博士学位论文

并 行 程 序 设 计 交 互 环 境

Development of an Interactive Parallel Programming Environment

作　　者　　俞一峻

专　　业　　计算机软件

研究方向　　并行处理

导　　师　　朱传琪

复旦大学并行处理研究所

1998 年 **4** 月 **10** 日

前序

在几年来朱传琪教授的悉心指导下，在朱洪教授, 招兆铿教授的帮助下，本人在复旦大学并行处理研究所从事了有关并行处理领域的博士学习和研究工作。在国家自然科学基金，863 研究攀登计划基金和博士点基金的资助下，本人参与完成了基于 AFT 编译技术的并行程序设计交互环境 ParaPIE，并在 ParaPIE 的成功经验基础上，作为主要研究开发人员之一，本人在欧洲联盟合作研究项目 ITDC'94-164 支持下完成了基于 FPT 编译技术的并行程序设计交互环境 PROFPAT。

在并行化编译及并行计算机体系结构的理论基础上，在博士研究期间中本人对并行程序设计交互环境进行了理论研究与开发、应用实践，从而对并行化编译技术有了更深入的认识。此论文为本人在这几年来对并行化编译和并行程序设计交互环境领域的一些问题的理解与总结，其中不免粗浅之处，望能包涵指正。

此外，在学习过程中, 本人还得到了 Erik D'Hollander 教授, 臧斌宇副教授的指导, 张福波博士, 陈彤, 王琦, 施武, 丁永华, 李剑慧等老师以及课题组其他同学的帮助, 没有他们的指导和帮助, 我很难顺利完成博士学习，在此由衷地表示感谢。

Abstract

First, the paper discusses the programming principles of how to exploit the parallelism underlying in sequential programs with the Parallel Programming Interactive Environment, explains the methods in designing the major functions and tools that are integrated in the PROFPAT(the PROgramming environment for the Fortran PArallel Transformer), and seperately demonstrates its application procedures such as interactive dependence analysis, loop parallelization, array privatization and unimodular transformation, etc..

Then the paper introduces the experiment procedures of applying PROFPAT to analyze three difficult SPECfp95 benchmark programs that automatic compilers can not parallelize, indicates the key obstacles and their solutions, and presents the effective results of the PROFPAT's application. The experiment shows the effectiveness of PROFPAT in analyzing the programs' parallelism and improving their performance by exploiting the parallelism and data locality.

The paper introduces several general practical new techniques, such as the analyzers of program profiles, maximum potential parallelism, array privatizability, and loop parallelism; the visualizers of the statement-level data dependence graph, procedural call-graph, loop iteration space data dependence graph; and interactive tools of array privatization and unimodular transformation; and summarizes the automatic techniques of the array privatization combined data dependence and coverage, the computation of parallelizing unimodular transforming matrix for multi-nested loops, the enhanced unimodular transformation with array reduction recognition, the non-loop level parallelism revealing technique, the dynamic dataflow analysis and the optimization of cross-loop local cache reuses, etc.

The paper not only treats the principles of these valuable auto or interactive analysis and compiling techniques, but also deduces several automatic algorithms to propel the automatic compiler research. Inversely, the improvement to automatic parallelization also becomes a driving force to the deeper study of interactive parallelization. Finally, the paper also throws lights on some of the possible objectives of the future Parallel Programming Interactive Environment.

Primary Keywords: Parallel Programming Interactive Environment, Parallelizing Compiler, Program Analysis, Program Transformation, Code Generation

Secondary Keywords: Intermediate Representation, Dependence Analysis, Dynamic Dataflow Analysis, Program Visualizing Interactive Analysis, Array Privatization, Automatic Parallelization, Unimodular Loop Transformation, Cache Reuse Optimization

摘要

首先，本文从理论上深入探讨了如何借助并行程序设计交互环境帮助程序员分析和利用潜藏于串行程序中的并行性的原理，详细介绍了并行程序设计环境 PROFPAT 中集成的主要功能和主要工具的设计实现方法，并结合交互相关性分析、交互循环并行化变换、交互数组私有化变换、交互幺模变换等程序并行化分析、变换的初步应用实践，说明了这些工具和功能的应用方法。

继而，本文介绍了如何用 PROFPAT 分析三个难以自动并行化的 SPEC95 测试程序 WAVE5, FPPPP 和 APSI 的实验过程和实验结果，指出了影响它们自动并行化的症结所在，提出了克服这些障碍的途径，取得了较好的并行化效果。这进一步说明了并行程序设计环境在分析利用程序并行性，提高计算性能方面的实际作用，证实了并行程序设计环境的有效性。

本文从而概括了若干具有一般性的实用新技术，如：程序计算量分析、最大潜在并行性分析、数组私有化分析、循环并行性分析等动态程序分析技术；语句数据相关图、过程调用图、循环迭代空间相关图、交互数组私有化、交互幺模变换等可视化语义交互分析变换技术等等；总结了若干自动并行化新技术，如：结合相关和覆盖的数组私有化方法、对多层嵌套循环计算幺模并行化变换矩阵的循环幺模变换方法、结合数组规约识别增强幺模变换的技术、发掘非循环级并行性的技术、动态数据流分析技术、优化利用循环间 Cache 数据局部性的技术等等。

本文不仅从理论上总结了这些有实用价值的自动并行化编译和交互并行程序分析新技术的原理，还概括了一些结合应用这些技术的自动算法，直接推动了自动并行化研究的进程；反过来自动并行化技术的提高也促进了交互并行化技术研究的深入开展。本文最后对并行程序设计交互环境的未来前景也作了有益的尝试性探讨。

主关键词 并行程序设计交互环境，并行化编译，程序分析，程序变换，代码生成

次主关键词 中间表示，相关性分析, 动态数据流分析，程序可视化交互分析，数组私有化分析，自动并行化变换，循环幺模变换，cache 优化，

1 序言

高科技关键技术越来越离不开计算。基础科学研究及实际应用领域都有一大批具有重大挑战意义的计算问题已经要求计算机在本世纪内达到每秒万亿次浮点运算(TeraFLOPS=10^{12})甚至更高(每秒千万亿次浮点运算 PetaFLOPS=10^{15})的计算能力**[15]**。这些“巨大挑战性(Grand Chanlleging)”的计算问题包括全球天气预报；分子及原子核结构，大气污染，燃料及燃烧机理，计算生物化学，新兴材料，医学解剖与模拟，新药研制，催化剂设计，天文数据图像处理，石油勘探开发以及设计国家安全的重大问题**[15][19]**。从计算模型数值模拟(如数值天气预报、核反应模型计算及核爆炸模拟试验等)，工程分析与设计 (如各种偏微分方程有限元分析、空气动力学计算)，人工智能与信息学模式识别(如博奕决策树搜索、密码破译、图像处理等)，到大规模数据处理(如能源地震波勘测、卫星遥感数据处理)等等诸多方面，都对超大规模计算(supercomputing)的提出了日益增长的需求，不断呼唤着具有更高性能计算机的问世。

经过 30 多年来的研究与发展，人们已经认识到而要提高计算机的性能，除了需要继续提高单机速度之外，更重要的还是需要在现有元件技术条件下，在并行计算机体系结构、并行程序设计语言及并行程序设计工具等方面给予充分的支持。归根到底必须在具体应用中有机地接合上述这些方面，才能真正充分体现出计算机的绝对高性能。在追求计算机高性能的过程中涌现出许许多多值得深入研究的关键技术，在硬件体系结构方面诸如向量计算机，超标量计算机，多级存储层次，共享主存多处理机，分布主存多处理机及大规模并行计算机等等；在软件方面诸如并行操作系统任务调度与同步，并行程序设计语言和并行算法，并行化及优化编译，并行程序设计环境等等。只有从概念上深刻体会这些技术之间的相互联系与相互作用，才可能综合运用，达到切实提高计算性能的最终目标。

并行处理被公认为是提高超大规模计算性能的关键途径，所以它已经成为本世纪末计算机学术界和工业界的关注焦点**[19][38]**。但是同样大家公认的是，并行软件难以开发、维护和移植。由于并行性与人们熟悉的串行编程思路完全不同，因此同一问题的并行算法比串行算法更复杂和易错。并行程序的不确定行为使之难以用传统方法调试和验证，而且并行程序的优化又依赖于目标机器的体系结构。针对开发并行软件的难题常常采用两条途径来解决。一条途径是设计新的并行算法，直接用并行程序设计语言或并行程序库（如 HPF**[32]**、PCF**[41]**、MPI、PVM**[33]**等）加以实现。但是，由于新的并行算法不易设计出来，也由于新的并行程序设计语言很难为习惯串行程序设计的程序员掌握，这就阻碍了程序员开发新的并行软件。而且，多年来已经积累了丰富的串行软件，没有理由忽视眼前的这一宝贵财富。所以，另一条较为理想的途径是通过编写新的或者修改已有的串行程序，使之在不改变语义的前提下尽量并行化和优化。然而，让并行化程序员手工去处理包含成千上万行代码的实际串行程序简直是不可能的，这就是为什么需要自动并行化编译器**[5][13][17][39][63][76][77]** 的原因。通过对串行源程序进行各种分析，自动并行化编译器能利用存在于这些程序中的潜在并行性，把串行程序自动转换成对应的并行形式，有效地提高它们在并行计算机上执行的性能。同时，自动并行化编译器解决了在不同并行计算机之间移植代码的难题，使在并行机上继承已有串行程序成为可能。作为连接并行计算机与应用程序之间的桥梁，并行化技术必然成为并行计算机上不可缺少的系统软件的组成要素。早期并行化系统 Parafrase-2**[47]**, KAP**[37]**, VAST**[23]**, PFC**[6]**, PTRAN**[3]**等和目前技术领先的自动并行化系统 SUIF**[54][60]**, Polaris**[17]**， FPT**[75]**和复旦大学开发的 AFT**[76][77]**等已经作出了一定效果。

尽管对于许多情形自动并行化编译技术都很有效，但是由于自动并行化编译器必须采取保守的处理方法，

即，如果编译器不能证明其并行执行能导致与串行执行相同的结果，它就只能产生串行代码，所以仍然存在一些场合不能运用自动并行化编译技术**[18][28][39][46][53]**。例如影响数据相关性分析精度的符号表达式和非线性下标表达式、过程调用和依赖于输入的复杂控制流、临时变量、递归变量和规约变量识别、下标化的数组等等都成为自动并行化的拦路虎。

要超越这些障碍，程序员的人工干预就十分有用了。由于自动并行化编译器对绝大多数较简单的情况自动利用并行性，节省了程序员大量时间，程序员反过来只要集中注意那些更复杂的情况来进一步提高目标程序的性能。

为了能结合自动并行化编译技术和用户交互提高并行化能力，就需要开展对并行程序设计交互环境的研究，如 PTOOL**[4]**, Rn**[22]**, ParaScope**[10][23]**，Faust**[35]**, SUPERB**[78]**, PAT**[8]**，ParaDyn，与 Gent 大学合作开发的 PEFPT**[58][59][71]**和复旦大学开发的 ParaPIE**[66][68]**等等。

并行程序设计环境 ParaPIE 的研究目标就是辅助程序员了解程序行为，在程序员帮助下解决自动并行化编译的问题；为研究者提供开放的技术研究平台，探索更有效的并行化技术。而本文采用的研究手段就是设计一个并行程序设计交互环境，用它来分析困难的串行程序，发掘其中的并行性及可推广的新并行化编译技术。

并行程序设计交互环境 ParaPIE(the PARAllel Programming Interactive Environment)课题起步于 1994 年，于 1995 年底通过鉴定验收。ParaPIE 以自动并行化编译器 AFT 为雏形，实现了程序编辑，程序语法分析，中间表示生成，程序变换，代码生成，程序计算量分析，最大潜在并行性分析等重要功能，为进一步开展并行程序设计交互环境的深入研究打下了坚实的基础。在 ParaPIE 已有工作的基础上，1996 年初，正式启动了与比利时 Gent 大学和欧洲 Simens 公司合作的欧盟国际合作研究项目 ITDC94-164 我方的工作：Gent 大学开发的自动并行化编译工具 FPT 的基础上，结合复旦大学 ParaPIE 前期工作的经验，完善和发展并行程序设计交互环境 PROFPAT(PROgramming environment for Fortran PArallel Transformer)，或简称 PEFPT(Programming Environment for FPT)。

PROFPAT 主要在以下方面改进了早期的 ParaPIE：

- 程序中间表示和可视化：除了 ParaPIE 已有的语法树，过程表，循环表等中间表示之外，又增加了过程调用图，语句级数据相关图的图形可视化工具(ParaPIE 中也有调用图和数据相关图中间表示，但没有提供可视化支持)，增加了循环迭代空间数据相关图的中间表示及图形可视化工具，增加了 FPT 的任务图的可视化工具。
- 目标程序代码生成：除了生成 SGI 和 SUN SMP 并行机上的并行 Fortran 程序，还增加了生成利用 PVM 原语和 Multi-thread 库实现的并行目标程序的功能。
- 程序分析：增加了可 DOALL 并行化循环分析，可私有化数组分析和循环迭代空间相关性分析等动态程序分析工具，结合可视化工具为分析程序并行性提供有效支持。
- 程序变换：增加了交互数组私有化变换和交互循环么模变换的程序变换功能。

1996 年秋，PROFPAT 项目在比利时布鲁塞尔通过了欧盟中期验收，标志着 PROFPAT 系统已经基本设计完成，开始进入了实际应用阶段。在随后的一年中，PROFPAT 被用来分析三个困难的 SPEC95 测试程序，取得了一定的并行化效果，并且发现了若干可以实现于自动并行化编译的新技术：如结合相关和覆盖的数组私有化方法、对多层嵌套循环计算么模并行化变换矩阵的循环么模变换方法、结合数组规约识别增强么模变换的技术、发掘非循环级并行性的技术、动态数据流分析技术、优化利用循环间 Cache 数据局部性的技术等。1997 年秋，PROFPAT 项目最终通过了欧盟委员会验收鉴定，并成为二十多个 ITDC 国际

项目中六个优秀项目之一参加了德国拿骚的展示会。

本文第 2 章探讨了如何借助并行程序设计交互环境帮助程序员分析和利用潜藏于串行程序中的并行性的原理，第 3 章详细介绍了并行程序设计环境 PROFPAT 中集成的主要功能和主要工具的设计实现方法，第 4 章结合程序交互并行化分析、变换技术，说明了 PROFPAT 用于分析利用三个难以自动并行化的 SPEC95 测试程序 WAVE5, FPPPP 和 APSI 的并行性的实验过程和实验结果。本文第 5 章重点介绍了相关-覆盖相结合的数组私有化新技术，第 6 章重点介绍了循环自动幺模并行化变换方面的若干新技术，第 7 章介绍了其他一些自动并行化新技术，如动态数据流分析，循环间 Cache 优化等。本文第 8 章提出了扩充和改进已有的并行程序设计交互环境的设想，展望了并行程序设计交互环境的广阔发展前景。

2 基本概念和原理

下面我们分“高性能计算机”，“程序分析”，和“程序并行化及优化变换”三个部分详细阐述并行程序设计环境研究的基本概念和原理。

2.1 高性能计算机

随着硬件技术的发展，人们已经在芯片上集成多达 12.5 亿个晶体管，时钟频率达到 1GHz。由于芯片集成度及时钟频率几十年来从不曾间断地持续迅速提高，我们完全有理由相信这种趋势仍然会在不久的将来延续下去。但是也要看到元器件的绝对频率和集成度受到光速及量子效应的局限，单机性能的发展速度已经无法跟上时代对超大规模计算的迅速增长的需要。

为了达到高性能，必须从体系结构、操作系统、编程语言编译器、编程工具和环境、程序设计及算法等方面充分利用并行处理技术，才能根本解决提高实际应用程序计算性能的问题**[63]** 。

高性能应用程序设计及算法
高性能程序设计工具及环境
高性能程序设计语言及编译
高性能操作系统
高性能体系结构
高性能元器件

图 2-1、高性能计算机技术的支持层次

2.1.1 体系结构支持

要把每一代元器件技术所赋予的性能优势的可能性转换为现实性，设计者必须针对应用市场的不同实际需要，采用各种体系结构制造出特定的计算机产品。每一种新体系结构技术都是针对一定的市场需要而诞生的，其中有些技术广泛地适用于各种计算机，如 Cache 存储器等，还有些技术则针对较狭窄的市场范围，如向量指令集等。所有这些体系结构技术的终极目的是以合理的代价达到高性能。

2.1.1.1 体系结构分类

通常根据计算机系统中有无多个处理机，将体系结构简单区分为单处理机(uniprocessor)的串行计算机和多处理机(multiprocessor)的并行计算机。而并行计算机又可根据处理机的多少分为小规模对称多处理机(SMP)、可伸展规模多处理机(scalable SMP)和大规模并行计算机(MPP)；根据存储器在处理机间的分布情况分为共享主存(shared memory)和分布主存(distributed memory)；根据指令流与数据流在处理机间的分配情况分为单指令流多数据流(SIMD)和单指令流多数据流(MIMD)**[14][31]** 。

