举出所有凸子图或枚举满足大小约束的凸子图的算法,该算法较文献[18]提出的算法快3.29倍.

(6)最大凸子图(Maximum Multiple Inputs Multiple Outputs,MAXMIMO):针对自定义指令枚举方法,大多数研究人员在凸性和I/O约束条件下枚举所有的子图,或者仅在约束条件下枚举最大的子图.通过实验发现,放松I/O的限制的情况下枚举出的自定义指令,往往能够带来更高的性能提升.因而,近年来,也有部分研究集中于枚举最大凸子图,而不考虑I/O的限制.文献[20]首次证明了MAXMIMO子图数目的上限为$2^{|V_I|}$,其中$|V_I|$为数据流图中禁忌节点的数目.文献[21]通过放松I/O的限制,枚举最大凸子图为可扩展处理器带来更高的性能提升.MAXMIMO子图作为自定义指令虽然能够带来更高的性能提升,但是其通常不具备良好的重用性或通用性.

表1给出了不同类型子图数目的上限.从表1可以看出,自定义指令枚举问题是算法复杂度非常高的问题.当问题的规模从单个应用扩大到领域内的多个应用时,枚举算法的运行效率显得尤为重要.因此,很有必要提出更加高效的自定义指令枚举算法.

表1　不同类型子图数目的上限

子图类型	子图数目上限	字母表达意义
所有子图	2^n	n为节点数目
所有凸子图(CS)	$2^n+n+1-3*2^{(n-1)/2}$(n为奇数) $2^n+n+1-2*2^{n/2}$(n为偶数)	n为节点数目
多输入多输出子图(MIMO)	$n^{\text{in}+\text{out}}$	n为节点数目,in为输入上限,out为输出上限
最大多输入多输出子图(MAXMIMO)	$2^{\|V_I\|}$	$\|V_I\|$为数据流图中禁忌节点的数目

连通子图或分离子图 考虑到枚举所有子图的时间复杂度可达到 $O(2^n)$,为了提升枚举效率,部分研究工作只考虑连通子图. 如图 2 所示,子图{1,4}为连通子图,而子图{7,8}则为分离子图. 文献[22]针对连通的子图,在输入和输出数目、面积和凸子图的约束条件下枚举模式. 文献[23]提出了通过合并向上和向下的椎体型子图构建连通子图. 然而,该算法可能使得同一个子图被枚举多次,因此需要额外的处理来删除冗余子图. 文献[24]提出基于邻近搜索的子图枚举算法,该算法用于枚举有向无环图内满足 I/O 端口约束的凸连通子图. 自定义指令可根据硬件本身实现指令的并行化处理. 通过搜索分离子图,实现分离子图并行化操作. 当 n 个指令的并行操作时,总的执行时间是 n 个指令中的最大值. 因此,可通过识别分离子图实现相当大的加速. 文献[25]是在连通子图的基础上通过组合连通子图来形成分离子图. 实验结果表明,此文献提出的算法的运行时间极大程度的依赖于连通子图枚举算法的效率. 文献[26]提出了一个能够根据用户要求灵活地枚举出连通子图或分离子图的算法. 实验结果显示,文献[26]在枚举所有子图的运行时间比文献[15]提出的算法快一到两个数量级.

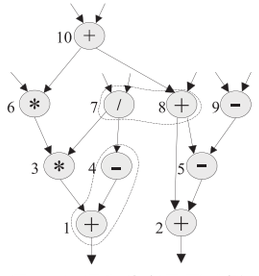

图2 连通和分离子图示例

2.3 自定义指令选择

自定义指令选择问题是从已枚举出的候选自定义指令当中,根据不同的设计目的选择一些满足约束条件的最佳子集作为最终的扩展指令集. 自定义指令选择问题同样是具有高复杂度的问题[27,28]. 按照束条件和设计目的不同,之前的研究可以如下分析.

2.3.1 约束条件

(1)面积约束:此前的大部分的研究集中于满足给定的资源(面积)约束条件下选择能够达到最大性能提升的候选自定义指令,这种情况下可将此类问题转化为经典的0/1 背包问题,可见自定义指令选择问题同样是算法复杂度很高的问题. 在没有面积约束的情况下,选择自定义指令能够带来更大的性能提升,因此,也有部分研究关注于在没有面积约束的情况下通过选择自定义指令达到最大的性能提升[29]. 对于此类问题,文献[30]根据候选自定义指令之间的重叠以及循环关系建立成一个相容图,由此将问题转化为经典的最大团问题,进而可以证明自定义指令的组合数目的上限为 $3^{m/3}$,其中 m 为候选指令的数目.

