[论文阅读]--在多面体模型中生成代码比你想象的要容易

摘要：多面体模型在自动并行化和优化方面取得了许多进展。广泛的研究表明，该计算模型为推理和应用程序转换提供了方便的抽象。然而，代码生成的复杂性一直是在优化编译器时使用多面体表示的一个障碍。首先，代码生成器很难处理生成的代码大小和控制开销，这可能会破坏转换所获得的理论上的好处。其次，这一步骤通常是耗时的，阻碍了在生产编译器中集成多面体框架或基于反馈的迭代优化方案。此外，当前的代码生成算法只覆盖了有限的可能转换函数集。本文讨论了处理非幺模、非可逆、非积分甚至非一致函数的一般变换框架，并提出了对最先进的代码生成算法的几个改进。研究了两个方向:生成代码的大小和代码生成器的效率。实验证明，改进后的方法能够处理实际问题。

1.介绍

通常的编译器中间表示(如抽象语法树)不适用于复杂的程序重构。简单的优化，例如常数折叠或标量替换可以实现，而不需要对这种刚性数据结构进行艰难的修改。但更复杂的转换，如循环反转、倾斜、平铺等，会修改执行顺序，这与语法相去甚远。一个基于程序和转换的线性代数表示的模型出现在80年代，以解决这个问题:多面体(或多面体)模型。

一句话：多面体模型是用于解决 *编译器中间表示(如抽象语法树)不能解决的* 复杂的程序重构难题（复杂的转换，如循环反转、倾斜、平铺等）。

这个模型因其丰富的数学理论和直观的几何解释而变得非常流行。此外，它添加了一类具有非常规则控制的代码，其中包括大量实际程序部件。

多面体框架基本上是常规编译过程的一个插件。它从抽象语法树开始，将适合模型的程序部分转换为线性代数表示。下一步是通过使用重新排序函数(调度、放置或分块函数)来选择新的执行顺序。寻找合适的执行命令是多面体模型研究的主要课题[4,5,9,12,13,20,22,24,27]。最后，代码生成步骤返回到抽象语法树或返回实现了重新排序函数所暗示的执行顺序的新的源代码。

多面体模型因其丰富的数学理论和直观的几何解释而变得非常流行，他其实是常规编译过程的一个插件。它从抽象语法树开始，将适合模型的程序部分转换为线性代数表示；然后通过使用重新排序函数(调度、放置或分块函数)来选择新的执行顺序（多面体模型研究的主要课题）；最后，代码生成步骤返回到抽象语法树或返回实现了重新排序函数所暗示的执行顺序的新的源代码。

到目前为止，多面体模型未能集成生产编译器，主要原因涉及到代码生成步骤。首先，大多数算法对重排函数要求严格的限制(例如，单模或可逆)，这减少了优化技术找到有效解决方案的机会。其次，用于构建循环的简单方案可能会生成大量包含and/or的低效的代码，从而抵消它们所能实现的优化。最后，该问题的复杂度对实际程序具有挑战性，并阻碍了迭代优化方案中框架的集成。在这篇文章中,我们将展示在多面体模型中如何处理一般的转换，从采用目前已知的最好的算法开始，展示我们是如何在一个合理的时间间隔产生有限的规模增长的高效的目标代码。

在这篇文章中,我们将展示在多面体模型中如何处理一般的转换，从采用目前已知的最好的算法开始，展示我们是如何在一个合理的时间间隔产生有限的规模增长的高效的目标代码。

论文组织如下: 第2节正式介绍了多面体模型。第3节介绍了这个模型中的一般程序转换框架。第4节描述了代码生成算法，提出了实现快速高效的、规模小的代码的新方法。第5节给出了算法实现的实验结果。最后，第6节讨论了相关工作，第7节总结了本文的主要贡献，并讨论了未来的工作。

2.背景和符号

多面体模型是顺序程序和并行程序的一种表示。它对应于像C或FORTRAN这样被称为静态控制程序的命令式语言的子集。这门课包括一系列的课程，薛教授对此进行了深入的讨论。它们的属性可以大致总结如下:(1)控制语句是带有仿射边界的do循环和带有仿射条件的if条件(实际上控制语句可以更复杂);(2)仿射边界和条件只依赖于外循环计数器和常量参数。任何程序中具有静态控制的最大连续语句集称为静态控制部分(SCoP)。图1中的内核是一个严格接受静态控制限制的示例，将用于说明进一步的概念。这种命令式语言中的循环可以用称为迭代向量的n项列向量来表示: $\vec{x} = (i_1, i_2, \ldots, i_n)^T$,

```
    do i=1, n
S1  | x = a(i,i)
    | do j=1, i-1
S2  | | x = x - a(i,j)**2
S3  | p(i) = 1.0/sqrt(x)
    | do j=i+1, n
S4  | | x = a(i,j)
    | | do k=1, i-1
S5  | | | x = x - a(j,k)*a(i,k)
S6  | | a(j,i) = x*p(i)
```

Figure 1. A Cholesky factorization kernel

考虑到静态控制类，程序执行可以通过对每条语句使用两个规范来完整描述:

- 迭代域D，即语句必须执行的迭代向量的值集。当一个语句被静态控件包围时，它的迭代域总是可以由定义多面体的一组线性不等式确定的。多面体一词在广义上用于表示格点的凸集(也称为z多面体或格-多面体)，即由仿射不等式限定的Z向量空间中的一组点:

$$\mathcal{D} = \left\{ \vec{x} \mid \vec{x} \in \mathbb{Z}^n, A\vec{x} \geq \vec{c} \right\},$$