2.1.1.2 单处理机

标量(scalar)计算机是最常见的单指令流单数据流(SISD)计算机体系结构。它的数据通路由 CPU，多级存储设备，总线拓扑，输入/输出设备等组成。

绝大多数串行计算机属于这种体系结构，其 CPU 分为复杂指令(CISC)类型(如 IBM390 类大型机，VAX 系列小型机，采用 Intel 芯片的早期 PC 机)和精简指令(RISC)类型(采用 Sparc, RS6000, HP PA-RISC, MIPS R-8000, Alpha AXP, Power PC 等等 RISC 芯片的工作站)。

即使对于单处理机的所谓串行计算机而言，利用硬件冗余，如多功能部件，流水线，高速缓存(cache)，多级存储器，超长指令(VLIW)，多线程，超标量(superscalar)，向量(vector)指令及链接技术等，也可为程序指令执行带来一定的并行性，从而提高计算速度性能。

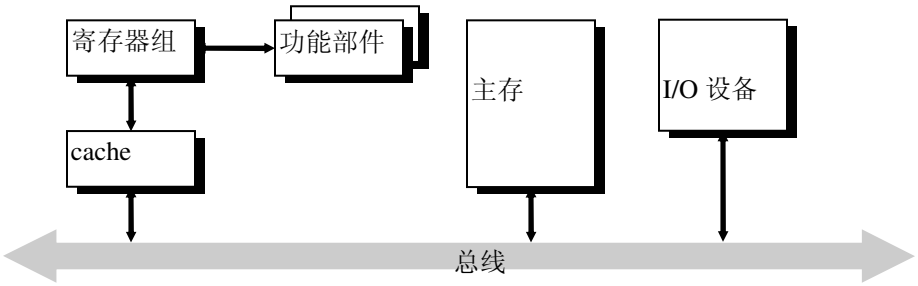


图 2-2、标量计算机的数据通路

例如，早期的超级计算机就广泛采用了向量化体系结构，如 Cray 1，银河 1 号等，适应了在科学计算中经常对向量数据进行计算的需要。向量处理机不仅提供传统的标量指令集和标量寄存器，还提供了向量指令集和向量寄存器。在向量计算机中，利用流水线的概念，将一个计算部件拆成几个流水功能部件，通过时间叠加实现并行加速，即平均在一个时钟周期内，一条向量指令就能处理多条标量指令才能处理的数据。因此将标量计算向量化可以为向量计算机带来更快的计算性能。

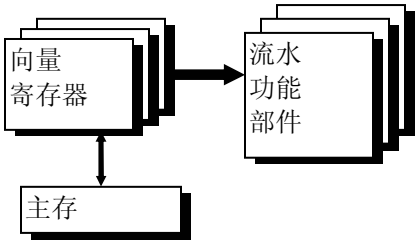


图 2-3、向量计算机的数据通路

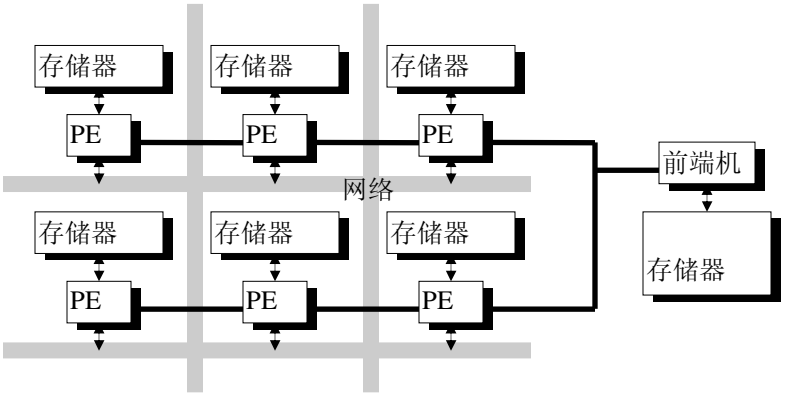


图 2-4、SIMD的处理机阵列、互联网络及前端处理机

这些单机并行化技术利用的是指令级小粒度(fine grain)的并行性。但是单纯依靠单机提高计算速度的潜力是有限的。要实现任务级大粒度(coarse grain)并行性，克服单机的性能局限，就需要应用已经逐渐成熟的多处理机并行技术。

2.1.1.3 单指令流多数据流(SIMD)大规模并行机

单指令流多数据流(SIMD)体系结构包括一个发布指令的前端控制单元和一个由多个执行同一指令的处理单元(PE)组成的阵列。

由于每个 PE 比较简单，便于使其数量达到很大规模，进行大规模并行计算(MPP)。典型的 SIMD 并行机如 Thinking Machine 公司的 CM-2，MaxPar 公司的 MP-2 等。

2.1.1.4 共享主存对称多处理机(SMP)

共享主存多处理机体系结构是最常见的 MIMD 多处理机系统，它由若干(不超过 30) 个共享单一主存的处理机组成。为了减少多处理机对共享主存的争用，通过宽频带多总线或交叉联接(crossbar)的方式连接这些多处理机。虽然共享主存，每个处理机还有各自私有的 cache，并且由硬件来维护同一主存单元在多个处理机 cache 之间的一致性。

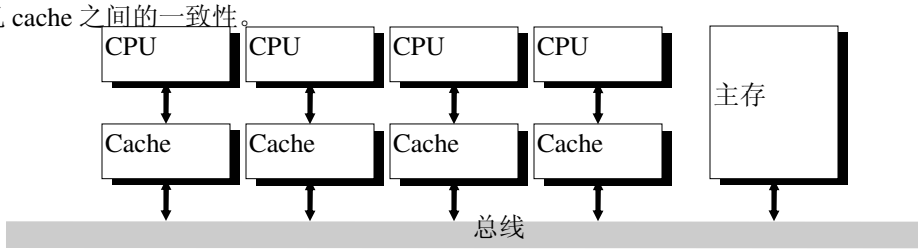


图 2-5、共享主存SMP中通过总线互联的局部cache与主存储器

Cray X-MP, Y-MP, SGI Power Challenge,银河 2 号等都是典型的共享主存多处理机系统。

2.1.1.5 分布主存可伸展多处理机(DSMP)

分布式存储可伸展多处理机是 MIMD 体系结构，每个结点处理机是一个完整的计算机，有自己的局部存储器，它们之间通过一定的拓扑结构(如总线结构、环形结构、Mesh 结构、超立方结构等)互联，借助消息传递在处理机之间分布数据。

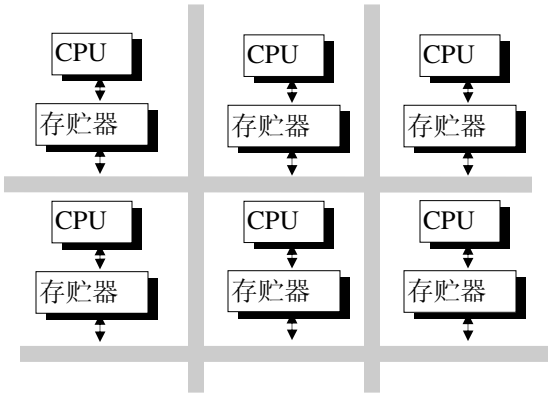


图 2-6、由处理机/存贮器对互联成DSMP

常见的分布主存多处理机有 IBM SP2, Intel Paragon, nCUBE, CM-5, Cray T3D，曙光 1000 等。

2.1.1.6 多级存储层次

结合寄存器，高速缓存(cache)，主存及外存等不同速度与容量的存储器构成了计算机中存放数据和代码的多级存储层次结构，如图 2-7所示。

上述存贮层次每两级之间速度往往相差若干数量级，而容量也同样相差若干数量级。前者是提高计算机存贮性能的关键，后者是降低存贮成本的关键。要提高计算机的速度性能，就必需尽可能多地重用高速存储设备，尽可能减少对低速存储设备的引用次数。

寄存器是由 CPU 指令直接存取的最快的存储设备，当然也是最昂贵，容量最小的存储设备。对于标量计算机，一个寄存器一次只能存贮一个标量，而向量计算机中引入了向量寄存器及向量运算部件，一次向量寄存器存取操作可以处理相当与许多标量的数据。所以当计算机体系结构支持向量运算时，就要尽量把对成批标量的运算转化成对向量的运算。

Cache 是仅次于寄存器的高速存储器，它作为 CPU 与主存贮器的中间缓冲，通过将频繁使用的数据和代码保存在 cache 中，减少对主存的访问，从而降低主存的平均访问延迟，增加有效带宽，有效地提高计算机的性能。为了维持 cache 与主存之间的数据一致，必需用适当的置换策略将不经常引用的数据写回主存，将经常重用的数据保留在 cache 中。这就是说，要充分利用程序指令和数据的局部性，提高 Cache 访问的命中率。

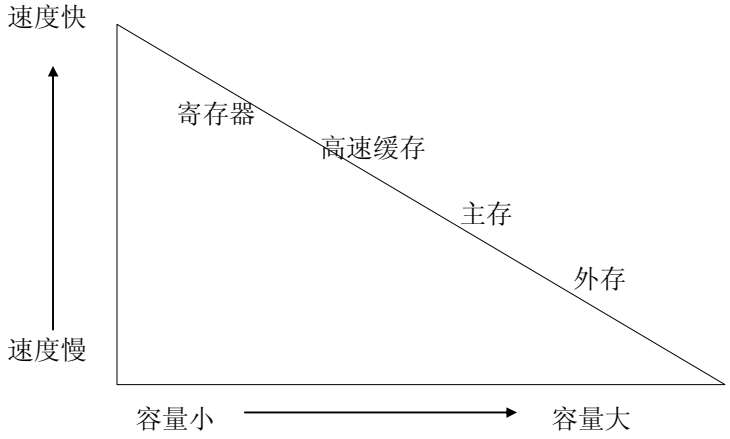


图 2-7、多级存储层次

在共享主存的多处理机系统中，每个处理机一般拥有自己的局部 cache，而可能还有一个全局 cache，这无疑能够提高各个处理机的性能。但是为了维护各个局部 Cache 互相之间及局部 Cache 与共享主存和全局 cache 之间的数据一致性，必须采用更复杂的置换策略。当程序并行执行时，处理机的调度策略有时难以做到把对同一存储单元内容的访问操作集中在同一处理机执行，显然当同一存储单元内容的访问操作被分配给不同的处理机执行时，cache 的读写命中率就会降低。近来的研究表明：多机系统往往比单机系统的 cache 命中率低**[26][27]**。另一方面，在多处理机系统中，cache 命中率低可能使程序并行执行的开销大大增加。这是因为：当两个或两个以上处理机的局部 cache 都保存了同一存储块的一个副本，而某一处理机执行的操作写引用了该存储块中某个存储单元的内容时，保证数据一致性的操作会带来额外的开销。因此提高 cache 命中率是改善多处理机系统性能的有效途径。

2.1.2 程序设计语言支持

由于历史原因，Fortran 仍是支配高性能计算的程序设计语言。为了迎合并行计算的需要，传统的串行 Fortran 程序设计语言(Fortran 77)在各种并行计算机上被扩展了，如 Cray Fortran, SGI Fortran 等，并且时刻在进行着标准化的工作(如 Fortran 90**[1]**, HPF, PCF)。程序设计语言扩展的一个主要方向是增加一些描述程序时间并行性与数据并行性的语句或元语，依靠相应语言编译器产生直接对应于相应计算机的目标代码(如 HPF)，另一个主要的扩展方向是增加一系列实现并行处理机任务分配、时序调度、数据分布和消息传递元语的库函数，用户程序通过调用这个并行程序库实现高性能计算(如 MPI 与 PVM)。不论采用何种方式，由于这些扩展的语句或库函数必须最终在目标计算机体系结构上执行，所以必须用符合目标体系结构特点的方式利用这些语言支持。

大多数并行程序设计语言可以在向量化语句、并行循环、并行任务划分等不同粒度级别上支持程序的时间并行。

我们不妨用并行程序设计语言改写矩阵乘法的串行程序为例，说明对这些并行性的语言支持。

```
DO i=1,n
  DO j=1,n
    DO k=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
    ENDDO
  ENDDO
ENDDO
```

2.1.2.1 语句级并行性

在 Fortran 90 中引入了向量化的语句，提供了向量加减，向量乘除标量，向量赋值语句，向量内积，向量归约等支持向量体系结构的语法。

用向量语句改写上例中的串行程序如下，

```
DO j=1,n
  DO k=1,n
    DO i1=1,n,B
      c(i1:i1+B-1,j)=c(i1:i1+B-1,j)+a(i1:i1+B-1,k)*b(k,j)
    ENDDO
  ENDDO
ENDDO
```

采用向量运算，一次运算批量处理数组的一个数据行或数据列上的 B 个元素，减少了最内重循环次数 B 倍。早期的自动向量化变换器，如 KAP，PTRAN 实现了对串行程序的自动向量化变换，证明可有效提高串行程序在向量计算机上的性能。

2.1.2.2 循环级并行性

一般串行 Do 循环语句语法如下：

```
<do_stmt> ::=
    DO <var> = <l_exp>, <u_exp>[,<s_exp>]
      <body>
    ENDDO
```

循环语句通常是构成程序主要计算量的关键部分。如果将一个或一批循环的迭代分配到不同并行处理机上串行执行，就可以利用循环级的并行性。

由于并行程序设计语言编译器必须在处理机任务间划分循环迭代，所以容易实现已知循环执行次数的 DO 循环。而循环执行次数无法静态决定的 WHILE 循环就不易划分循环迭代了，这时候一般把能分析出迭代次数的 WHILE 循环变形为 DO 循环处理。所以在并行程序设计语言中通常都对并行 DO 循环提供支持，显式指明其循环迭代是可并行执行的。一般提供两种并行 DO 循环语句：DOALL 和 DOACROSS。

```
<doall_stmt> :=
    DOALL <var> = <l_exp>, <u_exp>[,<s_exp>]
      <body>
    ENDDOALL
<doacross_stmt> :=
    DOACROSS <var> = <l_exp>, <u_exp>[,<s_exp>]
      <body>
    ENDDOACROSS
```

它们的语义是不同的。DOALL 循环被编译器认为是完全并行的循环，即所有循环迭代之间没有必须保持的执行次序。如果循环迭代之间存在存贮访问冲突(对同一存储单元赋值或运算是不可交换与结合的，必须按照一定次序进行)，则会带来计算结果的不确定性。所以一般必须保证 DOALL 循环的迭代之间不存在会影响计算结果的存储相关性。DOACROSS 循环是在循环迭代间进行同步的并行循环。处理机可以并行执行各循环迭代体，但是必须在结束循环迭代时与其他并行任务同步。

用 DOALL 语句改写上例中的串行程序如下，

```
DOALL i=1,n
  DO k=1,n
    DO j=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
    ENDDO
  ENDDO
ENDDOALL
```

根据对循环语句的相关性分析发现能 DOALL 并行化的循环，可以在多处理系统上有效发掘串行程序的并行性。

2.1.2.3 任务级并行性

Psection 语句说明的是可并行执行的任务语句段。它是更一般的并行性语言机制，通常通过 WAIT/POST 元语来实现同步互斥。

<PSection_stmt> :=

PSECTION BEGIN

BEGIN SECTION

<Section_Body$_1$>

END SECTION...

