软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(8):2371−2396 [doi: 10.13328/j.cnki.jos.005563]

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

基于多面体模型的编译“黑魔法”*

赵 捷[1,2], 李颖颖[1,3], 赵荣彩[1,4]

[1](解放军信息工程大学,河南 郑州 450001)

[2](Département d'Informatique, École Normale Supérieure, Paris 75005, France)

[3](数学工程与先进计算国家重点实验室(解放军信息工程大学),河南 郑州 450001)

[4](中原工学院,河南 郑州 450007)

通讯作者: 赵捷, E-mail: zjbc2005@163.com

摘 要: 基于多面体模型的编译技术发展近 30 年,已经在多个开源编译器和商业编译器中得到了应用和实现.与传统的编译优化模型相比,多面体模型具备应用范围广、表示能力强、优化空间大等优点,代表了程序自动并行化领域众多方向最先进的水平,成为国际上多个编译研发团队的研究热点;同时,多面体模型抽象程度高、实现难度大、面临问题多的特征,阻碍了基于该模型的编译技术在发展相对滞后地区的普及,形成国内专门从事该问题研究的团队屈指可数的现象.为了打开多面体模型的“黑盒子”,首先描述了多面体模型的原理,揭示了基于多面体模型的编译流程,并指出了该领域的主要研究内容;接下来,从程序并行性、数据局部性和其他领域上的扩展应用这 3 个方面对该领域上的研究进展进行了介绍;最后,对该研究领域当前面临的挑战和潜在的研究方向进行了总结.研究目的是通过回顾和总结基于多面体模型的编译技术研究进展,为国内编译研发团队提供重要参考,以期推动我国在该领域上的发展.

关键词: 多面体模型;并行性;局部性;依赖;调度;代码生成;循环分块;数组压缩

中图法分类号: TP314

中文引用格式: 赵捷,李颖颖,赵荣彩.基于多面体模型的编译“黑魔法”.软件学报,2018,29(8):2371−2396. http://www.jos.org.cn/1000-9825/5563.htm

英文引用格式: Zhao J, Li YY, Zhao RC. “Black magic” of polyhedral compilation. Ruan Jian Xue Bao/Journal of Software, 2018,29(8):2371−2396 (in Chinese). http://www.jos.org.cn/1000-9825/5563.htm

“Black Magic” of Polyhedral Compilation

ZHAO Jie[1,2], LI Ying-Ying[1,3], ZHAO Rong-Cai[1,4]

[1](PLA Information Engineering University, Zhengzhou 450001, China)

[2](Département d'Informatique, École Normale Supérieure, Paris 75005, France)

[3](State Key Laboratory of Mathematical Engineering and Advanced Computing (PLA Information Engineering University), Zhengzhou 450001, China)

[4](Zhongyuan University of Technology, Zhengzhou 450007, China)

Abstract: Polyhedral compilation has evolved for nearly three decades, being implemented as a building block or an optional extension of numerous open-source and/or commercial compilers. On the one hand, the polyhedral model is equipped with wider range of applications, more powerful expressiveness and greater optimization space when compared with those traditional models adopted for parallelizing compilers, thus representing the state of the art of almost each domain of parallelizing compilers and becoming a hot topic to

* 基金项目: 国家自然科学基金(61702546)
Foundation item: National Natural Science Foundation of China (61702546)
收稿时间: 2017-10-22; 修改时间: 2017-12-08; 采用时间: 2018-01-29; jos 在线出版时间: 2018-03-13
CNKI 网络优先出版: 2018-03-14 09:18:11, http://kns.cnki.net/kcms/detail/11.2560.TP.20180314.0917.001.html

a great number of international research teams working on compilers. On the other hand, the polyhedral model is also characterized as being difficult in theory, complex for manipulation, and diverse with challenges, hampering its adoption in underdeveloped countries and areas and drawing few researchers working on this topic from the domestic compiler community. Aiming at opening the "black-box" of the polyhedral model, this paper conducts a survey on the "black magic" of polyhedral compilation. First, the underlaying theory behind the polyhedral model is introduced along with a desciption of the general compilation process and an overview of the research directions. Next, the research progress of the polyhedral compilation targeting on parallelism, data locality, and extensions in various application domains is presented. Last but not least, open challenges faced by the polyhedral community and potential research directions on this topic are disussed. The purpose of this work is to provide an important reference for the domestic compiler community by reviewing and summarizing current trends on the polyhedral compilation, and to promote Chinese compiler researchers in making progress on this topic.

Key words: polyhedral model; parallelism; locality; dependence; schedule; code generation; loop tiling; array contraction

随着多核时代的来临,多核架构已成为当今处理器市场的主流.然而,由此引发的多核架构上的并行程序开发也变得异常困难;对于传统的面向单核处理器开发的应用程序,如何将其移植到这些多核架构上,成为一项艰巨的任务.与此同时,多核架构的出现为编译优化和代码生成提出了巨大的挑战,编译程序不仅要考虑如何发掘程序的并行性,而且还要考虑如何提升程序的数据局部性.

除此之外,不同处理器厂商针对各自的处理器架构设计并实现了相应的编程模型,无论是在多核架构上直接开发并行应用程序还是实现传统单核应用程序的移植任务,程序开发人员除了要深入了解目标架构的细节之外,还要越过处理器厂商设计的编程模型"门槛".

这些因素导致了并行应用程序的开发远远落后于处理器架构发展的这一事实,加之程序员固有的思维逻辑使得串行程序的开发显得非常直观和自然.因此,直接在多核架构上开发并行应用程序的技术往往只被少数人掌握.即便是专业人员,也不可能同时实现不同架构的并行应用程序开发.自动并行化工具是在多核架构上开发并行应用程序的一种行之有效但非常具有挑战性的方法,这种方法通过将串行程序自动转换为并行程序,能够让应用程序开发人员从复杂的并行代码编写任务中解脱出来,因此受到了广泛的关注.

1 引 言

基于多面体模型(英文翻译还可以对应为 polytope model、polyhedron model 等)的编译优化技术[1]是程序自动并行化领域的一个研究热点,是解决程序自动并行变换的一种有效手段.与传统的并行化编译器中所采用的幺模矩阵模型[2]相比,多面体模型具备以下几个方面的优点.

(1) 应用范围广:幺模矩阵只能处理完美循环嵌套,而多面体模型对输入程序的约束更少,不仅能够处理完美循环嵌套,而且对非完美循环嵌套的支持也比较完善.

(2) 表示能力强:幺模矩阵只能表示循环交换、倾斜和反转等优化,而多面体模型除了能够处理上述循环变换之外,还能自动实现包括循环分块、合并、分布等在内的几乎所有循环变换技术.

(3) 优化空间大:幺模矩阵 1 次只能实现 1 种循环变换,而多面体模型允许同时实现多个循环变换,极大地提升了并行化编译器优化程序的能力.

得益于这些优点,基于多面体模型的编译优化技术代表了程序自动并行化领域众多方向最先进的水平,成为国际上多个编译研发团队的研究热点.

然而,在这些优点的背后是多面体模型兼备的难点:首先,多面体模型的抽象程度高,涉及集合与映射、谓词逻辑等多个抽象解释领域的理论,多面体模型论文中的大量公式和抽象描述使研究人员很难理解;其次,多面体模型的实现难度大,当前,几乎所有基于多面体模型实现的编译优化技术在实质上都是线性整数规划问题,不仅难以实现,时间和空间复杂度也都很高;再次,多面体模型面临的问题多,考虑到多面体模型在发掘程序并行性和数据局部性方面的能力,越来越多来自不同领域的应用程序对多面体模型提出了更高的要求,这给多面体模型的扩展带来了许多挑战.

上述问题阻碍了基于多面体模型的编译技术在发展相对滞后地区的普及,而我国在编译程序方面的研究相对薄弱,最终形成了多面体模型发展至今已有约 30 年的历史,而国内专门从事多面体模型研究的团队屈指可

数的现象.然而,随着多面体模型的不断发展和完善,从事编译程序开发的研究人员对多面体模型的渴求也越来越强烈.

为了打开多面体模型的“黑盒子”、揭开多面体模型的神秘面纱,本文对当前基于多面体模型的编译技术研究进展进行了回顾和总结,揭示了该编译模型的原理,给出了基于多面体模型的编译流程;从多面体模型研究的两个基本目标——程序并行性和数据局部性的角度展开描述,介绍了其中的相关概念和研究内容;对多面体模型在其他领域上的扩展应用进行了梳理;在文章末尾,本文对多面体模型当前面临的挑战和潜在的研究方向进行了探讨.

本文是对基于多面体模型的编译优化技术进行系统、深入介绍的综述工作,目的在于通过对该研究问题的回顾和总结,为国内从事编译技术研发的团队和人员提供重要参考,以期推动我国在该领域上的发展.为了便于叙述,在无特殊说明的情况下,本文余下内容用多面体编译技术来指代基于多面体模型的编译优化技术,用多面体编译工具指代基于多面体模型的编译器或编译工具和框架.

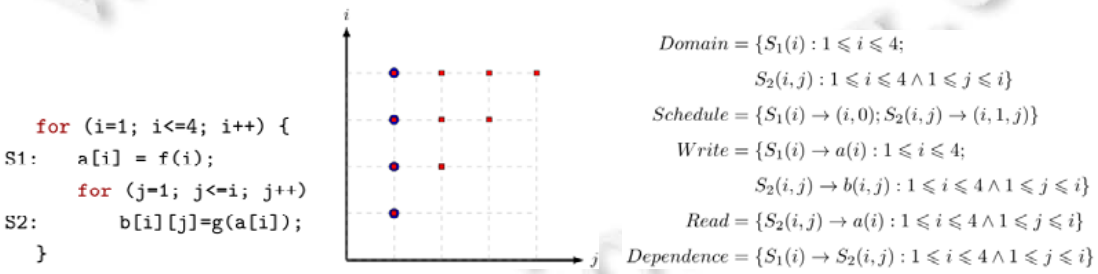
2 概 述

如上文所述,多面体编译技术具备多种优点,使得多面体模型在自动并行化领域得到广泛应用,其实质是利用了多面体模型对程序中的循环进行抽象表示.本节内容将首先介绍多面体编译技术的抽象模型,然后给出多面体编译工具的一般编译流程,最后概述多面体编译技术的主要研究内容.

2.1 多面体模型

多面体编译技术是指在循环边界约束条件下将语句实例表示成空间多面体,并通过这些多面体上的几何操作来分析和优化程序的编译技术,这种模型称为多面体模型.多面体模型通常利用迭代空间(domain)、访存映射(write and read)、依赖关系(dependence)和调度(schedule)表示程序及其语义.其中,迭代空间是循环中每条指令在一次循环迭代下的语句实例集合,访存映射用于表示语句实例与访存数据之间的映射关系,依赖关系指访存相同数据元素(其中至少有 1 个访存类型为写)的两个语句实例之间的偏序关系,调度则用于表示在满足依赖关系的前提下语句实例之间的偏序或全序执行顺序.

如图 1(a)所示循环嵌套,其对应的多面体表示如图 1(b),蓝色圆形对应 S_1 的语句实例,红色正方形对应 S_2 的语句实例;图 1(c)中列出了该循环对应的迭代空间、调度、访存映射关系和依赖关系.



(a) 示例循环嵌套 (b) 图 1(a)循环嵌套的多面体表示 (c) 图 1(a)循环嵌套的集合与映射

Fig.1 A loop nest with its polyhedral representations

图 1 循环嵌套及其对应的多面体模型

多面体编译技术自 20 世纪 90 年代初发展至今,在程序自动并行化领域取得了许多突破,在许多方面代表了这一领域当前最先进的研究水平.除了以 Pluto[3]和 PPCG[4]为代表的多面体编译工具外,多面体模型在标准开发和商业应用中的影响也逐步增大,如 GCC 的 Graphite 框架[5]、LLVM 的 Polly 模块[6]以及多面体模型在 Open64[7]和 IBM XL[8]编译器中的应用.多面体模型不仅可以作为一个编译遍或者模块嵌入到通用编译器中,而且还可以作为独立的编译工具,实现从源程序到源程序的翻译流程.这种独立性和兼容性,使多面体编译技术得

到了更广泛的关注和应用.

随着针对多面体模型研究的不断发展,多面体编译工具形成了一套编译流程.如图 2 所示为多面体编译工具的一般编译流程,虚线框内部分为多面体编译工具的构成部件.多面体编译工具可以用源程序作为输入语言,也可以以通用编译器中间语言作为输入;多面体编译工具的输出可以是源程序或者中间语言,中间语言可以再返回给上层通用编译器,而输出的源程序可以是添加了编译指示或者任意满足目标编程规范的代码.

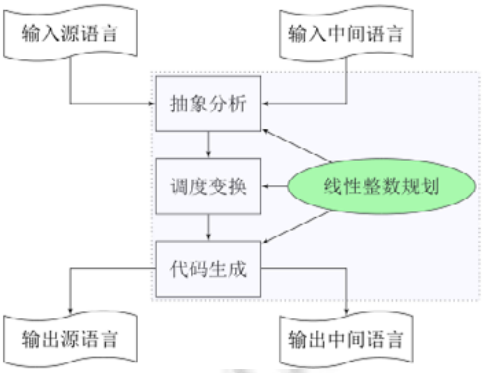


Fig.2 A general compilation process of polyhedral compilers

图 2 基于多面体模型的一般编译流程

多面体编译工具主要由 3 个部分组成.

- 首先,抽象分析构成多面体编译工具的前端,其作用是从输入语言中识别并抽取满足多面体表示约束的程序段,将该程序段表示成多面体形式,并计算迭代空间和访存映射;然后,根据迭代空间和访存映射,通过调用线性整数规划过程计算依赖关系.
- 其次,调度变换构成多面体编译工具的中间优化部分,其作用是在满足依赖关系的前提下,通过调用线性整数规划过程,计算一个充分利用目标程序语言和体系结构特点的调度,循环变换的结果也通过调度变换实现.
- 最后,代码生成构成多面体编译工具的后端,该部分的任务可以再细分为两个阶段:第 1 阶段以迭代空间和调度为输入,借助线性整数规划过程生成抽象语法树;第 2 阶段则是根据目标程序语言规范将抽象语法树转变成最终代码.