  图3(a)说明了图1中程序的语句S2的静态控制和多面体域之间的对应关系。

- 仿射函数(- x)，仿射函数为迭代域中的每个积分点指定对应语句实例的新坐标:

$$\theta(\vec{x}) = T\vec{x} + \vec{t},$$

  根据上下文，散射函数可能有几种解释:在空间中分布迭代，即跨越不同的处理器，及时对它们排序，或两者(通过组合)，等等。在空间映射的情况下，函数为给定语句实例返回的数字就是执行它的处理器的数量。在n维时间表,声明实例与逻辑日期(a1...一个)执行那些相关日期之前(b1...bn)敌我识别∃我1≤我< n (a1...ai) = (b1...bi)∧ai + 1 < bi + 1,即他们按照词典顺序。例如，我们可以很容易地通过使用这个程序[12]的抽象语法树来获取任何静态控制程序的顺序执行顺序:我们可以直接在图2所示的AST上为图1中的程序读取这样的函数，例如:xS1) = (0,i,0)T, S2(?xS2) = (0,i,1,j,0)T, S3(?= (0,i,2)Tetc。

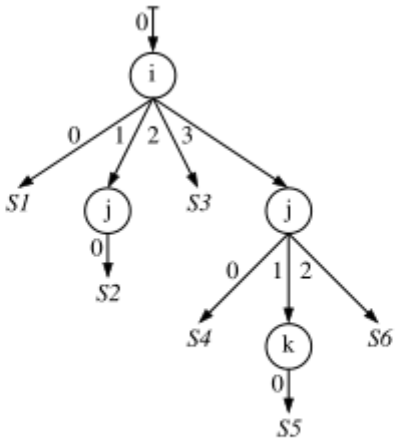


Figure 2. AST of the program in Figure 1

3.程序的转换

在多面体模型中的程序变换可以通过选择适当的仿射函数来指定。他们修改源多面体为包含相同的点，但在一个新的协调系统的 目标多面体，因此有一个新的字典顺序。实现这些转换是多面体框架的核心部分。因为目前的多面体代码生成算法只寻址一个可能函数的子集，所以缺乏灵活性。如何使用一般的仿射函数对原始多面体应用新的字典顺序将在3.1节中解释。第3.2节和第3.3节分别处理了非积分变换和非一致变换的特殊情况，并展示了如何在这个框架中处理它们。

3.1仿射变换

以前在多面体模型中代码生成的工作对散射函数有严格的限制，如单模1,17或至少可逆[20,27,22,5]。潜在的原因是，考虑到由A?x≥?c和导致目标指数的散射函数?y = T ?x，新坐标系中的多面体定义为(AT−1)?y≥?c，基底的改变。Griebl等人利用变换矩阵[14]的平方可逆扩展，提出了可逆性约束的第一次松弛。不幸的是，他们的方法实际上导致了非常高的控制开销。

在本文中，我们没有对转换函数施加任何约束，因为我们没有尝试执行原始多面体的基底到目标索引的变化。相反，我们通过在起始位置增加新的维度，将新的词典顺序应用到多面体。因此，从每一个多面体D和散射函数的俯仰角，可以建立另一个多面体T，并有合适的字典顺序:

$$\mathcal{T} = \left\{ \begin{pmatrix} \vec{y} \\ \vec{x} \end{pmatrix} \;\middle|\; \left[\begin{array}{c|c} Id & -T \\ \hline 0 & A \end{array}\right] \begin{pmatrix} \vec{y} \\ \vec{x} \end{pmatrix} \begin{matrix} = \\ \geq \end{matrix} \begin{pmatrix} \vec{t} \\ \vec{c} \end{pmatrix} \right\}$$

根据定义,当且仅当$\vec{y} = \boldsymbol{\theta}(\vec{x})$时 $(\vec{y}, \vec{x}) \in \boldsymbol{T}$ 。

新的多面体内的点按字典顺序排列，直到->y的最后一个维度，剩下的维度没有特定的顺序。

通过使用这种转换策略，原始迭代域和转换的数据都包含在新的多面体中。作为说明，让我们考虑图3(a)中的多面体ds2和散射函数S2(i, j) = 2i+j。相应的散射矩阵T =[2 1]是不可逆的，但可以推广到Griebl等人提出的T = h2 10 0 1 i。通常产生的多面体如图3(b)所示。如果我们选择自由维度的词典顺序，我们的策略将直接指向图3(c)中的多面体。在i和i上的投影将得到图3(b)中的结果。附加维度携带转换数据，即在这种情况下j = i−2i。这是很有帮助的，因为在代码生成期间，我们必须更新循环体中对迭代器的引用，当转换不可逆转时，这是必要的。这种转换策略的另一个特性是永远不要构建合理的目标约束系统。在变换函数非幺模的情况下，这一问题对以往的研究提出了很大的挑战。我们可以观察到图3(b)中的现象。没有重点的整数点在原始多面体中没有图像。通过→原始= T−1→目标，可以确定目标的原始坐标。因为T是非幺模的，所以T−1有有理元素。从而使一些整数目标点在原空间中有一个有理像;它们叫做洞。为了避免考虑这些洞，必须找到跨步(考虑积分点之间的步骤)。很多作品都提出用厄米特范式