BEGIN SECTION

<Section_Body$_n$>

END SECTION

PSECTION END

对于能够划分为 N 个计算量粒度为 G 的并行子任务的串行任务而言，其总计算量应为 O(N*G)。如果可供调度的任务数 N 少于处理机个数 P 时，就无法充分发挥多处理机的并行执行效率；但如果每个任务的计算量粒度小于一定的阈值时，每个任务实际调度的开销甚至超过任务体本身，则会降低并行化加速比，这时候应该适当减少任务数，增加每个任务的执行粒度；此外，如果处理机的计算能力之间存在有较大差异，则令并行任务间粒度的大小完全相同反而无法平衡处理机之间的计算负载，如果处理机是对称的，则应尽量让并行任务间粒度的大小相同来提高并行化效率。

2.1.3 程序的性能量度

2.1.3.1 CPU 时间

衡量程序性能的最主要指标是它的执行时间，程序执行时间分为系统时间和 CPU 时间，一般系统都提供 Profiler 工具来分析程序的计算性能。由于多任务操作系统中其他同时正在执行的任务争用系统计算资源的影响，对并行程序而言，CPU 时间的改变能够比较客观地反映出程序的性能改进。

2.1.3.2 时间加速比(Speedup Ratio)

如果串行程序 P_1 执行耗时为 T，在程序并行化为 P_2 后，如果 P_2 仍单机执行耗时为 T_1，如果 P_2 在 N 个处理机上并行执行耗时为 T_N，显然一般有：$T_N \le T \le T_1$。$T_N \le T$ 是因为如果并行化不能提高程序性能，则还不如选择原来的程序执行，$T \le T_1$ 是因为并行化后程序单机执行会增加一些额外的并行调度开销，因此一般会导致额外的计算时间。但是如果并行化时考虑了程序的优化，则有时会产生 $T_1 \le T$ 的例外情况，甚至产生超线性加速比。

我们研究的目的侧重于如何提高串行程序的性能，所以采用加速比 $S=T/T_N$ 作为衡量并行化效果的主要根据。

2.1.3.3 影响程序并行化效果的主要客观因素

影响程序并行化效果的因素有很多，首先决定于待分析串行程序本身是否具有一定的并行性，一个本质上是串行的待分析程序是很难取得并行化效果的。其次取决于占据主要计算时间的程序段是否能够并行化，如果不能对主要计算量并行化，程序性能很难取得实质的提高。最后，选择哪种并行化变换能取得较好的效果，也是一个重要因素，当程序并行任务粒度太小时并行化变换甚至还不如串行执行的效率高。

2.1.3.3.1 程序潜在并行性

如果一个程序段具有很多的并行性，主观上看，就应该能够将它并行化，取得理想的加速比。但实际情况往往是事与愿违，因为不了解程序到底是否对所有可能输入都具有并行性，这时候就不能草率地对程序并行化。比如，最一般意义上的数据相关性分析本质上是不可精确求解问题，必须在一定严格条件下对部分程序可精确求解。

2.1.3.3.2 程序计算量分布

程序计算量分布对加速比影响：如果程序段 $P=P_1;P_2;...;P_n$，它们占整个程序 P 串行执行时间的比例分别是 $p_1,p_2,...,p_n$，对它们并行化后取得加速比分别为 $s_1,s_2,...,s_n$，则整个程序 P 能取得加速比应为 $S=1/\sum_{i=1}^{n} p_i/s_i$ 。

可见，某程序段的计算量越大，它的加速比对整个程序的加速比影响也越大。

通过对实际程序执行时间的观察，可以发现程序的绝大多数计算量集中发生在循环中，所以并行化处理的主要对象应该是程序中的循环。而不同的循环各自的计算量之间也差别很大。

程序计算量分析主要是分析各个循环能否并行化对整个程序性能的影响程度。占整个程序执行时间较多的循环，首先应该成为并行化分析和变换的重点。自动并行化编译由于不能事先知道程序的计算量，所以经常将分析能力浪费在不重要的循环上，反而影响了对重要循环的分析，并行程序设计环境集成了程序循环一级的计算量分析工具，提示程序员注意分析那些最有分析价值的循环，很好地解决了这个问题。

2.2 程序分析

2.2.1 程序语法分析

在程序编译工具中，语法分析是不可缺少的环节，它除了判断源程序的语句是否符合程序语言语法外，还要进一步构造程序的中间表示，为下一步程序分析、变换和代码生成准备好内部数据结构。在 Fortran 程序的语法分析不仅是自动并行化工具的前提，也是并行程序设计环境的工作前提。

2.2.2 程序语义分析

对语法分析好的程序得到的中间表示，可以根据程序变换的需要进一步分析程序的语义，包括传统的数据流分析，分析并行语义的各种相关性分析等等。

2.2.2.1 数据流分析

数据流分析就是对程序中的数据变量(标量或数组)，分析每个变量引用点(Use)的被程序其他部分中哪些

定义点(def)影响，分析每个变量定义点会影响程序其他部分的哪些引用点。也就是说，分析在数据被引用(读)时可能到达该语句的所有可到达定义点集合(读集合 MayUse)，分析在数据被定义(写)时一定会被该语句重新赋值覆盖的可到达定义点集合(确定写集合 Kill)，分析在执行语句后可能被修改的变量集合(可能写集合 MayMod)。

对一个包含条件分支和循环的程序段而言，数据流分析是根据无环控制流图(删除循环回边)，通过对语句 s 建立数据流方程并通过初始条件(进入程序段或离开程序段时的数据流集合)迭代计算求解：

$$MayUse(s)=Local\_MayUse(s)\cup \bigcup_{p\in PRED(s)} MayUse(p)$$

$$MayMod(s)=Local\_MayMod(s)\cup \bigcup_{p\in PRED(s)} MayMod(p)$$

$$Kill(s)=Local\_Kill(s)\cup \bigwedge_{p\in PRED(s)} Kill(p)$$

其中，PRED(s)表示在控制流图上语句结点 s 的除了来自循环回边的结点外其他所有直接前驱结点集合。数据流分析必须保守计算，以确保正确性。换句话说，放大 MayUse 和 MayMod 集合是保守的，缩小 Kill 集合也是保守的，反之有可能导致错误的结果。

数据流分析的衍生结果还包括暴露集合及活跃变量集合，这些结果不仅对传统的编译优化变换，而且对并行化编译的相关性分析和数组的私有化分析都有重要作用。

2.2.2.2 数据相关性分析

2.2.2.2.1 数据相关性

在两个冲突的存储单元(程序变量)引用之间存在三种数据相关(data dependence)：

●先写后读的流(flow-)相关，

S1：V=.....

...

S2: .....=V

●先读后写的反(anti-)相关，

S1: .....=V

...

S2：V=.....

●先后都写的输出(output-)相关。

S1：V=.....

...

S2：V=.....

发生在两个程序段之间变量引用的数据相关性是影响这两个程序段并行执行的主要障碍，它定义了相关程序段之间正确执行的时间偏序关系。

2.2.2.2.2 循环迭代空间

一个循环迭代(iteration)是循环体中出现的循环下标变量替换为对应值的程序段实例。发生在同一循环迭代内两个变量引用之间的相关称为与该循环无关(loop-independent)相关。发生在两个不同循环迭代内两个

变量引用实例之间的相关称为跨循环(loop-carried)相关。

对一个规范 n 重循环而言，如表 2-7a ，其循环迭代空间(iteration space)是由各层循环的下标变量在给定上下界内所有取值向量所组成的集合$\{\mathbf{i}=[i_1,...,i_n]^T \in \mathbf{Z}^n \mid L_1 \le i_1 \le U_1，\ L_2(i_1) \le i_2 \le U_2(i_1)，\ ...，\ L_n(i_1,...,i_{n-1}) \le i_n \le U_n(i_1,...,i_{n-1})\}$。

串行循环的迭代空间是有序的（按照规范化迭代空间的字典序(lexigraphical order)从小到大依次执行：由元素 < 顺序决定的 n 维向量字典序$\angle$定义为 $\mathbf{i}=[i_1,...,i_n]^T \angle [j_1,...,j_n]^T=\mathbf{j}$，如果 $i_1<j_1$ 或 $i_1=j_1$ 且 $i_2<j_2$ 或, ...,或 $i_1=j_1$，...，$i_k=j_k$ 且 $i_{k+1}<j_{k+1}$, $k<n$）。DOALL 并行循环的迭代空间是无序的，因此，循环并行化变换后必须保证在原来迭代空间中存在跨循环相关的两个循环迭代保持串行迭代执行顺序。

2.2.2.2.3 相关距离

考察导致相关引用 A(**f**(**I**))和 A(**g**(**I**))的两组下标表达式 **f**(**I**)和 **g**(**I**)，如果 A(**f**(**I**))跨循环相关于 A(**g**(**I**))，那么在循环迭代空间 **I** 中必定存在两个迭代实例 $\mathbf{i}_1$ 和 $\mathbf{i}_2$，使得相关性方程成立：

$$\mathbf{f}(\mathbf{i}_1)=\mathbf{g}(\mathbf{i}_2)$$

相关距离向量(distance vector)**d** 定义为两个迭代向量 $\mathbf{i}_1$ 和 $\mathbf{i}_2$ 之间的差。迭代空间 I 中，相关的迭代点 $\mathbf{i}_1 \angle \mathbf{i}_2$ 之间的距离向量 $\mathbf{d}=\mathbf{i}_2-\mathbf{i}_1$, 不难看出，$\mathbf{i}_1 \angle \mathbf{i}_2$ 当且仅当 $\mathbf{0} \angle \mathbf{d}$。

2.2.2.2.3.1 常数相关距离

如果向量 $\mathbf{d} = \mathbf{i}_2 - \mathbf{i}_1$ 是不依赖循环迭代变量 **i** 的常数向量，并且

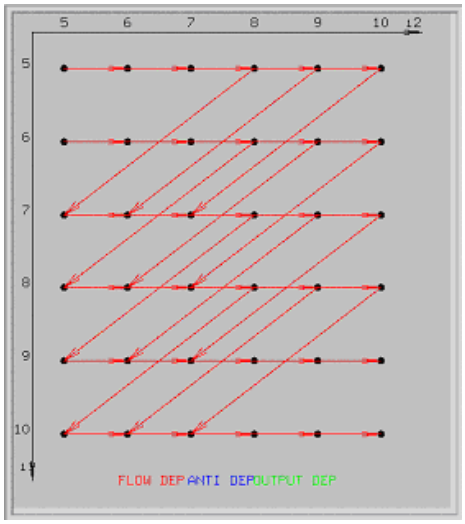
$$\mathbf{f}(\mathbf{i}_2-\mathbf{d})=\mathbf{g}(\mathbf{i}_2)$$

对所有相关方程的解都成立，则称相关距离 **d** 为常数相关距离。

例如，表 2-1循环中

表 2-1、常数相关距离

```
DO i1=5,10,1
   DO i2=5,10,1
      a(i1,i2)=a(i1,i2-1)+a(i1-2,i2+3)+a(i1-3,i2+7)
   ENDDO
ENDDO
```



a(i1,i2)和 a(i1,i2-1)的所有跨循环流相关解满足 $\mathbf{d} = \mathbf{i}_2 - \mathbf{i}_1=(0,1)^T$; a(i1,i2)和 a(i1-2,i2+3)的所有跨循环流相关解满足 $\mathbf{d} = \mathbf{i}_2 - \mathbf{i}_1=(2,-3)^T$ 。a(i1,i2)和 a(i1-3,i2+7)的所有跨循环流相关解满足 $\mathbf{d} = \mathbf{i}_2 - \mathbf{i}_1=(3,-7)^T$。

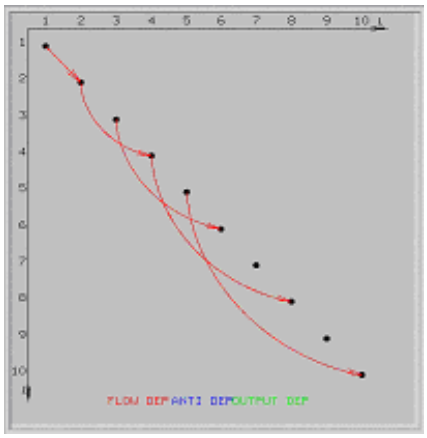
2.2.2.2.3.2 非常数(可变)相关距离

但是相关距离向量并不总是常数的，在实际测试程序中如果形如 $j=j+M_1$ 和 $k=k+M_2$ 的递归标量出现在数组下标表达式中($M_1 \neq M_2$), 则 A(j+N_1)和 A(k+N_2)之间的相关性就是非常数距离的,对应相关性方程为 $M_1*i+N_1 = M_2*i+N_2$。

比如表 2-2循环就是一例：

表 2-2、非常数相关距离

```
PARAMETER(n=10)
DO i=1,n
   a(2*i)=a(i)
ENDDO
```



如果把 $\mathbf{i_2}$ 看作 $\mathbf{i_1}$ 的函数(或把 $\mathbf{i_1}$ 看作 $\mathbf{i_2}$ 的函数)，相关性方程就可改写成：

$$\mathbf{f(i_1)=g(i_2(i_1))}\text{或者 }\mathbf{f(i_1(i_2))=g(i_2)}$$

从而有可能显式将 $\mathbf{i_2}$ 关于 $\mathbf{i_1}$ 的函数写出：

$$\mathbf{h(i_1)=i_2(i_1)}\text{ 或者 }\mathbf{i_1(i_2)=h(i_2)}$$

于是，可变距离相关向量可表达为关于 $\mathbf{i_1}$ 或 $\mathbf{i_2}$ 的函数：

$$\mathbf{d(i_1) = i_2 (i_1) - i_1}\text{ 或者 }\mathbf{d(i_2) = i_2 - i_1(i_2)}$$

对上述例子，按定义可变距离相关向量 $\mathbf{d(i_1) = i_2(i_1)-i_1}=2[i_1]-[i_1]=[i_1]$。

2.2.2.2.4 相关性测试

相关性测试为自动并行化的基础，并行识别及多种并行变换都需要用到相关性信息。相关性测试的主体为数组相关性测试，数组相关性测试的基本问题为判断在一个嵌套循环中同一数组的两次出现（其中一次为写）之间是否存在相关，即是否会对同一地址（数组下标）进行操作。由于数组下标可以看成是循环下标的函数，所以上述问题可以看作一个判断满足一定的约束条件的丢番图方程是否存在整数解的问题。下面我们对不同的相关性测试技术进行分别介绍。

2.2.2.2.4.1 GCD 测试

GCD 测试是最简单的一种测试方法：若一整数方程

$$a_1x_1+a_2x_2+...+a_nx_n = c$$

存在整数解，则 $\gcd(a_1,a_2,...,a_n)$ 必能整除 c，也就是说，若 $\gcd(a_1,a_2,...,a_n)$ 不能整除 c，则方程肯定没有整数解。

基于上述定理，GCD 测试对每一个丢番图方程考察其系数的最小公约数是否整除 c。若有一个丢番图方程不满足该条件，则必然没有整数解，也即相关性不存在，否则就保守地假设相关性存在。

2.2.2.2.4.2 Banerjee 测试

Banerjee 测试是被许多并行系统广泛采用的一种相关性测试方法。它的实现很方便，但它的测试范围并不是很广泛，同时也不是很精确。其实现方法如下：

对一整数 a，记 $a^+=\max(a,0)$, $a^-=\max(-a,0)$. 那么，我们有：

$$\text{若 } p<x<q,\ \text{则 } a^+p-a^-q<ax<a^+q-a^-p.$$

若对 i=1,2,...,n, 有 $p_i<x_i<q_i$, 则

$$\sum_{i=1}^{m}(a_i^+ p_i - a_i^- q_i) < \sum_{i=1}^{m} a_i x_i < \sum_{i=1}^{m}(a_i^+ q_i - a_i^- p_i)$$

那么，对整数方程

$$a_1x_1+a_2x_2+...+a_nx_n = c$$

来说，若

$$\sum_{i=1}^{m}(a_i^+ p_i - a_i^- q_i) > c \quad 或 \quad \sum_{i=1}^{m}(a_i^+ q_i - a_i^- p_i) < c$$

则说明该方程没有整数解。

2.2.2.2.4.3 Omega 测试

Omega 测试是由 William Pugh 于 1991 年提出的一种相关性测试方法**[48]**。Omega 测试较前两种测试方法，在精度上有很大改进(可以精确求解相关距离)。但它实现比较困难，在实际系统中较少得到应用。

Omega 测试实质上是基于 Fourier-Motzkin 消除法的一种整数规划问题算法。它的基本操作是投影。Omega 测试将约束分为等式约束与不等式约束，它首先将等式约束代入到不等式约束中，消去等式约束。接着 Omega 测试开始对不等式约束进行投影，每次投影将消去一个变量。直至整个问题仅剩下一个变量，我们就能很容易地判断它有没有整数解。

由于 Omega 测试监测的为整数解，而非实数解，所以在做投影时，若投影中含有整数解，并不能说明原来的约束中含有整数解，因为被消去的变量可能仅存在实数解。于是，Omega 测试将投影分为黑影(Dark Shadow)与实影(Real Shadow)，实影即为上面所说的投影，而黑影为实影的一个子集，它表示若它具有整数解，则原约束必有整数解。若某次投影仅有实影而不存在黑影，Omega 测试便将原来的问题转换为多个子问题，来求原约束是否存在整数解。

由于静态相关性测试的局限性，在并行程序设计环境中开发了一些动态相关性测试工具，如最大潜在并行性分析工具，数组私有化分析工具，循环迭代(跨循环)相关性分析工具等，加强了相关性分析的精确度。

2.3 程序变换

要进行合法的程序变换，必须是用语义上等价的程序去替换原有程序。

语义保持程序变换：将程序{P}S{Q}变换成{P'}S'{Q'}是允许的，当且仅当 P 蕴涵 P'并且 Q'蕴涵 Q。记为 S=>S'。如果 S'=>S 也成立，称 S 语义等价于 S'，记为 S<=>S'。