线性整数规划过程贯穿于多面体编译工具的整个流程,这是因为多面体编译工具的每个阶段最终都可以转化成线性整数规划问题的求解过程,本文后续内容将对这个问题进行具体描述.因此,多面体编译工具可以看作是一个多次重复解决线性整数规划问题的过程,而线性整数规划问题是一个 NP 难问题,由此可见多面体编译技术的实现难度.

2.2 研究内容

多面体编译的研究内容与图 2 所示的编译流程紧密相关.作为多面体编译工具的前端,抽象分析需要对输入语言进行分析,如果输入程序段满足静态仿射约束,那么将该程序段表示成空间多面体形式.这里的静态约束是指能够在编译阶段判定并能用有限的集合和映射表示程序的控制流和依赖关系.这些集合和映射最终构成多面体模型的迭代空间、访存映射、依赖关系和调度.

与标准编译器相同,多面体编译工具的前端不需要或很少对程序进行优化,抽象分析阶段通常以库实现的方式出现在不同多面体编译工具当中,如 pet[9]、clan[10]和 Polly 是当前多面体编译工具中最常用的 3 个抽象分析库.其中,pet 和 clan 是面向源程序设计的抽象分析库,Polly 是面向中间语言设计的抽象分析库.

为了扩展多面体模型的表示能力,来自国防科技大学的研究团队[11]曾从抽象解释领域对多面体凸集构造问题进行了研究.除此之外,依赖关系分析是多面体编译工具前端的一个重要研究内容,在多面体编译研究初

期,研究人员投入了大量的精力来提升依赖关系分析的能力.

调度变换阶段是多面体编译工具的核心,从多面体模型的几何表示角度来讲,调度变换过程是将程序多面体进行空间几何形状变换的过程,其目的是使变换后的多面体具备更好的程序并行性和数据局部性,而变换的过程就是各种循环变换的组合优化过程.

调度变换也是多面体编译工具最复杂的一个阶段:首先,调度变换与前端计算的依赖关系密切相关,因为调度不仅要满足程序必要的依赖关系来保证变换过程中的程序语义,还要利用部分依赖关系来达到提升数据局部性的目的;其次,调度变换不仅需要考虑所有的循环变换及这些循环变换的组合优化,而且要利用这些循环变换开发不同层次、不同粒度的并行性;再次,调度变换还要为后端代码生成的各种需求提供支持,例如生成CUDA 代码时,线程块内线程之间的同步指令的生成.

代码生成阶段的研究内容主要集中在从多面体表示到抽象语法树的生成部分,因为从抽象语法树到最后目标程序语言的代码生成只需要根据抽象语法树向目标文件中输出相关指令语句即可.抽象语法树的生成以变换后的调度和迭代空间为输入,前者规定了指令语句的执行顺序,后者限定了循环的边界和控制流的判定条件.从多面体表示到抽象语法树的生成过程也被称为多面体扫描,多面体的扫描方式和生成代码的优化是这部分的主要研究内容.

除了围绕上面 3 个阶段展开研究之外,线性整数规划工具是整个多面体编译工具的基础和各项研究内容的核心.和抽象分析库一样,线性整数规划工具一般以库的形式封装在多面体编译工具内.根据需求不同,各种线性整数规划工具的功能也不尽相同,目前,在各种多面体编译工具中使用的线性整数规划工具包括 isl[12]、Omega[13]、PIP[14]、Polylib[15]和 PPL[16]等.

上述几个研究内容主要针对程序并行性这一目标展开研究.随着多核和片上存储技术的不断演化,数据局部性成为多核架构上影响程序性能的关键因素,因此,提升数据局部性成为多面体模型甚至是程序自动并行化领域的另一研究目标.针对数据局部性,多面体编译研究人员也展开了大量的研究,这部分研究内容主要包括循环分块和数组压缩.

除此之外,多面体模型在其他领域中也不断得到渗透,包括在迭代编译、领域特定语言以及与运行时技术相结合的应用.因此,本文余下内容将从程序并行性、数据局部性以及多面体模型在其他领域的研究现状展开叙述.第 3 节针对以开发程序并行性为目的的研究内容进行描述,包括依赖关系分析、调度变换和代码生成.第 4 节介绍以数据局部性为目的的研究内容,包括循环分块和数组压缩.第 5 节陈述多面体模型在其他方向上的应用,包括迭代编译、针对多面体模型的优化和扩展、面向不同体系结构的后端、领域特定语言以及与运行时技术相结合等几个方面的内容.第 6 节分析多面体编译技术当前面临的挑战,并根据当前研究现状和现有的研究经验,探讨未来可能的研究方向.第 7 节总结全文.

3 程序并行性

多面体模型研究初期,多核技术的发展还处于初期阶段,为了简化模型,研究人员往往忽略通信或同步开销.然而,多核架构在当今市场上已随处可见,早期的一些多面体编译技术已不再适用于当前的体系结构;与此同时,随着多面体模型自身的发展,一些技术也已过时或被淘汰.限于篇幅,本文将主要针对最近十几年的多面体编译技术研究现状进行阐述;对于早期的多面体编译研究工作,将介绍开创性工作和目前仍在多面体编译研究领域中发挥重要作用的技术.

3.1 依赖关系分析

依赖关系是实现程序自动并行化的基础,包括循环变换在内的任何重排序变换只有在满足依赖的基本定理的前提下才是合法的[17].同样地,依赖关系分析在多面体编译技术中也发挥着至关重要的作用;不仅如此,多面体编译技术对依赖关系分析结果的精度要求更高,因为多面体模型考虑的是语句在循环一次迭代下的实例与另一个循环迭代中实例之间的依赖关系.

Feautrier[18]面向多面体模型首次实现了一项精确的依赖关系分析技术.Feautrier 指出,从循环嵌套内一个

语句实例 S_1 到另外一条语句实例 S_2 之间存在依赖,必须满足 4 个条件:(1) S_1 和 S_2 访存同一地址单元;(2) S_1 和 S_2 之中至少有 1 个是向该地址单元写入数据;(3) 存在一条从 S_1 到 S_2 的程序执行路径;(4) S_1 是所有满足前面 3 个条件中程序执行路径上离 S_2 最近的一个语句实例.这种将依赖关系的分析结果从语句精确到语句实例的方法被称为数据流分析.

数据流分析比传统的依赖关系分析技术如 Omega 测试[19]更精确,原因在于传统的依赖关系分析只考虑上述 4 个条件中的前 3 个,也就是说,传统的依赖关系分析只能精确到语句.虽然 Omega 测试本身是作为传统依赖关系分析出现,但 Omega 测试本身也可以看作是利用多面体模型来分析依赖关系的过程,因此,Omega 测试很容易扩展成数据流分析[20],也可以扩展成一种线性整数规划工具[13].

Maydan 等人[21]对 Feautrier 提出的数据流分析进行了进一步优化,提出了终写树的概念.Malsov[22]根据第 4 个约束条件,从程序执行路径上从后向前逆向寻找满足条件的依赖源点,在保证精确性的前提下,优化了时间复杂度.这些数据流分析技术在当前的多面体编译工具中仍发挥着重要作用,也将依赖关系的研究领域进一步细化为数据依赖关系和存储依赖关系.

由于数据流分析将依赖关系的分析结果从语句精确到语句实例,因此,其实现复杂度在传统依赖测试算法的基础上更难,这使得数据流分析在很长一段时间内只能适用于代码量很小的程序片段.Vasilache 等人[23]在大型测试集上实现了数据流分析,并利用分析结果计算出不同循环变换组合下潜在的依赖冲突,保证了生成代码的正确性;该技术后来被应用到 GCC 的 Graphite 模块中,用以消除存储依赖关系对开发程序并行性带来的负面影响[24].

多面体模型中的依赖关系分析为许多并行编程模型的分析和优化提供了基础.例如,Pellegrini 等人[25]利用多面体模型的依赖关系分析结果来实现 MPI 程序的通信优化;Yuki 等人根据同样的思想,提出了 X10 程序的数据流分析技术[26].这些工作的实现,进一步拓展了多面体模型的应用,尤其是数据流分析在其他编程模型方面发挥的作用.

除了提升依赖关系分析结果的精确性以外,多面体模型中的依赖关系分析研究工作还包括对特定的循环变换消除伪依赖,如,Baghdadi 等人面向循环分块提出了一种存储依赖消除的方法[27];Verdoolaege 和 Cohen[28]基于该技术实现了一种活跃变量生命周期的分析,在一些特定环境下能够提升调度算法的能力.

Omega 测试代表了传统依赖关系分析技术最先进的水平,而多面体编译中的数据流分析则可以看作是 Omega 测试的一种扩展.Omega 测试是求解线性整数规划问题的过程,所以多面体编译中的依赖关系分析也是一个线性整数规划问题的求解.由于精度要求的提高,多面体编译中的数据流分析的复杂度也比传统的依赖关系分析复杂度要高.

3.2 调度变换

调度变换是多面体编译技术的核心.调度变换是指在满足依赖关系的前提下,将一种调度转换成另外一种调度的过程.在程序自动并行化领域,这种变换过程一般是以提升程序的并行性和数据的局部性为目的.对于多面体编译工具而言,循环嵌套内的语句首先被表示成空间多面体形式,然后才进行后续分析和变换;多面体模型上的调度变换实质上就是多维空间几何的变基过程.

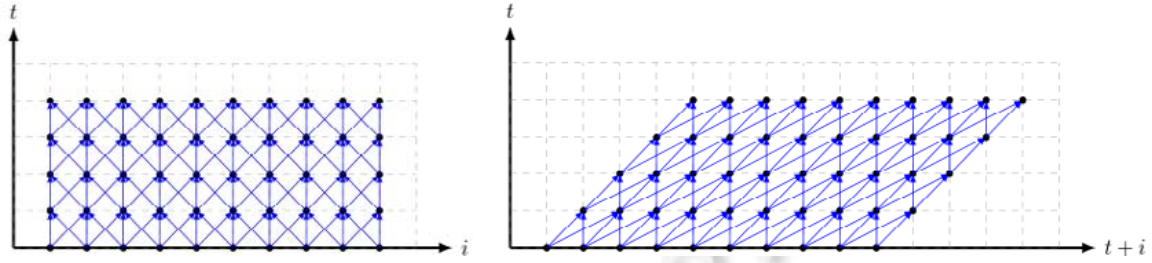
如图 3(a)所示是一维空间上 stencil 计算的循环嵌套.stencil 计算是一种通过利用空间上某一点的近邻元素对该点元素进行迭代更新的过程,在计算流体力学、图像处理、偏微分方程等领域都有十分广泛的应用,因此也是多面体编译技术优化的主要对象之一.

图 3(b)是该 stencil 计算调度变换前的空间多面体表示及其依赖关系,以 t 轴和 i 轴为该迭代空间的基;图 3(c)是调度变换后的空间多面体表示及其依赖关系,以 t 轴和 $t+i$ 轴为该迭代空间的基.图中每个点代表一次计算迭代,蓝色箭头表示迭代计算之间的依赖关系.在图 3(b)中,所有沿 i 轴平行的点都可以并行执行,但如果不对该循环嵌套进行循环变换的话,无法实现沿 t 轴和 i 轴方向的循环分块,这样就无法利用 t 轴上的数据局部性.对于图 3(c),$t+i$ 轴上的点并行性仍存在,与此同时,沿着 t 轴和 $t+i$ 轴的方向对迭代空间进行分块不会导致分块后的两个子区间存在依赖环,因此该图对应的调度可以有效利用 t 轴上的数据局部性.对比图 3(b)和图 3(c)不难发现,

该调度变换就是通过循环倾斜将原有的基(t,i)变换成新的基$(t,t+i)$.

```
for (t=0; t<T; t++)
    for (i=1; i<N-1; i++)
        A[t+1][i]=0.25*(A[t][i+1]+2.0*A[t][i]+A[t][i-1]);
```

(a) 一维空间上的 stencil 计算



(b) 调度变换前的空间多面体表示 (c) 调度变换后的空间多面体表示

Fig.3 One dimensional stencil computation with its polyhedral representations before and after scheduling

图 3 一维空间上的 stencil 计算及其调度变换前后的多面体表示

上述内容说明了调度变换的过程,从多面体模型被提出至今,研究人员针对调度变换进行了深入的研究,调度变换算法也可以分为 3 类.

(1) Feautrier 调度算法

Feaurtrier 最先针对多面体模型中的调度变换问题展开了研究[29,30],该算法将循环迭代抽象成执行序列,循环迭代在空间多面体上的坐标表示可以看作是该循环迭代对应的时间节点.由于输入的循环嵌套是串行程序,因此每个循环迭代都有唯一的坐标表示,即输入循环嵌套的每个迭代都有唯一的时间节点.Feautrier 调度算法的思想就是在满足依赖关系的前提下,对每个循环迭代计算一个新的执行时间节点,经过 Feautrier 调度算法变换后的循环迭代按照该执行时间节点按序执行.由于 Feautrier 调度算法可能对一些循环迭代的时间节点进行修改,因此调度变换后,不同的循环迭代可能具有相同的时间节点.此时,这些循环迭代就可以被并行执行,但不同的循环迭代在空间上被分布到不同的处理器上,以此达到提升程序并行性的目的.Feautrier 调度算法先对时间进行排序,然后再对空间进行划分,所以对于循环嵌套,该算法计算的调度往往实现的是内层循环的并行.

与此同时,由于循环嵌套外层被抽象成时间,外层循环串行执行,因此该算法计算出的调度无法直接沿外层时间调度进行循环分块,在局部性问题上具备一定的缺陷.为弥补 Feautrier 调度算法的这一不足,Griebl 等人[31]实现了一种前向通信算法,其思想是:当所有时间调度上的依赖方向都与程序执行顺序相同时,对时间调度进行分块,从而避免因循环分块导致的时间调度方向上的依赖环.