(2)重叠约束:允许重叠有时可能会改善程序运行时间,但同时增加不必要的功耗并难以生成新代码[31]. 如图3所示,$M1$ 与 $M2$ 中存在共同的节点4,故该图中存在重叠. 现有的研究中通常不允许选择的子图之间有重叠.

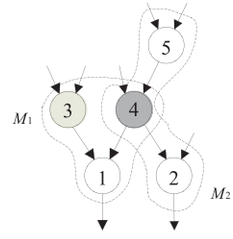

图3 重叠子图示例

(3)非循环约束:为了确保所选的子图与当前候选子图没有形成循环,应该进行无循环检查. 如果两个候选子图相互提供数据,则形成一个循环. 如图 4 所示,子图 $M1$ 和子图 $M2$ 之间存在循环. 在这种情况下,两个自定义指令之间发生死锁[32]. 因此,现有的研究中通常不允许选择的子图之间存在循环.

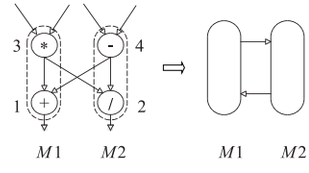

图4 循环子图示例

2.3.2 设计目的

(1)提升性能 专用硬件与通用处理器相比,其性能差异主要是因为二者的处理器的体系结构不同,从而相同的程序在专用硬件和通用处理器所花费的时钟周期不同. 在可扩展处理器中,经自定义指令识别后,选定的自定义指令在自定义功能单元中执行,而应用程序中未被包含进自定义指令的基本指令在基处理器(通用处理器)上执行. 提升性能是使用扩展指令集的最大好处之一[33]. 之前的大部分研究关注于性能的提升. 按照给出解的最优性,这些研究工作可以分为探索式算法和精确算法. 精确算法能够保证所给出解是最优的,但是当问题规模扩大的时候往往不能在合理的时间内给出最优解. 而探索式算法通常具有较高的求解效率,却不能保证解的最优性.

(2)降低功耗 随着集成电路的复杂度不断增加,功耗是当前电子产品设计时需要重点考虑的因素. 研究人员提出许多有关降低功耗的设计方法. 一种方法在满足性能约束的条件下使功耗最小化,另一种方法是在满足功率约束的同时最大化性能参数. 从自定义

指令识别的角度来看,这两种方法多数是通过限制指令大小、限制指令的执行次数、减少存储器访问次数来降低功耗[34,35]. Galuzzi 等人在文献[2]中指出,到目前为止,只有少数的研究关注于通过使用扩展指令集减少功耗. 考虑到功耗的重要性,已有越来越多的关于低功耗扩展指令集的相关研究出现[36].

(3)减少代码量　自定义指令通过将多个基本指令封装成一个复杂指令,可显著减少代码量,而代码量是嵌入式系统设计时需要考虑的重要问题之一,因此,减少代码量也可作为自定义指令识别的设计目的之一[37]. 为了最大限度地减少代码量,通常选择最少数目的子图集,一个子图对应一行代码,因而可以利用较少的子图就可以覆盖数据流图中的节点. 文献[38]提出了一种基于优先函数的子图选择方法,实验结果显示,该方法可平均减少71%的代码量,最高可减少84%的代码量.

2.3.3　选择方法

枚举出的子图被选择的原因多数是由于其在应用代码中可被频繁利用,或者与其他组件相比具有较高的性能提升,又或者可以显著减少功耗. 因此,提出良好的子图选择方法对于应用程序性能提升或功耗减少至关重要. 针对子图选择问题,从选择结果是否为最优,可分为精确方法和探索式方法.

精确方法　精确法作为子图选择的一种方法其特点是能够保证解的最优性. Cong 等人[10]从原始数据流图到扩展指令集的映射转换为布尔网络映射的面积最小化问题. 该研究在满足给定的面积(资源)约束条件下,选择能够达到性能提升最大化的候选子图,这种情况下,把子图选择问题转换为 0 - 1 背包问题. Clark 等人提出一种精确方法,旨在用最少数目的自定义指令覆盖每个节点,该算法将子图选择问题转换为单边覆盖问题[9]. Martin 等尝试通过使用约束编程来解决子图选择问题,该文献对自定义指令的选择通过两种相应的调度策略实现:时间约束调度和资源约束调度[39],实验结果显示该方法适用于问题规模较小的问题. 部分研究关注于在没有面积约束的情况下通过选择最佳候选自定义指令达到最大的性能提升. 文献[30]根据候选子图之间的重叠以及循环关系建立成一个相容图,由此将问题转化为经典的最大团问题. 针对子图选择问题,线性规划方法将问题转换为线性和非负性约束条件下符合线性函数的最大值或者最小值问题. 文献[14]将自定义指令选择问题转换为线性规划问题,并且在最小的面积约束条件下,求最性能提升最大化. 然而,当问题规模较大时,所用的线性规划工具无法在合理的时间内给出最优解.