对于并行程序而言，由于不可能事先确定各个并行任务的实际执行次序，要保证程序的正确性，就必须保证对并行任务的任意串行执行次序，都语义等价于并行程序。

正确的并行程序：假定并行程序 P 由子任务 $P_1,P_2,...,P_n$ 组成，记为 $P=P_1||P_2||...||P_n$，P 是正确的，当且仅当 $P_1,...,P_n$ 都是正确的，并且 P 语义等价于 $S=P_{i1};P_{i2};...;P_{in}$，其中$(i_1,i_2,...,i_n)$是(1,2,...,n)的任意置换。

结合上述定义，可知将串行程序{P}Seq{Q}变换成并行程序{P'}Para{Q'}，其中 $Para=P_1||P_2||...||P_n$，是**合法**的，要求不仅必须 P 蕴涵 P'，Q'蕴涵 Q，而且并行程序 Para 是正确的。

2.3.1 程序简化和优化变换

简化和优化变换是指所有从串行程序到串行程序的语义等价变换。执行简化变换不仅可以简化程序，方

便对程序行为进行分析；而且简化后的程序通常更容易并行化。执行优化变换则有助于提高程序最终的性能，但不应该为了优化串行程序而影响随后的并行化分析和变换。

表 2-3所示表达式化简、常数传播、向前替代的简单例子说明：简化变换通过削弱程序前置条件，方便了进一步分析。向前替代的例子还说明了，简化变换虽然可能会产生串行不够优化的代码，但如果应用相关性分析得知循环可并行化，则实际性能还是得到很大提高。

表 2-3、常数传播、表达式向前替代与表达式化简

程序变换	变换前	变换后
表达式化简	{N.NE.0} T=N/N {T.EQ.1}	{.T.} T=1 {T.EQ.1}
常数传播	{N.EQ.10} DO I=1,N A(I+10)=A(I)+... ENDDO	{.T.} DO I=1,10 A(I+10)=A(I)+... ENDDO
向前替代	{J.EQ.0} DO I=1,10 J=J+3 A(J)=A(3*I)+... ENDDO {J.EQ.30}	{.T.} DO I=1,10 A(3*I)=A(3*I)+... ENDDO J=3*10 {J.EQ.30}

2.3.2 程序规范化

2.3.2.1 程序结构化与规范化

对非结构化程序作任何控制相关分析和数据相关分析都是很困难的。虽然 F77 是支持结构化程序设计的程序设计语言，但是一个典型的实际串行 Fortran 程序仍然是含有大量 IF 语句和 GOTO 语句的非结构化程序[2]。

表 2-4、程序结构化

原 Fortran77 程序段	经结构化后的程序段

IF (JTYPE.EQ.3) GOTO 10
IF(JTYPE.GE.4) GOTO 20
ELSE GOTO 50
20 statement segment 1
IF(JTYPE.GE.5) GOTO 10
ELSE GOTO 50
10 statement segment 2
IF(JTYPE.GE.5) GOTO 40
ELSE GOTO 50
40 statement segment 3
50 CONTINUE
	IF(JTYPE.EQ.3) THEN
	statement segment 2
	IF(JTYPE.GE.5)THEN
	statement segment 3
	ENDIF
	ELSE IF (JTYPE .GE.4) THEN
	statement segment 1
	IF(JTYPE.GE.5) THEN
	statement segment 2
	IF(JTYPE.GE.5) THEN
	statement segment 3
	ENDIF
	ENDIF
	ENDIF

由于早期的 FORTRAN 缺乏足够的结构化描述语句，致使大量已存在的相当有价值的应用程序普遍采用 GOTO 语句。同时，FORTRAN 还提供了如 ENTRY，SAVE，算术 IF 等非规范语句。这些语句的使用使

得早期的应用获得相对较小的代码规模，但同时却降低了程序的可读性及可维护性。非规范语句的出现使得程序结构紊乱，出现多个程序入口，或使控制相关复杂，这些都影响系统的相关分析和并行转换。由于 GOTO 语句的出现,致使程序内部语句之间的控制相关关系变得更加复杂,同时影响了数据相关关系的分析,从而影响了并行转换技术的应用。我们从程序的流图出发，用循环及分支语句代替 GOTO 语句，从而消除了 GOTO 语句，如表 2-4所示。通过将程序结构化成仅由顺序块、块 IF 和循环组成的这样三种仅含单入口和单出口的语句块，不仅可以方便以后的语义分析，而且能够转化和识别出更多的 DO 循环，为进一步作优化和并行化作重要的准备。

自动程序结构化的研究不仅对自动并行化编译是十分重要的，对于程序员正确理解非结构化程序也有重要的意义。例如表 2-4。

对于其它非规范语句，如 ENTRY、SAVE、算术 IF、计算 GOTO、赋值 GOTO 等，由于它们在程序中出现的频率很低，所以我们在不改变其语义的情况之下，用其它语句来代替这些非规范语句。在不改变程序语义的前提下，改变程序结构来消除或用其它语句来替代非规范语句，我们称之为程序的规范化。

2.3.2.2 DO 循环迭代空间规范化

DO 循环规范化(normalization) ，使循环的遍历下标变量取值为一步长为+1 的整数序列，如表 2-5所示。

表 2-5、循环规范化

变换前	变换后

{.T.}
DO i=expL,expU,S
Body(i)
ENDDO
{.T.}
	{.T.}
	IF (expL.LE.expU.AND.S.GT.0.OR.
	expL.GE.expU.AND.S.LT.0)THEN
	DO i'=1,(expU-expL)/S+1,1
	i=expL+(i'-1)*S
	Body(i)
	ENDDO
	ENDIF
	{.T.}

2.3.3 程序并行化及优化变换

串行程序变换的目的是简化程序，因此往往削弱程序前置条件，而许多并行化变换则往往是有条件的（例如无数据相关），从而引入额外的前置条件，只有满足该条件时才能应用并行化变换技术，如表 2-6所示。

如果存在数据相关性，一般情况下必须对任务体变形，执行诸如变量私有化、循环分布、合并、展开、划分、交换、各种循环幺模变换等并行化变换，消除数据相关性后，才可能不需要同步，完全并行执行（DOALL），否则必须根据相关图尽量同步并行执行（DOACROSS）。但是也有少数例外情况，如果有相关性的任务体之间的任意执行次序语义相同，在保证对共享变量互斥写的简单同步前提下也可以完全并行化，如归约语句就是这种特殊情况。

各种自动并行化变换，如变量私有化（variable privatization）**[20][43][72][73]**、循环分布（loop distribution）与合并（fussion）**[74]**、展开（unfolding）、循环划分（strip-mining）、循环交换（interchange）**[61]**和循环幺模变换（unimodular transformation）等**[13][24][65]**都对要执行并行化的程序的前置条件提出了相应的要求，如果不满足条件，只能作保守的判断，即无法应用该技术。但是这些前置条件中出现的对符号变

量取值或取值范围的断言常常是难以自动分析得出的，因此自动并行化编译器经常拒绝并行化本可以并行化的程序段。

下面表 2-6的例子说明了假定无跨循环数据流相关时才能对循环作相应并行化变换。

表 2-6、DOALL并行化变换

变换前	变换后
{.T.} DO I=1,10 A(I+N)=A(I)+... ENDDO	{N=0 or N>=10 or N<=-10 } DOALL I=1,10 A(I+N)=A(I)+... ENDDO

对于已经并行化的程序应用循环划分（strip-mining）、循环交换（interchange）等变换技术提高任务粒度和利用数据局部性，可能进一步提高并行执行的性能。但是不适当地使用这些技术，会产生效率低的代码。在并行化技术之间甚至会相互影响，使得应用了某项变换技术（如循环划分）后就无法利用另一项实际效果更好的变换技术。因此，有时自动并行化编译器根据启发式规则反而会选择应用效率低的并行化变换技术**[18][28][46]**。

在并行程序设计交互环境中，借助程序员的提示和干预，可以选择适当的并行化变换技术应用于更多的循环。

2.3.3.1 循环迭代空间变换

只要整数变换 F 使 n 维迭代空间 I 中的迭代点$(I_1,...,I_n)^T$对应于到 m 维迭代空间 J 中的迭代点$(J_1,...,J_m)^T$，就称 F 是循环迭代空间变换。设 $\mathbf{i_1,...,i_n}$ 为迭代空间 I 在参照系 $\mathbf{R}^n$ 中的标架基向量(basis)，$\mathbf{j_1,...,j_m}$ 为迭代空间 J 在参照系 $\mathbf{R}^n$ 中的标架基向量，迭代空间 I 中的迭代点 $\mathbf{i}=(I_1,...,I_n)^T$ 对应于 $\mathbf{R}^n$ 中的向量 $\mathbf{V_i}=I_1\mathbf{i_1}+...+I_n\mathbf{i_n}$, 迭代空间 J 中的迭代点 $\mathbf{j}=(J_1,...,J_m)^T$ 对应于 $\mathbf{R}^n$ 中的向量 $\mathbf{V_j}=J_1\mathbf{j_1}+...+J_m\mathbf{j_m}$, I 空间到 J 空间的线性变换使标架向量乘以 m×n 矩阵 **F**: $(\mathbf{i_1,...,i_n})=(\mathbf{j_1,...,j_m})\mathbf{F}$，因为如果 $\mathbf{V_i}$ 和 $\mathbf{V_j}$ 是同一参照系中的同一向量，$(\mathbf{j_1,...,j_m})\mathbf{Fi}=(\mathbf{i_1,...,i_n})\mathbf{i}=\mathbf{V_i}=\mathbf{V_j}=(\mathbf{j_1,...,j_m})\mathbf{j}$ ，所以 **j=Fi**。如果循环变换使 I 空间中相关的两个迭代点 $\mathbf{i_1}\angle\mathbf{i_2}$ 变换为 $\mathbf{j_1}\angle\mathbf{j_2}$，则 I 空间相关距离向量 $\mathbf{d}=\mathbf{i_2}-\mathbf{i_1}$ 变换为 $\mathbf{d'}=\mathbf{j_2}-\mathbf{j_1}$, 对线性变换 F 有 **d'=Fd**。

表 2-7、规范循环的迭代空间变换

```
DO I1=L1,U1,1                                  DO J1=LL1,UU1,1
  DO I2=L2(I1),U2(I2),1                          DO J2=LL2(J1),UU2(J1),1
    ...                                            ...
    DO In=Ln(I1,...,In-1),Un(I1,...,In-1),1        DO Jm=LLm(J1,...,Jm-1),UUn(J1,...,Jm-1),1
      Body(I1,I2,...,In)             <==>              Body(J1,J2,...,Jm)
    ENDDO                                          ENDDO
    ...                                            ...
  ENDDO                                          ENDDO
ENDDO                                          ENDDO
        (a)                                            (b)
```

2.3.3.2 幺模(Unimodular)变换

如果迭代空间 I 到 J 的线性变换为整数变换，即对 I 中任意一迭代点 **i** 和 J 空间中的迭代点 **j** 都有 **Fi=j**，而且变换前后循环步长始终保持 1；则变换后的循环程序较易书写。幺模(unimodular)变换都是一一对应的整数变换，它等价于对迭代下标向量乘以一个幺模矩阵，而不改变迭代空间的体积和维数。特别地，

所有循环步长为 1 的规范多重循环幺模变换后仍为步长为 1 的规范多重循环(参见性质 3)，如表 2-7b。由于幺模变换具有这些适合循环变换的良好性质，因此成为循环变换研究的重点**[13][24][65]**。

2.3.3.2.1 幺模矩阵和基本幺模变换

幺模矩阵 **U** 是行列式为±1 的整数方块矩阵。循环斜错(skewing)、循环反序(reversal)、循环交换(interchange)是最基本的幺模变换。其变换矩阵分别是：

$$\begin{bmatrix} 1 & & & \\ & 1 & a & \\ & & 1 & \\ & & & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & -1 & \\ & & & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix}$$

(a) skewing　　(b) reversal　　(c) interchange

2.3.3.2.2 幺模矩阵和幺模变换的性质

若 **U** 是幺模矩阵，它具有下列性质，可以用线性代数方法证明，限于篇幅本文不再赘述：

性质 1：若 $\mathbf{U}_1$ ，$\mathbf{U}_2$ 是幺模矩阵，则 $\mathbf{U}=\mathbf{U}_1\mathbf{U}_2$ 仍是幺模矩阵。

性质 2：若 **U** 是幺模矩阵，则 $\mathbf{U}^{-1}$ 存在且仍是幺模矩阵。

性质 3：若 U 是循环幺模变换矩阵，则所有循环步长全为 1 的规范多重循环仍变换为所有循环步长全为 1 的规范多重循环。

性质 4：若 **U** 是幺模矩阵，存在一系列基本幺模矩阵 $\mathbf{U}_1,...,\mathbf{U}_T$, $\mathbf{U}=\mathbf{U}_T\mathbf{U}_{T-1}...\mathbf{U}_1$。

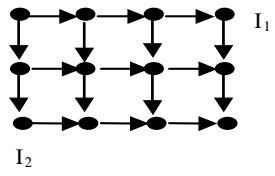
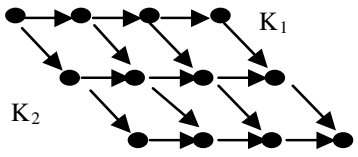
性质 5：列向斜错矩阵 **V** 可以由行向斜错矩阵 **U** 和交换阵 $\mathbf{T}_1$ 和 $\mathbf{T}_2$ 乘积计算：$\mathbf{V}=\mathbf{T}_1\mathbf{U}\mathbf{T}_2$

所以严格地说，基本幺模矩阵由行向斜错、反序、交换组成。

2.3.3.2.3 幺模并行化变换实例

例如，应用幺模变换中的一种斜错基本变换，就可以将表 2-8a 中的循环变换为 b 中的循环，且通过观察循环迭代空间相关图的变化可以看到 b 的内层循环可以并行化。

表 2-8、幺模并行化变换前后的循环迭代空间

(a)原循环	(b)幺模并行化
DO I_1 = 0,3,1 　DO I_2 = 0,2,1 　　$A(I_1,I_2) = A(I_1 - 1,I_2)+A(I_1,I_2-1)$ 　ENDDO ENDDO	DO K_1 = 0,5,1 　DOALL K_2 = max{0,K_1-3},min{2,K_1},1 　　$I_1=K_1- K_2$ 　　$I_2=K_2$ 　　$A(I_1,I_2) = A(I_1 - 1,I_2)+A(I_1,I_2-1)$ 　ENDDOALL ENDDO
	

2.3.3.3 数组私有化(Array Privatization)变换

2.3.3.3.1 临时变量和私有变量

传统串行程序设计往往通过重用一些变量，以减少存储空间，我们称这些被以后程序重用的变量为临时变量，也就是说，临时变量在程序段中所使用(读引用)值不再重用前面执行的程序段所(临时)定义的变量值，而总是在本程序段中被重新定义(写覆盖)的新的变量值。如在表 2-9a 的循环语句中，每个循环体中(第 4 行)使用的变量 S 都是在本次循环迭代中(第 3 行)定义的，而不是在其它循环迭代所定义的值。变量 S 就是循环中的临时变量。

表 2-9、标量的数组扩张和私有化变换

(a)串行程序	(b)数组扩张	(c)私有化变换
1 REAL S	1 REAL S	1 REAL S
2 DO I = 1, N	2 REAL SA(N)	2 DOALL I=1,N
3 S= ...	3 DOALL I = 1, N	3 LOCAL REAL SP
4 ...=S	4 SA(I)=...	4 SP=...
5 ENDDO	5 ...=SA(I)	5 ...=SP
	6 ENDDOALL	6 IF (I.EQ.N) THEN
	7 S=SA(N)	7 S=SP
		8 ENDIF
		9 ENDDOALL

并行程序中的私有变量是不同于临时变量的概念。在并行计算机中，私有变量指局部于一个处理机的存储单元，共享变量指处理机间可共享的存储单元。现在的并行程序设计语言一般都直接支持私有变量说明，如在表 2-9c 中，循环 I 是并行 DOALL 循环。程序第 1 行说明的标量 S 是循环 I 的共享变量，其存储空间对各处理机是相同的。循环体内的 LOCAL 语句说明的标量 SP 是循环 I 的私有变量，并行执行该循环的每个处理机都有一独立的空间存放 SP。因此，并行循环中的私有变量不能取循环迭代以外所定义的值，且本次循环迭代所定义的变量值只在循环体中有效，不能为循环迭代以外引用。显然，私有变量不能作为不同处理机之间共享的临时变量。