(2) 基于划分平面的调度算法

Feautrier 调度算法的前提是处理器之间的通信或同步开销相对于计算可以忽略或完全不计,其计算结果也往往是趋向于内层并行,通信和同步频率较大,然而这个前提条件在当今多级存储的体系结构环境下显然不再适用.

由于调度变换实质上是变基过程,如图 3 中所示将原基(t,i)变换新基$(t,t+i)$,因此调度变换就是计算将原基(t,i)变换成新基$(t,t+i)$所需的一个 2×2 大小的矩阵.Lim 等人提出了一种以外层并行为目的的调度算法[32,33],其思想是:将该矩阵抽象成多面体,构成该矩阵的每一行抽象成不同的划分平面,那么调度变换的过程就可以转换为计算每一个划分平面的所有元素的过程.由于调度变换需要满足依赖关系,因此每个划分平面与依赖距离向量的乘积必须大于等于 0,从而所有的划分平面和依赖距离向量的乘积可以构成多个不等式,这些不等式构成计算线性整数规划问题的前提条件,最终利用线性整数规划工具计算出一个满足所有不等式的整数解集合,即矩阵的所有元素.在计算划分平面的过程中,该算法尽量让依赖关系在内层得到满足,这样,外层划分平面对应的

循环就可以被并行执行,从而提高并行粒度,并减少通信和同步频率.

(3) Pluto 算法

Lim 等人提出的划分算法虽然能够提高并行性,并减少通信和同步开销,但这种最优化是将通信数据量最优化到某个阶,如将通信数据量从 c_1n^3 降低到 c_2n^2,其中,c_1 和 c_2 为常数系数,n 为通信数据相关变量.也就是说,Lim 等人的调度算法没有实现常数系数 c_2 的最小化.针对这一问题,Bondhugula 等人[34]设计了一种以通信数据量最优化为目的的代价模型,并基于该代价模型依次求解调度变换所需的划分平面.与 Lim 等人基于划分平面的算法思想相同,该算法也是以计算出为寻求变基的矩阵中所有元素为目的,不同的是,该算法在上述多个不等式的基础上又添加了一个由循环边界等程序常数参数限定的不等式,这个新的不等式就是该算法的代价模型.这种代价模型的引入,导致线性整数规划过程未知变量的个数有所增加,但计算出来的矩阵与 Lim 等人计算出的矩阵不同,最终得到的调度变换结果也比 Lim 等人计算的调度具备更少的通信量.该调度算法是实现 Pluto 编译器的核心内容,因此被称为 Pluto 算法.同时,由于该算法中引入了代价模型,在一些文章中也用基于代价模型的调度变换算法来指代 Pluto 算法.

Pluto 算法的代价模型实现了通信数据量常数系数的最优化,但该算法求解出的划分平面中不允许负数的存在,因此,Pluto 算法无法实现循环反转或带有负数系数的循环倾斜.为了弥补 Pluto 算法的不足,Acharya 等人在 Pluto 算法基础上实现了 Pluto+算法[35,36].

上述几种调度变换算法具有不同的使用范围,但在多面体编译工具中,这些调度算法有时会协同工作,以期获得最优的调度变换结果.例如,PPCG 中不仅可以调用 Pluto 算法,也可以调用 Feautrier 调度算法,这样可以满足不同并行层次的挖掘.

然而,调度变换也是一个线性整数规划问题.调度变换可被看作是求解由划分平面构成的矩阵中所有元素的过程,求解新的调度需满足程序的依赖关系,这种约束被抽象成该矩阵与依赖向量之间的矩阵向量乘运算,运算结果的每个分量都大于等于 0.将这些约束构成的不等式传递给线性整数规划工具,由线性整数规划工具计算出满足这些约束条件的最小解.因此,调度变换的时间复杂度较高.如何降低调度变换的时间复杂度,也是研究人员关注的一个话题,例如,Upadrasta 和 Cohen[37]实现了一种在多项式时间内计算简单调度变换的算法.

3.3 代码生成

代码生成也称为多面体扫描,是多面体编译技术面向不同目标平台实现优化的依据.与调度变换不同,代码生成并未对程序进行变换,因此不需要考虑依赖关系.代码生成的输入是调度变换阶段生成的调度以及程序原有的迭代空间.代码生成的任务就是将迭代空间上的每个点,按照调度指定的顺序和规则生成抽象语法树.代码生成的核心问题是如何生成每个循环的边界以及如何生成每个控制流的判定条件.

多面体模型的代码生成可以划分为两种方法,为便于说明,假设有如图 4(a)所示的迭代空间,其中,红色正方形代表语句 S_1 的所有迭代,蓝色圆形代表语句 S_2 的所有迭代.我们将利用该图来说明两种代码生成方法的不同.

(1) 聚合代码生成方法

经过调度变换算法之后,程序在新基上进行多面体表示,而代码生成将对这个新的多面体表示进行扫描.在对多面体表示进行扫描时,最关键的问题是如何生成循环的边界,即如何确定多面体的区间.

Ancourt 和 Irigoin[38]提出的方法是多面体模型代码生成方面的开创性工作,他们的方法主要针对单个循环嵌套进行分析,在生成循环边界时,利用 Fourier-Motzkin 消去法由外向内依次生成循环的边界.这种方法仅适用于完美循环嵌套,并且要求循环内所有语句的调度完全一致.此外,该方法还要求循环步长为 1.

尽管 Ancourt 和 Irigoin 的方法有很多局限性,但是该方法的思想为后续聚合代码生成方法提供了依据.虽然后来有一部分学者针对上述局限性(如循环步长必须为 1 的情况)进行了完善,但这些方法生成的代码都是一个完美循环嵌套下包含多个控制流的形式,这些控制流用于保证生成的代码只执行必要的迭代.

Kelly 等人针对 Ancourt 和 Irigoin 的方法要求循环内所有语句都必须采用完全相同调度的问题,提出了一种通过代码复制来降低或消除完美循环嵌套中控制流开销的代码生成方法 CodeGen+[39].对于采用不同调度的语句,如图 4(a)所示的两个语句 S_1 和 S_2,其对应的多面体表示不完全一致,因此,CodeGen+算法首先计算出这两

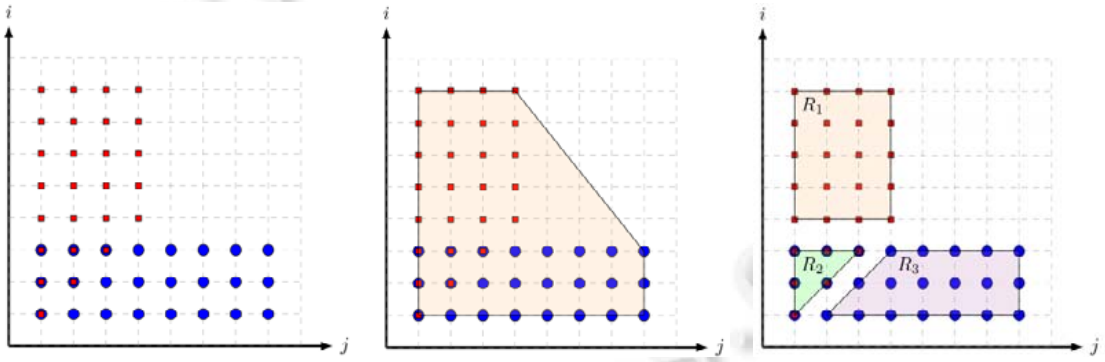
个多面体表示的凸包,如图 4(b)所示,该凸包对应一个完美循环嵌套;与 Ancourt 和 Irigoin 的方法不同,CodeGen+算法是在必要时将完美循环嵌套分成多个部分,从而生成非完美循环嵌套,这样可以避免生成的代码先迭代整个凸包然后再利用控制流语句来消除凸包内的无用迭代的问题.

由于 CodeGen+算法是先计算语句多面体表示的凸包然后再由最内层循环依次向上来消除控制流开销的过程,因此我们将此类方法称为聚合代码生成方法.Ancourt 和 Irigoin 的方法及其改进可以看作是聚合代码生成的特例.Chen[40]在 CodeGen+算法基础上实现了一种对生成代码控制流开销和代码规模进行折衷的方法,该方法与 Presburger 运算结合,使 Codegen+算法可以在迭代空间上实现各种复杂的优化策略.Codegen+算法是 Omega 库的代码生成算法.

(2) 分割代码生成方法

与聚合代码生成方法首先计算多面体凸包的方法不同,Quilleré 等人提出了一种首先对语句多面体表示进行分割,然后再逐个扫描这些多面体的代码生成方法[41].如图 4(c)所示是这类方法对多面体表示进行分割的示意图,该方法首先将语句 S_1 和 S_2 的多面体表示分割成 3 个部分,其中,R_1 部分只包含语句 S_1 的迭代,R_2 部分同时包含语句 S_1 和 S_2 的迭代,R_3 部分只包含语句 S_2 的迭代.对这 3 部分依次扫描后,可以得到 3 个不同的循环嵌套,并且无需对生成的代码进行控制流语句的消除.一般情况下,相对于聚合代码生成方法,Quilleré 等人的算法生成的代码规模较大.

Bastoul[42]将 Quilleré 等人提出的分割算法进行改进后,实现了一个代码生成工具 CLooG,被包括 Pluto,PPCG 在内的多个多面体编译工具作为代码生成阶段的算法.Bastoul 不仅在 CLooG 中实现了 Quilleré 等人的算法,而且针对生成代码规模较大的问题进行了优化.由于代码生成算法依赖于变量消去法,因此即使利用分割代码生成方法也存在大量的控制流.针对这个问题,Vasilache 等人[43]对 CLooG 算法进行了改进,该方法同时考虑了循环变换本身的特征,例如利用循环分段(strip-mining)的特征来消除取模操作带来的控制流开销.针对 CLooG 算法中无法避免的控制流开销问题,Razanajato 等人[44]提出了一种在 CLooG 算法基础上再进行分割的方法,即对图 4(c)中分割出的 3 个部分再进行分割,该方法目前还在研发当中.



(a) 示例迭代空间 (b) 聚合代码生成 (c) 分割代码生成

Fig.4 An illustrative iteration space and its diagrams of different code generation algorithms

图 4 示意迭代空间及其代码生成方法示意图

随着面向异构系统的多面体编译工具 PPCG 的诞生,多面体模型的代码生成对存储布局优化和代码多版本控制有了更明确的要求.Grosser 等人[45]基于分割多面体表示的思想,针对 PPCG 的代码生成模块进行了优化,不仅实现了代码生成阶段的微调功能,使得生成的代码在规模和控制流上得到了改进,而且他们还设计了调度树表示来支持异构系统编程模型中线程之间同步语句的生成.

无论是聚合代码生成方法还是分割代码生成方法,其实质上都是线性整数规划问题.这是因为这些代码无论是通过聚合还是分割之后,都要扫描 1 个或多个多面体表示,而扫描单个多面体表示时需要利用变量消去法

逐步求解循环边界,这个过程实质上是一个线性整数规划问题.以 Ancourt 和 Irigoin 提出的方法为例,该方法使用 Fourier-Motzkin 消去法来消除变量,其他变量消去法还包括对偶单纯形法等.

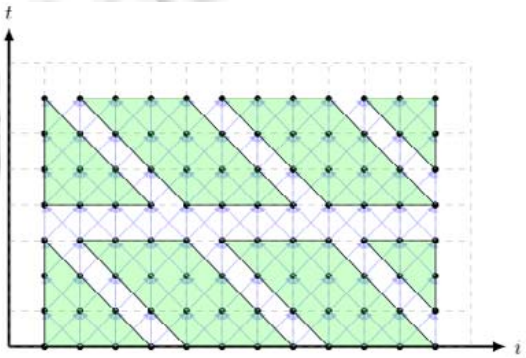
4 数据局部性

在多面体编译技术研究初期,利用多面体模型发掘程序的并行性是其最主要的研究内容.随着异构系统和多级存储结构的发展,数据局部性成为影响程序性能的另一个关键因素.循环分块和数组压缩是多面体编译技术中面向数据局部性研究的两种最重要的技术,可以通过基于划分平面的调度算法计算出满足条件的循环分块和数组压缩方案[46],因此,循环分块和数组压缩也可以看作是求解线性整数规划问题的过程.

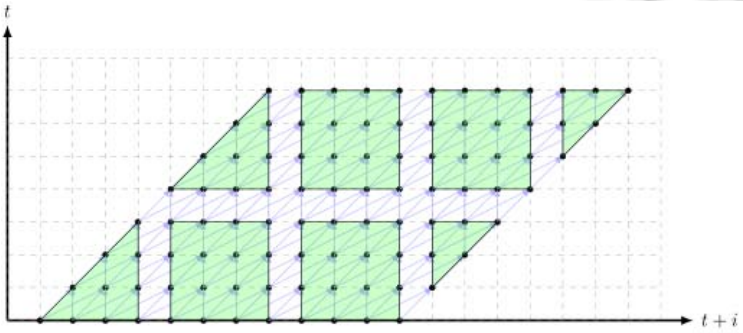
4.1 循环分块

循环分块是指通过增加循环嵌套的维度来提升数据局部性的循环变换技术,循环分段可以看作是循环分块在单层循环上的特例,区别是循环分块需在循环分段的基础上再进行循环交换.循环分块是多面体模型提升数据局部性最关键的一个循环变换[47],也是多面体编译技术核心内容——调度变换的主要目标之一.循环分块的研究可以分为面向分块形状的研究和面向分块大小的研究.

当前最先进的调度变换算法,如 Pluto 算法可以自动实现循环分块,然而这些分块技术往往只能实现简单的平行四边形分块.如图 5 所示是 Pluto 算法对图 3(a)中所示一维 stencil 计算实现的循环分块,当以调度变换前的基为参照时,分块为平行四边形,如图 5(a)所示;如果以调度变换后的基为参照,则分块仍是长方形,如图 5(b)所示.当描述分块形状时,往往是以调度变换前的基为参照.



