

# [文献阅读03]-模板计算的有效自动并行化

## 论文基本情况：

题目：Effective Automatic Parallelization of Stencil Computations

模板计算的有效自动并行化

作者：Sriram Krishnamoorthy, Muthu Baskaran, Uday Bondhugula, J. Ramanujam, Atanas Rountev, P Sadayappan

俄亥俄州立大学计算机科学与工程系

路易斯安那州立大学电子与计算机工程学系计算与技术中心

出版物：[PLDI '07: Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation](https://doi.org/10.1145/1250734.1250761) June 2007 Pages 235–244 <https://doi.org/10.1145/1250734.1250761>

第28届ACM SIGPLAN编程语言设计和实现会议论文集，2007年6月，第235-244页

ACM SIGPLAN编程语言和设计会议起源于1979年编译器构建研讨会，并且已经发展成为该领域的首屈一指的会议。尽管计算机科学已经成熟，可以包括许多其他子领域，但PLDI所代表的核心领域是保持生命力和最新性的需求。计划委员会在2007年的艰苦工作，以及创纪录数量的论文入选，是这种活力的可喜迹象。

ACM编程语言特别兴趣小组（SIGPLAN）探索了编程语言的概念和工具，重点是设计，实现和有效使用。它的成员是编程语言用户，开发人员，实施者，理论家，研究人员和教育工作者。ACM SIGPLAN通告以电子或印刷形式每月提供（将于2018年12月停产），其中包含常规专栏和技术报告，以及来自多个SIGPLAN会议的会议记录和摘要。SIGPLAN每年赞助四次主要会议，另外还有许多针对特定兴趣领域的会议和讲习班。

出版人：美国纽约计算机协会

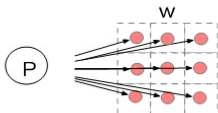
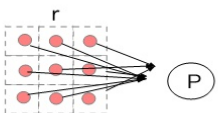
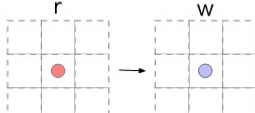
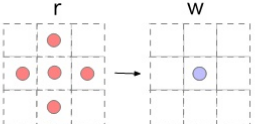
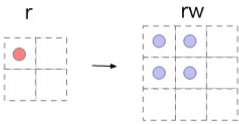
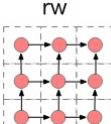
摘要：模版计算的性能优化已在文献中得到广泛研究，因为它们出现在许多计算密集型的科学和工程应用中。还开发了可以转换顺序模板代码以优化数据局部性和并行性的编译器框架。但是，通常需要循环倾斜才能沿时间维度分块模板代码，从而导致分块流水线并行执行中的负载不均衡。在本文中，我们开发了一种模板代码自动并行化的方法，该方法明确解决了分块过程的负载均衡执行问题。提供的实验结果证明了该方法的有效性。

关键词:模板计算，分块，自动并行化，负载均衡

## 1.介绍

模板计算是许多科学/工程代码中出现的一类非常重要的计算。涉及模板的计算领域包括那些使用显式时间积分方法计算偏微分方程数值解的领域(例如，气候/天气/海洋建模，使用有限差分域方法的计算电磁学代码)，以及执行平滑和其他基于相邻像素的计算的多媒体/图像处理应用。计算机科学领域已经有一些解决模板计算性能优化的前期工作(例如[24, 19, 18, 10])。由于模板计算的特点是有有一个规则的计算结构，它们可以进行自动编译时分析和转换，以优化并行性利用和数据局部性。然而，正如稍后通过一个例子所阐述的，现有的编译器框架在生成针对并行性和数据局部性进行优化的高效代码方面存在局限性。

模板计算对于求解大型线性方程组是十分重要的，模板（Stencil）计算是数值模拟程序中常见的循环运算模式，其特点是遍历计算区域，每个位置均执行相同的计算操作。个人猜测，可能因为每个位置都执行相同的指令“模板”，所以将这种计算称作模板计算。下面是模板计算的分类：

BroadCast	Reduce	Point-wise
 <p>e.g.</p> <pre>do i=1,N   a(i) = para end do</pre>	 <p>e.g.</p> <pre>do i=1,N   para += a(i) end do</pre>	 <p>e.g.</p> <pre>do i=1,N   b(i) = 2 * a(i) end do</pre>
Jacobi-stencil	Stagger-stencil	Sweep-stencil
 <p>e.g.</p> <pre>do j=1,N   do i = 1,N     b(i,j) = a(i-1, j) + a(i+1, j)              + a(i,j-1) + a(i,j+1)   end do end do</pre>	 <p>e.g.</p> <pre>do j=1,N   do i = 1,N     b(i,j)   += 0.25*a(i,j)     b(i+1,j) += 0.25*a(i,j)     b(i,j+1) += 0.25*a(i,j)     b(i+1,j+1) += 0.25*a(i,j)   end do end do</pre>	 <p>e.g.</p> <pre>do j=1,N   do i = 1,N     a(i,j) = a(i-1, j) + a(i, j-1)   end do end do</pre>

