DEPENDENCE ANALYSIS FOR SUPERCOMPUTING

超级计算中的依赖关系分析

【美】UTPAL BANERJEE 著

沈志宇　赵克佳译

湖南科学技术出版社

超级计算
中的依赖关系分析

〔美〕UTPAL BANERJEE 著
沈志宇　赵克佳　译

湖南科学技术出版社

超级计算中的依赖关系分析

沈志宇　赵克佳译

责任编辑：夏可军

*

湖南科学技术出版社出版发行
（长沙市展览馆路8号）
湖南省新华书店经销　湖南省新华印刷二厂印刷

*

1991年1月第1版第1次印刷
开本　787×1092毫米　1/32　印张：4.75　字数：102,000
印数：1—1,300

ISBN 7—5357—0796—3
O·85　定价：1.90元

地科 60—026

中译本序

巨型计算机在不同时期有不同的含义，但都是指当时速度最快、容量最大的一类计算机。就现代巨型机而言，1976年美国克雷研究公司研制的Cray－1可以说是一个新起点。十五年来巨型机发展迅速，今天的最高水平的巨型机，如Cray－3和日本NEC的SX－3，不论在速度和容量方面，比十五年前，均提高150倍以上。据宣布，Cray－3的速度达160亿次（峰值），主存5亿字；SX－3的速度达200亿次（峰值），主存10亿字节。现代巨型机在体系结构上大都采用由含多组流水线的单机复合而成的多机系统。这种机器的速度主要来自多组流水线和多台单机的同时并行工作。因此，发挥这种机器的效率的关键在于充分利用其各种可能的并行性，但这恰恰是个很难的卡脖子问题。这个问题可以说是现今巨型机应用的瓶颈。并行性差使得巨型机有时连1/10的速度（峰值）也达不到。

从使用巨型机的角度来看，最重要的是对各类问题进行并行算法的研究。但是，目前已有大量现成的大型传统串行程序，希望能在巨型机上运行并获得所期望的高效。另外，许多程序员不习惯于并行化程序设计，宁愿沿用传统的设计方式。于是，“自动并行化”便成为软件领域的一个重要研究课题。

串行程序并行化包括同步并行和异步并行两方面。粗略地讲，同步并行（也即向量化）旨在发挥流水线的效率，异步并

行旨在发挥多机协同工作的效率。

自动并行化的工作主要是依照源（串行）程序中的数据依赖关系，进行等价变换，使结果程序含有尽可能大的并行成分。在这一过程中，数据依赖关系的分析是最重要的一环。沈志宇、赵克佳同志翻译的《超级计算中的依赖关系分析》就是专门研究数据依赖关系的分析方法及相关的数学基础的。这本书所讨论的概念、方法及相关的数学基础主要是面向向量化方面的。

FORTRAN几十年来一直是科学/工程计算中最重要的语言，因此，绝大多数的自动并行化工作是针对FORTRAN源程序的。本书讨论了四种主要的依赖关系。数据依赖关系的发掘主要是通过求解数组元素下标式的有关方程组来确定的，由于FORTRAN语言限制下标式只能是简单的线性式a＊I＋b，因此线性丢番图方程的理论便是很好的工具。本书的重要特点之一是：数学基础是自足的。

本书的后一部分讨论了编译时数据依赖关系的两种类型的测试法：精确测试和近似测试。这是两类有效的基本测试法。一般而言，当难于用精确的方法确定一方程组是否有满足给定限制条件的通解时，人们求助于近似解法。书中详细介绍了一些行之有效的重要近似解法。这些构成了本书内容的基本部分。

本书兼具基础性和实用性。它对巨型机、并行算法和巨型机软件系统的研究人员、工程人员和高校师生均有重要参考价值。甚至可用作计算机专业研究生的教材或教学参考资料。

本书作者U. Banerjee是有名的计算机专家，美国控制数据公司（CDC）顾问；译者沈志宇同志前年曾是美国Illinois大学

的访问学者，并在并行化方面作过不少工作，经验丰富。本书译文准确、通顺、易读，不失为一本好书，特向读者推荐。

陈火旺
国防科大研究生院
1990年5月

原著前言

本书论述依赖关系的概念和依赖关系测试的一般方法。这里依赖关系指的是数据依赖关系，而测试指的是编译时进行的测试。我们认为建立关于这一课题的确定的理论，从而给研究界提供一个能使问题的各个方面更好地统一起来的、一致的概念结构的时机已经成熟。当然，我们能在多大程度上成功地达到这一目的还要看情况的发展。我们并不试图在本书中介绍至今为止已知的关于这一课题的所有细节，也不涉及所有优秀的程序员都愿意使用的那些窍门。但我们确实试图说服读者认识到，数据依赖关系分析的数学基础是线性函数边界理论和线性丢番图方程；数据依赖关系分析中的层次和方向向量概念是相当自然地引出的；不同的依赖关系测试方法确实是一些一般测试方法的特例；等等。