2.3.3.3.2 临时变量私有化

使用临时变量虽然可以在串行程序中减少存储空间占用，然而如果临时变量所在的程序段分配到不同处理机，就会在并行程序中引起非本质存储相关问题，阻碍并行化的实现。一般而言可以通过对临时变量换名增加存储空间的方法来消除并行任务之间的非本质存储相关。特别对串行循环而言，如果不同循环迭代读写同一个临时变量(象表 2-9a 的 S)，就会引起跨循环的反相关(先读后写)和输出相关(先写再写)，但不会引起真正意义上[1]跨循环的流相关(先写后读)。在分析串行循环中的并行性时，我们看到通过适当的循环程序变换(如数组扩张和变量私有化)消除临时变量，同时也就消除了由临时变量引起的跨循环反相关和输出相关，从而可以用并行循环语句(DOALL)代替原串行循环语句(DO)。

临时变量的数组扩张变换通过对每一个临时变量(标量或数组)说明一个更高维数的数组，将对原标量或数组元素的每个引用替换为一个以循环变量为扩张维下标的数组元素，使不同循环迭代存取数组不同元素集，从而消除变换前的临时变量。如表 2-9b 中，通过把共享标量 S 扩张为共享数组 SA，使循环 I 并行

1 数据相关可分为基于地址的和基于数值的两种计算方法。如果对同一个存储地址单元，在两个(基于地址的)相关引用执行之间还存在另一个写引用，则不认为两者之间存在基于数值的相关。又因为反相关和输出相关是可以消除的，所以，只有基于数值的流相关才被认为是真正意义上的数据相关。

化。

如前所述，并行循环语句中的私有变量是局部于单个循环迭代的存储单元，因此它们不会产生任何跨循环存储相关。所以在串行循环中指定临时变量为私有变量也可以消除临时变量，使循环语句并行化，这就是临时变量私有化变换，如表 2-9c。私有化和数组扩张的都能消除临时变量，但是两者相比，私有化显然少占用宝贵的存储空间，特别对于循环迭代次数远大于处理机个数时更是如此。

除了临时变量的识别以外，变量私有化变换还要需要解决初值复制和终值复制的问题。并行程序要求在每个并行程序段内私有变量只有被定义后才能被读引用，在定义前的引用是没有意义的，所以临时变量的初值必须在对应并行程序段中复制给私有变量，这就是初值复制问题。并行程序中的私有变量在离开并行程序段后就不可见，但如果串行程序中的临时变量在程序段结束后仍然被使用，那么并行程序中私有变量的终值就需要在对应并行程序段结束前复制给共享变量，如表 2-9c 第 6-8 行语句，这就是终值复制问题。

表 2-10、数组的数组扩张和私有化变换说明

(a) 串行程序	(b) 数组扩张	(c) 数组私有化变换
1 REAL A(1000)	8 REAL A(1000)	20 REAL A(1000)
2 A(1) =0.0	9 REAL XA(1000, N)	21 A(1)=0.0
3 DO I = 1, N	10 A(1)=0.0	22 DOALL I = 1, N
4 DO J = 2, M	11 DOALL I = 1, N	23 LOCAL REAL PA(1000)
5 A(J)= A(J-1) +F(I,J)	12 XA(1,I) = A(1)	24 PA(1) = A(1)
6 ENDDO	13 DO J = 2, M	25 DO J = 2, M
7 ENDDO	14 XA(J,I) = XA(J-1,I)+F(I,J)	26 PA(J) = PA(J-1)+F(I,J)
	15 ENDDO	27 ENDDO
	16 ENDDOALL	28 IF (I .EQ. N) THEN
	17 DO J = 2, M	29 DO J = 2, M
	18 A(J)= XA(J,N)	30 A(J)= PA(J)
	19 ENDDO	31 ENDDO
		32 ENDIF
		33 ENDDOALL

2.3.3.3.3 数组私有化变换与暴露集

当需要私有化变换的临时变量是数组时称为数组私有化变换。对于临时变量识别、初值和终值复制的问题，数组私有化会需要比标量私有化变换更复杂的计算。

临时数组在每个循环迭代内使用的数组元素值或者是在该循环迭代内定义的，或者是在循环外定义的，而不能在其它的循环迭代中生成。表 2-10a 所示的串行程序中，从写引用的角度看，在每个循环迭代内，数组段 A(2:M)[2]都被重写，而数组元素 A(1)的值始终不变；从读引用的角度看，在每个循环迭代内使用的数组段 A(2:M-1)的值都在本次循环迭代内生成，而使用的数组元素 A(1)的值是在循环外产生的，所以数组 A 是 I 循环中的临时变量。考虑了初值和终值复制后，表 2-10b 和 c 分别给出数组扩张和数组私有化变换结果。

2 数组元素的集合可以有不同表示方式，最常见的是正则段和不等式组。但任何求解数组元素集合以及关于这类集合上的运算都是近似的或只对某些情况有效。不论如何选择表示方式都不影响本文后面将介绍的相关-覆盖方法，所以我们可以将集合的表示及其运算看成是实现的细节。为了简洁直观，本文的示例均用正则段表示：数组段 A(1:M)表示集合{A(1),...,A(M)}，当 M<1 时集合为空。

对于串行循环体中同时存在读写引用的标量，如果所有读引用都被本循环体内的写引用覆盖(先写后读)则可以认为该标量为可私有化的临时变量，否则，如果标量的某个读引用暴露于本循环体的写覆盖之外，则必然被其它循环迭代的写引用覆盖，从而存在跨循环迭代的真正意义上的流相关，该标量便不是可以私有化的临时变量。

类似地，数组私有化时也可以利用暴露集的概念去识别可私有化的临时数组。

定义 2-1:在程序段 P 中被读引用的某个数组元素，如果该元素先写后读，那么称该数组元素为 P 中**非暴露的**；否则称该数组元素为 P 中**暴露的**。P 中所有暴露的数组元素构成 P 的**暴露集**。

暴露集是所有数组私有化判定方法的重要概念。非暴露的数组元素的值是程序段中产生的，而暴露元素的值是在程序段之外产生的。如在表 2-10a 串行程序中，数组元素 A(1)对所有循环迭代都是暴露的，而其它 A(2:M)中的数组元素则对于所有循环迭代都是非暴露的。

2.3.3.3.4 数组私有化基本判定准则

如前所述，为了并行循环的数组私有化识别，我们必须找出循环中所有可私有化的临时数组，其中每个元素要么对所有循环迭代都是暴露的，要么对所有循环迭代都是非暴露的。换句话说，如果临时数组的元素在某个循环迭代内被读引用，它要么在本次循环迭代不暴露，要么不能被前面的循环迭代定义。据此有下面数组私有化识别的基本准则。

基本准则：在循环 L 中，如果对任意 2≤k≤BL 均有$UE_{L_k} \cap \bigcup_{i=1}^{k-1} Set_{L_i}$（$W_{L_i}$）=∅，那么数组是可私有化的。

如在表 2-9串行程序中，$Set_{\text{I 循环体}}(W_{\text{I 循环体}})$= A(2:M)，$UE_{\text{I 循环体}}$={A(1)}。所以每个迭代的写集都是 A(2:M)，每个迭代的暴露集都是{A(1)}。由于{A(1)}∩A(2:M)=∅，所以 A 是 I 循环的可私有化数组。

推论：在循环 L 中，如果对任意 2≤k≤BL 均有UE_{L_k} =∅，那么数组是可私有化的。

2.3.3.4 数组归约(Reduction)

复杂的数组归约识别是并行化研究的另一项实用技术，将它同么模变换理论相结合，具有理论和实践的重要意义。因为对归约语句的并行化处理可以采用特定的技术，所以在进行么模变换之前，可以不考虑归约引起的数据相关，这就扩展了么模变换技术的适用范围。

定义 2-2：假设 S 为语句,a 为变量(标量或数组元素), e 为表达式，⊕为运算符，若下列条件成立：

1. ⊕为具有单位元且满足结合律及交换律的运算符。
1. 有或经变换后具有形式 a＝a⊕e ；

那么，称语句 S 具有归约形式，⊕为归约运算符, a 为归约变量, 称 e 为归约表达式。

定义 2-3：循环 L 中具有归约形式的语句，若满足下列条件，则称为循环 L 的归约语句：

1. 归约变量与不具有归约形式的语句及所有的归约表达式在循环 L 中没有流相关；
1. 若两归约变量之间在循环 L 中存在相关，那么它们的归约运算符相同。

定义 2-4：若归约语句本身或归约语句之间具有跨循环 L 的相关，则称归约语句对循环 L 有效，否则称其对循环 L 无效。

定义 2-5：循环 L 中所有同时满足定义 2-2、定义 2-3、定义 2-4的语句构成的语句集，称为循环 L 的归约语句集。归约语句集中语句之间的相关，称为循环 L 的归约相关。

若一个循环除了归约相关外没有跨循环相关，则可通过归约变形实现循环的并行化；若存在其它跨循环相关，可以考虑引入么模变换来消除这种相关，以实现并行化。

2.3.3.4.1 归约识别与变换

现在我们来详细地解释这些定义的含义。定义 2-2描述了实际程序中大量出现的一种基本现象—归约操作，也即不同的循环实例对某一存储单元，进行一系列可结合且可交换的数据操作。操作的结合律和交换律保证了无论用什么顺序执行循环，归约变量的最终结果都不改变[3]。对归约语句而言，循环虽然可以乱序执行，但却不能并行(同时)执行，否则将产生语义上的歧义。因此归约语句可以通过设置临界区获得一定的并行性，但其并行度小于可完全并行的语句。定义 2-2给出了归约操作的迭代性、结合律和交换律。

表 2-11、归约识别

<table>
<tr>
<td>DO I = 1,9,1
A[I]=A[I]+F(I)
A[I+1]=A[I+1]+G(I)
ENDDO

a 归约语句</td>
<td> 
b 归约数组 A 的读写区域</td>
</tr>
<tr>
<td>DO J=I,M
DO I=I,N
A[J]=A[J]+3; ……s1
A[J]=A[J]*5; …… s2
B[g(J)]=B[g(J)]+f(A[J]); …… s3
ENDDO
ENDDO
c 不满足定义 2-3的归约形式单句</td>
<td>DO I=1,N
A(I)=A(I)+C(I) …… s1
ENDDO

d 不存在跨循环相关的归约语句</td>
</tr>
</table>

例如：表 2-11(a)所示的循环中，两个语句都为归约语句，循环对归约数组 A 的若干元素进行累加操作；表 2-11(b)中的阴影部分表示数组 A 被读写的元素，从中我们可以看到，数组区域 A[1:9]上的每一数组元素都被写过两次，且无论如何交换循环的执行次序，每个数组元素 A(I)的最终结果都为:A(I)的初值+G(I)+F(I)。但若两个循环并行执行，将导致两个循环实例同时执行一句归约语句，因此结果有可能被错算为 A(I)的初值+G(I)或 A(I)的初值+F(I)，因此该循环不能并行执行，但可以用么模变换的技术获得一定的并行性。

循环按任意顺序执行虽然可以保证归约变量终值的正确性，但不能保证中间结果的正确性。定义 2-3保证了归约变量的中间结果除归约操作外不再被引用。若不满足定义 2-3中的第一个条件，则说明归约变量的中间结果将被引用。若不满足定义 2-3中的第二个条件，那么两个归约语句的归约变量相同而归约运算符不相同，而不相同的归约运算符之间没有相互的交换律和结合律，因此不能保证循环乱序执行的正确性。例如：表 2-11c 中的 s1 与 s2 都满足定义 2-2，但它们都不是归约，因为 s2 的产生值在 s3 中被引用，不满足定义 2-3的第一条；s1 到 s2 有流相关，但两者的操作符不同，不满足定义 2-3的第二条。而 s3 为归约语句。

归约语句应该对某些存储单元反复读写。定义 2-4说明了这一点。例如：表 2-11d 中的 s1 语句是一个归约语句，但它对每个数组元素只进行了一次读写，每个循环实例对不同的数组元素进行读写，因此

[3] 这仅仅是理论上的保证，实际上改变执行顺序有可能带来误差积累。

我们没有必要将它看成归约，该语句不影响循环本身的并行循环。

前面我们谈到归约语句具有一定的并行性，在并行执行循环的不同迭代时只要保证归约语句不被同时执行，就可以保证结果的正确性。

表 2-12、归约变换

DOACROSS I=1,9 Critical_section_A A(I)=A(I)+F(I) end section Critical_section_A A(I+1)=A(I+1)+G(I) end section ENDDOACROSS a 利用加锁进行归约变换	REAL A1(1:10,1:9) DOALL I=1,p,1 /*p 为处理机个数 */ DOI$_1$=1,10 A1(I$_1$,I)=0 /*0 为"+"的单位元*/ ENDDO DO J=$\lfloor 9/p*(I-1)\rfloor$+1,$\lfloor 9/p*I\rfloor$,1 A1(J,I)=A1(J,I)+F(J) A1(J+1,I)=A1(J+1,I)+G(J) ENDDO ENDDO DO I=1,p,1 DO I$_1$=1,10,1 A(I)=A1(I$_1$,I)+A(I) ENDDO ENDDO b 利用数组扩张来进行归约变换

现在我们介绍两种使含归约的循环并行化的归约变换方法。

第一种方法就是让相应循环直接并行化，仅对归约单句加锁使归约变量的操作互斥即可。为保证最大并行性，我们只需对同一组归约语句，即变量名和操作符相同的归约语句，加相同的锁，对不同组的归约语句加不同的锁。表 2-12a 描述了这种方法。

加锁操作需要并行语言的支持，但有些并行计算机的并行语言不支持锁操作，这时我们需采用变量扩张技术。这种技术对每一组归约，都为每个并行进程分别引入的一个临时数组，该临时数组每个元素的初值为归约运算的单位元；然后各进程完全并行执行，并将每组归约的部分结果记录在各自的临时数组元素内；在并行循环结束后，再通过每个的临时数组的部分结果求出归约语句的最终结果。应该注意的是，变量扩张方法在引入临时数组前需要计算出临时数组的大小，当归约变量是数组而我们又无法确切计算出临时数组的大小时，我们可以根据整个归约数组的大小来进行数组扩张。表 2-12b 描述了这种方法：其归约运算符为+，归约运算的初值为 0，数组的归约区域为 A[1:10]。

2.3.3.5 cache 优化与数据局部性利用

Cache 的容量较主存的容量小得多，如果 cache 中的数据在被重用之前就已经被置换出 cache，那么 cache 就得不到很好的利用。通过循环合并(loop fusion)、循环分块(loop strip-mining)、循环颠倒(loop reversal)等技术加强数据的局部性，使 cache 中的数据在被置换出 cache 前尽可能得到重用，能够提高单处理机系统的性能。在共享主存的多处理机系统中，各处理机有自己的局部 cache，因此要提高 cache 的命中率，不仅要考虑各处理机局部 cache 中的数据在被置换出 cache 前尽可能得到重用，还要考虑在处理机的调度策略下如何使得对同一存储单元内容或相邻存储单元内容的访问操作尽量集中在同一处理机执行。

为了利用循环程序中的数据局部性，人们开发了一系列循环 cache 优化技术。当并行执行某一循环时，不同处理机对同一存储单元内容或相邻存储单元内容的引用操作有可能引起真共享或假共享 cache 行的抖动。一旦 cache 抖动现象发生，程序并行执行与串行执行的加速比不仅达不到理想值，严重时，程序并行

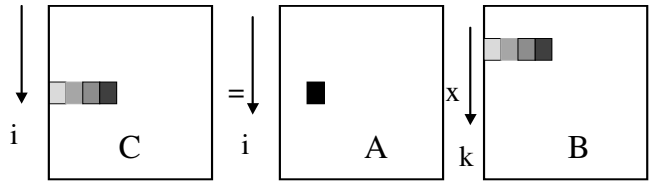
执行的时间甚至会超过程序串行执行的时间。由于 cache 抖动严重影响了并行处理系统性能的发挥，多年来，并行处理领域的研究人员一直在努力寻求加强数据局部性，减少以至消除 cache 抖动的各种技术，如循环的交换、块化、错位、加边等方法，并取得了一定的成效。

以表 2-13矩阵乘法为例，由于 i,k,j 三重循环中 i,j 循环都可并行化，如果选择内层 j 循环并行化，则会导致相邻共享数组单元(在同一 cache 行中)被不同处理机使用，从而导致所谓 cache 行抖动；反之，选择并行化外层 i 循环，因为不同处理机使用的数据相距较远，减少了 cache 行的冲突或重用，可以很好地避免 cache 抖动。

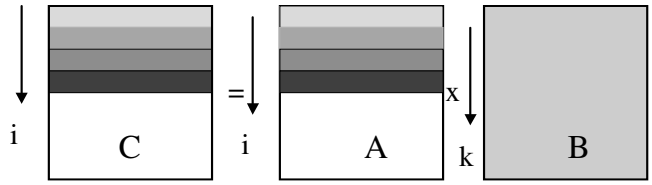
表 2-13、多处理机上循环的数据局部性利用示意：前四个并行循环迭代

<table>
<tr>
<td>

```
DO i=1,n
   DO k=1,n
      DOALL j=1,n
         c(i,j)=c(i,j)+a(i,k)*b(k,j)
      ENDDOALL
   ENDDO
ENDDO
```