循环分块是实现模板代码并行化和数据局部优化的关键转换。许多关于迭代空间分块的研究已经发表[17, 29, 28, 26, 8, 25, 21, 22, 14, 7, 15, 9, 16, 3]。除了少数例外情况(例如Griebel [11, 12]的工作)，对使用分块的性能优化的研究通常集中在两个互补方面中的一个或另一个:(a)数据局部性优化[2, 3, 28, 26, 8]; 或者(b)并行执行的图块大小/形状优化[25, 21, 6, 14, 7, 15, 9, 16]。用于数据局部性优化的分块涉及数据重用的最大化，即沿着数据依赖向量的方向分块。但是这种分块可能导致分块之间的依赖性，从而抑制不同处理器上分块的并行执行。**据我们所知，以前没有任何工作以集成的方式解决数据局部优化的分块问题和并行执行的负载均衡问题。**我们首先用一维Jacobi代码的简单例子来说明这个问题，并介绍两种避免这个问题的方法：**交叉分块和分裂分块**。作为模板计算的一个例子，让我们考虑图1所示的一维雅可比代码。优化此模板计算以减少缓存未命中需要循环融合和分块；为了融合两个内环，需要环路偏斜。先前已经提出了针对不完全嵌套循环的数据局部性优化的框架。使用艾哈迈德等人[3, 4]提出的方法，通过首先将不完全嵌套循环中的迭代嵌入到完全嵌套的迭代空间中，可以将循环嵌套转换为图2所示的循环嵌套。循环转换和分块然后可以应用于转换后的完美嵌套迭代空间。通过减少/消除控制开销，转换后的迭代空间随后可以转换成高效的代码[20]。**在本文中，我们将重点放在分块迭代空间的负载均衡的并行执行上，这些空间已经使用[4]中开发的技术嵌入到完全嵌套的迭代空间中。**

图3显示了通过向数组A添加一个附加维数而获得的一维Jacobi代码的单语句形式。此代码中的流依赖性与先前显示的版本相同，但对映体反依赖性。因此，如图1和2所示，在循环主体中使用一个语句代替两个语句的序列就足够了，分别用于更新和复制。尽管这种内存效率低的代码在实践中不会使用，但实际上更多方便使用单语句迭代空间来解释本文的主要思想。但是，已开发的方法不限于此类单语句循环，而是适用于一般的多语句模板代码，例如图1中的代码。该方法的通用性是用于内存效率更高的多语句版本的代码。附录中有解释。稍后介绍的实验结果也使用了内存高效的多语句版本。

```

for t = 0 to T-1
  for i = 1 to N-1
    B[i] = (A[i-1]+A[i]+A[i+1])/3; (S1)
  for i = 1 to N-1
    A[i] = B[i]; (S2)

```

---

**Figure 1.** Example: 1-D Jacobi code

```

for t = 0 to T-1
  for i = 1 to N
    if(i>=1 and i<=N-1)
      B[i] = (A[i-1]+A[i]+A[i+1])/3; (S1)
    if(i>=2 and i<=N)
      A[i-1] = B[i-1]; (S2)

```

---

**Figure 2.** Fused 1-D Jacobi code

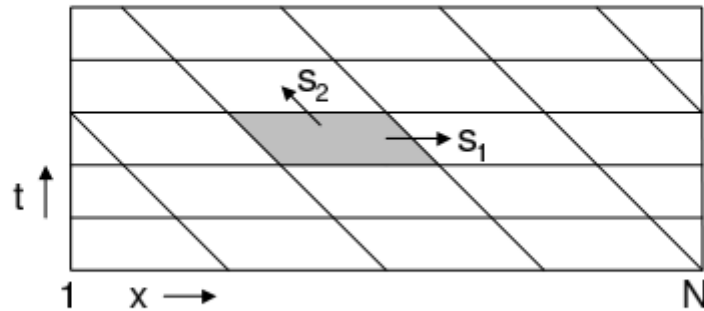
```

for t = 0 to T-1
  for i = 1 to N-1
    A[t,i] =
      (A[t-1,i-1] + A[t-1,i] + A[t-1,i+1])/3;

```

---

**Figure 3.** Single-statement form of 1-D Jacobi code




---

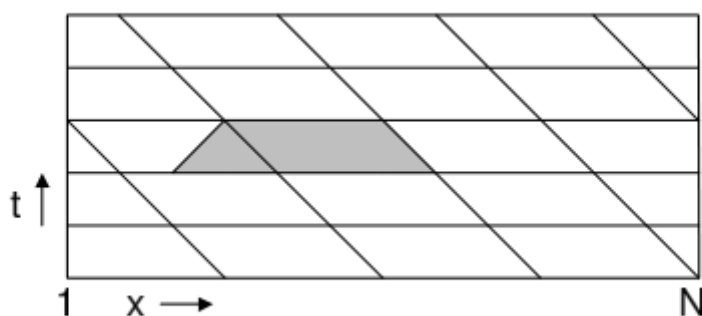
**Figure 4.** Standard tiling for one-dimensional Jacobi.  $s_1$  and  $s_2$  denote the inter-tile dependences.

图3的完美循环嵌套具有恒定的依存关系  $(1,0)$  ,  $(1,1)$  和  $(1,-1)$  。用于数据重用优化的切片（例如，使用[2]中介绍的方法）将产生如图4所示的形状的切片。水平轴对应于空间维度，时间沿垂直维度。沿时间维度使用足够大的切片大小有助于在缓存/寄存器内进行大量的数据重用。然而，在水平方向上存在瓦片间的依赖性，从而抑制了由不同处理器并行执行瓦片。但是，如果将垂直图块的大小减小为一（即，沿时间维消除了平铺），则可以同时执行沿空间维（与x轴相邻）的所有图块。因此，在实现良好的数据重用与并行执行的负载平衡之间需要权衡。

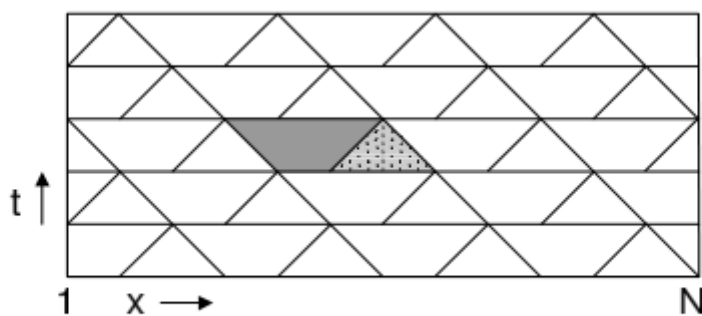
代替上述标准的平铺，请考虑图5中所示的平铺。从由相同超平面形成的平铺开始，在平铺的左侧添加一个额外的三角形区域，与相邻的右端的点重叠瓦。通过这种平铺，平铺处理的迭代点不再分离。一些迭代由两个相邻的图块冗余执行。这导致计算成本增加。但是这样做消除了沿着水平方向的图块之间的依赖性。所有处理器都可以开始并行执行，从而消除了图4中图块的管道并行执行导致的初始处理器空闲状态。