(a) 以调度变换前的基为参照的分块示意图



(b) 以调度变换后的基为参照的分块示意图

Fig.5 Diagrams of rectangular tiling and parallelogram tiling

图 5 长方形分块和平行四边形分块示意图

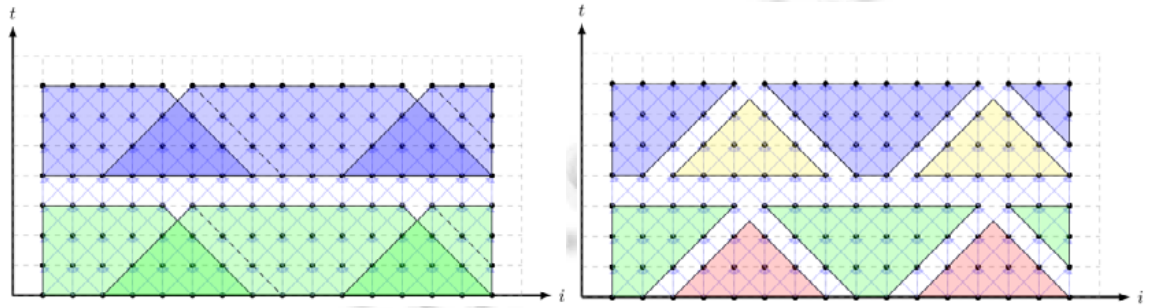
平行四边形分块虽然提升了数据局部性,但分块之间只能实现流水并行.为了克服平行四边形分块的缺点,

使循环分块后分块之间也能完全并行,研究人员在多面体模型下研究了以下 3 种形状的循环分块.

(1) 交叉分块和分裂分块

Krishnamoorthy 等人针对 stencil 计算提出了交叉分块和分裂分块[48],如图 6(a)和图 6(b)所示分别是交叉分块和分裂分块的示意图.对于交叉分块,其实质是先实现平行四边形分块(如图 6(a)中虚线所示是原来平行四边形分块形成的形状),然后对于每个分块,将在前一个分块内的所有依赖源点也划分到该分块中,这样就形成了两个分块之间的交叉.由于每个分块所需的依赖源点都在该分块内,因此水平方向上的分块都可以并行执行,但交叉部分被重复计算.

为了消除交叉分块导致的冗余计算问题,可以将交叉分块进行分裂,如图 6(b)所示,原来交叉的部分作为分块的第 1 阶段,未交叉部分作为分块的第 2 阶段.将分块分裂后,水平方向上所有分块的第 1 阶段先并行执行,然后再并行执行所有分块的第 2 阶段.分裂分块的两个阶段之间必须串行执行.



(a) 交叉分块示意图 (b) 分裂分块示意图

Fig.6 Diagrams of overlapped tiling and split tiling

图 6 交叉分块和分裂分块示意图

Krishnamoorthy 等人在 CPU 上实现了这两种循环分块方法,为了能够在 GPU 上实现分块之间的并行,Holewinski 等人[49]在 GPU 上实现了交叉分块,Grosser 等人[50]则在 GPU 上实现了分裂分块.

(2) 钻石分块

Bandishti 等人[51]针对 stencil 计算提出了钻石分块,如图 7 所示是钻石分块的示意图.与交叉分块和分裂分块不同,钻石分块不会试图寻找平行四边形分块,而是通过 Pluto 算法寻找能够满足钻石分块的划分平面,然后沿着这些划分平面对迭代空间进行划分,从而得到如图 7 所示的钻石形状的分块.图中所有相同颜色的分块都可以并行执行.

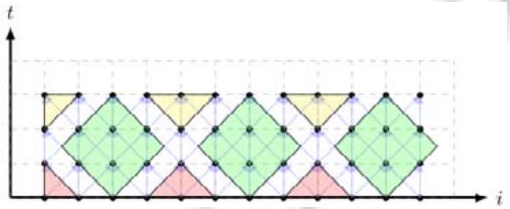


Fig.7 Diagram of diamond tiling

图 7 钻石分块示意图

由于钻石分块既没有重复计算的问题,也不会涉及到块内分裂而不同阶段必须串行执行的麻烦,因此,钻石分块被扩展到其他计算领域.Pananilath 等人[52]利用钻石分块来优化计算流体力学中的格子玻尔兹曼方法,Ghysels 等人[53]利用钻石分块来优化数值分析中求解微分方程的多重网格方法,Bondhugula 等人[54]对潜水方程的有限差分模型进行了优化.

为了保证分块之间能够并行执行,钻石分块往往会排除 Pluto 算法计算出的最优调度变换划分平面,这些由 Pluto 算法计算出的最优划分平面虽然在并行性上不如钻石分块计算出的划分平面,但在数据局部性上优于后者.当利用钻石分块计算出的划分平面对调度进行变换后,分块内的循环迭代可以再使用 Pluto 算法来提升数据局部性.针对这个问题,Shretha 等人[55]提出了一种实现钻石分块后再对分块内进一步分块的多级分块技术.

(3) 六角形分块

Grosser 等人[56]面向 stencil 计算提出了六角形分块.如图 8 所示是六角形分块的示意图.钻石分块可被看作是六角形分块的一种特殊形式,与钻石分块相比[57],六角形分块的顶端可以沿着水平方向按需扩展,这种可扩展性带来的好处是,分块内部一次迭代中可以使用向量化等手段实现分块间和分块内的多级并行.

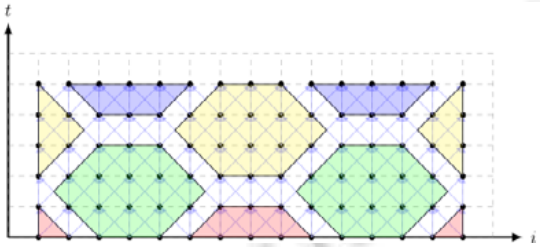


Fig.8 Diagram of hexagonal tiling

图 8 六角形分块示意图

在处理多维 stencil 计算时,Grosser 等人只在时间和空间的第 1 维上进行六角形分块,而在剩余的空间维度上进行平行四边形分块,这种混合分块模式在保证分块之间沿时间维上的并行性的同时,减少了生成的代码中控制流开销,提升了程序的性能.这种混合分块模式也为后续研究提供了基础,Bondhugula 等人[58]将这种混合模式应用到钻石分块中.

分块大小是多面体模型下循环分块的另一个研究内容.循环分块的大小往往与目标体系结构的参数相关,如果在生成分块代码时指定静态参数,那么用户必须通过多次调整参数大小来确定最佳选择.循环分块的大小无法在编译阶段获得,因此研究人员要解决的问题是如何生成循环分块大小为运行时参数的代码.

针对该问题,Renganarayanan 等人[59,60]通过将这些约束分块间和分块内的边界添加到代码生成阶段的线性整数规划问题中,利用代码生成工具生成分块大小为运行时参数的代码.Kim 等人[61]在 Renganarayanan 等人工作的基础上实现了多级循环分块带有运行时参数分块大小的代码自动生成,但他们的方法只能处理完美循环嵌套.

Hartono 等人[62]实现了非完美循环嵌套带有参数分块大小的代码自动生成,但他们的方法生成的代码只能串行执行,也就是说,他们的工作只关注如何提升数据局部性;之后,为了提升程序的并行性,他们又在前期工作基础上实现了并行代码的自动生成[63].

Mehta 等人[64]在多级分块的基础上,在代码生成时将分块内的向量化问题也考虑在内,基于 Pluto 编译器实现了一个循环分块的代码生成算法.与前面几个方法生成带有运行时参数的循环分块大小不同,Mehta 等人的方法是根据目标体系结构计算出一个最优的循环分块大小,生成的代码中并不包含动态参数.

无论是面向分块形状的研究还是面向分块大小的研究,这些问题实质上都是线性整数规划问题.这是因为:首先,循环分块形状的选择是通过调度变换的划分平面寻找到分块的方向,调度变换本身是线性整数规划问题,所以循环分块形状的选择也是线性整数规划问题;其次,循环分块大小的选择则是通过向代码生成的线性整数规划问题添加新的约束来生成带有参数的循环分块大小,因此循环分块大小的研究也是线性整数规划问题.

另一个值得一提的问题是关于循环分块形状的参照基.当前,所有的循环分块形状都是参照调度变换前迭代空间的基来描述的.事实上,如果以调度变换后的迭代空间的基为参照,那么这些循环分块在调度变换后的迭代空间上仍然是长方形.因此,循环分块形状的选择实际上也是调度变换的过程.

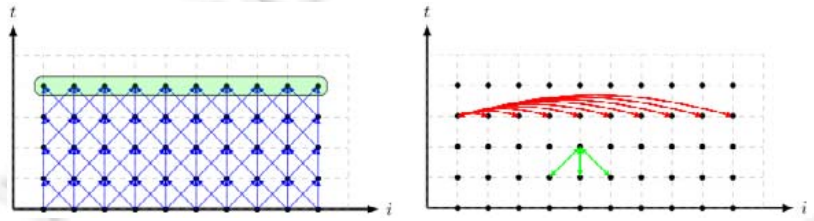
4.2 数组压缩

如图 3(a)所示的 stencil 计算代码,迭代过程中使用了与语句迭代实例个数相同的数组,即该代码中语句实例的个数与其写入数组元素的个数相同,每个语句迭代实例都向各自的数组元素中写入数据.因此,迭代空间和数据存储空间是完全一致的,如图 9(a)是图 3(a)所示 stencil 计算的数据存储空间示意图,其中最后一次迭代存储的数据构成该循环嵌套的向下暴露集.

在多核架构和多级存储结构中,缓存空间往往十分有限,如果程序所需的数据存储空间大于缓存空间,存储延迟会成为阻碍程序性能提升的关键.因此,为了减少程序的数据存储空间,研究人员考虑的问题是数据存储空间是否有必要和语句迭代空间完全一致.如图 9(a)所示,向下暴露集中的数据必须在计算结束后仍处于活跃状态,而所有中间迭代写入的数据都是临时数据.数组压缩研究的问题就是如何根据临时数据活跃变量周期之间的关系来减少数据存储空间的大小.

多面体模型中的数组压缩基于数组下标冲突关系[65]设计实现.数组下标冲突是指不同数组下标对应的数据元素在相同调度下同时处于活跃状态而不能存储在同一个地址单元的冲突,由所有这样的数组下标构成的集合称为数据冲突集.如图 9(b)所示是图 9(a)中数据中的冲突关系,红色弧形双向箭头代表一次迭代中数据之间的冲突关系,绿色直线双向箭头代表不同迭代之间数据之间的冲突关系.

从图 9(b)中可以看出,相同迭代内任意两个数据之间都存在冲突关系,因此数据沿空间方向无法压缩;不同迭代内的数据,由于当前迭代写入的数据只和上一次迭代的数据之间存在冲突关系,因此只有相邻两个迭代之间的数据无法存储在同一个数据单元.从而可以得到如图 9(c)所示的代码,该代码需要存储的数据个数为 $2N$,当迭代次数较多时,这种技术可以极大地减少所需的存储空间.



(a) 数据空间存储 (b) 迭代间和迭代内数组下标冲突关系

```
for (t=0; t<T; t++)
    for (i=1; i<N-1; i++)
        A[(t+1)%2][i]=0.25*(A[t%2][i+1]+2.0*A[t%2][i]+A[t%2][i-1]);
```

(c) 数组压缩后一维空间上的 stencil 计算

Fig.9 Diagrams of the data space and conflict relations of the one-dimensional stencil computation and its transformed code

图 9 一维 stencil 计算数据存储空间和数组下标冲突关系示意图及其变换后的代码

当前多面体模型下的数组压缩大多都是利用上述模运算来压缩数据空间,主要可以分为 3 类.

(1) 基于 UOV(universal occupancy vector)的压缩方法

在还没有数组下标冲突关系的定义以前,Strout 等人[66]提出了一种基于全局共享向量 UOV 的数组压缩方法.所谓全局共享向量,是指能够利用相同存储单元存储但相互之间有依赖关系的两次迭代之间的依赖向量,由于 stencil 计算的特性,这种依赖关系可以扩展到整个数据空间.对于具有全局共享向量的两次迭代,依赖汇点必须在依赖源点的生命周期结束之后才能执行,也就是说,依赖汇点不仅与依赖源点之间存在冲突,而且与依赖源点在其生命周期内所有的读引用存在冲突,这种传递冲突关系的过程称为传递闭包.

全局共享向量具有较大的局限性:首先,全局共享向量在整个数据空间上必须一致;其次,向量的每个分量必须是常数.Thies 等人[67,68]扩展了 Strout 等人的工作,用调度变换的方法计算更一般的依赖关系条件下的数组

压缩问题,但是他们的调度变换算法只能压缩多维数组的某一维.

(2) 基于格的压缩方法.

Darte 等人[65]对数组下标冲突关系给出了定义,并在此基础上提出了基于格的数组压缩方法,其主要思想是利用格求解能够将数组空间压缩到最小维度的模运算映射函数.假设 n 维空间上有 m 个线性无关整数向量,用这 m 个线性无关整数向量构成一组基,那么该 n 维空间上所有能用该组基表示的整数集合被称为一个 m 阶格.另一方面,他们将模运算映射函数抽象成范畴论中的态射问题,利用态射问题的核与格之间的关系求解模运算映射函数,并最终计算出满足条件的模运算映射关系.

Alias 等人[69]对基于格的压缩方法进行了实现:首先,利用 ROSE 编译器中的 Bee 模块来分析程序中活跃变量的生命周期等相关信息;然后,再利用 Cl@k 工具来寻找满足条件的格,并由此求解最终的模运算映射.最近,Darte 等人对他们提出的这种基于格的方法进行了扩展[70],用于处理循环分块、流水并行等情况下的数组压缩问题.基于格的数组压缩方法主要的问题在于只能处理数组内的压缩.

(3) 基于划分平面的压缩方法.