以不同的方式来解决这个问题.相反，我们不改变原始多面体的基础，但我们只适用一个适当的字典顺序。因此，我们的目标系统总是积分的，并且在相应的多面体上没有洞。步幅信息以方程的形式明确地包含在约束系统中。

这种方法的代价是给多面体增加新的维度。这可能是一个相关的问题，因为首先，它增加了扫描步骤的复杂性，其次，它增加了约束系统的大小，而高级代码生成通常需要大量内存。在实践中，用第4节中介绍的方法处理附加维常常是琐碎的。最终，与基于其他方法的原型相比，我们的原型效率更高，所需内存更少(见第5节)。

3.2 有理变换

一些自动分配程序或调度程序需要使用有理变换。因此仿射函数可以有一个更一般的形式:

$$\theta(\vec{x}) = (T\vec{x} + \vec{t})/\vec{d},$$

其中/表示整数除法，->d是一个常数向量，每个元素都能除法出(->x)对应的尺寸。在实践中，约数通常对应于资源约束(例如处理器的数量，功能单元的数量等)。Wetzel提出了解决这个问题的第一个解决方案，但对整个散射函数只有一个除数值，导致了复杂的控制。

同样，我们建议增加维度来解决这个问题。对于(T->x)/->d中的每一个有理元，我们引入一个辅助变量来表示除法的商。例如，让我们考虑图4(a)中的原始多面体和调度函数lai (i) = i/3 + 1。我们引入q和r，如i = 3q + r，根据定义0≤r = i−3q≤2。然后我们可以处理一个等价的积分变换，即:在0≤i - 3q≤2的情况下，取->(q) = q + 1。如图4(b)所示，这相当于对尺寸i进行露天开采。对于几个非整数系数，我们只需要更多的辅助变量来表示除法的结果

3.3非一致的转换

随着时间的演变程序分析的能力在不断增强，同时为了解决新的优化挑战，程序转换也变得越来越复杂。从单个循环嵌套的简单转换开始，它们发展到语句式函数，最近发展到每个语句几种转换，它们中的每一个都适用于迭代域的子集。因此，迭代域为D的语句的散射函数可以是如下形式:

$$\theta(\vec{x}) = \begin{cases} \text{if } \vec{x} \in \mathcal{D}_1 \text{ then } T_1\vec{x} + \vec{t_1} \\ \text{if } \vec{x} \in \mathcal{D}_2 \text{ then } T_2\vec{x} + \vec{t_2} \\ \dots \\ \text{if } \vec{x} \in \mathcal{D}_n \text{ then } T_n\vec{x} + \vec{t_n} \end{cases}$$

其中Di，1 ≤ i ≤ n是d的一个分块。至少当代码生成器有效地处理一个以上的多面体时，通过显式地将所考虑的多面体分成多个分块来处理这种变换是非常简单的。当使用仿射条件分割迭代域时，就像在索引集分割中一样，建立分区是很琐碎的，但是只要我们能够将每个子分区表示为多面体，就有可能用非仿射条件进行更一般的分区。例如，斯拉马合金。找到最佳并行化需要非一致转换的程序，例如，θ(i) = if (i m o d d = n)然后...其他...其中d是标量值，n是可能的参数常数。他们提出了一个专门解决这个问题的代码生成方案。通过添加新的维度，在我们的框架中处理这个问题是可能的。例如，对应于θ(i)的然后部分的迭代域将是具有附加约束i = jd+n的原始域，而其他部分的附加约束可以是I≤JD+n-1和I≥JD+n+1-d。然后我们可以将转换应用于结果多面体，如第3.1节所示。

4.扫描多面体

我们在前面的章节中展示了任何静态控制程序都可以使用一组迭代域和分散函数来指定，这些函数可以合并来创建具有适当字典顺序的新多面体。在多面体模型中生成代码相当于找到一组嵌套循环，按照这个顺序一次且只能一次访问每个多面体的每个积分点。这是框架中的一个关键步骤，因为最终的程序有效性高度依赖于目标代码质量。特别是，我们必须确保糟糕的控制管理不会破坏性能，例如产生冗余条件、复杂的循环边界或未充分利用的迭代。另一方面，我们必须避免代码爆炸，因为大量的代码可能会污染指令缓存。

目前，当我们必须为几个多面体生成一个扫描码时，奎尔勒等人的方法给出了最好的结果。该技术保证在扫描散射维度时避免冗余控制。然而，它也有一些局限性，例如高复杂性和不必要的代码爆炸。下面，我们针对这些缺点提出一些解决方案。我们在第4.1节中介绍了通用算法，并对其进行了一些修改。我们在第4.2节中讨论了在不影响代码效率的情况下减少代码大小的问题。最后，在4.3节中，我们讨论了复杂性问题。

4.1扩展Quillere等人的算法

Quillere等人最近提出了第一个代码生成算法，直接构建没有冗余控制的目标代码，而不是从简单的代码开始并试图改进它[21]。因此，这种方法总是能够去除防护装置，并且处理更容易。最终，它会更高效地生成更好的代码。该算法依赖于多面体运算，多面体运算可以通过例如PolyLib1[26]来实现。