虽然在这种情况下标准切片可以增强数据局部性，但是重叠切片可以改善数据局部性并消除流水线并行性的开销，但以稍微增加计算时间为代价。但是，增加的计算成本与图块大小无关。因此，分数计算开销在重叠平铺的方向上与图块大小成反比，并且如果沿时间维度选择足够大的图块大小，则可以使其微不足道。

另一种方法，如图6所示，将每个图块的内部分为两个子图块，其中两个子图块（阴影图）中只有一个的点取决于相邻图块中的点，而其他子图块不依赖于任何相邻图块中的点，因此可以并行执行。通过这种方法，每个标准图块都分为两个子区块，并且可以按照两个步骤的顺序进行负载均衡的并行执行：首先并行执行所有非依赖子区块并与邻居区块进行通信，然后所有依赖子区块都同时进行被执行。



**Figure 5. Overlapped tiling for 1-D Jacobi.**



**Figure 6. Split tiling for 1-D Jacobi.**

本文的结构如下：第2节定义了本文中解决的问题。在第3节中，我们描述了分块的迭代空间可以从交叉/分裂分块中受益的条件。在第4节中，我们展示了如何变换给定的分块迭代空间，以使交叉/分裂分块适用。第5节讨论代码生成，第6节分析交叉分块的成本和收益。第7节提供的实验结果证明了交叉/分裂分块的好处。在第8节中，我们讨论相关工作，并在第9节中进行总结。

## 2.背景和问题陈述

本节介绍了有关多面计算模型的一些标准背景，并定义要解决的问题。考虑一个具有 $n$ 个嵌套级别的完美循环嵌套。多面体迭代空间定义了一组 $n$ 维点集，其特征是一组有界超平面，并建模为 $B.l \geq b$ ，其中 $l$ 是迭代向量。  $B$ 中的 $b_i$ 行则定义了对应有界超平面的法线。例如，一维雅可比迭代示例的迭代空间为

$$\begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} t \\ i \end{pmatrix} \geq \begin{pmatrix} 0 \\ -T+1 \\ 1 \\ -N+1 \end{pmatrix}$$

在向量分析中，**雅可比矩阵**是函数的一阶偏导数以一定方式排列成的矩阵，其行列式称为**雅可比行列式**。

在代数几何中，代数曲线的**雅可比行列式**表示雅可比簇：伴随该曲线的一个代数群，曲线可以嵌入其中。

超平面是n维欧氏空间中余维度等于一的**线性**子空间，也就是必须是(n-1)维度。这是平面中的直线、空间中的平面之推广（n大于3才被称为“超”平面），是纯粹的数学概念，不是现实的物理概念。因为是子空间，所以超平面一定经过原点。

在**几何体**中，超平面是一维小于其环境空间的**子空间**。如果空间是3维的，那么它的超平面是二维平面，而如果空间是二维的，则其超平面是一维线。该概念可以用于定义子空间维度概念的任何一般空间。

在高中的数学课本，我们应该听过，“点动成线，线动成面，面动成体”。

我们先从低维空间出发，在低维空间中简单理解超平面

在一维空间中，只有一个维度，一维坐标系

$$a_1 x_1 + b = 0$$

在一维空间上确定了一个点

点是一维空间上的超平面

在二维空间上，有两个维度，平面直角坐标系

$$a_1 x_1 + a_2 x_2 + a_3 x_3 + b = 0$$

在二维空间上确定了一条直线

直线是二维空间上的超平面

在三维空间上，有三个维度，三维坐标系

$$a_1 x_1 + a_2 x_2 + a_3 x_3 + b = 0$$

在三维空间上确定了一个平面

平面是三维空间上的超平面

以此类推到n维空间上

作者：思想永不平凡

链接：<https://www.jianshu.com/p/2dadd6f8cdbc>

来源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

雅克比迭代法的计算公式简单，每迭代一次只需计算一次矩阵和向量的乘法，且计算过程中原始矩阵**A**始终不变，比较容易并行计算。其迭代过程为：首先将**方程组**中的**系数矩阵A**分解成三部分，即：**A = L+D+U**，如图1所示，其中**D**为对角阵，**L**为下**三角矩阵**，**U**为上三角矩阵。

对  $Ax=b$

$$A = \begin{bmatrix} 0 & & & & \\ a_{21} & 0 & & & \\ a_{31} & a_{32} & 0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & 0 \end{bmatrix} + \begin{bmatrix} a_{11} & & & & \\ & a_{22} & & & \\ & & \ddots & & \\ & & & a_{nn} & \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ & 0 & a_{23} & \cdots & a_{2n} \\ & & 0 & \cdots & a_{3n} \\ & & & \ddots & \vdots \\ & & & & 0 \end{bmatrix}$$

下三角阵                      对角阵                      上三角阵

$$= L + D + U$$

$\because a_{ii} \neq 0$  (Jacobi假设)  $\therefore D$ 可逆

进一步, 有:  $(L+D+U)x=b$

$$Dx = -(L+U)x + b$$

Baidu 百科

$$x = -D^{-1}(L+U)x + D^{-1}b \rightarrow \text{理论分析}$$

之后确定迭代格式,  $X^{(k+1)} = B^*X^{(k)} + f$ , (这里 $\wedge$ 表示的是上标, 括号内数字即迭代次数), 如图2所示, 其中 $B$ 称为迭代矩阵, 雅克比迭代法中一般记为 $J$ 。(  $k = 0, 1, \dots$  )