探索式方法　子图选择是复杂度非常高的问题,精确法虽然能够保证解的最优性,但是却无法在合理的时间内得到解. 当图的规模变大,动态规划等精确方法的效率较低,因此,利用探索式的方法能够高效地解决该问题. Kastner 等人通过沿着最频繁出现的边生成子图来选择具有高出现频率的子图[40]. Guo 等人提出基于冲突图的模式选择算法,此算法用最少数目的模式来覆盖图. 该算法在子图之间不能有重叠的前提下,使用一个目标函数贪婪地选择子图[41]. Kamal 等人将候选自定义指令集进行分割,然后从分割后的自定义指令集独立选择近似最优子集[42]. Bozorgzade 等人[43]提出子图选择由调度决定,使重用性高子图获得更高的优先级. 文献[38]将扩展指令集应用于高层次综合之中,根据不同的选择目的,提出了基于最少子图数目、出现频率的模式选择以及关键路径选择的三种启发式方法. 近年来,一些群体智能算法也用来解决子图选择问题[44]. 文献[44]将模拟退火、禁忌搜索、蚁群算法、遗传算法、粒子群算法等群体智能算法用于解决子图选择问题,并从算法运行时间和性能提升两方面进行了比较,实验结果显示模拟退火算法在算法运行时间及性能提升方面效果较好.

从以上的分析可以看出,自定义指令选择问题同样是高复杂度的问题,当问题规模扩大时,算法很可能不能在合理的时间内提供满意的结果. 此外,上述子图选择算法或者是为了减少代码量、或者是为了减少面积、或者是为了提升运算性能,只能满足单一的设计目标. 然而,这些研究却缺乏能够综合考虑以上三个方面的子图选择算法.

2.4　代码转换

根据所选择的子图,自动生成功能相同的、包含了自定义指令的新代码. 新代码将作为高层次综合工具的输入. 在此阶段,需要判断所选择的子图哪些是功能和结构相同的. 这些功能和结构相同的子图将由同样的自定义功能单元来实现. 判断子图的功能和结构是否相同的问题可以看成图的同构问题[45]. 文献[46]采用约束编程方法实现子图同构的识别,约束编程方法是将问题的描述和问题的求解分离,通过定义图的节点、边的变量信息以及约束条件后,利用约束编程求解器识别同构子图. 文献[29,36]通过实现 Cordella 等人提出的 VF2 算法[47]来进行子图同构判断.

在将功能等效的子图映射成相同的自定义指令之后,需要在表示自定义指令的代码之前增加一个特定的编译指令. 自定义指令用程序语言的方法来表示. 这样,高层次综合工具对自定义指令会像基本指令一样,对自定义指令进行调度和绑定. 对于高层次综合工具 CtoS Cadence,所有非内联函数都被默认为自定义指令,即不需要特殊编译指令. 对于高层次综合工具 CataplutC,需要在自定义指令对应的方法表示前加入#pragma 编译指令.

生成的自定义指令的代码格式如图5所示.

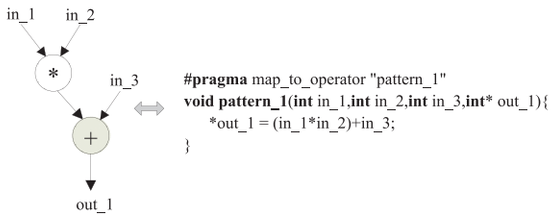

图5 自定义指令代码格式

3 自定义指令的应用

早期限于专用芯片设计周期长、硬件开发难以调试以及成本高等特点,微处理器的发展缓慢. 随着可扩展处理器的开发及深入研究,可扩展处理器可在信号处理、图像处理和信息安全等不同的领域中使用,并针对相同领域内的应用集合来自动生成自定义指令集,而不局限于某个特定的应用.

(1)信号处理

可扩展处理器具有较好的灵活性及计算能力,有关可扩展处理器在信号处理方面的自定义指令选择和调度成为近年来的热点话题. 将传感器与可扩展处理器结合,可保证处理器在复杂计算的情境下依然能够有较强的稳定性. 与此同时,对于领域内专用的可扩展处理器,其程序具有结构和功能的相似性,通过识别和判断相似的计算代码块,并将该类代码在自定义功能单元中运行,从而提高处理器的利用率,节约硬件成本. Arslan 等人将子图同构和全局约束的方法结合起来用于指令调度,并在 IDCT(Inverse Discrete Cosine Transform),MESA(Maximum Entropy Spectral Analysis)等用于信号处理的测试程序中验证方法的有效性[48]. Sisto 等人提出了一种专用于汽车应用领域的传感器信号可扩展处理器设计方法[49]. Eissa 等人通过自定义指令实现加速 SHA-3(Secure Hash Algorithm 3)算法,实验结果显示,该方法可获得21% ~43%的加速[50].