Lefebvre 等人[71]是多面体模型下面向数组压缩研究的先驱,他们的工作为后续研究奠定了基础.他们最主要的贡献在于提供了一种实现数组压缩的充分条件,即在进行数组压缩之前需要将数据空间扩展成与语句迭代实例一一对应的大小,以此消除反依赖和输出依赖的影响,然后沿着数据空间的基的方向寻找模运算的基,所有模运算的基的乘积就是压缩后的数组空间大小.他们还利用相交图的方法给出了一种数组之间空间压缩的方法.

Lefebvre 等人的方法只能沿着数据空间基的方向寻找模运算的基,即他们的方法只能沿着图 9(a)中 t 轴和 i 轴的方向压缩数组.为了解决这个问题,Bhaskaracharya 等人[72]将数组压缩问题转换成寻找数据空间上的划分平面问题,利用类似调度变换的方法来寻找任意方向上数据压缩的可能性.他们的方法在处理数组内的压缩问题时优于 Lefebvre 等人的方法,但无法处理数组之间的压缩.Bhaskaracharya 等人[73]又面向数组之间的压缩问题扩展了他们的方法:首先寻找满足目标问题的划分平面,使满足条件的划分平面个数最少,然后沿着每个划分平面寻找模运算的基.

从以上几种数组压缩方法中不难发现,数组压缩问题往往被转换成调度变换问题,或者利用与调度变换相似的思想求解数据空间的划分平面,因此数组压缩也是求解线性整数规划问题的过程.

5 其 他

随着多面体编译技术在提升程序并行性和优化数据局部性方面不断取得突破,多面体模型在越来越多的领域得到应用,多面体模型能够处理的问题也越来越广泛.除此之外,针对多面体模型本身的受限条件和目前多面体编译技术中还无法实现的优化,研究人员也投入了大量的精力,以期进一步扩大多面体模型的影响.

5.1 迭代编译

体系结构的不断发展,衍生出越来越多的循环变换技术,编译器开发人员面临的一个主要问题是选择哪些循环变换以及按照怎样的顺序实施循环变换才能获得最好的性能.除此之外,循环变换中的参数选择也是影响程序性能的一个重要因素,如循环分块的大小.这些循环变换的排列组合问题形成了迭代编译这一研究方向,多面体模型在迭代编译领域也发挥着重要作用.

Cohen 等人[7,74]基于 Open64 编译器实现了一个半自动的迭代编译框架,该工作可被看作是多面体模型和迭代编译结合的先驱.他们的工作将原有的迭代编译处理的循环变换范畴从幺模变换扩展到仿射变换.该工作在早期多面体模型的基础上,用迭代空间、调度、仿射函数和数据布局空间这 4 个特征描述程序,并分析不同循环变换可能改变的程序特征,利用这些特征表示循环变换及其组合.

Vasilache 等人[75]基于数据流分析计算出各种循环变换组合下潜在的依赖冲突,实现了一种对导致依赖冲突的循环变换自动修正的算法;与此同时,他们还对迭代空间分裂技术进行了改进.判定循环变换的组合优化带来的性能提升是一个非线性规划问题,他们的工作在一定程度上降低了该问题的复杂性.

与上述自顶向下的方法不同,Pouchet 等人[76,77]在多面体模型基础上提出了一种自底向上的迭代编译优化框架.所谓自顶向下是编译器本身提供了各种循环变换和优化的选项,通过迭代编译多遍测试选择其中合法并且能够最大限度提升性能的循环变换组合序列.而 Pouchet 等人提出的自底向上的方法则是先通过构建可能的合法循环变换组合序列,从这些组合序列构成的空间中寻找最优解.这种方法减少了搜索空间的范围,同时无需考虑如何消除或修正不合法变换带来的影响.

Pluto 算法的提出,使得调度变换算法在程序并行性和数据局部性得到了很大的改善.为了充分利用 Pluto 算法来减少迭代编译过程中的搜索空间,Pouchet 等人[78,79]结合 Pluto 算法对多面体模型下的迭代编译优化做出了进一步改进,在他们原有工作的基础上进一步缩减了搜索空间的范围;与此同时,在 Pluto 算法实现循环分块的基础上,提出了不同的循环合并策略.

除了上述以多面体模型为核心开展研究的迭代编译工作之外,Park 等人[80,81]还将基于多面体模型的迭代编译与机器学习问题相结合,以进一步优化迭代编译面临的问题.

5.2 优化与扩展

多面体模型长期受限于静态仿射约束的限制,与此同时,作为通过静态编译阶段的信息来实现程序优化的工具,多面体编译自然也无法避免静态分析面临的缺陷.为了克服多面体编译技术的缺陷,研究人员在针对多面体模型的优化和扩展方面投入了大量的精力.

Benabderrahmane 等人[82]实现了一种面向 while 循环的多面体模型处理方法,借助条件谓词将控制流转换为数据依赖,并在代码生成阶段插入中断指令强制循环提前结束,从而消除引入的冗余循环迭代次数.他们的方法将多面体模型从受限于静态仿射约束的局面扩展到动态控制流下的自动应用.

最近几年,Strout 等人[83,84]和 Vinkat 等人[85−87]的研究团队致力于稀疏矩阵向量乘运算的多面体编译优化技术研究,扩展了多面体模型在处理非线性仿射数组下标领域的应用.他们的方法是基于 Chill 编译框架[88]实现一种稀疏矩阵向量乘运算的 inspector/executor 并行模式,能够处理稀疏矩阵的多种压缩存储形式,优化效果能够达到手工编写库函数的水平.

Chill 编译框架的实现为多面体编译技术和人工交互的实现提供了基础,Hall 等人[89]基于 Chill 框架实现了一种允许用户指导实现的自动调优系统,用户可以通过交互界面指定和选择相应的循环变换,也可以对循环变换的参数进行配置,然后交由多面体编译工具自动生成代码.类似的工作还有 Yuki 等人[90]提出的 AlphaZ 系统、Namjoshi 等人[91]提出的 Loopy 以及 Bastoul[92]和 Bagnères 等人[93]的研究,这些工作也有助于用户理解多面体模型中实现的循环变换,从而在帮助多面体编译工具实现程序变换决策时发挥作用.

数据局部性的重要性在第 4 节中已经阐述,在多核架构下,数据布局优化和局部性的开发在提升程序性能方面的作用也越来越重要.在这方面,Baskaran 等人[94]、Lu 等人[95]、Fauzia 等人[96]以及 Ramashekar 和 Bondhugula[97]实现了数据传输和布局等方面的优化,他们的工作都是基于现有的多面体编译工具来实现,随着 PPCG 编译器的问世,这些工作中实现的技术也都陆续集成到了该编译器当中.

Pluto 算法提出之后,多面体编译工具实现循环变换的能力得到了大幅提升,但循环合并一直未能得到很好的解决.Bondhugula 等人[8]和 Mehta 等人[98]先后提出了几种不同的循环合并策略,这些循环合并策略可以通过编译阶段的参数来选择和控制.Mehta 和 Yew[99]还针对多面体模型中时间复杂度的问题进行了研究,通过将多个语句抽象成一个宏语句来减少依赖边的个数,从而达到简化线性整数规划问题求解的目的.

在扩展应用领域方面,Zou 等人[100]利用多面体模型来实现累积加操作的自动并行化,这种操作可以看作是归约操作更一般化的体现;Pradelle 等人[101]则实现了一种二进制代码的多面体编译优化技术,首先将二进制代码恢复成 C 程序,为多面体模型提供必要的信息,然后将经过多面体模型优化的 C 程序再变换成二进制代码.Shirako 等人[102]试图将多面体模型编译技术与传统的在抽象语法树上直接实现变换的编译技术相结合,不过,他们的方法只针对有限的循环变换(主要是循环合并和循环交换)进行了讨论.Moll 等人[103]利用多面体模型表示 OpenCL 程序,对输入的 OpenCL 程序的控制流分支和访存连续问题进行了优化.值得一提的是,来自中国科学院计算技术研究所的研究团队[104]利用多面体模型实现了一种 GPU 架构上的存储优化,该工作也是近些年来

国内研究人员利用多面体模型实现优化的一个代表性工作.

在处理非连续迭代空间时,传统的多面体模型往往会采取向上近似的手段来简化过程.所谓非连续迭代空间是指循环嵌套的多面体表示中存在不被该循环嵌套执行的点,这些点被称为“洞”.为了简化分析过程,过去的做法是假设这些“洞”也被循环嵌套执行.为了克服这个缺点,Gupta 等人[105]提出了 z-polyhedal 模型,该模型后来由 Seghir 等人[106]和 Cilardo 等人[107]进行了优化和扩展应用.这些工作和 isl 库中对一阶谓词逻辑 Presburger 运算的支持和存在量词的使用功能相似.目前,isl 库在多面体模型领域的应用比前者更广泛.

5.3 面向不同体系结构的后端

由于多面体模型研究人员的不断努力,多面体编译工具在循环优化方面的功能不断得到完善,编译器开发和许多其他领域的研究人员也陆续将多面体模型的应用扩展到各种硬件结构上.近年来,异构架构成为多面体模型扩展的最重要的一个方向,研究人员面向异构架构实现了各种多面体编译工具的开发,这些工作包括文献[49,108−110],其中,Konstantinidis 等人的工作[110]是首次将多面体模型扩展到异构架构,Leung 等人[109]在商用多面体编译工具 R-Stream 上实现了面向异构架构的代码生成,然而,这些工具被后来的 PPCG 超越.当前,面向异构架构的工作大多基于 PPCG 实现,例如,Juega 等人[111]在 PPCG 上实现了一种带有运行时参数分块大小的方法;Shirako 等人[112]根据 CUDA 代码线程块和线程两级并行结构的特点,针对 PPCG 中的调度变换算法进行了优化.

Grosser 等人[113]基于 Polly 实现了一种从中间语言到中间语言的代码翻译工具 Polly-ACC,并最终生成 OpenCL 或 CUDA 代码.Polly-ACC 和 PPCG 在本质上都是通过调度树来实现调度变换和代码生成,但 PPCG 中实现了对 Pencil 语言[114]的支持,能够采用“编译制导+多面体编译优化”的模式对程序进行分析和优化,使其能够处理更复杂的应用程序.

面向分布存储的自动并行化一直都是自动并行化领域的难点,Bondhugula 等人针对分布存储结构进行了一些初步的研究[115−118].与传统方法相比,面向分布存储结构的多面体编译工具首先在依赖关系分析能力上有了进一步的提升,最重要的一点是,Bondhugula 等人的工作利用 Pluto 算法实现了循环分块,并基于此实现了一种数据空间的分块,这种数据空间的分块减少了冗余通信数据量,在性能上有了一定的提升.不过,他们的工作需要运行时技术的支撑.

向量化一直以来都不是多面体模型研究人员关注的主要问题,但随着多面体编译技术的不断成熟,研究人员开始考虑如何结合多面体模型和向量化.Trifunović 等人[119]首先实现了多面体模型和向量化的结合,通过一种代价模型来平衡循环变换和向量化挖掘,其中最重要的变换是循环展开.Kong 等人[120]在 Pluto 算法基础上实现了循环分块后分块内的向量化技术,并进一步优化了分块内的数据局部性.

可重构架构是多面体模型应用的一个重要领域,来自北京大学和清华大学的研究团队在这个方向上利用多面体模型取得了一些成果.Bondhugula 等人[121]实现了一种面向 FPGA 的代码生成算法,但该算法只能处理完美循环嵌套,而且对循环内的依赖关系要求比较苛刻.Natale 等人[122]等人利用多面体模型实现了 stencil 计算的 FPGA 代码自动生成,来自清华大学微电子所的研究团队[123]面向粗粒度可重构架构实现了一种基于多面体模型的循环优化,他们的工作并没有对多面体模型本身进行优化.

考虑到多面体模型对循环优化的处理能力,北京大学丛京生教授带领的团队和 Pouchet 等人利用多面体模型在高层次综合领域取得了许多研究进展[124−127].与前面的两个工作相似,这些工作大部分都没有对多面体模型本身进行优化,而是针对他们的目标架构为多面体模型添加了相应的代码生成后端,其中,Zuo 等人的工作[125]对代码生成进行了优化.考虑到多面体模型在代码生成时循环边界上经常出现 min/max 操作,而这些操作会影响代码在 FPGA 上的性能,因此,他们针对该问题对多面体模型的代码生成模块进行了优化.此外,Meeus 和 Stroobandt[128]也在多面体模型和高层次综合的结合方面做出了贡献.

5.4 领域特定语言

领域特定语言的优势在于允许用户通过编程语言来表达一些编译阶段很难甚至是无法获取的信息,这些

信息为编译优化提供了重要依据.近些年来,多面体编译技术研究人员通过结合多面体模型和领域特定语言的优势,进一步拓展了领域特定语言的影响.

Bhaskaracharya等人[129]针对图形化编程环境LabVIEW设计并实现了一个多面体编译框架PolyGLoT,他们的贡献在于将多面体模型从传统的文本编程为核心扩展到图形化编程领域特定语言上,是多面体模型和领域特定语言结合的先驱工作.Henretty 等人[130]面向 stencil 计算领域特定语言实现了一种代码自动生成工具,通过对连续数组的布局进行转置优化实现了数据连续访存,并在他们的工具中实现了分裂分块.

Sbîrlea 等人[131]针对一种数据流图语言实现了一种多面体编译优化策略,这种语言的设计使数据之间的依赖关系更加明显,因此无需调用多面体模型中的依赖关系分析模块,简化了分析过程.Mullapudi 等人[132]针对面向图像处理设计的领域特定语言 Halide[133]设计并实现了一种多面体编译工具 PolyMage,用于自动生成 Halide 语言中的调度,将由 Halide 语言编写的程序转换成 C 程序,并在生成的代码中实现了交叉分块.Chugh 等人[134]实现了 PolyMage 面向 FPGA 上的代码生成.