J格式:  $X^{(k+1)} = B_J X^{(k)} + f_J$

$$\begin{cases} x_1^{(k+1)} = (-a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \dots - a_{1n}x_n^{(k)} + b_1)/a_{11} \\ x_2^{(k+1)} = (-a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \dots - a_{2n}x_n^{(k)} + b_2)/a_{22} \\ \dots \\ x_n^{(k+1)} = (-a_{n1}x_1^{(k)} - a_{n2}x_2^{(k)} - \dots - a_{n,n-1}x_{n-1}^{(k)} + b_n)/a_{nn} \end{cases}$$

类似可写浓缩式:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \sum_{j=1}^n a_{ij} x_j^{(k)}, i = 1, 2, \dots, n$$

Baidu 百科

再选取初始迭代向量 $X^{(0)}$ , 开始逐次迭代。

计算中的相关性可以用一个矩阵 $D$ 表示, 其中每列定义一个依赖向量。一维雅可比迭代示例中的依赖向量为

$$D = \begin{pmatrix} d_1 & d_2 & d_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

假设我们得到了一组分块迭代空间的分块超平面。这些超平面由矩阵 $H$ 编码, 其中每一行代表分块超平面的法向量。例如, 对应于图4的平铺超平面被编码为

$$H = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$



如果每个分块可以原子执行，并且存在有效的分块总顺序，则由一组分块超平面定义的分块是合法的。从直观上讲，如果两个分块没有相互影响，则分块就是合法的。这个有效性条件可以由 $H.D \geq 0$ 证明。

如果所有处理器都可以并行地开始执行而没有流水线启动开销，那么这个调度将具有并行启动的能力。这样的调度被称为并行启动调度。

问题陈述：在本文中，我们对以下问题感兴趣：考虑给定的（非分块）迭代空间，在该空间中可以并行启动调度。然而，对于由一组分块超平面定义的该空间的给定分块，相应分块的迭代空间中的分块依赖性可能会阻止并行启动。我们考虑以下问题：如何在分块的迭代空间中实现并行启动？我们的第一个目标是分析描绘出阻止并行启动的情况的特征。接下来，我们定义两种方法，即交叉分块和分裂分块，这两种方法可以在分块空间中并行启动并恢复由于分块而丢失的负载均衡的特性。

### 3.禁止并行启动

如果原始非分块迭代空间不可以并行启动调度，那么它的分块不能并行启动。但是，如果在没有分块的情况下可以同时并行启动，则分块的引入可能会阻止并行启动。此部分描述了非分块空间支持并行启动调度的条件，然后推导了分块空间的并发启动禁止条件。为了简化表示，本讨论假定使用单个语句的迭代空间，但是我们为多语句迭代空间定义了该技术的通用版本（在附录中概述）。

#### 3.1非分块空间中的并行启动

首先，我们描述了原始非分块迭代空间中并发启动存在的条件。考虑下面的例子，依赖向量是 $(1,0)$ 和 $(0,1)$ 。具有这些依赖性的两个迭代空间如图7所示。在图7(a)中，并行计算必须从原点 $(0,0)$ 开始，并且存在流水线启动开销。另一方面，所有处理器可以从灰色边界开始并行遍历图7(b)中的迭代空间。

通常，迭代空间能否并行启动取决于迭代空间生成的多面体的边界。如果迭代空间存在一个不包含依赖项的有限超平面，即迭代空间中包含所有依赖项，则该迭代空间支持并行启动。如果依赖关系的源和目标迭代点都包含在超平面中，则该超平面包含依赖关系。由于 $B$ 中的 $b_i$ 行定义了边界超平面的法线向量，因此该属性可以由下面的条件表示

$$\exists b_i \in B : \forall d_j \in D : b_i \cdot d_j > 0$$

注意，此条件与分块超平面无关。我们将此属性称为逐点并发开始条件。如果不满足此条件，则非分块的迭代空间不能并行启动。对于一维雅可比迭代的例子，该条件成立是因为存在一个有限超平面的法向量 $b_1 = (1 \ 0)$ 对于所有依赖矢量 $d_j$ 都满足 $b_1 \cdot d_j > 0$ 。

#### 3.2分块空间中并发启动被禁止的条件

接下来，我们考虑在分块迭代空间中并行启动被禁止的条件。给定分块超平面及其法线向量 $h_i \in H$ ，我们将具有 $h_i$ 法向量的超平面的位移向量 $s_i$ 定义为连接同一超平面的两个实例的向量，同时与所有的其他超平面可以并行移动。显然，位移向量的集合 $S$ 的定义如下：

$$\forall s_i \in S : \forall j \neq i : h_j \cdot s_i = 0$$

对于一维雅可比的例子，我们将用如图四所示的位移向量。

$$S = \begin{pmatrix} s_1 & s_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}$$

如果存在一个依赖向量 $d_j$ ，使得对于与 $d_j$ 相关的一些迭代点 $i_1$ 和 $i_2$ ，点 $i_1$ 在其中一个块中，点 $i_2$ 在另一个块中，那么两个相邻块的执行应该有序。请注意，只有当存在一个依赖项通过分离成两个分块的超平面的时候，换句话说，如果满足以下条件，才有可能

$$\exists d_k \in D : h_i \cdot d_k \neq 0$$

当给定的 $h_i \in H$ 超平面满足此条件时，沿着该维度的位移向量 $s_i$ 带有块间依赖关系。对于一维雅可比的例子， $s_1$ 和 $s_2$ 都携带块间依赖关系（例如， $h_1 \cdot d_1 > 0$ 和 $h_2 \cdot d_1 > 0$ ）。