(2)图像处理

在信息处理方面,海量的数据和复杂的应用使处理器系统面临极为严峻的考验. 随着图像处理技术的蓬勃发展,神经网络和支持向量机等学习型的机制虽然在图像处理方面有着一定的优势,但是针对庞大的数据量,这些优化算法仍需要大量的时间处理数据信息. 国内外研究发现,将自定义指令应用于图像处理领域,能够显著地提升图像处理算法的性能. 朱方研究面向嵌入式处理器中的移动视频监控技术,其图像噪声多、目标检测精度低等问题得到了明显的改善,并且该技术实现了 ASIP 较高的数据吞吐量和较低的内存访问量[51]. 在嵌入式视觉领域内,对于图像的计算能力有着较高的要求,需要在较短的时间内对图像识别和处理. Mori 等人提出了用于实时图像处理和计算机视觉算法的 ASIP 设计[52]. Rakanovic 等人提出了加速前馈神经网络的 ASIP 设计流程,通过对神经网络算法进行自定义指令识别,使算法性能提升了 20 至 40 倍[53]. Edwards 等人通过将自定义指令集应用到实时目标检测系统中,使得处理速度提高了 1.5 至 6.8 倍[54].

(3)信息安全

随着通信技术的迅速发展,信息安全受到社会各界的重视. 在密码学领域,一些安全性较高的加密解密技术缺乏实用性,其原因是密钥过大或者密钥难以扩展,因而通过自定义指令提升加密算法的执行效率是近几年的研究热点. Rawat 等人对原始加密算法 KECCAK 进行指令分析综合,结果表明指令分析综合方法能够快速地运行加密算法并且减少面积花销[55]. Youssef 等人主要研究对称密钥算法的指令集扩展,加密指令集扩展技术降低了嵌入式处理器对于面积和能耗的要求,该实验结果表明,指令集扩展技术能够适用于嵌入式服务路由器、实时多媒体应用程序和智能芯片中的安全领域[56]. 胡绵江等人提出一种面向加密算法共性子图的指令定制方法,该方法通过研究加密算法中的共性计算模式,减少算法的指令数目,从而提高算法的效率[57]. 夏辉等人将椭圆曲线密码应用于专用指令集处理器中,该处理器提升了椭圆曲线标量乘法运算的运算效率[58].

4 未来研究方向展望

随着自定义指令自动识别问题研究的深入,可扩展处理器的自定义指令识别技术未来的发展方向是自定义指令自动识别云端化,实现并行机制的计算模式加速人工智能领域的关键应用,从而进一步设计出高性能、低功耗的电子产品. 在此背景下,对于数据流图的处理仍有许多问题亟待解决,如节点的负载均衡问题、领域专用指令集问题、约束条件和约束目标多样化以及领域专用指令集通用性问题等. 因此,未来专用指令集的研究方向可从如下4点展开:

(1)自定义指令自动识别云端化研究

云计算已经成为当前的研究热点之一,而且已经成功应用到诸多领域. 电子设计自动化领域(Electronic Design Automation,EDA)领域和计算机辅助设计领域已经开始进行基于云计算设计工具的研究和开发[32,59,60]. EDA 行业的公司也相继布局 EDA 云应用软件. 例如,2015 年,IBM 和 Cadence 分别发布了其在芯片设计领域首个电子设计自动化云和推出了业内第一个云硬件仿真加速器 Palladium Z1. 将 EDA 应用软件部署于云平台,利用云计算大规模并行计算的优势可显著缩短设

计周期,云计算资源的弹性缩放使得 EDA 工具的使用成为像水电等公共服务一样,按需付费,从而极大节省设计成本.

由于自定义指令的识别所涉及的自定义指令枚举问题和自定义指令选择问题具有高计算复杂度,当问题域扩大到多个应用程序时,传统的串行算法很可能不能提供满意的结果,而算法的效率决定了整个设计周期. 因此,基于云计算的扩展指令集并行识别研究可作为未来扩展指令集自动识别的研究方向之一. 采用并行计算时,需要考虑如何保证计算节点的负载均衡,从而更加高效地完成自定义指令的枚举和选择.

在未来研究工作中,可考虑通过分析自定义指令枚举问题和自定义指令问题复杂度模型,分别建立自定义指令枚举问题和自定义指令选择算法运行时间预测模型,并依据算法运行时间预测模型将复杂问题划分为若干子问题,采用云计算框架 Spark 将子问题分发给计算节点并行独立地处理.