Baghdadi 等人设计并实现了一种编译制导语言 Pencil[114],不仅可以作为领域特定语言编译器的目标语言生成,为后续多面体编译工具优化提供领域相关信息,而且还可以像 OpenMP 一样作为编译指示由用户添加到通用编程语言中,帮助多面体编译工具识别并行区域并进行相关优化.这种“编译制导+多面体编译优化”的方法为解决许多传统多面体模型无法处理的问题提供了基础,例如,Reddy 等人[135]通过扩展 Pencil 语言的编译制导指令,在多面体模型上实现了归约操作的优化,其特点是能够处理由用户定义的归约操作,而这种归约操作往往无法由现有的手工库函数来进行优化.

Spampinato等人面向基础线性代数运算设计了一种领域特定语言,并实现了针对该语言的编译器LGen用来自动生成对应的C程序[136].后来,他们将LGen编译器与多面体模型相结合,首先利用isl的集合和映射来优化LGen的中间表示,然后再借助多面体模型代码生成工具来优化生成的代码[137].

5.5 运行时技术

Simburger 等人[138]通过实验对多面体模型和多面体编译技术进行了一个全面的比较和分析,并对多面体模型潜在的研究方向进行了讨论,指出,动态运行时技术是多面体模型的一个发展方向.

事实上,在 Pluto 算法提出之后,研究人员已经意识到:即便 Pluto 算法能够自动实现循环分块用以充分利用程序的数据局部性,但循环分块之后分块之间仍只能通过流水并行甚至只能串行执行.与此同时,在处理非完美循环嵌套以及多个语句的情况时,Pluto 算法给出的分块策略往往存在负载不均衡的问题.为了解决这个问题,Baskaran 等人[139]利用一种动静结合的方式来实现循环分块之间的动态调度,其思想是,先利用 Pluto 算法实现循环分块,然后在编译阶段计算循环分块之间依赖关系的有向无环图,将相关信息嵌入到生成的代码中;在运行时阶段,通过解析该有向无环图中循环分块之间的关系,并最终实现循环分块的动态调度.该技术后来由 Dathathri 等人[140]进一步扩展,实现面向分布存储结构的动态优化.

Kong 等人[141]也针对循环分块之间的并行性进行了动态运行时的优化,与 Baskaran 等人和 Dathathri 等人的工作不同之处在于,他们是利用 OpenStream 语言[142]的同步子句对生成代码中的同步操作进行优化,而且 Kong 等人的工作是基于 PPCG 实现的.从解决问题的目标角度来看,Kong 等人的工作更接近于第 4.1 节中面向各种复杂循环分块形状的研究所要解决的问题,即他们要解决的问题是循环分块之间的并行启动问题.

Bao 等人[143]通过与动态运行时技术结合,实现了一种自动验证多面体模型实现的循环变换合法性的方法.他们的方法也是先通过静态编译阶段在生成代码中嵌入用于验证依赖关系的辅助代码,再通过运行时检测这些辅助代码中的依赖关系信息是否得到满足来确定生成的代码,即多面体模型实现的循环变换是否正确.类似地,Sampaio 等人[144]针对静态编译的保守性分析问题,提出了一种通过判定依赖关系是否得到满足的谓词条件实现多版本代码生成的动态分析技术.当某种循环变换所需的依赖关系得到满足时,运行时执行实现了该循环变换的代码版本,否则执行未实现该循环变换的代码版本.虽然这种技术可能导致代码膨胀问题,但是有效克服了静态分析过于保守的弊端.

Doerfert 等人[145]则实现了一种“编译制导+多面体编译优化”的技术.他们的方法是通过在源程序添加必要

的编译指示来帮助多面体编译工具分析程序.他们的方法实际上和“Pencil+PPCG”的方法类似,不同之处在于,他们的工作是在中间语言上实现的.

投机并行是自动并行化领域的另一个重要研究方向,这种方法对动态运行时技术的依赖性很高.为了弥补多面体编译技术静态分析面临的不足,研究人员也试图将多面体编译技术和投机并行技术结合.Jimborean 等人[146]的工作和Sukumaran-Rajam等人[147]提出的Apollo系统是这方面工作的代表,他们的共同点在于都是先利用投机并行技术来假设程序能够并行执行,然后在运行时阶段利用多面体模型动态实施循环变换.

6 面临的挑战

前文对多面体模型和多面体编译技术当前的研究现状进行了系统深入的总结和分析,从这些研究成果中不难发现,基于多面体模型的编译技术研究在程序自动并行化领域取得了许多突破,这些研究成果为并行编程领域的发展起到了至关重要的作用.然而也正如前文中所介绍的,多面体模型有其固有的弊端和不足,距离研究人员对程序自动并行化的需求目标还有一定的差距.通过对多面体编译技术研究工作的研究和总结,我们认为,多面体编译技术的研究工作目前还面临以下几个方面的挑战.

首先,多面体模型能够自动实现各种循环变换,但多面体模型本身受限于仿射约束,对输入程序具有一定的要求.如何扩展多面体模型在非仿射约束下的应用,是当前多面体编译技术研究人员正在解决的问题.以处理非线性数组下标问题为例,本文在第5.2节中介绍了相关的研究工作,同时,第5.4节中领域特定语言,尤其是Pencil语言和PolyMage系统的提出,对“编译制导+多面体编译优化”模式和“领域特定语言+多面体编译优化”模式的可行性提供了重要依据.然而这些工作仍存在不少缺点,例如,Venkat 等人针对稀疏矩阵运算的多面体编译技术还需要借助与用户的交互来实现,而 Pencil 语言目前还处于雏形阶段,针对该语言进行扩展或另外设计新的编译制导语言来帮助多面体解决更一般性的问题是扩展多面体模型应用领域的一种有效途径,也将是当前多面体编译技术研究的一个重要方向.

其次,正如 Simburger 等人[138]通过实践表明的结果,实现静态多面体编译优化与动态运行时技术的结合将成为多面体模型研究领域的另一个重要研究方向.虽然当前研究人员已经开始着手研究了多面体模型与动态运行时技术的结合,但与其他传统的多面体编译技术相比,这些研究工作的发展还处于十分初级的阶段.另外,多面体模型的代码生成模块本身存在代码膨胀问题,这些动态运行时技术的应用还要借助额外的辅助代码来实现目的,这些方法对代码生成质量的影响还不清楚,对程序性能的影响也没有相关的讨论.

第三,多面体编译技术在其他领域的影响逐渐扩大,随着人工智能研究领域掀起的新一代变革,多面体编译技术也面临着如何与人工智能领域的技术相结合并发挥作用的问题.深度学习、机器学习等方向需要解决的并行化问题都可以借助多面体编译技术的优化来实现,Google 公司发布的 TensorFlow 系统就是基于数据流图实现的数值计算库,而第5.4节中提到的面向图形化编程环境和数据流图语言的多面体编译工具为人工智能和多面体模型的结合提供了重要依据.此外,2016 年 Google 公司也对外宣布了专门用于张量运算的加速运算部件TPU(tensor processing unit),多面体模型已经在面向 GPU 的代码自动生成领域取得了巨大的进展,这为面向TPU的代码自动生成也奠定了一定的基础.

最后,本文在介绍多面体编译技术面向程序并行性和数据局部性的研究内容时说明了这些研究内容在实质上都是求解线性整数规划问题的过程,而线性整数规划问题是一个 NP 难问题.也就是说,多面体编译技术无法避免时间复杂度较高的问题.当前,几乎所有多面体编译技术的研究工作都没有办法避免这个问题,虽然这个问题在现在可能不会影响多面体模型的应用,但随着多面体编译技术的发展,其面临的输入程序将变得越来越复杂,编译时间将会成为一个令研究人员头疼的问题.Mehta 等人[99]对该问题进行了一个十分初步的研究,但要实现降低多面体模型的编译时间复杂度还远远不够.

另外,多面体编译技术在国内的发展还十分落后,虽然国内也有一些研究团队在该领域中取得了一些研究成果,但这些工作基本上都是借助多面体模型实现一些其他优化,并未对多面体模型本身进行深入系统的研究.从这个角度来讲,在国内开展多面体编译技术的研究也是一个值得关注的问题.

7 总 结

多面体模型是程序自动并行化领域的一个研究热点,本文对多面体模型和基于该模型的编译技术研究进展进行了系统深入的总结,介绍了多面体模型的工作原理,基于多面体模型的编译工具一般编译流程,然后对该领域面向程序并行性和数据局部性方面的研究内容进行了列举,并梳理了多面体模型在其他领域的应用和发展.最后,作者分析了多面体模型及其相应编译技术当前面临的挑战.考虑到多面体模型在国内的研究还处于十分初级的阶段,本文对该领域的研究进展进行了总结,目的是为国内从事编译研究工作的人员提供一个参考,并期望借助此文推动国内编译研究团队在该领域中取得新的进展.

致谢 本文部分工作在法国 INRIA PARKAS 实验室完成,感谢 PARKAS 实验室 Albert Cohen 教授对本文工作的指导;感谢 KU Leuven 大学的 Sven Verdoolaege 研究员、ETHz 的 Tobias Grosser 博士、MIT 的 Riyadh Baghdadi 博士、ANL 实验室的 Michael Kruse 博士、PARKAS 实验室的 Chandan Reddy 博士生的讨论,帮助我们加深对多面体模型的理解;感谢 Indian Institute of Science 的 Uday Bondhugula 教授、Intel 公司的 Anand Venkat 博士,在我们使用他们的工具时所给予的解答和帮助.

References:

[1] Feautrier P, Lengauer C. Polyhedron model. In: Proc. of the Encyclopedia of Parallel Computing. 2011. 1581−1592.

[2] Wolf ME, Lam MS. A loop transformation theory and an algorithm to maximize parallelism. IEEE Trans. on Parallel and Distributed Systems (TPDS), 1991,2(4):452−471.

[3] Bondhugula U, Hartono A, Ramanujam J, Sadayappan P. A practical automatic polyhedral parallelizer and locality optimizer. In: Proc. of the 29th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2008. 101−113.

[4] Verdoolaege S, Carlos Juega J, Cohen A, Gómez JI, Tenllado C, Catthoor F. Polyhedral parallel code generation for CUDA. ACM Trans. on Architecture and Code Optimization (TACO), 2013,9(4):54:1−54:24.

[5] Trifunović K, Cohen A, Edelsohn D, Li F, Grosser T, Jagasia H, Ladelsky R, PoP S, Sjödin J, Upadrasta R. Graphite two years after: First lessons learned from real-world polyhedral compilation. In: Proc. of the 2nd GCC Research Opportunities Workshop (GROW). 2010.

[6] Grosser T, Groesslinger A, Lengauer C. Polly—Performing polyhedral optimizations on a low-level intermediate representation. Parallel Processing Letters, 2012,22(4):No.1250010.

[7] Cohen A, Sigler M, Girbal S, Temam, O, Parello, D, Vasilache N. Facilitating the search for compositions of program transformations. In: Proc. of the 19th Annual Int'l Conf. on Supercomputing (ICS). ACM Press, 2005. 151−160.

[8] Bondhugula U, Gunluk O, Dash S, Renganarayanan L. A model for fusion and code motion in an automatic parallelizing compiler. In: Proc. of the 19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). ACM Press, 2010. 343−352.

[9] Verdoolaege S, Grosser T. Polyhedral extraction tool. In: Proc. of the 2nd Int'l Workshop on Polyhedral Compilation Techniques (IMPACT). 2012.

[10] Clan—A polyhedral representation extractor for high level programs. 2017. http://icps.u-strasbg.fr/~bastoul/development/clan/docs/clan.html

[11] Chen LQ, Wang J, Liu WW. Weak join for the constraint-based polyehdra abstract domain. Ruan Jian Xue Bao/Journal of Software, 2010,21(11):2711−2724 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3664.htm [doi: 10.3724/SP.J.1001.2010.03664]

[12] Verdoolaege S. isl: An integer set library for the polyhedral model. In: Proc. of the ICMS 2010. LNCS 6327, Berlin, Heidelberg: Springer-Verlag, 2010. 299−302.

[13] Omega project source release. Version 1.00, 2017. http://www.cs.umd.edu/projects/omega/release-1.0.html

[14] The parametric integer programming's home. 2017. http://www.piplib.org/

[15] PolyLib—A library of polyhedral functions. 2017. https://icps.u-strasbg.fr/polylib/

[16] The parma polyhedra library. 2017. http://bugseng.com/products/ppl/

[17] Allen R, Kennedy K. Optimizing compilers for modern architectures a dependence-based approach. 2001.

[18] Feautrier P. Dataflow analysis of scalar and array references. Int'l Journal of Parallel Programming (IJPP), 1991,20(1):23−53.

[19] Pugh W. The Omega test: A fast and practical integer programming algorithm for dependence analysis. In: Proc. of the 1991 ACM/ IEEE Conf. on Supercomputing (ICS). ACM Press, 1991. 4−13.

[20] Pugh W, Wonnacott D. Eliminating false data dependences using the Omega test. In: Proc. of the ACM SIGPLAN 1992 Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 1992. 140−151.

[21] Maydan DE, Amarasinghe SP, Lam MS. Array-Data flow analysis and its use in array privatization. In: Proc. of the 20th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). ACM Press, 1993. 2−15.

[22] Maslov V. Lazy array data-flow dependence analysis. In: Proc. of the 21st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). ACM Press, 1994. 311−325.

[23] Vasilache V, Bastoul C, Cohen A, Girbal S. Violated dependence analysis. In: Proc. of the 20th Annual Int'l Conf. on Supercomputing (ICS). ACM Press, 2006. 335−344.

[24] Trifunović K, Cohen A, Razya L, Li F. Elimination of memory-based dependences for loop-nest optimization and parallelization: Evaluation of a revised violated dependence analysis method on a three-address code polyhedral compiler. In: Proc. of the 3rd GCC Research Opportunities Workshop (GROW). 2011.

[25] Pellegrini S, Hoefler T, Fahringer T. Exact dependence analysis for increased communication overlap. In: Proc. of the 19th European Conf. on Recent Advances in the Message Passing Interface (EuroMPI). Springer-Verlag, 2012. 89−99.