**块间依赖关系会引入原始迭代空间中不存在的依赖关系方向。**如果在某个边界 $b_i$ 上并行启动的条件在原始迭代空间中是满足的，而在分块迭代空间中的块内依赖关系中是不满足的，则并行启动在分块的迭代空间中是被禁止的。如果出现以下情况，分块则不能并行启动

$$\exists b_i \in B, h_j \in H, d_k \in D : b_i \cdot D > 0 \wedge b_i \cdot s_j = 0 \wedge h_j \cdot d_k \neq 0$$

当上述条件为真时，在平行于边界 $b_i$ 的超平面内存在依赖关系，从而排除了边界中所有分块的并行执行。因此，即使原始迭代空间支持并行启动，分块也无法这样做。对于1-D Jacobi示例发生这种情况是由于边界平面法线 $b_1 = (1 \ 0)$ ，分块超平面法线 $h_1 = (1 \ 0)$ 以及任何依赖 $d_k$ ， $k = 1 \dots 3$ 。

#### 4.交叉分块

交叉分块的基本思想是通过“复制”原始迭代空间中的点来消除某些分块间的依赖关系。结果，相同的迭代点可以是两个相邻分块的成员（即，分块可以重叠）。本节概述了一个具有建设性的过程，该过程可以确定具体的不含有块间依赖关系的分块，从而消除对并行启动的抑制。关键步骤是构建同伴（协同）超平面，该超平面消除了沿所需方向的依赖性。沿着消除依赖关系的方向，新分块将没有任何依赖关系。

在标准平铺中，具有法向矢量 $h_i$ 的超平面定义了平铺的两个面。我们将这些面表示为 $h_i(l)$ （背面）和 $h_i(l+1)$ （正面）。沿着与移动矢量 $s_i$ 定义的移动方向，与后续图块共享正面。如果 $h_i \cdot D \geq 0$ ，则背面 $h_i(l)$ 没有传入相关性。另一方面，根据平铺有效性条件，正面 $h_i(l+1)$ 没有传入相关性。如果瓦片的背面被具有法向矢量 $h_1$ 的重叠超平面代替，则可以消除超平面之间的所有依赖性。等

$$\forall d_j \in D : h'_i \cdot d_j \leq 0$$

请注意，超平面跨越了迭代空间和迭代空间中的任何矢量。因此，伴随超平面可以被定义为现有超平面的线性组合。缩放给定的超平面向量 $h_i$ 并不能消除任何其他依赖性。另外，我们对形成瓦片背面的伴随超平面感兴趣。因此，它是通过在其他超平面上“向后”移动来构造的，由超平面的负线性组合表示，并由下式给出：

$$h_i \cdot D \geq 0 \Rightarrow h'_i = h_i - \sum_{j \neq i} k_j \cdot h_j \wedge h'_i \cdot D \leq 0 \wedge k_j > 0$$

这样的伴随超平面消除了沿位移矢量的依赖性。对于每个禁止同时启动的超平面/移位矢量重复此过程。

##### 4.1重叠切片的成本分析

考虑具有 $n+1$ 维迭代空间的 $n$ 维雅可比迭代，以及沿每个维长度为 $N$ 的 $n$ 维数据空间。令 $B$ 为沿着 $n$ 维空间的每个空间分块大小。令 $p$ 为在 $n$ 维网格中组织的处理器数量。  $B = N / n \sqrt[p]{n}$ 。  $t$ 是时间片大小。



**重叠切片的调度要求处理器循环以保持负载均衡。我们用一个简单的变化说明了通信频率的确定。**从正交拼贴开始，两个平面都可以部分旋转以形成用于1-D Jacobi的梯形瓷砖和用于2-D Jacobi的方形金字塔，其顶视图如图8所示。这种重叠的拼贴方案具有相同的功能。通讯量是原来的一倍，但创业公司的数量却翻了一番。但是，由于不需要循环，因此对于这种情况，代码生成更为简单。通信量越高，启动次数越多；对于空间瓦片大小进入体积的高维Jacobi（大于1）尤其如此。

考虑从正交切片获得的重叠切片方案。给定处理器与其在处理器空间中任何相邻处理器的坐标之间的逐点差异是一个n向量，其n个分量中的每个分量均为1、0或-1。撇开全零的情况，我们有 $3^n - 1$ 个邻居。因此，每个图块（无转发）的通信启动次数由下式给出：

$$S_1 = 3^n - 1$$

例如，对于3-D Jacobi，我们有8个角，12个边缘和6个面，即总共26个（=  $3^3 - 1$ ）邻居向/从/从其接收数据以计算重叠的图块。

通过通信转发，每个磁贴的通信启动次数可以减少到 $2n$ （每个面一个）。

$$S'_1 = 2n$$

同样，不进行转发和进行转发的原始计划的启动次数为：

$$S_2 = 2^n - 1$$

$$S'_2 = n$$

假设正交拼贴的确切通信量由下式给出：

$$\begin{aligned} V &= \sum_{i=1}^n \binom{n}{n-i} 2^i B^{(n-i)} f(i, t) \\ &\approx 2ntB^{n-1} \text{ when } t \ll B \\ f(k, t) &= \sum_{i_{n-k+1}=1}^{t-1} \dots \sum_{i_n=1}^{i_{(n-1)}} i_n \end{aligned} \quad (6)$$

原始调度的通信量减少为：

$$\begin{aligned} V &= \sum_{i=1}^n \binom{n}{n-i} B^{(n-i)} f(i, 2t) \\ &\approx 2ntB^{n-1} \text{ when } t \ll B \end{aligned} \quad (7)$$

对于更高的维度，通信时间表和正在通信的数据可能会非常复杂。在通信量中添加少量点可大大简化代码生成。在图8中，四个角中的每个角都是可以添加的点。这样，总通信量将变为：

$$\begin{aligned}
V' &= (B + 2t)^n - B^n \\
&= {}^nC_1 B^{n-1} (2t) + {}^nC_2 B^{n-2} (2t)^2 \\
&\quad + \dots + (2t)^n
\end{aligned} \tag{8}$$

$$\begin{aligned}
&\approx 2ntB^{n-1} \text{ if } t \ll B \\
&= \Theta(tB^{n-1})
\end{aligned} \tag{9}$$