(2)约束条件和优化目标多样性问题

从 2.2 节和 2.3 节的分析可以看出,已有研究对于约束条件和优化目标的处理呈现单一化[61]. 针对子图枚举问题,此前大部分研究只能枚举出满足特定约束条件子图[17,20,23,24],当约束条件发生变化时(增加或者删除),算法将不再适用. 因此,如何将研究问题的描述和求解过程分离,提供一般化的自动求解方式仍有待进一步研究. 针对子图选择问题,已有大部分研究只考虑提升性能,或者只追求降低功耗的设计目的[1,14,29,31,32],缺少对多个目标同时优化,提出更有效同时能实现多目标优化的方法很有必要.

由于约束编程方法可将问题的描述和问题的求解分离,其在问题的建模方面具有较好的灵活性,在未来的研究工作中,针对子图枚举问题和子图选择问题,可分别建立符合约束编程范式的问题模型,在此模型下,可灵活增加或删除约束条件和优化目标,并采用约束规划求解器对问题进行自动求解.

(3)领域专用指令集的重用性、通用性问题

由于应用程序之间的结构和功能差异性,针对特定应用程序生成的扩展指令集并不能给领域内其他应用程序带来满意的性能提升. 最近的一些研究开始针对同领域内的应用集合来自动生成扩展指令集[36,62~65],而不是仅针对某个特定的应用. 其中,文献[62,63]通过使用编译优化领域的泰勒扩展图从多个应用中识别和判断相似的计算代码块. Karunarathna 等人试图直接从汇编代码序列中找出领域适用的自定义指令[64],相比于从数据流图中枚举和选择自定义指令,此方法不能发现和充分利用基本指令之间并行关系,由此带来的性能提升可能有限. 文献[65]首先从领域内的两两应用之间识别出共同的计算代码块作为自定义指令候选,然而,仅从两两应用之间识别自定义指令仍然不能保证所生成的扩展指令集适用于其他的应用.

但是,这些研究存在一个共性的问题:虽然生成的扩展指令集可以适用于设计时考虑到的这些应用,但是对设计时未考虑到的应用却不一定适用. 或者,设计时考虑到的应用有更新时,更新后的应用可能不能很好地利用已生成的扩展指令集. 因此,已有方法所生成的扩展指令集缺乏一定的灵活性和通用性. 针对领域专用指令集的通用性问题,可考虑引入字符串匹配领域中的编辑距离和图像处理领域中的匹配度的来量化候选自定义指令之间的差异性和相似性,并对相似度较高的自定义指令进行合并,进而使得合并生成的自定义指令具有更好的通用性.

(4)面向机器学习领域关键应用的扩展指令集识别

由于可扩展处理器在性能和功耗等方面具有良好的表现,近年来,可扩展处理器在信号处理、图像处理及网络信息安全等领域中得到了广泛使用. 当前,我国已将将人工智能作为发展新一代信息技术的主要方向. 针对机器学习领域的关键算法的加速研究,目前主要的方法是通过 FPGA、ASIC 或 GPU 实现对算法运行进行加速[66]. 然而,在机器学习领域还缺乏可扩展处理器的应用研究. 通过观察发现,机器学习领域的很多算法在样本训练时需要大量的计算,如 DNN 算法和 CNN 算法等. 针对机器学习领域的算法中计算密集的代码块,可考虑通过识别自定义指令对算法进行加速,并从性能提升和功耗降低两方面分析自定义指令的应用效果. 在未来研究工作中,可首先分析机器学习算法的特点,采用程序性能分析器对机器学习算法进行分析和确定算法计算密集的代码块,并针对确定的计算密集型代码块自动识别自定义指令. 面向机器学习领域关键应用的扩展指令集识别的研究将推动可扩展处理器在机器学习领域的应用,因此,将具有较好的应用前景.

5　总结

本文主要论述了可扩展处理器的自定义指令自动识别问题. 针对自定义指令识别过程中涉及的关键问题,尤其是自定义指令枚举问题、自定义指令选择问题的高复杂度,归纳和分析了降低枚举问题和选择问题复杂度的方法,并在此基础上,针对自定义指令解决可扩展处理器性能、功耗以及代码量方面的问题,做出了详细的总结和分析,给出了自定义指令的主要应用,并对自定义指令识别未来研究方向进行了展望.

参考文献

[1] Xiao C, Casseau E, Wang S, et al. Automatic custom in-

struction identification for application-specific instruction set processors[J]. Microprocessors & Microsystems,2014,38(8):1012 – 1024.

[2] Galuzzi C, Bertels K. The instruction-set extension problem:a survey[J]. ACM Transactions on Reconfigurable Technology & Systems,2011,4(2):1 – 28.

[3] P Faraboschi et al. Lx:A technology platform for customizable VLIW embedded processing[A]. Proceedings of International Symposium on Computer Architecture [C]. 2000. 203 – 212.