它的基本原理是，从要扫描的多面体列表开始，递归地生成扫描代码(AST)的抽象语法树的每一级。AST的节点用一个多面体T来标记，并且有一个子节点列表(符号T →(...))。树子节点用一个多面体和一个陈述(符号TS)来标记。每个递归都构建一个AST节点列表，如图5中的算法所示。它从以下输入开始:(1)要扫描的变换多面体的列表(TS1，...，TSn)；(2)上下文，即全局参数上的约束集；(3)第一维d = 1。从AST生成代码是一个简单的步骤:如果约束涉及对应于节点级别的维度或不涉及对应于节点级别的维度，那么标记每个节点的约束系统可以分别直接转换为循环边界和相关条件。

该算法与Quillere等人在中提出的算法及在其改进版本有些不同；我们的两个主要贡献如下:减少代码大小而不降低代码性能(步骤7)和通过使用模式匹配减少代码生成的处理时间(步骤3)。

CODEGENERATION: Build a polyhedra scanning code without redundant control AST.

Input: a polyhedron list $(\mathcal{T}_{S_1}, ..., \mathcal{T}_{S_n})$, a context C, the current dimension d.
Output: the abstract syntax tree of the code scanning the polyhedra inside the input list.

1. Intersect each polyhedron $\mathcal{T}_{S_i}$ in the list with the context C in order to restrict the domain (and subsequently the code that will be generated) under the context of the surrounding loop nest.
2. Compute for each resulting polyhedron $\mathcal{T}_{S_i}$ its projection $\mathcal{P}_i$ onto the outermost d dimensions and consider the new list of $\mathcal{P}_i \rightarrow \mathcal{T}_{S_i}$.
3. Separate the projections into a new list of disjoint polyhedra: given a list of m polyhedra, start with the first two polyhedra $\mathcal{P}_1 \rightarrow \mathcal{T}_{S_1}$ and $\mathcal{P}_2 \rightarrow \mathcal{T}_{S_2}$ by computing $(\mathcal{P}_1 - \mathcal{P}_2) \rightarrow \mathcal{T}_{S_1}$ (i.e. S_1 alone), $(\mathcal{P}_1 \cap \mathcal{P}_2) \rightarrow (\mathcal{T}_{S_1}, \mathcal{T}_{S_2})$ (i.e. S_1 and S_2) and $(\mathcal{P}_2 - \mathcal{P}_1) \rightarrow \mathcal{T}_{S_2}$ (i.e. S_2 alone), then for the three resulting polyhedra, make the same separation with $\mathcal{P}_3 \rightarrow \mathcal{T}_{S_3}$ and so on.
4. Build the lexicographic ordering graph where there is an edge from a polyhedron $\mathcal{P}_1 \rightarrow (\mathcal{T}_{S_p}, ..., \mathcal{T}_{S_q})$ to another polyhedron $\mathcal{P}_2 \rightarrow (\mathcal{T}_{S_v}, ..., \mathcal{T}_{S_w})$ if its scanning code has to precede the other to respect the lexicographic order, then sort the list according to a valid order.
5. For each polyhedron $\mathcal{P} \rightarrow (\mathcal{T}_{S_p}, ..., \mathcal{T}_{S_q})$ in the list:
   (a) Compute the stride that the inner dimensions impose to the current one, and find the lower bound by looking for stride constraints in the $(\mathcal{T}_{S_p}, ..., \mathcal{T}_{S_q})$ list.
   (b) While there is a polyhedron in $(\mathcal{T}_{S_p}, ..., \mathcal{T}_{S_q})$:
      i. Merge successive polyhedra with another dimension to scan in a new list.
      ii. Recurse for the new list with the new loop context $C \cap \mathcal{P}$ and the next dimension $d + 1$.
6. For each polyhedron $\mathcal{P} \rightarrow (\text{inside})$ in the list, apply steps 2 to 4 of the algorithm to the *inside* list in order to remove dead code. Then consider the concatenation of the resulting lists as the new list.
7. Make all the possible unions of *host* polyhedra with *point* polyhedra to reduce code size.
8. Return the polyhedron list.

Figure 5. Extended Quilleré et al. Algorithm

代码生成:构建一个没有冗余控制的多面体扫描代码。

输入:一个多面体列表(TS1，...，TSn)，一个上下文C，当前维度d。

输出:扫描输入列表中多面体的代码的抽象语法树。

1.将列表中的每个多面体与上下文C相交，以便在相关的循环嵌套的上下文中限制迭代域(以及随后将生成的代码)。

2.计算每个生成的多面体TSi在最外层的d维上的投影Pi，并计算新的Pi→ TSi列表。

3.将投影分离成一个新的不相交多面体列表:给定m个多面体的列表，从最初的两个多面体P1→TS1和P2→TS2开始计算(P1-P2)→TS1(即S1alone)、(P1∩ P2) → (TS1，TS2)(即S1和S2)和(P2-P1)→TS2(即S2alone)，然后对于三个多面体，用P3→TS3进行同样的分离，以此类推。

4.如果从一个多面体P1→(TSp，...，TSq)到另一个多面体P2→(TSv，...，TSw)有一条边，则构建一个字典排序图。如果它的扫描代码必须在另一个之前以遵守字典顺序，那么根据有效的顺序对列表进行排序。

5.对于列表中的每个多面体P → (TSp，...，TSq):

(a)计算内部维度施加给当前维度的跨度(stride)，并通过在(TSp，...，TSq)列表中查找跨度约束来找到下界。

(b)当有一个多面体在(TSp，..., TSq)中时:

ⅰ.将连续的多面体与另一个维度合并，以便在新列表中扫描。

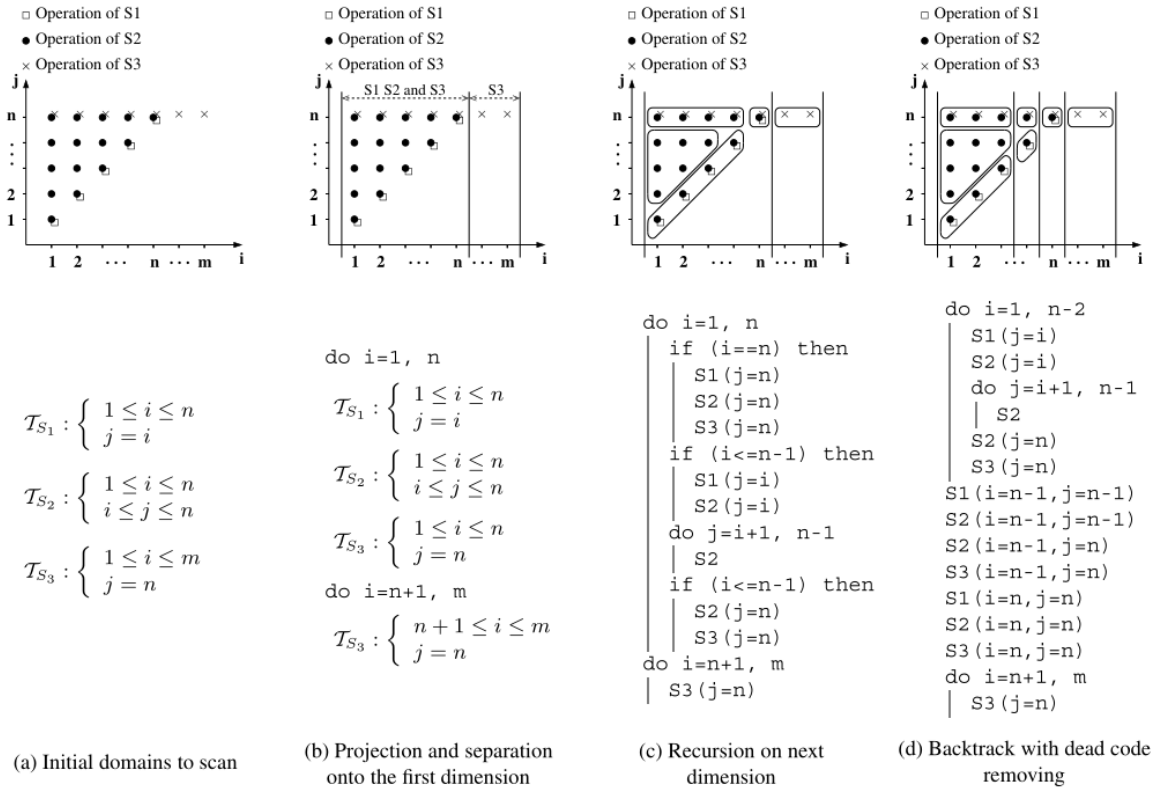
ii.递归处理带有新的循环上下文C∩P和下一个维度d + 1的新列表。

6.对于列表中的每个多面体P→(inside)，对inside列表应用算法的第2步到第4步，以去除死代码。然后将结果列表连接作为新的列表。

7.将所有可能的主（host）多面体与点(point)多面体结合，以减少代码大小。

> 8.返回多面体列表。

我们建议通过图6中的例子来说明这个算法(没有步骤7)。我们必须为图6(a)中的三个多面体生成扫描代码。为了简单起见，我们将直接显示节点约束系统到源代码的转换。我们首先计算与上下文的交集(即，在这一点上，对参数的约束，假设为n ≥ 2和m ≥ n)。我们把多面体投影到第一维I上，然后把它们分成不相交的多面体。如图6(b)所示，这导致了两个不相交的多面体。我们现在可以为这个第一维度生成扫描代码。然后我们在下一个维度上递归，对每个多面体列表重复这个过程(在这个例子中，现在有两个列表:每个generatedouter循环内部一个)。我们将每个多面体与新的上下文相交，即外环迭代域；然后我们将得到的多面体投影到外部维度上。最后，我们将这些投影分离成不相交的多面体。最后一个过程对于第二个列表来说并不重要，但是对于第一个列表来说会产生几个域，如图6(c)所示。然后我们生成与新维度相关联的代码，由于这是最后一个维度，因此会完全生成一个扫描代码。最后，我们通过在递归回溯期间应用新的投影步骤来移除死代码(例如，在图6(c)中的第一个循环嵌套中，迭代i = n只对循环体的一小部分有用)。最终代码如图6(d)所示。