For  $n = 2$ :

$$V' = 4tB + 4t^2 \tag{10}$$

## 4.2分割分块

重叠的切片通过冗余计算切片的各个部分，消除了拼贴间的依赖性。在消除依赖性的同时，这种方法增加了总体计算量。在本节中，我们将利用依赖抑制的思想来开发一种称为拆分平铺的替代方法，以实现并发启动而无需计算开销。在分割分块中，不是沿一个维度冗余地计算前面分块的一部分，而是执行前面分块的处理器首先计算该部分，并将结果沿该维度发送到其后继。

我们表明，对于模板计算，可以识别出一个分块子区域，以便可以在所有分块中并行执行该子区域。这将启用并发启动。我们概述了一种算法，该算法将一个分块划分为多个子区域，并调度计算和通信以实现并发启动和负载均衡执行，其中所有处理器在调度的所有步骤中执行相同数量的工作。

### 4.2.1分块区域

模板计算中的图块以超平面实例为边界：

$$\forall I, B.I \geq b, h_j \in H : h_j.I \geq lo_j, h_j.I \leq hi_j$$

其中定义了每个超平面的两个并行实例，一个实例沿该维度在下方限制图块，而另一个实例从上方限制于图块。

沿着维度 $j$ ，依赖性抑制可识别伙伴超平面，从而可以独立于图块的其余部分计算由伙伴超平面 ( $h? j$ ) 沿正方向 ( $h? j \geq lo? i$ ) 包围的区域。在重叠切片法中冗余计算。

定义1.沿维度 $j$ 的独立区域由 $\neg j$ 表示。沿着该区域的图块的其余部分将用 $j$ 表示。

在随后的讨论中，从上下文中应该清楚， $j$ 是指维还是沿该维的独立区域的补码。通过使伙伴超平面沿该维度从下方限制边界来定义区域 $j$ ：

$$\forall I, B.I \geq b, h_k \in H, k \neq j : h_k.I \geq lo_k, h_k.I \leq hi_k$$

$$\forall I, B.I \geq b : h'_j.I \geq lo'_k, h_j.I \leq hi_j$$

注意，沿所有其他维度的超平面保持不变。

可以沿每个尺寸将图块划分为这两个区域。这些区域的各种相交将图块分为 $2^k$ 个分量，用于 $k$ 个这样的尺寸。我们仅考虑可能存在依赖性抑制的维度，这将消除时间维度。例如，以 $x$ 和 $y$ 为维度的二维雅可比矩阵中的图块可以划分为分量 $x \ y$ ,  $y \ y$ ,  $y \ y$ 和 $x \ y$ 。

根据独立区域的定义，图块组件 $i$ ...并不取决于其沿尺寸 $i$ 的前身。因此，可以并行计算作为独立切片区域沿所有处理器的交集的切片组件，而无需任何通信-也就是说，所有处理器可以并行开始执行此操作，从而导致并发启动。

考虑图块分量 $i$  independent ..., 其中所有其他图块区域都是独立的。除 $i$ 之外, 此图块组件在任何维度上均没有任何依赖性。在此, 它所依赖的前片中的gion的推导是, 沿所有其他维度具有与面片组件相同的超平面的面片组件, 并且沿尺寸 $i$ 的超面被该面片的下界超平面所代替。边界超平面, 用于依赖性抑制的伙伴超平面成为较低边界的超平面。这就是瓦片分量 $i \cap \dots$ 。因此, 一旦沿前 $i$ 的 $i \cap \dots$ 计算出的沿 $i$ 的边界, 就可以计算出瓦片分量 $i \cap \dots$ 。

通常, 对于瓦片分量所依赖的每个维度 $i$ , 由通过将 $i$ 替换为 $i$ 而获得的前一瓦片中的瓦片分量来计算瓦片间边界。例如, 二维的瓦片分量 $x \ y$ 。可以在沿着 $x$ 从前任接收到带有 $\cap xy$ 的共享边界, 并且沿着 $y$ 从前任接收到带有 $x \cap y$ 的共享边界之后, 可以计算Jacobi。

1. If  $(n==1)$ , say a dimension  $x$ . Compute  $\neg x$ , send and receive the result along the  $x$  dimension, compute  $x$  and return.
2. Execute algorithm for  $(n-1)$ -dimensional stencil computation for all dimensions except one, say  $z$ . Thus all values computed will be for those independent along  $z$  (all tile sections have  $\neg z$  as the  $z$  dimension component).
3. Send all computed values along the  $z$  dimension.
4. Execute algorithm for  $n$ -dimensional stencil computation for all dimensions except  $z$ . But this time, all values computed will be dependent for dependent regions along  $z$ .

---

**Figure 9.** Computation/communication scheduling algorithm for split-tiling

图9分割分块的计算/通信调度算法

图9展示了一种用于 $n$ 维模板计算的具有 $2n-1$ 个通信步骤的调度算法。在此递归公式中, 通信步骤数由下式给出:

$$L(n) = 2 * L(n - 1) + 1$$

其中 $L(1) = 1$ ;即,  $L(n) = 2n-1$ 。注意, 该方法不会产生任何额外的计算成本。另外, 仅传达空间维度中的小块间边界, 因此产生与标准平铺相同的通信量成本。

## 5.代码生成

在本节中, 我们将讨论具有重叠和拆分图块的迭代空间代码的生成。我们描述了利用Ancourt和Irigoin [5]描述的代码生成框架所必需的参数的推导。

平铺的迭代空间中的每个平铺都由平铺原点标识。切片迭代空间的执行定义为切片的起源遍历遍历, 以及遍历映射到每个切片的迭代的执行。

平铺迭代空间的原点定义为原始迭代空间的原点。给定原点，可以将所有图块的原点列举为位移向量的线性组合。瓦片大小定义为沿着移位矢量的瓦片原点之间的距离，并且嵌入在移位矢量本身的规范中。