[4] R E Gonzalez. Xtensa:A configurable and extensible processor[J]. IEEE Micro,2000,20(2):60 – 70.

[5] Z A Ye. et al. Chimaera:A high-performance architecture with a tightly-coupled reconfigurable functional unit [A]. 2000 International Symposium on Computer Architecture [C]. Vancouver:IEEE,2000. 225 – 234.

[6] 陈虎,陈书明,陈胜刚,等. GISEES:面向嵌入式系统的扩展指令集自动产生方法[J]. 电子学报,2011,39(9):2026 – 2033.
Chen Hu, Chen Shu-ming, Chen Shen-gang, et al. GISSES: automatic generation of instruction-set extensions for embedded systems[J]. Acta Electronica Sinica,2011,39(9):2026 – 2033. (in Chinese)

[7] Sartor J B, Eeckhout L. MInGLE:An efficient framework for domain acceleration using low-power specialized functional units[J]. Acm Transactions on Architecture & Code Optimization,2016,13(2):17.

[8] Wang C, Li X, Zhang H, et al. Hot spots profiling and data-flow analysis in custom dataflow computing SoftProcessors [J]. Journal of Systems and Software, 2017, 125: 427 – 438.

[9] Clark N, Hormati A, Mahlke S, et al. Scalable subgraph mapping for acyclic computation accelerators [A]. 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems [C]. Seoul:ACM,2006. 147 – 157.

[10] Cong J, Fan Y, Han G, et al. Application-specific instruction generation for configurable processor architectures [A]. 2004 ACM International Symposium on Field-Programmable Gate Arrays[C]. Monterey:ACM,2004. 183 – 189.

[11] Galuzzi C, Bertels K. The instruction-set extension problem:a survey [A]. 2008 International Workshop on Reconfigurable Computing:Architectures, Tools and Applications[C]. London:Springer,2008. 209 – 220.

[12] Alippi C, Fornaciari W, Pozzi L, et al. A DAG-based design approach for reconfigurable VLIW processors[A]. 1999 Conference on Design, Automation and Test in Europe[C]. Munich:ACM,1999. 57.

[13] Galuzzi C, Bertels K, Vassiliadis S. A linear complexity algorithm for the generation of multiple input single output instructions of variable size[A]. 2007 International Workshop on Embedded Computer Systems[C]. Berlin:Springer,2007. 283 – 293.

[14] Galuzzi C. Automatic selection of application-specific instruction-set extensions[A]. 2007 International Conference on Hardware/software Codesign and System Synthesis[C]. Salzburg:IEEE,2007. 160 – 165.

[15] 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.

[16] Xiao C, Casseau E. Exact custom instruction enumeration for extensible processors[J]. Integration, the VLSI journal,2012,45(3):263 – 270.

[17] 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.

[18] Balister P, Gerke S, Gutin G, et al. Algorithms for generating convex sets in acyclic digraphs[J]. Journal of Discrete Algorithms,2009,7(4):509 – 518.

[19] Wang S, Xiao C, Liu W. A faster algorithm for enumerating connected convex subgraphs in acyclic digraphs[J]. IEEE Embedded Systems Letters,2017,9(1):9 – 12.

[20] Atasu K, Luk W, Mencer O, et al. FISH:Fast instruction synthesis for custom processors[J]. IEEE Transactions on Very Large Scale Integration Systems,2011,20(1):52 – 65.

[21] Giaquinta E, Mishra A, Pozzi L. Maximum convex subgraphs under I/O constraint for automatic identification of custom instructions[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,2015,34(3):483 – 494.

[22] Huynh H P, Sim J E, Mitra T. An efficient framework for dynamic reconfiguration of instruction-set customization [J]. Design Automation for Embedded Systems,2009,13(1 – 2):91 – 113.

[23] Yu P, Mitra T. Scalable custom instructions identification for instruction-set extensible processors[A]. 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems[C]. Washington:ACM,2004. 69 – 78.

[24] 薄拾,葛宁,林孝康. 一种高效的凸连通子图枚举算法[J]. 软件学报,2010,21(12):3106 – 3115.
Bo Shi, Ge Ning, Lin Xiao-kang. Efficient algorithm for convex connected subgraph enumeration[J]. Journal of Software,2010,21(12):3106 – 3115. (in Chinese)

[25] Yu P, Mitra T. Disjoint pattern enumeration for custom in-

structions identification[A]. 2007 International Conference on Field Programmable Logic and Applications[C]. Amsterdam:IEEE,2007. 273 – 278.

[26] Xiao C,Casseau E. Efficient custom instruction enumeration for extensible processors[A]. 2011 IEEE International Conference on Application-Specific Systems, Architectures and Processors[C]. Santa Monica:IEEE,2011. 211 – 214.