(a) Initial domains to scan

(b) Projection and separation onto the first dimension

(c) Recursion on next dimension

(d) Backtrack with dead code removing

Figure 6. Step by step code generation example

4.2减少代码大小

多面体模型中优化方法的功能对于嵌入式系统编译特别有意义。由于固有的硬件限制，目标代码的大小是此类应用程序的主要限制之一。为此目的或为了避免指令高速缓存污染，可以控制生成的代码大小。可以使用迭代代码生成方法轻松地对其进行管理：它们从幼稚的（低效率的）短代码开始，并通过选择要删除的条件并执行代码提升（在选定条件下拆分代码并消除代码）来消除控制开销。在两个分支中复制原始的受保护代码）。因此，停止代码吊起将停止代码增长。**使用本文讨论的递归代码生成方法，总是可以选择不分离多面体，并在有条件的情况下生成较小的代码。**这些技术始终以生成效率较低的代码为代价。本节以其他方式介绍，对控制开销的影响很小，并且可能显着改善了代码大小。它基于一个简单的观察结果：分离多面体通常会导致某些点之间的隔离，但这并非总是必要的。图6显示了这种现象的一个生动的例子（与Omega CodeGen一样，基于提升的代码生成器也必须满足相同的问题）。**如果可能，将这些点集成到主循环中将通过添加新的迭代来减少代码大小。** Bouchebaba首先在二维环巢融合[4]的特殊情况下指出了这个问题。他提取了14种不应该将顶点与循环融合的情况，并在其他情况下应用了融合。下面提供了一种基于图5中代码构造算法的属性的通用代码生成解决方案。

为了确保分离步骤不会导致不必要的多面体剥离，有必要计算此分离。另外，由于投影在分离过程中隐藏了其中的某些尺寸，因此我们必须在每个尺寸上实现递归。因此，我们可以在每次递归结束时删除孤立的点（步骤7）。在给定的递归深度下，将对列表中的每个循环节点应用删除过程（即，使得与当前深度对应的尺寸不是恒定的）：1.定义要与该节点合并的点候选：节点按深度优先顺序分支，并在叶子中构建语句列表。候选语句必须适合该语句列表，因为在消除死代码后可以保证叶子中的每个语句至少执行一次。因此，只有具有此结构的点可以与节点合并。 2.检查该点是否直接在步骤4和6中构建的词典顺序图中的节点之前或之后（有关此图的详细信息，请参见[21]）。该图仅基于投影尺寸，但是如果候选点直接跟随排序图中的节点并且无法合并，则意味着输入多面体不是凸的，这是一个矛盾。 3.如果先前测试成功，则使用多面体将合并候选点与节点，并从多面体列表中删除点。我们可以将此过程应用于图6中的示例。删除死代码后AST的转换与图6(c)中的代码相同。j循环的候选语句是S2。我们可以合并S2点在这个循环之前和之后。那么去维死码只会分离出对应于i = n的点，新的候选点为S1S2S3。可以合并它，最终的代码如图7所示，对象代码大小为176B，而图6(d)中的前一个代码为464B(每个语句是一个二维数组条目增量)。

4.3复杂问题

代码生成过程中的主要计算核心是分离出不相交的多面体(第3步)，给定n个多面体列表，最坏情况复杂度为O(3n)个多面体操作(指数本身)。此外，内存使用非常高，因为我们必须为每个分隔的域分配内存。对于这两个问题，我们提出了一个局部解决方案。

我们使用模式匹配来减少多面体的计算次数:在给定深度域通常是相同的(这是输入代码的属性,这17%的操作发生在第五节中给出的指标集),或分离(这是调度矩阵的性质,这为36%的操作发生在基准组第五节)。因此我们多面操作之前检查这些属性快速通过比较直接的元素约束系统(这使得寻找平等的75%),通过比较具有固定值的未知数(这允许找到94%的析取词)。当证明了其中一个性质时，我们可以直接给出该操作的平凡解。该方法将性能提高了近2倍。

为了避免内存分配大爆炸，当我们检测到高内存消耗时，我们用一种更简单的算法继续为剩余的递归生成代码，生成效率较低的代码，但使用的内存要少得多。我们没有将投影分割成不相交的多面体(算法的第3步)，而是在它们的交点不为空时将它们合并。然后我们使用一组联合，比一组不相交的多面体要小得多。算法的其他部分未作修改。该方法的缺点是产生了代价昂贵的条件来决定是否扫描一个积分点。这种方法可以与使用多面体的凸壳进行比较[15,25,14,5]，但由于它可以处理不描述凸多面体的复杂边界(通常是参数化仿射约束的最大值或最小值，如max(m，n))，因此更为通用。

5. 实验结果

我们实现了这个算法，并将其集成到Open64/ORC[3]内部的一个完整的多面体转换基础设施中。这样一个现代的编译器提供了许多步骤来提取大型的静态控制部分(例如，内联函数、标准化循环、goto消除、归纳变量替换等)。在本节中，我们将研究所提出的框架对大型、有程序代表性的SCoPs的适用性，这些SCoPs是从SPECfp2000和PerfectClub基准中提取的。所选择的方法是执行所有这些静态控制部分的代码再生。

图8给出了一些关于SPECfp 2000和PerfectClub基准测试的代码再生问题的信息。前两列给出了对应基准测试中的SCoPs和迭代域的总数。这些问题被认为是困难的:之前Omega和LooPo的**经验表明，在没有时间或内存爆炸的情况下，仅仅为10个左右的多面体生成高效代码就是很具有挑战性的。**代码生成部分的两列显示了在Intel Pentium III 1GHZ和512 MB内存的架构上，由于内存爆炸和总的代码生成时间，可以以次最优化的方式部分重新生成多少部分的SCoPs。*这三个具有挑战性的问题具有大量的自由参数（13或14），从而导致较高的代码版本控制。 lucas（超过1700个域）中最大的一个花费了22分钟和1 GB RAM才能在Itanium 1 GHz计算机上最佳生成。*这些结果非常令人鼓舞，因为代码生成器证明了其利用数百条语句和许多自由参数的重新生成现实问题的能力。尽管存在最坏情况的指数算法复杂性，但是代码生成时间和内存需求都是可接受的。