移位矢量矩阵指定图块原点的遍历顺序。移位向量被排序为沿着存在内部同步性-外部同步性的方向进行外循环。

给定图块原点 $x_0$ （根据位移矢量等效地定义或作为原始迭代空间中的迭代点定义），可以通过其中的一个点来标识界定图块的每个超平面。对于不需要识别出任何重叠的超平面 $h_{ialong}$ ，组成该图块的迭代空间中的迭代点 $x$ 满足以下条件：

$$h_i.x \geq h_i.x_0 \wedge h_i.x < h_i.(x_0 + s_i)$$

请注意， $x_0$ 是构成图块背面的所有非重叠超平面上上的一个顶点。 $x_0 +$ 表示所有超平面 $h_i$ 的图块正面上的一个点。由于重叠不会改变正面，因此对于利用重叠的超平面也是如此。

当沿着一个维度识别出一个重叠的超平面时，我们用重叠的超平面 $h?$  替换原始超平面的背面。一世。从 $h$ 开始？由 $h_{iby}$ 构造的 $iis$ 仅沿其他超平面（点 $x_0 + ?$ ）移动。不管 $h$ 的选择如何， $j? = isi$ 是一个有效点。一世。因此，这些超平面的图块的边界条件由下式给出：

$$h_i.x \geq h_i.(x_0 + \sum_{j \neq i} s_j) \wedge h_i.x < h_i.(x_0 + s_i)$$

给定图块的起源和遍历以及重叠图块的形状，Ancourt和Irigoin [5]的代码生成过程可用于生成代码。生成的代码将具有 $n$ 个外部图块空间循环（每个循环对应于平铺超平面），以及内部循环枚举属于图块的所有迭代。让我们假设已经识别出 $n$ 个超平面中的 $k$ 个用于重叠平铺。重叠的平铺通过消除沿着该超平面的所有块间依赖关系，可以沿着超平面并发启动。因此，与其余 $n-k$ 个超平面相对应的图块空间循环带有所有块间相关性，并且可以作为外部循环顺序运行，并且与重叠的平铺超平面相对应的 $k$ 个图块空间循环都可以通过映射并行运行到 $k$ 维或更小的处理器空间。

分割切片的切片原点的遍历与标准切片的遍历相同。如前所述，通过扫描通过指定绑定图块组件的适当超平面实例而得出的多面体，为各个图块组件生成块内代码。如前所述，子区块之间的适当超平面边界定义了要在处理器之间进行通信阶段通信的数据。

## 6.实验评估

两种拟定的分块方案（交叉分块和分割分块）都使模板代码负载均衡地分块执行，且能够内在地满足并行启动标准。两种方案的并行程度是相同的。它们相对于标准分块（流水线分块）的计算/通信开销有所不同。使用交叉分块时，将有少量的计算开销，并且总通信量也会有少量增加。分割分块不需要额外的计算，并且总通信量也与标准切片完全相同，但是需要其他消息，即会产生更高的消息启动成本开销。

下面，我们展示了将一维雅可比代码的交叉/分割分块与标准（流水线）分块进行比较的实验结果。实验是在由32个计算节点组成的集群上进行的，每个集群都是运行Linux内核2.6.9的，处理器由两个主频为2.8 GHz皓龙254（单核）处理器组成，具有4GB 内存和1MB L2缓存。我们在实验中每个节点使用一个处理器。该代码使用带有-O3优化标志的Intel C语言编译器进行编译。

一维雅可比的迭代空间具有空间维度和时间维度。本文实现了两种版本的流水线调度：（i）一种是处理器空间沿时间维度映射、时间沿空间维度映射；（ii）另一种版本是分布式的处理器以块循环方式沿着时间维度执行分块。

首先，我们进行了实验，以确定两个流水线计划的最佳的时间片大小和空间片大小。本文在32个处理器上进行了1000个时间步长，总问题大小为64000个元素的条件下进行了实验。执行时间如图10和11所示。通信启动的次数随着空间分块的大小的增加而减少。这通常会导致执行时间的减少，同时空间分块的大小也会增加。但是，对于较大的空间分块，流水线启动成本会增加，从而占据主导地位并增加执行时间。时间片大小的增加会减少时间片的数量，从而减少同步的数量。但是，较大的时间片大小（和较

大的空间片大小)会增加流水线启动成本。因此,时间片大小的增加会减少执行时间,直到流水线启动成本开始占据主导地位。从实验中推断出,两个流水线调度的执行时间对于时间片大小16和空间片大小1000来说是最小的。因此,时间片大小16和空间片大小1000用于后续的评估该计划。

对于重叠和拆分切片,空间切片大小固定为 $N / nproc$ ,其中 $N$ 是空间尺寸大小, $nproc$ 是用于并行执行的处理器数。时间片大小选择为16,以匹配流水线方案。

给定空间和时间片大小的这些后,图12显示了针对各种问题大小的四种方案的性能。与流水线分块解决方案不同,交叉分块和分割分块方案导致执行时间随问题大小线性增加。通过分割和交叉的分块方案,随着问题大小的增加,执行时间的改善是由于对数据局部性的更好利用。此外,与流水线调度不同,通信成本与问题大小无关。

由于没有流水线启动成本,交叉分块和分割分块方案的可扩展性得到了改善,如图13所示。问题的大小固定为每个处理器20000个元素。改变处理器的数量以测量各种方案的扩展能力。平行于x轴的直线对应于线性扩展。分割分块的解决方案性能最佳,其次是交叉分块的解决方案。随着处理器数量的增加,流水线调度的性能会下降。