为了很好地理解本书，读者应有一定的数学基础，主要是微积分和线性代数的基础。我们相当深入地涉及了丢番图方程，并描述了一些不广为人知的矩阵理论概念。熟悉线性规划的读者将能较快地弄懂本书中的几个概念。

我们从M·Wolfe以及K·Kennedy和R·Allen的研究成果中学到了许多东西。Wolfe在伊利诺依大学的博士论文及Kennedy和Allen关于FORTRAN程序向量化的论文对这个课题仍然是非常有用的。他们在这个领域及其相关领域做了大量的工作，他

们引入了新的概念并完善了原有的概念。对原有的gcd测试的改进和对原有的方向向量不等式测试方法的扩充就是两个例子。

作者感激 David Kuck 教授，是他引导作者研究了这一课题。作者感谢 Dan Gajski 教授所给予的鼓励。David Padua 教授使用这部书的初稿在伊利诺依大学讲授了一门课，并提出了许多有益的意见；作者所在的控制数据公司的前任经理 Erwin Huntley为作者提供了很好的工作条件；Larry Bumgarner，Jack Neuhaus以及 Fred Kunz 审阅了部分书稿；Rich Ragan和Jeff Lanam 在书稿准备工作中给予了很大的帮助，在此作者也要向他们表示感谢。

非常欢迎读者给本书提出各种意见。

Utpal Banerjee

目 录

第一章　引论

任何时期的超级计算机都是在那个时期可供使用的最快的计算机。很多应用领域需要对大量的数据进行大规模的计算，因此它们对超级计算机系统的计算能力有着极大的和不断增强的需求。这些领域包括气象、国防、核能、石油、航空和航天、以及基础科学研究。随着人们认识到由于物理定律的限制，串行计算机的速度已经达到了极限，并行处理时代就开始了。当今所有的超级计算机都采用了某种形式的并行处理技术。并行处理在于开发一个或多个层次（例如作业、程序、子程序、循环、语句或指令层次）的并行性，而它的应用并不仅限于超级计算机。

并行处理通常是由硬件系统采用设置多功能部件、一个或多个向量流水线、或多个互相连接在一起的CPU等方法来支持的。近几年来已经出现了一些支持并行处理的并行语言，但还没有一种被大家广泛地接受。用FORTRAN（或其它串行语言）书写的现有软件包已经花费了巨大的投资。直至今天，多数用于超级计算机的程序仍然是用 FORTRAN 语言编写的（有些 FORTRAN 语言已经扩充了并行结构）。因此有必要使编译程序能够检测和开发串行程序中的并行性。为了发挥超级计算机在系统结构上的优势，超级计算机厂商至少应当提供一个能够自动检测程序中的并行性，并通过重构这个程序来生成并行代

码的FORTRAN编译程序。

编译程序进行并行性测试和程序重构是为了实现向量化或并行化，或既向量化又并行化。向量化意味着以发现向量结构为目标来重构程序。这种结构的一个例子是：

```
A(1:100:1) = B(3:102:1) + C(1:199:2)
```

这个例子包含三个一维数组，等价于下面的由一个语句和100个独立迭代构成的循环：

```
do  I = 1,100
      A(I) = B(I + 2) + C(2 * I - 1)
enddo
```

面向多处理机系统的并行化则是通过识别那些不同迭代可以独立地或以某种方式重叠地在不同处理机上同时执行的循环而实现的。例如，如果适当的同步可以保证处理机间通讯的正确性，则下述循环的各个迭代可以在多个处理机上同时执行。

```
do  I = 1,100
      B(I) = A(I) + C(I)
      A(I + 2) = D(I) * E(I)
enddo
```

假如处理机 P_0 正执行迭代 $I=i$，而处理机 P_2 正执行迭代 $I=i+2$，那么处理机 P_2 必须等待，直至 $A(i+2)$ 的值被 P_0 计算出来并能够为 P_2 所用时为止。