	SCoPs		Code Generation		Robustness	
	Total	Domains	Sub.	Time (s)	CodeGen	LoopGen
applu	25	757	0	28.16	39%	53%
apsi	109	2192	1	42.13	98%	98%
art	62	499	0	1.50	99%	100%
equake	40	639	0	6.80	73%	73%
lucas	11	2070	1	47.58	1%	1%
mgrid	12	369	0	4.53	54%	54%
swim	6	123	0	0.58	100%	100%
adm	109	2260	1	43.94	92%	92%
dyfesm	112	1497	0	14.81	84%	86%
mdg	33	530	0	4.52	82%	100%
mg3d	63	1442	0	18.56	85%	85%
qcd	74	819	0	28.23	79%	86%

Figure 8. Code generation of static control parts in high-performance applications

图8.高性能应用中静态控制部件的代码生成

我们相比结果通过代码生成器,CLooG2,先前实施Quillere e t人算法,LoopGen 0.4 [21]（CLooG之间的差异和LoopGen改进本文中讨论的直接后果),和使用最广泛的代码生成器的多面体模型,即ωCodeGen 1.2[15]。由于固有的限制(主要是内存爆炸)，这些生成器不能处理基准测试集中的所有实际代码生成问题，图8中的健壮性部分给出了它们能够处理输入问题的百分比3。这些结果说明了对代码生成方案可伸缩性的现有需求。因此，只对公共子集进行比较。这两个值是代码生成时间和相对于原始代码大小的生成代码大小。结果如图9所示。它表明，直接生成一个没有冗余控制的代码比尝试改进一个幼稚的代码要有效得多。我们的模式匹配策略证明了它的有效性，因为我们观察到在CLooG和LoopGen之间显著地加速了4.05，在CLooG和LoopGen之间显著地加速了2.57。由LoopGen生成的代码大小通常比CodeGen结果平均大38%，因为它以代码大小为代价消除了更多的控制开销。本文提出的代码大小改进方法在保持生成代码有效性的同时，显著降低了平均6%的增长率。