## 6.1多语句模板

现在,我们考虑典型的多媒体应用程序的多语句模板代码。该代码是一系列的循环嵌套,相邻循环之间具有生产者-消费者关系,如图14所示。“并行”的实现利用了每个循环嵌套中的全部并行性,即在每个循环嵌套之后进行同步。将利用流水线实现有限语句并行的解决方案限制在五个处理器上。图14和15显示了针对此代码使用重叠和拆分切片所测得的性能。从图15可以看出,分割和重叠切片的性能要比直接并行实现更好。由于利用了数据局部性并允许并发启动,因此重叠和拆分切片的加速是超线性的。

## 7.相关工作

最近的一些工作提出了关于模板计算的手动优化和实验研究[19, 18, 10]。迭代空间平铺[17, 29]是一种将多个循环迭代聚合到切片中的方法,其中切片是原子执行的;与其他处理器的通信(或同步)发生在图块之前或之后,而不是在图块迭代的执行期间发生。几项工作已使用切片来利用数据局部性[2、3、28、26、8]。其他人已经解决了瓦片形状和大小的选择,以最大程度地减少整体执行时间[25、21、6、22、14、7]。切片的大小会影响并行性和通信量:较小的切片会通过降低流水线启动成本来提高并行度,而较大的切片会减少处理器之间的通信频率。许多研究人员对此进行了研究[6、22、14、15、9、16]。Griebel [11, 12]提出了一个集成框架,用于优化平铺使用中的数据局部性和并行性。但是,不考虑流水线问题。

Sawdey和O'Keefe [24]描述了TOPAZ这个工具,该工具在SPMD执行模版代码的过程中探索边界值的重复计算,其中用户标记了要复制的代码区域。然后,该工具会分析并生成正确的代码。这种方法有助于降低通信成本并改善负载平衡。Adve等。[1]描述了dHPF编译器中使用的计算分区策略,该策略使用dHPF中可用的LOCALIZE指令来利用复制计算。与我们的自动并行化方法不同,这两种方法都依赖于复制计算的用户规范。

## 8.结论

通过优化数据局部性以及利用嵌套循环的并行性,迭代空间平铺已引起了相当大的关注。迭代空间块的形状选择可能会导致块间依赖性,从而抑制块在不同处理器上的并发执行,从而导致流水线启动开销。本文已经解决了通过具有恒定依赖性的循环计算的平铺执行来增强并发性的问题。提出了两种方法,即重叠切片和拆分切片,这些方法可以消除碎片间的依赖性,从而实现额外的并发性。实验结果证明了所提方案的有效性。

致谢这项工作得到了美国国家科学基金会的部分支持,获得的奖项为0121676、0121706、0403342、0508245、0509442和0509467。我们感谢David Callahan建议使用分块拼贴技术。

## 附录：多语句迭代空间的处理

通过重叠/拆分切片来增强并发可行性的特征可直接应用于单语句迭代空间，例如图2的Jacobi code的简化（但在空间上无效）版本，但是Jacobi代码的有效版本包含两个不同的语句。在本附录中，我们讨论了重叠/分割切片如何与多语句迭代空间一起使用。

考虑图2的Jacobi代码。两个语句S1和S2嵌套在t和i循环内。如果我们将嵌套循环的整个主体（即S1和S2）作为定义依赖性的基础，则数据依赖性为  $(0,1)$ ， $(0,2)$ ， $(1,0)$ ， $(1,-1)$  和  $(1,-2)$ 。如前所述，对应于  $t = 0$  的边界超平面b，即法向量  $(1,0)$ ，我们发现依赖向量  $(0,1)$  和  $(0,2)$  与b的点积为零。换句话说，不满足逐点并发开始条件。这里的问题是由于在基于整个循环体定义依赖关系时使用了粗粒度。取而代之的是，可以采取更细粒度的视图，分离出由于S1和S2实例引起的依赖性。从S1到S2有流量相关性  $(0,1)$ ，即从S1  $(t, i)$  到S2  $(t, i + 1)$  有流量相关性，而反相关性有  $(0,0)$ ， $(0,-1)$  和  $(0,2)$  从S1到S2。从S2到S1，我们具有流量相关性  $(1,0)$ ， $(1,-1)$  和  $(1,-2)$ ，以及反相关性  $(1,-1)$ 。通过检查S1和S2实例之间的依赖关系（而不是在每个迭代空间点从S1和S2进行汇总计算）可以清楚地看出，对于特定值t的S1的所有实例都可以同时执行：没有直接依赖关系在它们之间，所有传入的依赖关系都来自时间步t-1处的S2实例。在给定时间步长的S2实例也可以同时执行，因为传入的依赖关系全部来自同一时间步长的S1实例。

给定具有语句S1, S2, ..., Sk的模板计算的多语句迭代空间，我们首先在语句之间形成强连接的组件。对于每个强连接的组件，所有自传递依赖关系均从某条语句开始计算，形成所有可能的依赖关系链，这些链终止于同一条语句的实例中。这些自传递的依赖关系然后用于检查并发启动，而不是第4节中假定的单语句依赖关系。此技术允许通常用于重构完美嵌套循环的基于超平面的方法可以被使用。在这种情况下应用。

考虑图2的Jacobi示例，可以从S1到S2返回S1，反之亦然（S2到S1到S2）计算自传递相关性。从S1到S2的依存关系是  $(0,0)$ ， $(0,1)$  和  $(0,2)$ ，而S2-S1的依存关系是  $(1,0)$ ， $(1,-1)$  和  $(1,-2)$ 。形成从S1到S1的所有可能的传递依赖性，我们得到  $(1,2)$ ， $(1,-1)$ ， $(1,0)$ ， $(1,1)$  和  $(1,2)$ 。利用这些自传递依赖性，可以看出它们的点积与  $t = 0$  边界超平面（法线  $(1,0)$ ）始终为正，即，对于此迭代空间而言，逐点并发启动是可行的。