</td>
<td> 
</td>
</tr>
<tr>
<td>

```
DOALL i=1,n
   DO k=1,n
      DO j=1,n
         c(i,j)=c(i,j)+a(i,k)*b(k,j)
      ENDDO
   ENDDO
ENDDOALL
```

</td>
<td> 
</td>
</tr>
</table>

3 PROFPAT 的设计与实现

3.1 PROFPAT 的设计目标

并行程序设计交互环境 PROFPAT 结合自动并行化编译技术和用户交互与一体，用来提高目标程序的性能。它应达到下列设计目标：

- 帮助程序员继承过去岁月里积累的串行程序资源。
- 减轻程序员开发并行程序的负担。
- 减少在不同并行计算机间移植代码的难度。
- 帮助程序员理解和分析程序的并行语义，更有效地利用优化和并行化技术。
- 提供研究并行化编译技术的平台，验证各种并行化技术的正确性和有效性。

3.2 PROFPAT 的主要功能概述

为了达到以上这些设计目标，PROFPAT 必须将许多有用的交互或自动编程工具，如程序编辑、程序并行语义分析和显示、并行和优化程序变换、并行程序性能分析等等，有效地集成在一个统一的程序设计交互环境中，如图 3-1所示的数据流图比较了并行程序设计环境 PROFPAT 和自动并行化编译工具 AFT 功能上的不同。

在 AFT 中程序员很难交互地干预自动并行化过程, 与 AFT 不同, PROFPAT 的功能包括更多的用户交互。用户反馈能修正系统对程序的不精确分析，从而导致更有效的程序并行化。

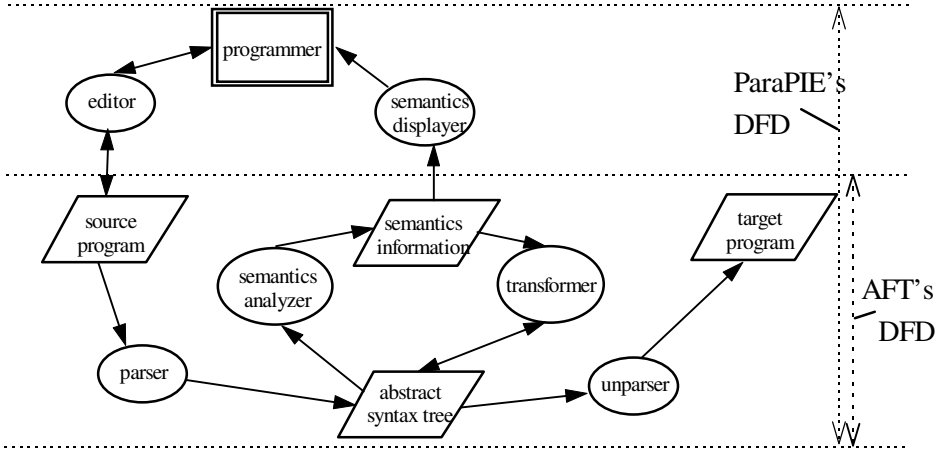


图 3-1、并行程序设计环境PROFPAT和自动并行化编译工具AFT数据流图的比较

3.3 并行程序设计环境的实现方法

3.3.1 环境集成方法

我们在 PROFPAT 中试图基于并行化技术，实现基于数据分解的程序内部中间表示，并在保证用户界面独立性和环境集成一致性的原则下为用户提供帮助开发高性能并行程序的一系列工具。

具体地说，在 PROFPAT 中通过项目数据库统一地存取、管理和维护程序的各种分析信息，如正文，表格，图形等。以一致的项目数据命名方法，保证了工具模块性好、耦合度低。以相同的图形用户界面风格设计所有工具，并为所有工具提供通用的正文和图形表示构造和处理模块。选用具有标准性、兼容性和开放性的支持软件开发，PROFPAT 在 UNIX 工作平台和 X 窗口系统下确保了充分的可移植性。

系统层次结构反映了基本服务和编程工具之间的集成界面层次，如图 3-2所示。.编程工具利用集成框架的服务，这些服务调用底层支持软件实现。

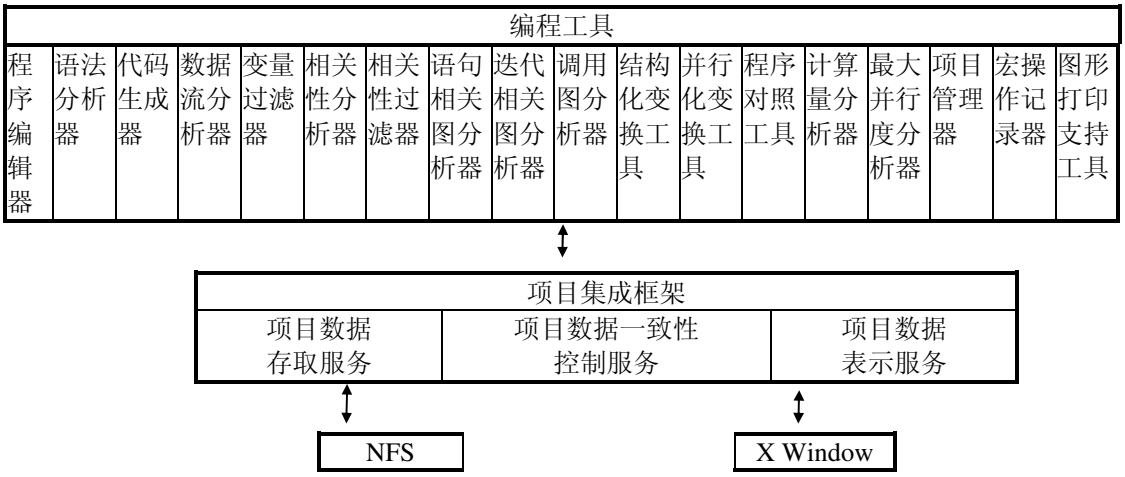


图 3-2、PROFPAT的系统结构

在自动并行化编译过程中，*中间表示*(IR) 反映了工具互相之间交换信息的不为用户所见的内部数据结构。在交互编程环境中，用户不可见的许多中间表示 IR 必须转换为终极用户可以见到的*外部表示*(ER)，从而允许程序员通过直接修改外部表示 ER 干预中间表示，辅助和引导系统完成各种功能，如图 3-3所示，反映了 PROFPAT 工具与程序员之间的相互关系。

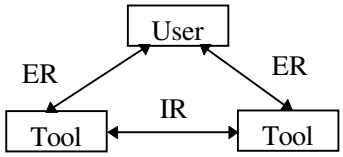


图 3-3、PROFPAT系统工具与程序员之间的关系

表 3-1反映了 PROFPAT 的功能侧面。在象 AFT 这样的自动并行化编译器中，没有考虑如何让用户干预系统的自动分析和变换过程。与 AFT 不同，PROFPAT 的功能涉及更多用户交互，从用户得到的反馈能用来修正系统对程序的解释，导致对程序更有效的并行化。

表 3-1、PROFPAT系统集成的主要工具和数据集

工具名称	利用的中间表示	可视化的外部表示	用户交互操作
程序编辑器 Program Editor	程序正文缓冲区	程序正文	浏览和编辑程序正文
语法分析器 Parser	抽象语法树	串行源程序	打开文件并初始化项目数据表
代码生成器 Unparser	抽象语法树	并行目标程序	选择目标代码类型来保存文件
数据流分析器 Dataflow Analyzer	变量数据流信息	变量信息表	指定私有化变量和数组范围

变量过滤器 Variable Filter	同上	同上	过滤出感兴趣变量信息
相关性分析器 Dependence Analyzer	相关变量对、语句对及相关向量等数据相关性信息	变量相关信息表	删除虚假相关对、检测递归相关语句
相关性过滤器 Dependence Filter	同上	同上	根据语句范围、变量范围、相关类型过滤出感兴趣相关信息
语句相关图分析器 Stmt. Dep. Analyzer	语句对数据相关性信息	语句相关图	语句相关图浏览、过滤查询
迭代相关图分析器 Iter. Space Analyzer	跨循环迭代数据相关性信息	循环迭代相关图	为模拟执行提供符号值，迭代相关图浏览和过滤查询
调用图分析器 Callgraph Analyzer	过程调用关系	过程调用图	查找过程
结构化变换工具 Structurizer	语法树	结构化的源程序	对指定程序结构化
并行化变换工具 Parallelizer	结构化程序的语法树、数据流和数据相关信息	程序段是否可并行化及可套用的并行化技术	用指定并行化技术对指定程序段并行化或自动并行化
程序对照工具 Program Contraster	源-目标语法树的语句对照关系	语句对照表	对照变换前后的源程序和目标程序
计算量分析器 Profiler	插入计时语句	过程及循环计算量分析表	数据排序和统计图表显示
最大并行度分析器 Max.Para.Detector	插入映射变量和计时语句	循环最大并行度表	同上
项目管理器 Project Manager	工具及外部数据项间依赖关系	项目数据项表	调用相应工具新建、修改项目数据项
宏操作记录器 Macro Recorder	用户操作事件序列	操作宏记录文件	记忆和重放一个操作序列
图形打印支持工具 Graphics Printer	绘图命令原语	Postscript 格式文件	生成、显示和打印各种图形

3.3.2 程序中间表示

AFT 读人 Fortran 源程序，经语法分析后将其转换为中间表示，所有的分析与变形都在该中间表示上进行，最后将其反语法分析后还原为并行 Fortran 程序。在 AFT 中，中间表示主要由以下几个数据结构组成：函数表，符号树，变量表，标号表。它们的关系如图所示。对于每个函数或过程，都建有对应的函数表，函数表除了给出该函数的一些特征，如函数名，虚参名，全局及局部变量名等等之外，还指出了该函数对应的语法树，变量表，标号表等等。语法树中记录了该函数或过程的函数体的语法信息，它将过程体中每一条语句都转换为语法树中的一棵子树，它如实地反映了源程序中的语法结构。语法树是整个中间表示中的主体。变量表记录该函数或过程中的变量的特性，如变量是否同名，数组的维数及上下界等等。标号表则管理该函数或过程中的标号。

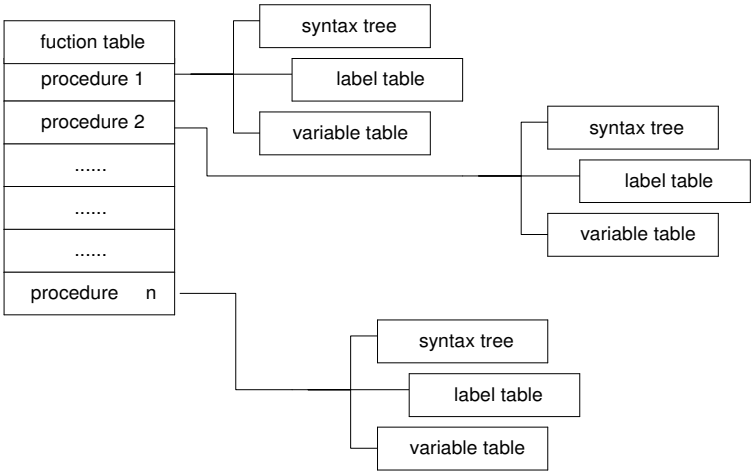


图 3-4、AFT的主要中间表示

在并行程序设计环境 PROFPAT 系统中，程序中间表示是由为表示程序语法和语义信息而设计的数据结构组成的。这些数据结构包括：抽象语法树，符号、过程和 DO 循环表，数据相关图，循环迭代空间相关图，过程调用图。其中抽象语法树、过程表、标号表、变量表是 AFT 的中间表示，如图 3-4所示。DO 循环表，数据相关图，循环迭代空间相关图，过程调用图是为了与用户交互而提供的外部表示。

抽象语法树(AST)是用来表示源程序和目标程序语法信息的关键数据结构。它是由程序语法分析器计算得到的程序结构。一旦分析出抽象语法树，就可以用各种树操作修改抽象语法树来反映对程序的重构。另外，许多有用的语义信息已经加入抽象语法树，如循环界限、数组维数、常数值等等。为了方便同程序员交互，PROFPAT 还把语句行号信息加入抽象语法树的语句结点。有了行号，程序员可以清楚地告诉系统所关心的是哪一条语句。当程序重构时则记录对应的原程序行号信息，提示程序员对照变换前后的程序。

符号表中存放程序中变量或标识符的名字和值的信息，以及如读/写，可私有化等属性。过程表是对抽象语法树的补充，它为每个过程记录对应子语法树及其他有用信息。在 DO 循环表中列出了程序结构化后系统找到的所有 DO 循环以及循环计算量、最大潜在并行度等性能分析调试信息，以供应用循环并行化变换技术时参考。

数据相关图(DDG)来表示数据相关性分析的结果。 在语句相关图中，数据相关被看成是语句对之间的关系。用图形结点表示语句，结点上的标记说明语句行号。用图形弧边表示数据相关关系，边上的标记是关于语句之间跨循环方向相关向量及相关变量对的信息。边上的不同颜色可以区分不同相关类型。迭代空间相关图是语句相关图的补充，用图形结点表示迭代循环体，用图形弧边表示跨循环相关关系，边上的不同颜色可以区分不同相关类型。它提供了对 DO 循环体存在的跨循环数据相关的生动写照，可以用来向程序员说明某些循环并行化变换前后的效果。

过程调用图是对过程间调用和被调用关系的图形表示。这些关系对于理解调用过程与被调用过程之间跨过程影响是非常有用的。由于 FORTRAN 程序不允许递归调用，它的过程调用图不包括回路。通过对有向无环图的自动布局，用户可以看到程序自上而下的调用关系。

3.3.3 用户界面设计

作为并行程序设计交互环境，PROFPAT 为了将中间表示信息可视化，以使程序员能与系统交互，把 PROFPAT 系统应用窗口组织成几个窗口工作区域：菜单条、弹出菜单、状态栏、程序编辑窗口和对照窗口、正文信息窗口、表格编辑窗口、图形编辑窗口。

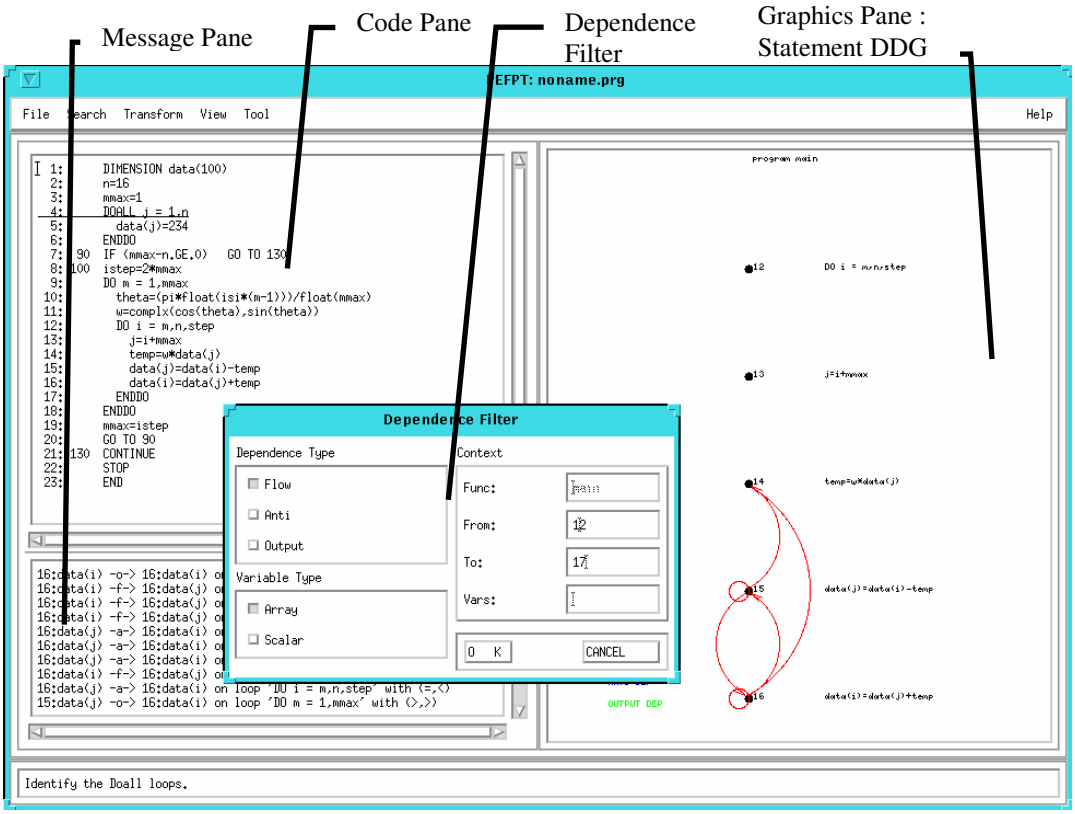


图 3-5、PROFPAT的主窗口用户界面:正在显示的程序是FFT.F

最顶层的程序编辑窗口是分析程序的出发点，如图 3-5所示。它的左工作区类似程序员熟悉的程序正文编辑浏览工具，它的右工作区是图形画板，显示语义可视化工具的图形输出，如各种数据相关图、过程调用图等等，如图 3-5的语句数据相关图、图 3-7的过程调用图、图 3-8的循环迭代数据相关图所示。