[26] Yuki T, Feautrier P, Rajopadhye S, Saraswat V. Array dataflow analysis for polyhedral X10 programs. In: Proc. of the 18th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). ACM Press, 2013. 23−34.

[27] Baghdadi R, Cohen A, Verdoolaege S, Trifunović K. Improved loop tiling based on the removal of spurious false dependences. ACM Trans. on Architecture and Code Optimization (TACO), 2013,9(4):52:1−52:26.

[28] Verdoolaege S, Cohen A. Live-Range reordering. In: Proc. of the 6th Int'l Workshop on Polyhedral Compilation Techniques (IMPACT). 2016.

[29] Feautrier P. Some efficient solutions to the affine scheduling problem. Part I: One-Dimensional time. Int'l Journal of Parallel Programming (IJPP), 1992,21(5):313−347.

[30] Feautrier P. Some efficient solutions to the affine scheduling problem. Part II: Multidimensional time. Int'l Journal of Parallel Programming (IJPP), 1992,21(6):389−420.

[31] Griebl M, Feautrier P, Größlinger A. Forward communication only placements and their use for parallel program construction. In: Proc. of the Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC). Springer-Verlag, 2002. 16−30.

[32] Lim AW, Lam MS. Maximizing parallelism and minimizing synchronization with affine transforms. In: Proc. of the 24th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). ACM Press, 1997. 201−214.

[33] Lim AW, Cheong GI, Lam MS. An affine partitioning algorithm to maximize parallelism and minimize communication. In: Proc. of the 13th Int'l Conf. on Supercomputing (ICS). ACM Press, 1999. 228−237.

[34] Bondhugula U, Baskaran M, Krishnamoorthy S, Ramanujam J, Rountev A, Sadayappan P. Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model. In: Proc. of the Int'l Conf. on Compiler Construction (CC). Springer-Verlag, 2008. 132−146.

[35] Acharya A, Bondhugula U. PLUTO+: Near-Complete modeling of affine transformations for parallelism and locality. In: Proc. of the 20th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). ACM Press, 2015. 54−64.

[36] Bondhugula U, Acharya A, Cohen A. The Pluto+ algorithm: A practical approach for parallelization and locality optimization of affine loop nests. ACM Trans. on Programming Languages and Systems (TOPLAS), 2016,38(3):12:1−12:32.

[37] Upadrasta R, Cohen A. Sub-Polyhedral scheduling using (unit-) two-variable-per-inequality polyhedra. In: Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). ACM Press, 2013. 483−496.

[38] Ancourt C, Irigoin F. Scanning polyhedra with DO loops. In: Proc. of the 3rd ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). ACM Press, 1991. 39−50.

[39] Kelly W, Pugh W, Rosser E. Code generation for multiple mappings. In: Proc. of the 5th Symp. on Frontiers of Massively Parallel Computation (Frontiers). IEEE Computer Society, 1995. 332−341.

[40] Chen C. Polyhedra scanning revisited. In: Proc. of the 33rd ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2012. 499−508.

[41] Quilleré F, Rajopadhye S, Wilde D. Generation of efficient nested loops from polyhedra. Int'l Journal of Parallel Programming (IJPP), 2000,28(5):469−498.

[42] Bastoul C. Code generation in the polyhedral model is easier than you think. In: Proc. of the 13th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). IEEE Computer Society, 2004. 7−16.

[43] Vasilache N, Bastoul C, Cohen A. Polyhedral code generation in the real world. In: Proc. of the Int'l Conf. on Compiler Construction (CC). Springer-Verlag, 2006. 185−201.

[44] Razanajato H, Loechner V, Bastoul C. Splitting polyhedra to generate more efficient code. In: Proc. of the 7th Int'l Workshop on Polyhedral Compilation Techniques (IMPACT). 2017.

[45] Grosser T, Verdoolaege S, Cohen A. Polyhedral AST generation is more than scanning polyhedra. ACM Trans. on Programming Languages and Systems (TOPLAS), 2015,37(4):12:1−12:50.

[46] Lim AW, Liao SW, Lam MS. Blocking and array contraction across arbitrarily nested loops using affine partitioning. In: Proc. of the 8th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming (PPoPP). ACM Press, 2001. 103−112.

[47] Zhang JL, Di P, Jiang CF, Zhang W, Xu XH, Wan J, Ren YJ. A parallel finite difference stencil algorithm based on iterative space alternate tiling. Ruan Jian Xue Bao/Journal of Software, 2010,21:270−283 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/10028.htm

[48] Krishnamoorthy S, Baskaran M, Bondhugula U, Ramanujam J, Rountev A, Sadayappan P. Effective automatic parallelization of stencil computations. In: Proc. of the 28th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2007. 235−244.

[49] Holewinski J, Pouchet LN, Sadayappan P. High-Performance code generation for stencil computations on GPU architectures. In: Proc. of the 26th ACM Int'l Conf. on Supercomputing (ICS). ACM Press, 2012. 311−320.

[50] Grosser T, Cohen A, Kelly PHJ, Ramanujam J, Sadayappan P, Verdoolaege S. Split tiling for GPUs: Automatic parallelization using trapezoidal tiles. In: Proc. of the 6th Workshop on General Purpose Processor Using Graphics Processing Units (GPGPU). ACM Press, 2013. 24−31.

[51] Bandishti V, Pananilath I, Bondhugula U. Tiling stencil computations to maximize parallelism. In: Proc. of the 2012 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society, 2012. 40:1−40:11.

[52] Pananilath I, Acharya A, Vasista V, Bondhugula U. An optimizing code generator for a class of Lattice-Boltzmann computations. ACM Trans. on Architecture and Code Optimization (TACO), 2015,12(2):14:1−14:23.

[53] Ghysels P, Vanroose W. Modeling the performance of geometric multigrid stencils on multicore computer architectures. SIAM Journal on Scientific Computing, 2015,37(2):C194−C216.

[54] Bondhugula U, Bandishti V, Cohen A, Potron G, Vasilache N. Tiling and optimizing time-iterated computations over periodic domains. In: Proc. of the 2014 23rd Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT). IEEE Computer Society, 2014. 39−50.

[55] Shrestha S, Gao GR, Manzano J, Marquez A, Feo J. Locality aware concurrent start for stencil applications. In: Proc. of the 13th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). IEEE Computer Society, 2015. 157−166.

[56] Grosser T, Cohen A, Holewinski J, Sadayappan P, Verdoolaege S. Hybrid hexagonal/classical tiling for GPUs. In: Proc. of the Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). ACM Press, 2014. 66−75.

[57] Grosser T, Verdoolaege S, Cohen A, Sadayappan P. The relation between diamond tiling and hexagonal tiling. Parallel Processing Letters, 2014,24(3):No.1441002.

[58] Bondhugula U, Bandishti V, Pananilath I. Diamond tiling: Tiling techniques to maximize parallelism for stencil computations. IEEE Trans. on Parallel and Distributed Systems (TPDS), 2017,28(5):1285−1298.

[59] Renganarayanan L, Kim DG, Rajopadhye S, Strout MM. Parameterized tiled loops for free. In: Proc. of the 28th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2007. 405−414.

[60] Renganarayanan L, Kim D, Strout MM, Rajopadhye S. Parameterized loop tiling. ACM Trans. on Programming Languages and Systems (TOPLAS), 2012,34(1):3:1−3:41.

[61] Kim DG, Renganarayanan L, Rostron D, Rajopadhye S, Strout MM. Multi-Level tiling: *M* for the price of one. In: Proc. of the 2007 ACM/IEEE Conf. on Supercomputing (SC). ACM Press, 2007. 51:1−51:12.

[62] Hartono A, Baskaran MM, Bastoul C, Cohen A, Krishnamoorthy S, Norris B, Ramanujam J, Sadayappan P. Parametric multi-level tiling of imperfectly nested loops. In: Proc. of the 23rd Int'l Conf. on Supercomputing (ICS). ACM Press, 2009. 147−157.

[63] Hartono A, Baskaran MM, Ramanujam J, Sadayappan P. DynTile: Parametric tiled loop generation for parallel execution on multicore processors. In: Proc. of the 2010 IEEE Int'l Symp. on Parallel & Distributed Processing (IPDPS). IEEE Computer Society, 2010. 1−12.

[64] Mehta S, Beeraka G, Yew PC. Tile size selection revisited. ACM Trans. on Architecture and Code Optimization (TACO), 2013, 10(4):35:1−35:27.

[65] Darte A, Schreiber R, Villard G. Lattice-Based memory allocation. IEEE Trans. on Computers (TC), 2005,54(10):1242−1257.

[66] Strout MM, Carter L, Ferrante J, Simon B. Schedule-Independent storage mapping for loops. In: Proc. of the 8th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM Press, 1998. 24−33.

[67] Thies W, Vivien F, Sheldon J, Amarasinghe S. A unified framework for schedule and storage optimization. In: Proc. of the ACM SIGPLAN 2001 Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2001. 232−242.

[68] Thies W, Vivien F, Amarasinghe S. A step towards unifying schedule and storage optimization. ACM Trans. on Programming Languages and Systems (TOPLAS), 2007,29(6):34:1−34:45.

[69] Alias C, Baray F, Darte A. Bee+Cl@k: An implementation of lattice-based array contraction in the source-to-source translator rose. In: Proc. of the 2007 ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES). ACM Press, 2007. 73−82.

[70] Darte A, Isoard A, Yuki T. Extended lattice-based memory allocation. In: Proc. of the 25th Int'l Conf. on Compiler Construction (CC). ACM Press, 2016. 218−228.

[71] Lefebvre V, Feautrier P. Automatic storage management for parallel programs. Parallel Computing, 1998,24(3):649−671.

[72] Bhaskaracharya SG, Bondhugula U, Cohen A. Automatic storage optimization for arrays. ACM Trans. on Programming Languages and Systems (TOPLAS), 2016,38(3):11:1−11:23.

[73] Bhaskaracharya SG, Bondhugula U, Cohen A. SMO: An integrated approach to intra-array and inter-array storage optimization. In: Proc. of the 43rd Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). ACM Press, 2016. 526−538.

[74] Girbal S, Vasilache N, Bastoul C, Cohen A, Parello D, Sigler M, Temam O. Semi-Automatic composition of loop transformations for deep parallelism and memory hierarchies. Int'l Journal of Parallel Programming (IJPP), 2006,34(3):261−317.

[75] Vasilache N, Cohen A, Pouchet LN. Automatic correction of loop transformations. In: Proc. of the 16th Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT). IEEE Computer Society, 2007. 292−304.

[76] Pouchet LN, Bastoul C, Cohen A, Vasilache N. Iterative optimization in the polyhedral model: Part I, one-dimensional time. In: Proc. of the Int'l Symp. on Code Generation and Optimization (CGO). IEEE Computer Society, 2007. 144−156.

[77] Pouchet LN, Bastoul C, Cohen A, Cavazos J. Iterative optimization in the polyhedral model: Part II, multidimensional time. In: Proc. of the 29th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2008. 90−100.

[78] Pouchet LN, Bondhugula U, Bastoul C, Cohen A, Ramanujam J, Sadayappan P. Combined iterative and model-driven optimization in an automatic parallelization framework. In: Proc. of the 2010 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society, 2010. 1−11.

[79] Pouchet LN, Bondhugula U, Bastoul C, Cohen A, Ramanujam J, Sadayappan P, Vasilache N. Loop transformations: Convexity, pruning and optimization. In: Proc. of the 38th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). ACM Press, 2011. 549−562.

[80] Park E, Cavazos J, Alvarez MA. Using graph-based program characterization for predictive modeling. In: Proc. of the 10th Int'l Symp. on Code Generation and Optimization (CGO). ACM Press, 2012. 196−206.

[81] Park E, Cavazos J, Pouchet LN, Bastoul C, Cohen A, Sadayappan P. Predictive modeling in a polyhedral optimization space. Int'l Journal of Parallel Programming (IJPP), 2013,41(5):704−750.

[82] Benabderrahmane MW, Pouchet LN, Cohen A, Bastoul C. The polyhedral model is more widely applicable than you think. In: Proc. of the Int'l Conf. on Compiler Construction (CC). Springer-Verlag, 2010. 283−303.

[83] Strout MM, Georg G, Olschanowsky C. Set and relation manipulation for the sparse polyhedral framework. In: Proc. of the Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC). Springer-Verlag, 2012. 61−75.

[84] Strout MM, LaMielle A, Carter L, Ferrante J, Kreaseck B, Olschanowsky C. An approach for code generation in the sparse polyhedral framework. Parallel Computing, 2016,53:32−57.

[85] Venkat A, Shantharam M, Hall M, Strout MM. Non-Affine extensions to polyhedral code generation. In: Proc. of the Annual IEEE/ ACM Int'l Symp. on Code Generation and Optimization (CGO). ACM Press, 2014. 185−194.

[86] Venkat A, Hall M, Strout M. Loop and data transformations for sparse matrix code. In: Proc. of the 36th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2015. 521−532.

[87] Venkat A, Mohammadi MS, Park J, Rong H, Barik R, Strout MM, Hall M. Automating wavefront parallelization for sparse matrix computations. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society, 2016. 41:1−41:12.

[88] Chen C, Chame J, Hall M. CHiLL: A framework for composing high-level loop transformations. Technical Report, 08-897, University of Southern California, 2008.

[89] Hall M, Chame J, Chen C, Shin J, Rudy G, Khan MM. Loop transformation recipes for code generation and auto-tuning. In: Proc. of the Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC). Springer-Verlag, 2009. 50−64.

[90] Yuki T, Gupta G, Kim DG, Pathan, T, Rajopadhye S. AlphaZ: A system for design space exploration in the polyhedral model. In: Proc. of the Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC). Springer-Verlag, 2012. 17−31.

[91] Namjoshi KS, Singhania N. Loopy: Programmable and formally verified loop transformations. In: Proc. of the Int'l Static Analysis Symp. (SAS). Springer-Verlag, 2016. 383−402.