[27] Prakash A,Clarke C T,Lam S K,et al. Rapid memory-aware selection of hardware accelerators in programmable SoC design[J]. IEEE Transactions on Very Large Scale Integration(VLSI) Systems,2017. 1 – 12.

[28] Jordans R,Jóźwiak L,Corporaal H,et al. Automatic instruction-set architecture synthesis for VLIW processor cores in the ASAM project[J]. Microprocessors and Microsystems,2017,51:114 – 133.

[29] 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,2013,32(1):34 – 46.

[30] Wang S,Xiao C,Liu W,et al. Selecting most profitable instruction-set extensions using ant colony heuristic[A]. 2015 Conference on Design and Architectures for Signal and Image Processing[C]. Krakow:IEEE,2015. 1 – 7.

[31] Kamal M, Afzali-Kusha A, Safari S, et al. Yield and speedup improvements in extensible processors by allocating extra cycles to some custom instructions[J]. ACM Transactions on Design Automation of Electronic Systems,2016,21(2):28.

[32] Xiao C,Wang S,Liu W,et al. Parallel custom instruction identification for extensible processors[J]. Journal of Systems Architecture,2017,76(C):149 – 159.

[33] Severance A, Edwards J, Omidian H, et al. Soft vector processors with streaming pipelines[A]. 2014 ACM International Symposium on Field-programmable Gate Arrays[C]. Monterey:ACM,2014. 117 – 126.

[34] Ham T J, Wu L, Sundaram N, et al. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics[A]. 2016 International Symposium on Microarchitecture[C]. Taipei:IEEE,2016. 1 – 13.

[35] Kulkarni A,Page A,Attaran N,et al. An energy-efficient programmable manycore accelerator for personalized biomedical applications[J]. IEEE Transactions on Very Large Scale Integration(VLSI) Systems,2018,26(1):96 – 109.

[36] González-Álvarez C, Sartor J B, Alvarez C, et al. Automatic design of domain-specific instructions for low-power processors[A]. 2015 International Conference on Application-specific Systems, Architectures and Processors[C]. Toronto:IEEE,2015. 1 – 8.

[37] Mitra T. Handbook of Hardware/Software Codesign[M]. Berlin:Springer,2017. 377 – 409.

[38] Xiao C,Casseau E. Improving high-level synthesis effectiveness through custom operator identification[A]. 2014 IEEE International Symposium on Circuits and Systems[C]. Melbourne:IEEE,2014. 161 – 164.

[39] Martin K, Wolinski C, Kuchcinski K, et al. Constraint-driven instructions selection and application scheduling in the DURASE system[A]. 2009 IEEE International Conference on Application-Specific Systems, Architectures and Processors[C]. Boston:IEEE,2009. 145 – 152.

[40] Kastner R,Kaplan A,Memik S O,et al. Instruction generation for hybrid reconfigurable systems[J]. ACM Transactions on Design Automation of Electronic Systems(TODAES),2002,7(4):605 – 627.

[41] Guo Y,Smit G J M,Broersma H,et al. A graph covering algorithm for a coarse grain reconfigurable system[J]. ACM SIGPLAN Notices. ACM,2003,38(7):199 – 208.

[42] Kamal M,Afzali-Kusha A,Safari S,et al. OPLE:A heuristic custom instruction selection algorithm based on partitioning and local exploration of application dataflow graphs[J]. ACM Transactions on Embedded Computing Systems,2015,14(4):1 – 23.

[43] Bozorgzadeh E,Memik S O,Kastner R,et al. Pattern selection:customized block allocation for domain-specific programmable systems[A]. 2002 International Conference on Engineering of Reconfigurable Systems and Algorithms[C]. Las Vegas:IEEE,2002. 190 – 196.

[44] Wang S,Xiao C,Liu W,et al. A comparison of heuristic algorithms for custom instruction selection[J]. Microprocessors & Microsystems,2016,45:176 – 186.

[45] Mishra A,Agarwal M,Asati A R,et al. Using graph isomorphism for mapping of data flow applications on reconfigurable computing systems[J]. Microprocessors and Microsystems,2017,51:343 – 355.

[46] Bukchin Y,Raviv T. Constraint programming for solving various assembly line balancing problems[J]. Omega,2017(7):1 – 32.

[47] Cordella L P,Foggia P,Sansone C,et al. A(sub)graph isomorphism algorithm for matching large graphs[J]. IEEE transactions on pattern analysis and machine intelligence,2004,26(10):1367 – 1372.

[48] Arslan M A,Kuchcinski K. Instructionselection and scheduling for DSP kernels on custom architectures[A]. 2013 Euromicro Conference on Digital System Design[C]. Los Alamitos:IEEE,2013. 821 – 828.