只要选定一个程序，其程序源代码就显示在左边的程序正文区，其图形语义信息就显示在右边的图形区。窗口底部的消息区提示程序员系统反馈的数据表格或程序工具执行结果。通过窗口顶部的主菜单条可以调用系统不同工具的功能，如循环变换工具、程序分析工具及其他有用的编程工具。交互查询对话功能和在程序语句、过程调用图、数据相关图等不同信息之间的导航功能有力支持了系统的环境集成。工具间信息的一致性则由底层的项目管理工具支持。

此外，为了更好地跟踪说明并行化变换前后的源程序语句和目标程序的对照关系，我们还实现了一个程序对照主窗口，如图 3-6所示，上、下两个编辑浏览窗口分别显示变换前后的源程序语句和目标程序，强调显示的语句行是变换前后的对应语句。

3.3.4 利用并行语义信息

3.3.4.1 信息的可视化和导航

目前系统中支持的可视化信息有：源程序和目标程序(图 3-6)，过程调用图(图 3-7)，语句相关图(图 3-5)，数据相关表(**表 4-3**)，迭代空间相关图(图 3-8，图 3-12)，过程和循环计算量条形和饼形统计图(图 3-9, 图 3-10)，变量数据流信息表(**表 4-4**)，可私有化数组表(**表 4-4**)，最大潜在并行性表(图 3-11)等等。

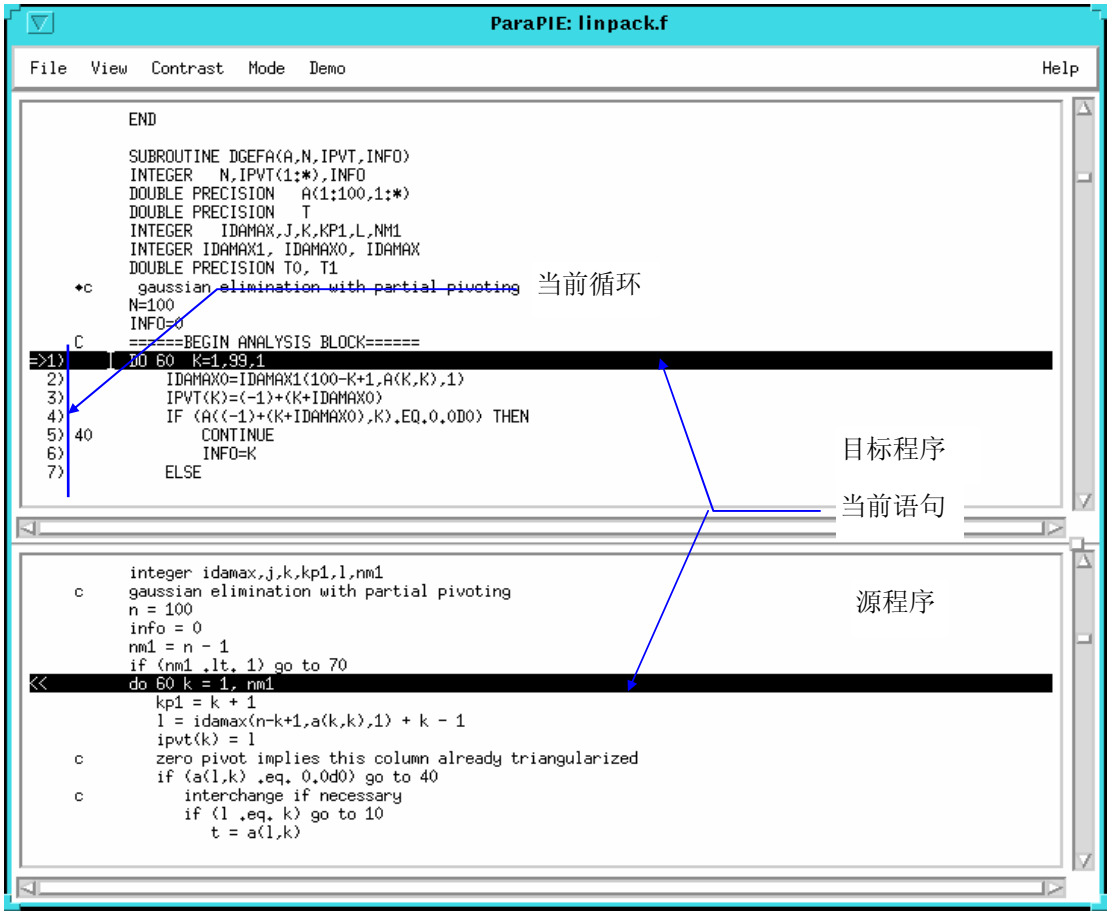


图 3-6、PROFPAT的程序对照工具窗口用户界面：正在显示的是Linpack.f

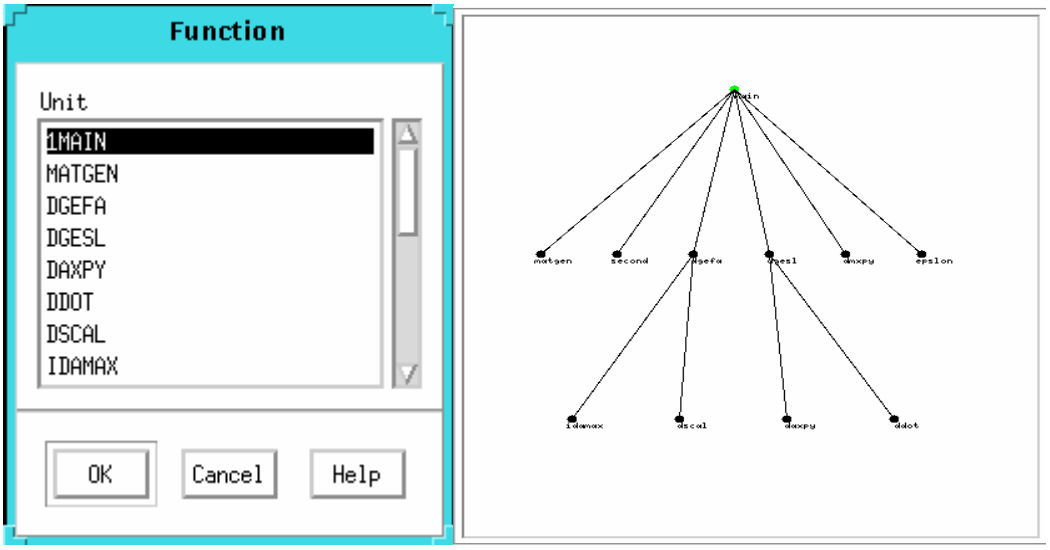


图 3-7、PROFPAT对程序LINPACK.f显示的过程表及过程调用图

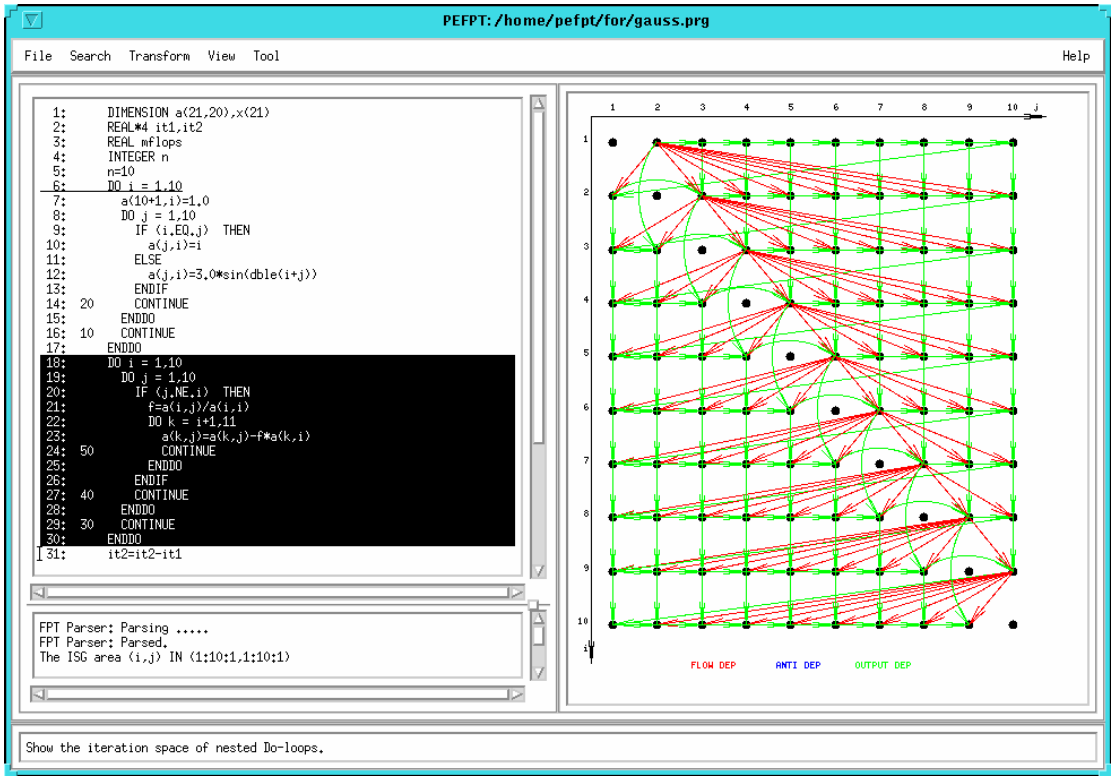
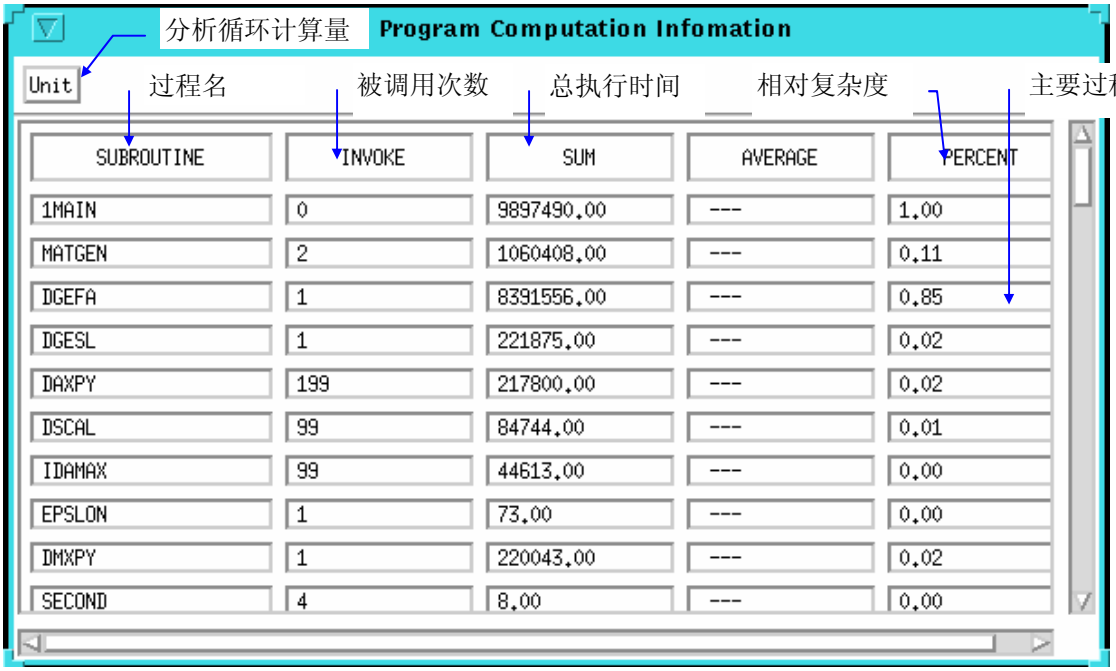


图 3-8、PROFPAT对程序GAUSS.f显示的循环迭代空间数据相关图



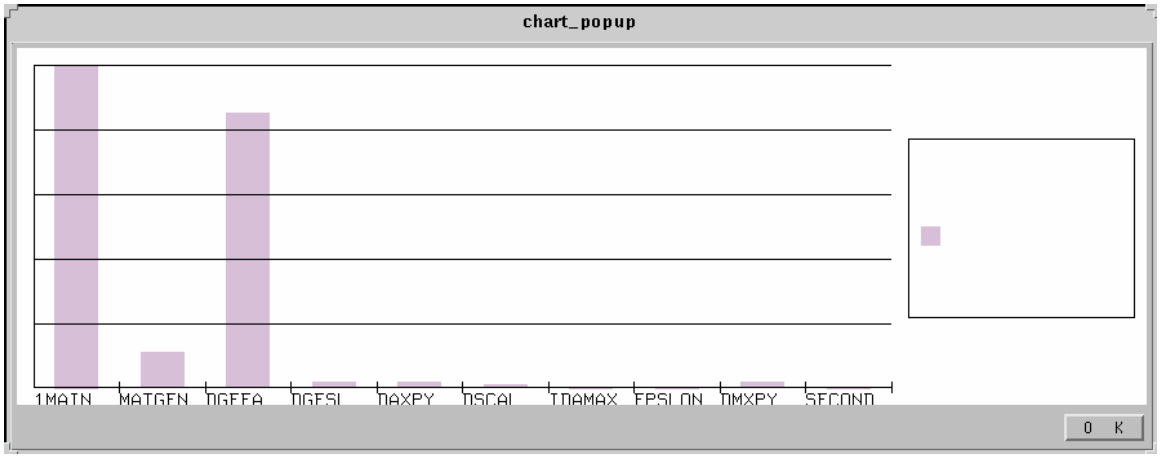


图 3-9、对linpack.f程序计算量分析的过程计算量分布表和分布图，
从中可以清楚看到占据主要计算量的过程是DGEFA

因为不同类型的可视化信息之间互相有联系，上下文相关的导航机制可以帮助用户知道怎样查找和获取相关的信息。例如在程序编辑窗口中的光标位置能告诉用户他正在访问程序的哪个部分。用户提供过程名和语句行号的部分信息可以查找和查看程序的其他部分，使用循环表可以在程序前后循环之间遍历。利用语句对照表可以在两个对照窗口中查看变换前后的对应语句。在图形窗口中单击调用图结点时，程序窗口能够自动将光标移动至相应过程的代码行；单击语句相关图结点或相关边时，程序窗口能够刷新显示相应的代码行或在正文信息窗口中显示详细的相关性信息。