[92] Bastoul C. Mapping deviation: A technique to adapt or to guard loop transformation intuitions for legality. In: Proc. of the 25th Int'l Conf. on Compiler Construction (CC). ACM Press, 2016. 229−239.

[93] Bagnères L, Zinenko O, Huot S, Bastoul C. Opening polyhedral compiler's black box. In: Proc. of the 2016 Int'l Symp. on Code Generation and Optimization (CGO). ACM Press, 2016. 128−138.

[94] Baskaran MM, Bondhugula U, Krishnamoorthy S, Ramanujam J, Rountev A, Sadayappan P. Automatic data movement and computation mapping for multi-level parallel architectures with explicitly managed memories. In: Proc. of the 13th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). ACM Press, 2008. 1−10.

[95] Lu Q, Alias C, Bondhugula U, Henretty T, Krishnamoorthy S, Ramanujam J, Rountev A, Sadayappan P, Chen YJ, Lin HB. Data layout transformation for enhancing data locality on nuca chip multiprocessors. In: Proc. of the 18th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). IEEE Computer Society, 2009. 348−357.

[96] Fauzia N, Pouchet LN, Sadayappan P. Characterizing and enhancing global memory data coalescing on GPUs. In: Proc. of the 13th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). IEEE Computer Society, 2015. 12−22.

[97] Ramashekar T, Bondhugula U. Automatic data allocation and buffer management for multi-GPU machines. ACM Trans. on Architecture and Code Optimization (TACO), 2013,10(4):60:1−60:26.

[98] Mehta S, Lin PH, Yew PC. Revisiting loop fusion in the polyhedral framework. In: Proc. of the 19th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). ACM Press, 2014. 233−246.

[99] Mehta S, Yew PC. Improving compiler scalability: Optimizing large programs at small price. In: Proc. of the 36th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2015. 143−152.

[100] Zou Y, Rajopadhye S. Scan detection and parallelization in inherently sequential nested loop programs. In: Proc. of the 10th Int'l Symp. on Code Generation and Optimization (CGO). ACM Press, 2012. 74−83.

[101] Pradelle B, Ketterlin A, Clauss P. Polyhedral parallelization of binary code. ACM Trans. on Architecture and Code Optimization (TACO), 2012,8(4):39:1−39:21.

[102] Shirako J, Pouchet LN, Sarkar V. Oil and water can mix: An integration of polyhedral and AST-based transformations. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society, 2014. 287−298.

[103] Moll S, Doerfert J, Hack S. Input space splitting for OpenCL. In: Proc. of the 25th Int'l Conf. on Compiler Construction (CC). ACM Press, 2016. 251−260.

[104] Li J, Liu L, Wu Y, Liu XH, Gao Y, Feng XB, Wu CY. Pragma directed shared memory centric optimizations on GPUs. Journal of Computer Science and Technology (JCST), 2016,31(2):235−252.

[105] Gupta G, Rajopadhye S. The Z-polyhedral model. In: Proc. of the 12th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). ACM Press, 2007. 237−248.

[106] Seghir R, Loechner V, Meister B. Integer affine transformations of parametric Z-polytopes and applications to loop nest optimization. ACM Trans. on Architecture and Code Optimization (TACO), 2012,9(2):8:1−8:27.

[107] Cilardo A, Gallo L. Improving multibank memory access parallelism with lattice-based partitioning. ACM Trans. on Architecture and Code Optimization (TACO), 2015,11(4):45:1−45:25.

[108] Baskaran MM, Bondhugula U, Krishnamoorthy S, Ramanujam J, Rountev A, Sadayappan P. A compiler framework for optimization of affine loop nests for GPGPUs. In: Proc. of the 22nd Annual Int'l Conf. on Supercomputing (ICS). ACM Press, 2008. 225−234.

[109] Leung A, Vasilache N, Meister B, Baskaran M, Wohlford D, Bastoul C, Lethin R. A mapping path for multi-GPGPU accelerated computers from a portable high level programming abstraction. In: Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU). ACM Press, 2010. 51−61.

[110] Konstantinidis A, Kelly PHJ, Ramanujam J, Sadayappan P. Parametric GPU code generation for affine loop programs. In: Proc. of the Int'l workshop on Languages and Compilers for Parallel Computing (LCPC). Springer-Verlag, 2013. 136−151.

[111] Juega JC, Gómez JI, Tenllado C, Catthoor F. Adaptive mapping and parameter selection scheme to improve automatic code generation for GPUs. In: Proc. of the Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). ACM Press, 2014. 251−261.

[112] Shirako J, Hayashi A, Sarkar V. Optimized two-level parallelization for GPU accelerators using the polyhedral model. In: Proc. of the 26th Int'l Conf. on Compiler Construction (CC). ACM Press, 2017. 22−33.

[113] Grosser T, Hoefler T. Polly-ACC transparent compilation to heterogeneous hardware. In: Proc. of the 2016 Int'l Conf. on Supercomputing (ICS). ACM Press, 2016. 1:1−1:13.

[114] Baghdadi R, Beaugnon U, Cohen A, Grosser T, Kruse M, Reddy C, Verdoolaege S, Betts A, Donaldson A, Ketema J, Absar J, Haastregt SV, Kravets A, Lokhmotov A, David R, Hajiyev E. Pencil: A platform-neutral compute intermediate language for accelerator programming. In: Proc. of the 2015 Int'l Conf. on Parallel Architecture and Compilation (PACT). IEEE Computer Society, 2015. 138−149.

[115] Bondhugula U. Compiling affine loop nests for distributed-memory parallel architectures. In: Proc. of the Int'l Conf. on High Performance Computing, Networking, Storage and Analysis (SC). ACM Press, 2013. 33:1−33:12.

[116] Ravishankar M, Dathathri R, Elango V, Pouchet LN, Ramanujam J, Rountev A, Sadayappan P. Distributed memory code generation for mixed irregular/regular computations. In: Proc. of the 20th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). ACM Press, 2015. 65−75.

[117] Reddy C, Bondhugula U. Effective automatic computation placement and data allocation for parallelization of regular programs. In: Proc. of the 28th ACM Int'l Conf. on Supercomputing (ICS). ACM Press, 2014. 13−22.

[118] Dathathri R, Reddy C, Ramashekar T, Bondhugula U. Generating efficient data movement code for heterogeneous architectures with distributed-memory. In: Proc. of the 22nd Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). IEEE Computer Society, 2013. 375−386.

[119] Trifunović K, Nuzman D, Cohen A, Zaks A, Rosen I. Polyhedral-Model guided loop-nest auto-vectorization. In: Proc. of the 18th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). IEEE Computer Society, 2009. 327−337.

[120] Kong M, Veras R, Stock K, Franchetti F, Pouchet LN, Sadayappan P. When polyhedral transformations meet SIMD code generation. In: Proc. of the 34th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2013. 127−138.

[121] Bondhugula U, Ramanujam J, Sadayappan P. Automatic mapping of nested loops to FPGAs. In: Proc. of the 12th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). ACM Press, 2007. 101−111.

[122] Natale G, Stramondo G, Bressana P, Cattaneo R, Sciuto D, Santambrogio MD. A polyhedral model-based framework for dataflow implementation on FPGA devices of iterative stencil loops. In: Proc. of the 2016 IEEE/ACM Int'l Conf. on Computer-Aided Design (ICCAD). IEEE Computer Society, 2016. 1−8.

[123] Liu D, Yin S, Liu L, Wei S. Polyhedral model based mapping optimization of loop nests for CGRAs. In: Proc. of the 50th ACM/EDAC/IEEE Design Automation Conf. (DAC). IEEE Computer Society, 2013. 1−8.

[124] Pouchet LN, Zhang P, Sadayappan P, Cong J. Polyhedral-Based data reuse optimization for configurable computing. In: Proc. of the ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays (FPGA). ACM Press, 2013. 29−38.

[125] Zuo W, Li P, Chen D, Pouchet LN, Zhong S, Cong J. Improving polyhedral code generation for high-level synthesis. In: Proc. of the Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ ISSS). IEEE Computer Society, 2013. 1−10.

[126] Wang Y, Li P, Cong J. Theory and algorithm for generalized memory partitioning in high-level synthesis. In: Proc. of the 2014 ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays (FPGA). ACM Press, 2014. 199−208.

[127] Zuo W, Kemmerer W, Lim JB, Pouchet LN, Ayupov A, Kim T, Han K, Chen D. A polyhedral-based SystemC modeling and generation framework for effective low-power design space exploration. In: Proc. of the IEEE/ACM Int'l Conf. on Computer-Aided Design (ICCAD). IEEE Computer Society, 2015. 357−364.

[128] Meeus W, Stroobandt D. Automating data reuse in high-level synthesis. In: Proc. of the Conf. on Design, Automation & Test in Europe (DATE). European Design and Automation Association, 2014. 298−301.

[129] Bhaskaracharya SG, Bondhugula U. Polyglot: A polyhedral loop transformation framework for a graphical dataflow language. In: Proc. of the Int'l Conf. on Compiler construction (CC). Springer-Verlag, 2013. 123−143.

[130] Henretty T, Veras R, Franchetti F, Pouchet LN, Ramanujam J, Sadayappan P. A stencil compiler for short-vector SIMD architectures. In: Proc. of the 27th Int'l ACM Conf. on Supercomputing (ICS). ACM Press, 2013. 13−24.

[131] Sbîrlea A, Shirako J, Pouchet LN, Sarkar V. Polyhedral optimizations for a data-flow graph language. In: Proc. of the Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC). Springer-Verlag, 2015. 57−72.

[132] Mullapudi RT, Vasista V, Bondhugula U. Polymage: Automatic optimization for image processing pipelines. In: Proc. of the 20th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM Press, 2015. 429−443.

[133] Ragan-Kelley J, Barnes C, Adams A, Paris S, Durand F, Amarasinghe S. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In: Proc. of the 34th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2013. 519−530.

[134] Chugh N, Vasista V, Purini S, Bondhugula U. A DSL compiler for accelerating image processing pipelines on FPGAs. In: Proc. of the 2016 Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT). IEEE Computer Society, 2016. 327−338.

[135] Reddy C, Kruse M, Cohen A. Reduction drawing: Language constructs and polyhedral compilation for reductions on GPU. In: Proc. of the 2016 Int'l Conf. on Parallel Architectures and Compilation (PACT). ACM Press, 2016. 87−97.

[136] Spampinato DG, Püschel M. A basic linear algebra compiler. In: Proc. of the Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). ACM Press, 2014. 23−32.

[137] Spampinato DG, Püschel M. A basic linear algebra compiler for structured matrices. In: Proc. of the 2016 IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). IEEE Computer Society, 2016. 117−127.

[138] Simburger A, Apel S, Grosslinger A, Lengauer C. The potential of polyhedral optimization: An empirical study. In: Proc. of the 2013 IEEE/ACM 28th Int'l Conf. on Automated Software Engineering (ASE). IEEE Computer Society, 2013. 508−518.

[139] Baskaran MM, Vydyanathan N, Bondhugula U, Ramanujam J, Rountev A, Sadayappan P. Compiler-Assisted dynamic scheduling for effective parallelization of loop nests on multicore processors. In: Proc. of the 14th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP). ACM Press, 2009.

[140] Dathathri R, Mullapudi RT, Bondhugula U. Compiling affine loop nests for a dynamic scheduling runtime on shared and distributed memory. ACM Trans. on Parallel Computing (TOPC), 2016,3(2):12:1−12:28.

[141] Kong M, Pop A, Pouchet LN, Govindarajan R, Cohen A, Sadayappan P. Compiler/Runtime framework for dynamic dataflow parallelization of tiled programs. ACM Trans. on Architecture and Code Optimization (TACO), 2015,11(4):61:1−61:30.

[142] Pop A, Cohen A. OpenStream: Expressiveness and data-flow compilation of OpenMP streaming programs. ACM Trans. on Architecture and Code Optimization (TACO), 2013,9(4):53:1−53:25.

[143] Bao W, Krishnamoorthy S, Pouchet LN, Rastello F, Sadayappan P. PolyCheck: Dynamic verification of iteration space transformations on affine programs. In: Proc. of the 43rd Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). ACM Press, 2016. 539−554.

[144] Sampaio DN, Pouchet LN, Rastello F. Simplification and runtime resolution of data dependence constraints for loop transformations. In: Proc. of the Int'l Conf. on Supercomputing (ICS). ACM Press, 2017. 10:1−10:11.

[145] Doerfert J, Grosser T, Hack S. Optimistic loop optimization. In: Proc. of the 2017 IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). IEEE Computer Society, 2017. 292−304.

[146] Jimborean A, Clauss P, Dollinger JF, Loechner V, Caamaño JMM. Dynamic and speculative polyhedral parallelization using compiler-generated skeletons. Int'l Journal of Parallel Programming (IJPP), 2014,42(4):529−545.

[147] Sukumaran-Rajam A, Clauss P. The polyhedral model of nonlinear loops. ACM Trans. on Architecture and Code Optimization (TACO), 2016,12(4):48:1−48:27.

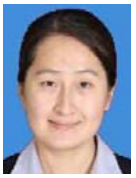
附中文参考文献:

[11] 陈立前,王戟,刘万伟.基于约束的多面体抽象域的弱接合.软件学报,2010,21(11):2711−2724. http://www.jos.org.cn/1000-9825/3664.htm [doi: 10.3724/SP.J.1001.2010.03664]

[47] 张纪林,狄鹏,蒋从锋,张伟,徐向华,万健,任永坚.一类基于迭代空间条块的并行有限差分 Stencil 算法.软件学报,2010,21:270−283. http://www.jos.org.cn/1000-9825/10028.htm

赵捷(1987－),男,内蒙古通辽人,博士,讲师,CCF 专业会员,主要研究领域为先进编译技术.

赵荣彩(1957－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为先进编译技术,网络安全,逆向工程.



李颖颖(1984－),女,讲师,CCF 专业会员,主要研究领域为先进编译技术.