[49] Sisto A,Pilato L,Serventi R,et al. Application specific instruction set processor for sensor conditioning in automo-

tive applications[J]. Microprocessors and Microsystems, 2016,47:375 – 384.
[50] A S Eissa, et al. SHA-3 instruction set extension for a 32-bit RISC processor architecture[A]. 2016 IEEE International Conference on Application-Specific Systems, Architectures and Processors[C]. London: IEEE, 2016. 233 – 234.
[51] 朱方. 基于 MPSoC 的移动视频监控关键技术研究[D]. 南京:东南大学,2016. 1 – 124.
Zhu Fang. Research on key technology of mobile video surveillance on MPSOC[D]. Nanjing: Southeast University, 2016. 1 – 124. (in Chinese)
[52] Mori J Y, Kautz F, Hübner M. Efficient camera input system and memory partition for a vision soft-processor[A]. 2016 International Symposium on Applied Reconfigurable Computing[C]. Rio De Janeiro: Springer, 2016. 328 – 333.
[53] Rakanovic D, Struharik R. Implementation of application specific instruction-set processor for the artificial neural network acceleration using LISA ADL[A]. 2017 East-West Design & Test Symposium[C]. Novi Sad: IEEE, 2017. 1 – 6.
[54] Edwards J, Lemieux G G F. Real-time object detection in software with custom vector instructions and algorithm changes[A]. 2017 IEEE International Conference on Application-Specific Systems, Architectures and Processors[C]. Seattle: IEEE, 2017. 75 – 82.
[55] Rawat H K, Schaumont P. SIMD instruction set extensions for keccak with applications to SHA-3, Keyak and Ketje[A]. 2016 Hardware and Architectural Support for Security and Privacy[C]. Seoul: ACM, 2016. 1 – 7.
[56] Youssef N B H, Youssef W E H, Machhout M, et al. Instruction set extensions of AES algorithms for 32-bit processors[A]. 2014 International Carnahan Conference on Security Technology[C]. Rome: IEEE, 2014. 1 – 5.
[57] 胡绵江,窦勇,倪时策,等. 一种面向加密算法共性子图的指令定制方法[J]. 计算机研究与发展,2012,49(s1):299 – 304.
Hu Jin-jiang, Dou Yong, Ni Shi-ce, et al. A way using common subgraph to customise instruction for encryption algorithms[J]. Journal of Computer Research and Development, 2012, 49(s1): 299 – 304. (in Chinese)
[58] 夏辉,于佳,秦尧,等. 嵌入式领域 ECC 专用指令处理器的研究[J]. 计算机学报,2017,40(5):1092 – 1108.
Xia Hui, Yu Jia, Qin Rao, et al. The researches on the ASIP of ECC in embedded domain[J]. Chinese Journal of Compouter, 2017, 40(5): 1092 – 1108. (in Chinese)
[59] D Wu, et al. Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation[J]. Computer-Aided Design, 2015, 59: 1 – 14.
[60] Wang S, Xiao C, Liu W. Parallel enumeration of custom instructions based on multi-depth graph partitioning[J]. IEEE Embedded Systems Letters, 2019, 11(1): 1 – 4.
[61] Ananthanarayana T, Lopez S, Lukowiak M. Power analysis of HLS-designed customized instruction set architectures[A]. 2017 IEEE International Parallel and Distributed Processing Symposium Workshops[C]. Orland: IEEE, 2017. 207 – 212.
[62] C González-Álvarez, J B Sartor, C Álvarez, D Jiménez-González, L Eeckhout. Accelerating an application domain with specialized functional units[J]. ACM Trans. Archit. Code Optim, 2013, 10(4): 1 – 25.
[63] G Cecilia, et al. MInGLE: An efficient framework for domain acceleration using low-power specialized functional units[J]. ACM Trans. on Architecture and Code Optimization, 2016, 13(2): 1 – 26.
[64] M Karunarathna, Y Tian, C Fidge. Domain-specific application analysis for customized instruction identification[J]. Elsevier Microprocessors and Microsystems, 2014, 38(7): 637 – 648.
[65] A Pulli, C Galuzzi, G Gaydadjiev. Towards domain-specific instruction-set generation[A]. 2014 International Conference on Field Programmable Logic and Applications[C]. Munich: IEEE, 2014. 1 – 4.
[66] Wang C, Gong L, Yu Q, et al. DLAU: A scalable deep learning accelerator unit on FPGA[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, 36(3): 513 – 517.

作者简介

肖成龙 男,1984 年出生,湖南祁阳人,副教授,主要研究方向为软硬件协同设计.
E-mail: chenglong. xiao@ gmail. com

王珊珊 女,1985 年出生,河北任丘人,副教授,主要研究方向为计算机体系结构.
E-mail: celine. shanshan. wang@ gmail. com