为了实现重构，一个重构编译程序要对程序进行一系列的程序转换。有几种转换现在已经确认是有益的，另一些新的转换则正在研究之中。为适应超级计算机系统结构的发展，今后可能会出现更多的程序转换技术。为了保证正确性，一个给定程序的任何特定转换都必须遵循这个程序中关于存储引用的依赖关系的限制。这些依赖关系是由程序变量定义和使用的方式

规定的。为了保证结果正确，变量的定义和使用顺序必须满足一定的约束条件。一般地说，编译程序所掌握的程序的依赖关系信息越精确和越完整，它在寻求并行性时进行程序转换的能力就越强。依赖关系分析的目的是收集程序中潜在的依赖结构的有用信息，并以适当形式来描述这些信息。

由于许多研究人员的贡献，对这个课题的研究这些年来一直在深入。有关这个课题的主要参考文献是：[Cohagan 1973]，[Lamport 1974]，[Towle 1976]，[Banerjee 1976]，[Banerjee et al. 1979]，[Banerjee 1979]，[Kuhn 1980]，[Wolfe 1982]，[Allen et al. 1983]，[Allen 1983]，[Burke & Cytron 1986]，[Wolfe & Banerjee 1987]，及[Allen & Kennedy 1987]。在本书中，我们试图综合许多已经发表了的研究成果，创建一个严密的依赖关系分析理论，并用有条理的形式来描述它。

上面提到的依赖关系可以看成是各个变量出现、各个语句、各个循环迭代、以及各个语句块之间的关系。在本书中我们把依赖关系作为赋值语句之间的关系来研究。对控制流本书不予考虑。我们所取的程序模型是仅含赋值语句的单个或嵌套FORTRAN DO循环。在本书中变量指的是标量或数组元素。假定不同的名字表示不同的变量。本书所阐述的概念和技术可以应用于更一般的程序和其它程序设计语言。尽管对于理解本书来说不是必需的，但为了理解为什么我们要计算一些有关依赖关系的特殊信息，读者最好能有一些关于程序转换的知识。(见[Kuck et al. 1981]，[Wolfe 1982]，[Padua & Wolfe 1986]，[Allen & Kennedy 1987]，及[Wolfe & Banerjee 1987]。这些论文还给出了进一步的参考文献。)

基本的依赖关系问题是要判明两个下标含循环控制变量的

数组元素在给定条件下是否表示同一个存储单元。当我们考虑以某种方式表示的一些存储区域并研究它们相交的可能性时，我们研究的就是更为一般的依赖关系问题了。循环控制变量都是整变量，因此这些问题简化为在一组不等式的限制下求一组丢番图方程的整数解。如果在编译时不能得到有关这组方程和不等式的完整的信息，编译程序就不可能作出较好的依赖关系分析。即使这组方程和不等式在编译时是完全有定义的，当组中含有非线性函数时仍然不能作出准确的依赖关系分析。幸亏实际用户程序中的数组元数下标表达式和循环的初值和终值通常都是线性函数。因此我们在本书中将把注意力集中于这类线性问题。

在线性情况下依赖关系是否存在可用整数规划来确定。现有的几种不等式消去法都致力于寻找方程和不等式组的实数解，因此它们仅在没有实数解时才是确定的。然而，由于依赖关系问题涉及的方程和不等式组通常较为简单，且在程序转换过程中要频繁地进行依赖关系测试，可以说所有已知方法的开销都太大，因而在编译程序中应用它们是低效的。我们的注意力集中在那些利用依赖关系问题的特殊性质来求解，且已经使用了一段时间的简单而实用的方法上。它们包括在通常情况下能准确地测试出依赖关系是否存在的精确算法，以及已被证明对于实际的程序是相当有效的近似算法。由寻求线性函数边界的一般算法（第四章）和解线性丢番图方程的一般算法（第五章），我们可以系统地推导出上述这些算法（第六章）。基本概念和基本记法则在第二章和第三章给出。

这里我们要说明的是，本书中对某些算法的“证明”可能有些象是“解释”或“示例”。例题是本书的一个很重要的部分。如果某些例题中的程序看起来不实际的话，那只是因为设计它

们的目的就是简洁且能代表典型的情形。

本章的剩余部分由一组例题构成。这些例题可以使读者一瞥后面各章的内容。每个例题给出由循环和赋值语句构成的一段程序。请记住每个循环假定都是串行执行的，且程序中的存储引用次序有一定的限制，这些限制正是我们试图发现和分析的。

例1.1

```
        do  I = 1,100
S:          A(I) = B(I + 2) + 1
T:          B(I) = A(I - 1) - 1
        enddo
```