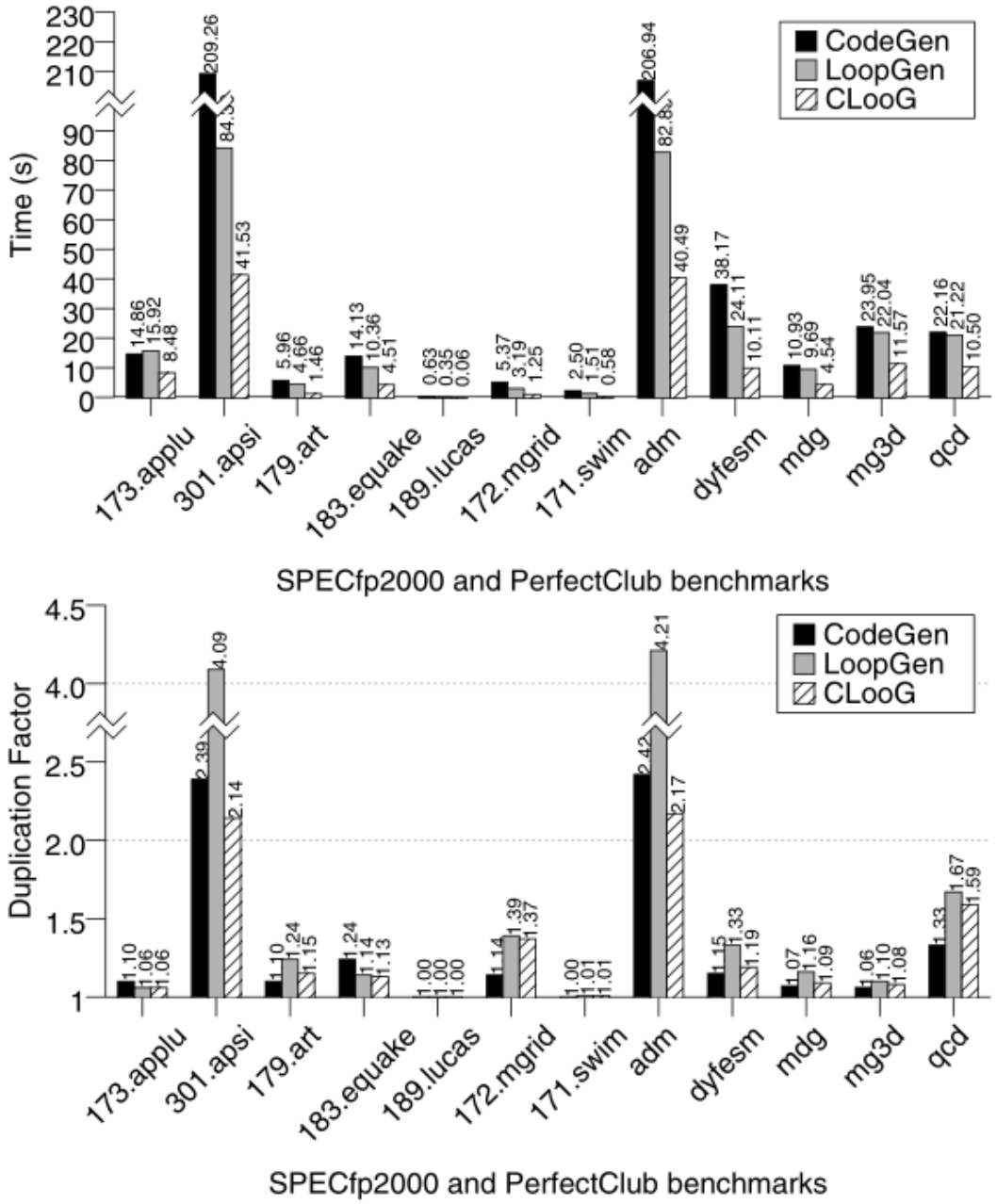


Figure 9. Code generation times and sizes

总之，我们的算法比CodeGen快得多，明显比LoopGen快得多。LoopGen生成更大的代码，而我们的代码和代码原代码是差不多的大小。我们仍然需要比较运行时开销:我们的代码与原始代码具有相同的性能，并且我们相信对于LoopGen来说这也是正确的。由于技术上的原因，评估CodeGen的性能很困难，只能留待以后再做。

6. 相关工作

Ancourt和Irigoin[1]提出了多面体扫描问题的第一个解决方案。他们的开创性工作是基于fourie - motzkin的成对消除。他们的方法的范围是非常有限的，因为它只能应用于一个带有单模变换(散射)矩阵的多面体。其基本思想是应用变换函数作为循环索引的基底的改变，然后对于每一个新的维度，将多面体投影到轴上，从而找到相应的环界。这种方法的缺点是大量的冗余控制。大多数关于代码生成的后续工作都试图扩展这第一种技术，以便处理更一般的转换。Li和Pingali [20]， Xue [27]， Darte[9]和Ramanujam[22]将单模性约束放宽为可逆约束，然后提出了处理非单位步长(环增量可以是不同于1的东西)。他们都使用Hermite范式[23]来寻找步长，并使用经典的fourie - motzkin消去法来计算环界。此外，Li和Pingaliproposeda用一个完整的算法从一个部分矩阵中构建一个非幺模变换函数，比如对于依

赖[20]的变换保持合法。同理，Griebl等人放宽了可逆性约束，提出利用该矩阵[14]的平方可逆扩展来处理任意矩阵。摘要给出了一般仿射变换函数在无单模性、可逆性甚至正则性约束的情况下的处理方法。

作为fourie - motzkin消去法的替代，Collard等人[7]提出了一种基于对偶单纯形算法[10]的参数化版本的环界计算技术。另一种方法使多面体在轴上连续投影，如在[1]中，但是使用Chernikova算法[18]来处理一个表示为一组射线和顶点[19]的多面体。这两种技术的优点是不需要任何冗余控制就可以生成代码(只针对一个多面体)，但是当第二种技术可以生成非常紧凑的代码时，第一种技术可以在长度上迅速激增。首先通过生成简单的完美嵌套代码，然后(部分)消除冗余保护[15]，解决了在同一代码中扫描多个多面体的问题。另一种方法是分别为每个多面体生成代码，然后将它们合并[14,5]。此解决方案生成大量冗余控制，即使分离的代码中没有冗余。

Quillere等人提出通过将多个多面体的并集分离为不相交的多面体子集，并从最外层到最内层[21]生成相应的循环巢，从而递归生成一组扫描多个多面体并集的循环巢。这种稍后的方法提供了目前最好的解决方案，因为它保证了没有冗余控制。然而，它也有一些局限性，例如高度的复杂性或不必要的代码爆炸。本文提出了解决这些问题的方法。

7. 结论

程序优化的当前趋势是将优化转换的选择及其对源代码的应用分开。大多数转换都是重新排序，然后可选地对语句本身进行修改。程序转换器必须被告知所选的重新排序，这通常是通过指令来完成的，如tile、fuse或skew。很难决定一组指令的完整性，或者理解它们的交互。我们认为给出一个仿射函数是另一种指定重新排序的方法，并且它与指令法相比有很多的优点。它的精度更高，具有更好的组合特性，并且在许多情况下可以自动选择仿射函数。为此，本文通过消除对仿射函数的任何附加约束，为最先进的并行化和优化技术提供了一个灵活的转换框架。唯一的缺点是，从一个仿散函数推导出一个程序需要时间，并且可能引入很多运行时开销。我们相信像CLooG这样的工具已经消除了这些困难。整个从源代码到多面体再到源代码的转换已经成功地应用到12个基准测试中，相对于最广泛使用的代码生成器，它能够处理的基准测试部分的速度显著提高了4.05。

正在进行的工作旨在从代码生成步骤推理上游。指出最密集的计算部分在源程序将允许驱动代码生成器，以避免无意义的，时间和代码大小消耗控制开销删除。减少复杂性和代码版本控制的另一种方法是找到每个静态控制部件参数上和之间的仿射约束。

翻译自：第13届平行架构与编译技术国际会议论文集(PACT ' 04) 1089-795X/04 $ 20.00(Proceedings of the 13th International Conference on Parallel Architecture and Compilation Techniques (PACT'04) 1089-795X/04 $ 20.00 IEEE)

write by sheen