对于I在{1，2，…,100}中的每一个值i，语句S和T都有一个实例，分别表示为S(i)和T(i)。部分地展开这个循环（见表1.1）将有助于我们确切地观察数组A和B的不同元素是如何被这些实例引用的。在实例S(1)中计算的数组元素A(1)的值在实例T(2)中被使用，S(2)计算的A(2)的值则在T(3)中被使用，等等。一般地说，由S(i)计算的A(i)的值在T(i+1)中被使用，$1 \leqslant i \leqslant 99$。这使得语句T**流依赖**于语句S，记作$S\delta^{f}T$。这个依赖关系是由S中的变量A(I)和T中的变量A(I-1)引起的。这一依赖关系的**相关迭代对**集合是：

$$\{(i, j): j = i + 1,\ 1 \leqslant i \leqslant 99\}$$

依赖距离为常数1，因为对每一个迭代对来说，$j - i = 1$。

```
S(1):  A(1) = B(3) + 1
T(1):  B(1) = A(0) - 1
S(2):  A(2) = B(4) + 1
T(2):  B(2) = A(1) - 1
S(3):  A(3) = B(5) + 1
```

```
T(3):   B(3) = A(2) - 1
S(4):   A(4) = B(6) + 1
T(4):   B(4) = A(3) - 1
 ⋮        ⋮
S(100):  A(100) = B(102) + 1
T(100):  B(100) = A(99) - 1
```

表1.1 展开后的例1.1的循环

S(1)所用的B(3)的值是在这个循环执行之前定义的，而不是由T(3)计算出来的。换句话说，S(1)应当在T(3)改变B(3)之前使用B(3)的值。类似的，S(2)应当在T(4)改变B(4)之前使用B(4)的值，等等。一般地说，S(i)应当在T(i+2)改变B(i+2)之前使用B(i+2)的值，$1 \leqslant i \leqslant 98$。这使得语句T**反依赖**于语句S，记作$S\delta^{a}T$。这个依赖关系是由S中的变量B(I+2)和T中的变量B(I)引起的，其相关迭代对集合是：

$$\{(i, j): j = i + 2,\ 1 \leqslant i \leqslant 98\}$$

这里依赖距离为常数2。

如果我们首先同时地执行语句S的所有迭代，然后同时地执行语句T的所有迭代，则T能使用由S计算的正确的A(1)，A(2)，…，A(99)的值，而S也能在T改变B(3)，B(4)，…，B(102)之前使用它们的值。

例1.2

```
        do  I = 1,100
T:          C(I) = A(3*I+1) + 1
S:          A(2*I+7) = B(I) - 3
        enddo
```

变量A(2I+7)和A(3I+1)可能导致语句T流依赖于语句S，但我们不能一眼看出T是否依赖于S，以及如果依赖，哪些迭代

与这个依赖关系有关，依赖距离又是多少。完全展开这个循环可以得到这些问题的答案，但应用分析的方法来解决这些问题更好。语句T的实例T(j)使用由语句S的实例S(i)计算的值，当且仅当迭代I=j在迭代I=i之后执行且变量A(2i+7)和A(3j+1)表示数组A的同一个元素，即当且仅当i<j，且

$$2i+7=3j+1，或2i-3j=-6。$$

这是一个含两个整变量的线性丢番图方程。因为i，j都是循环控制变量I的值，它们还应当满足

$$1\leqslant i\leqslant 100和1\leqslant j\leqslant 100。$$

如果存在一个满足所有这些条件的整数解（i，j），则语句S中的变量A(2I+7)和语句T中的变量A(3I+1)使得T流依赖于S。上述方程的通解，如果存在的话，将给出所有与这个依赖关系有关的迭代对和依赖距离。（依赖距离不一定总是常数。）

注意变量A(2I+7)和A(3I+1)也可能使得语句S反依赖于语句T。在这种情况下i和j扮演的角色相反。如果我们仍使用上述记法，那么因为现在对变量A(3j+1)的引用将在对变量A(2i+7)的定值之前，所以除了j≤i，而不是i<j外，解这个依赖关系问题的丢番图方程和条件都和前面给出的相同。

例1.3

```
      do  I1=1,100
          do  I2=1,50
S:              A(I1, I2)=A(I1, I2-1)+B(I1, I2)
          enddo
      enddo
```

首先考虑这个两重循环的迭代执行顺序。在下列两种情况下迭代$(I_1, I_2)=(i_1, i_2)$先于迭代$(I_1, I_2)=(j_1, j_2)$执行：

$$i_1<j_1，或i_1=j_1且i_2<j_2$$