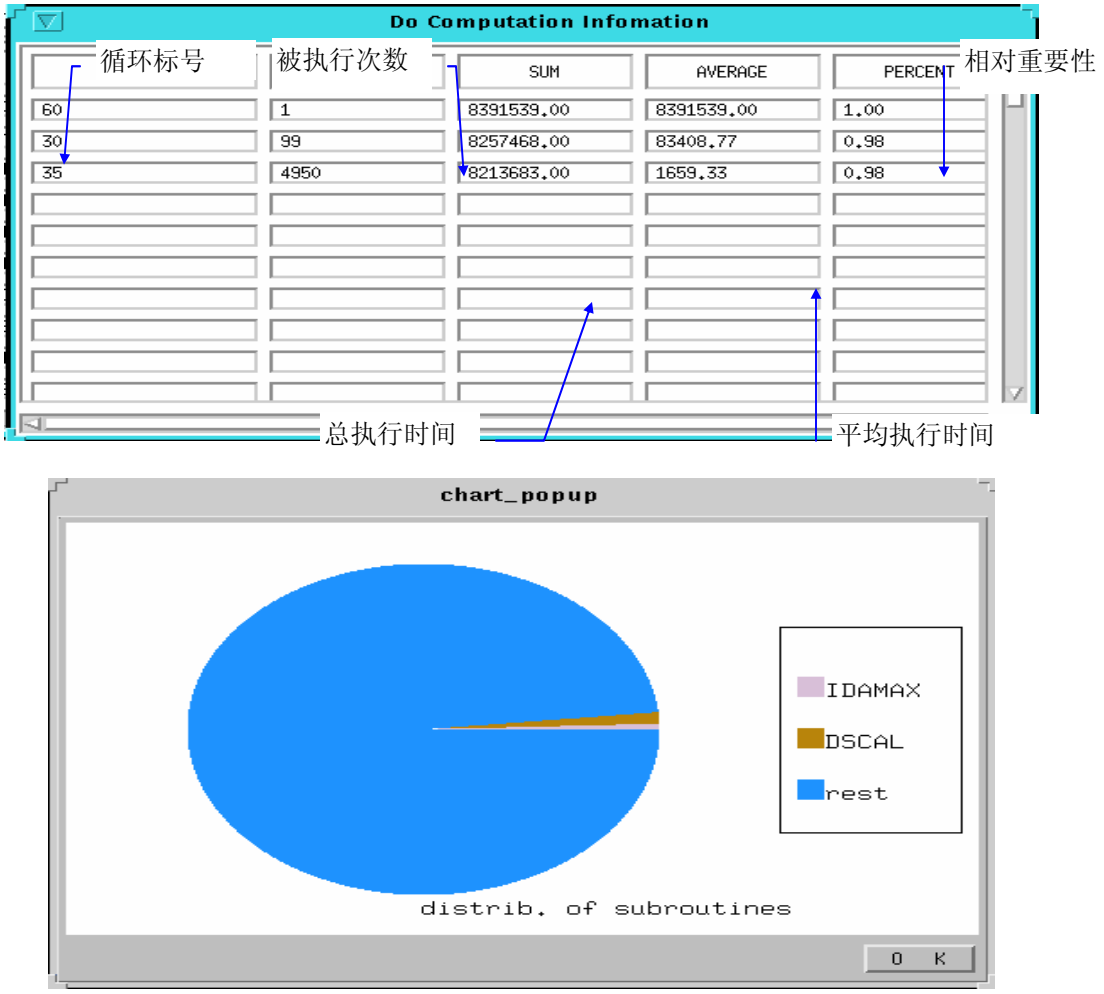


图 3-10、对过程DGEFA程序计算量分析的循环计算量分布表和子过程计算量分布图，从中可以清楚看到其子过程IDAMAX和DSCAL不占据主要计算量

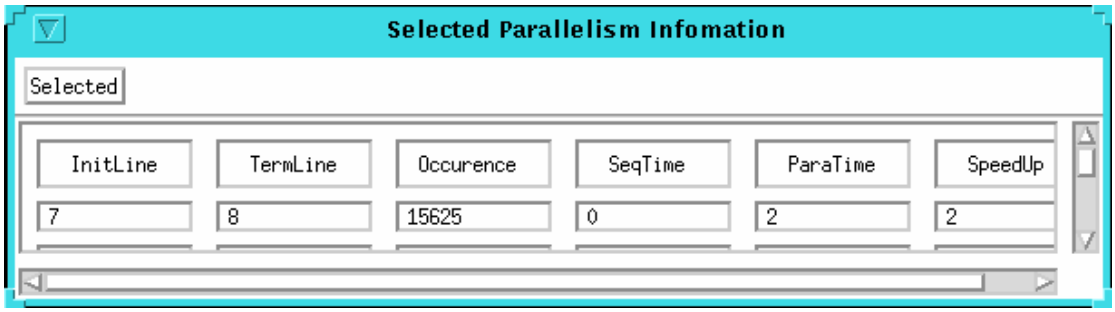


图 3-11、对MXM1.f的两个串行语句(语句7和8)分析最大潜在并行性，从中可以清楚看到它们可以并行执行，最大操作级潜在并行性为2

3.3.4.2 信息的过滤

如果程序语义信息太多，让用户一下子无法识别，PROFPAT 使用信息过滤机制来帮助用户观察他所关心的信息。例如，根据相关类型、语句范围可以从众多相关边中过滤出用户关心的数据相关信息，使数据相关图得到更清楚的显示。通过选定部分程序段或循环，可以作该程序段的动态计算量和最大并行性分析。

3.3.4.3 信息的修正

如果一些自动计算出来的数据相关实际并不存在，只是由于系统做了保守估计才错误得出的。这时候程序员提供一些诸如符号值间关系的提示，就可以使系统根据这些启示重新对数据相关性进行更精确的计算。另外，用户也可以在数据相关图中直接删除某些实际不存在的相关边，来引导后续的并行化变换。通过在数组表格中指定可私有化数组，可以将数组私有化变换应用于更多的情况。

3.4 主要工具的实现方法

3.4.1 串行源程序的各种测量分析

PROFPAT 为源程序的测量分析提供了程序计算量分析工具、潜在并行性分析工具、循环并行性分析工具、以及可私有化数组分析工具。对于程序员来说，这些工具对并行化串行大程序是很有用的分析工具。这些程序分析工具的实现方法是在程序中自动插入时间检查点的方法来获得所需测量结果的。下面就逐一讨论这些重要功能的实现。

3.4.1.1 分析程序计算量

由于大多数 Fortran 应用程序的规模都很大，绝大多数实际应用程序由上千行代码组成，如果并行程序设计者试图把程度的各个部分都尽可能地并行化，这种不分轻重缓急的办法必然会造成工作量急剧增大，而且由于注意力的分散，很可能对程序中计算量大的部分的并行性挖掘得不够深，而把精力消耗在一些小计算量部分的并行化上，这样就可能使并行化后的效果不好。程序员很难完全分析程序所有的部分。而实际上程序总体加速比是所有程序单元加速比关于程序计算量的加权平均，程序计算量较高的程序单元能否取得加速比对整个程序性能的提高是更关键的，需要程序员更仔细地加以分析。因此，如果有了关于程序单元程序计算量的信息，程序员就可以专注于那些对程序总体性能产生重要影响的耗费计算时间的程序单元。

程序设计的经验表明科学计算程序的大量计算时间更多地是用在循环语句中。因此分析程序之前不仅需要知道过程的程序计算量，也需要知道循环的程序计算量。

程序计算量分析工具的主要思想就是在不改变源程序语义的情况下，设置一时钟参数，在每个待分析过程及 DO 循环的开始与结束点分别插入计时变量和计时语句，将对各个子程序和 DO 循环的执行进行时钟累计计数，在程序执行后得到统计结果。最后结果包括各子程序被调用次数、累计执行时间和占总执行时间的百分比，各 DO 循环累计进入次数、执行时间和占所在程序单元执行时间的百分比，在 PROFPAT 中所得的数据表格如图 3-9所示。对其实现方法简要描述如下：

- 对 DO 循环和子程序定义相应的数据结构

因为我们对程序计算量进行的分析主要是针对 DO 循环和子程序而言，因此必须有相应的数据结构来标识不同的循环和子程序，这里我们是用链表进行索引的。在程序运行时，把进入当前结构的系统时间和退出当前结构的系统时间分别写入相对应的数据结构中去进行计算。

- 对模拟系统时间建立相应的说明与初始化

如果采用真实的机器时间进行记录及计算，我们发现许多程序中的大多数结构的运行时间都比较短，在

系统时间表示精度不高的情况下运行时间都接近于 0 ，造成最后的结果不能精确地比较不同的两部分的计算量。鉴于这一原因，我们定义了一个模拟系统时间，另外对每种运算的时间开销进行估算。在程序运行时，对模拟系统时间进行累加和记录，这样，在进行计算量比较时会得到更为直观和准确的结果。

- 对程序中所有语句的分类处理

程序计算量分析工具主要处理的语句分为：ASSIGN、IF 、DO 、CALL 、GOTO、STOP，见表 3-2。

表 3-2、对程序中所有语句的分类处理

主要语句	处理方法
ASSIGN	对运算时间开销进行计算，累加到模拟系统时间上
IF	对条件判断语句的开销进行计算，累加到模拟系统时间上。
DO	在 DO 语句前，当前模拟系统时间进栈，在 DO 语句后将相应的时间差累加到标识 DO 循环的数据结构中去。
CALL	类似于 DO 语句。不过模拟系统时间也作为 CALL 语句的一个参数传入子程序中。
GOTO	由于在 DO 循环中出现的 GOTO 语句会造成储存时间的栈的状态混乱，因此我们必须对 GOTO 语句跳转的 DO 循环的层次进行扫描，对栈进行相应的修改以保持一致性。
STOP	对程序非正常出口的情况下增加输出结果语句。

- 初始化

在程序的开始部分增加一些输出结果的初始化语句。

- 输出结果

在主程序的结束语句前加入将计算结果输出到指定文件的语句。

3.4.1.2 分析最大潜在并行性

程序段操作级最大潜在并行性决定了该程序段在理想并行化时可能达到的最大加速比。最大潜在并行性信息告诉程序员哪些程序单元值得去并行化。在 PROFPAT 中我们实现了对给定输入数据时程序段的操作级最大潜在并行性的动态分析工具。

该工具主要采用了数据流计算机的基本思想：

- 对输入的串行源程序，为每种运算操作预先定义时间，用以模拟程序执行时间，并为源程序中每个变量定义两个时间映射变量：读映射和写映射，用来定义相应变量的最后读写时间，在操作数变量的基础上统计所有操作的时间的总和。对于 COMMON 变量，其映射变量也定义为 COMMON 变量。
- 计算相应的映射变量的值的语句是插入到源程序中的，如：

  C＝A op B

  则相应的映射变量由下列公式修改：

  $Twc = T_{op} + MAX(Twc,Trc,Twa,Twb)$

  Tra=MAX(Tra,Twc)

  Trb=MAX(Trb,Twc)

  或：若不考虑非本质的输出相关和反相关，改为

  $Twc = T_{op} + MAX(Twa,Twb)$

  然后将以上语句插入源程序中。

这种对时间映射变量的修改反映了数据相关，对控制相关的处理类似。经过上述处理结束后的输出结果是一个增改过的 Fortran 源程序，它可以被编译，与预定义的并行性检测库(PDT)连接，再串行执行，对

计算时间进行模拟。执行结果除原程序运行结果外，还跟踪计算各个映射变量的值，这些时间映射变量值事实上构成了在抽象机上对程序全体操作的最合理调度，其结果可得到模拟的原程序串行执行时间和在抽象机上并行执行时间，两者之比，即为原程序在抽象机上的加速比，也就是原程序固有的理想并行度。

如果某程序或程序的某部分的内在并行性很小，那么我们可以寻求新方法重新编写源程序；或者虽然有可观的理论并行度，但是在真实机器上没有得到很好的挖掘，那么也许我们能重新安排数据结构和程序控制结构，以便程序员仔细分析识别，找出合理利用并行性的途径。

3.4.1.3 分析循环并行性以及可私有化数组

程序中的大量并行性往往存在于耗费计算量的循环语句中。通过前面所述的程序计算量分析工具，可以发现那些耗时的循环，通过前面所述的最大潜在性分析工具，可以告诉程序员这些循环是否有必要和值得进一步分析和并行化。

接下来对 DO 循环并行化时，循环级并行性是十分关键的启发信息。如果循环不存在跨循环的流相关，该循环可以并行化为 DOALL 循环。

但是如果存在跨循环反相关和输出相关，要使这些潜在可 DOALL 并行化的循环正确执行，必须应用某些程序变换，如数组扩张和私有化消除这些非本质相关性。变量私有化是循环并行化实践中最有效的技术之一**[43][72][73]**，通过为每个共享变量在并行执行的每个处理机上复制一个私有变量，就可以消除该共享变量引起的非本质数据相关。

在 PROFPAT 中，我们通过分析跨循环数据相关性找出可完全并行化的 DOALL 循环以及潜在可私有化数组。

3.4.1.3.1 实现原理

首先，我们为了监控程序执行，设置一个计时时钟，每经过一次操作将它递增一个时间单位。对循环中的每个形如：

$V = OP(V_1,...,V_n)$

的赋值语句，如果 $T_{loop}<TWV_i<T_{iter}$ (i=1,...,n), 就存在一个由变量 V_i 引起的跨循环流相关。类似地，要知道变量 V 是否可私有化，必须分别检查 $T_{loop}<TWV<T_{iter}$ 或 $T_{loop}<TRV<T_{iter}$，才能知道是否存在跨循环输出相关或反相关。其中 T_{loop} 和 T_{iter} 分别记录进入执行当前循环和当前循环迭代的时间。对变量 V 来说, TWV(TRV)是根据模拟时钟记录的最近一次写(读)该变量的时间映射变量**[50][52]** 。

3.4.1.3.2 分析循环迭代相关性的实现方法

我们在系统中可以采用两种实现方法：静态分析和动态模拟。与静态模拟方法相比，动态模拟方法对程序的分析比较精确。虽然动态模拟方法耗时较长，不适合在自动并行化工具中用来分析整个程序，但是可以结合在 PROFPAT 中，通过程序员交互分析一个程序段，根据需要用不同角度观察循环迭代相关图。为了程序员可视化分析循环及跨循环数据相关，我们针对要分析的循环构造一个局部数据结构来刻划迭代空间相关图**[58][59]**。通过模拟执行的方法将计算得到的跨循环数据相关填入该数据结构中。一旦用户选择了待分析循环，PROFPAT 首先向程序员交互地收集需要分析的循环迭代空间边界以及符号变量的数值，然后通过计算每个循环迭代所引用的变量集合和比较不同循环迭代对变量的引用集合是否相交来得出跨循环相关信息。

3.4.1.3.2.1 从程序员交互获得交互视图：

- 指出待分析循环程序段，程序员可通过选择最外层循环指出需要分析的循环迭代空间。
- 指出需察看的循环迭代空间的边界，程序员可为每层嵌套循环提供虚拟的下界和上界，从而指定需要分析的循环迭代子空间，减少模拟计算时间。
- 指出需显示的循环嵌套，由于实现的循环迭代数据相关图是二维图形，所以支持同时显示 1 维到 2 维的循环嵌套。这相当于对多维循环迭代空间的二维投影。如果待分析循环是多于 2 重，我们可以通过分析其中的任意两重循环嵌套间是否存在跨循环数据相关，供程序员参考。

```
1   首先对循环中出现的变量 A 的每个元素说明对应读/写时间的映射变量 RA(i1,...,im)和
    WA(i1,...,im)，该变量记录最近发生引用的循环迭代，标量可以看作一个单元素数组。
2   将时间映射变量初始化为最小值ϕ。
3   FOREACH 循环迭代 I1,...,In，计算循环迭代空间 I1,...,In 的索引值 Key=IStoKey(I1,...,In)
4   FOREACH 循环体中的变量引用点 R，计算每维的引用下标表达式 S1,...,Sm：
5   IF R 是读引用，
      IF WA(S1,...,Sm)=ϕ THEN
      无跨循环相关性，CONTINUE
      ELSEIF WA(S1,...,Sm)≠Key THEN
        WA(S1,...,Sm)→Key 之间存在跨循环流相关性，记录中间结果：
        KeytoIS(WA(S1,...,Sm))，I1,...,In，A，S1,...,Sm ，FLOW
      ENDIF
      IF RA(S1,...,Sm)=ϕ THEN
        无跨循环相关性，CONTINUE
      ELSE
        RA(S1,...,Sm)→Key 之间存在跨循环输入相关性[4]，记录中间结果：
        KeytoIS(RA(S1,...,Sm))，I1,...,In，A，S1,...,Sm ，INPUT
      ENDIF
      RA(S1,...,Sm):=Key
    ENDIF
    IF R 是写引用，
      IF RA(S1,...,Sm)=ϕ THEN
      无跨循环相关性，CONTINUE
      ELSE
      RA(S1,...,Sm)→Key 之间存在跨循环反相关性，记录中间结果：
          KeytoIS(RA(S1,...,Sm))，I1,...,In，A，S1,...,Sm ，ANTI
      ENDIF
      IF WA(S1,...,Sm)=ϕ THEN
        无跨循环相关性，CONTINUE
      ELSEIF WA(S1,...,Sm)≠Key THEN
        WA(S1,...,Sm)→Key 之间存在跨循环输出相关性，记录中间结果：
        KeytoIS(WA(S1,...,Sm))，I1,...,In，A，S1,...,Sm ，OUTPUT
      ENDIF
      RA(S1,...,Sm):= ϕ
      WA(S1,...,Sm):=Key
    ENDIF
```

[4] 输入相关一般不影响循环并行化，但是作为中间结果，这里输入相关能帮助精确计算反相关。

3.4.1.3.2.2 根据交互视图自动生成测试程序：

3.4.1.3.2.3 编译执行测试程序，获得中间执行结果供 PROFPAT 分析

通过 Fortran 77 编译上述生成的测试程序，同预定义库相连结，然后执行，结果存放在中间临时文件中，在提供 PROFPAT 循环迭代空间可视化工具之前还要利用输入相关结果修正反相关结果，使之正确：

1 以前面的中间结果为输入，维护相关记录表 **D={(<I_1>，<I_2>，<var>，<subscripts>，<type>)}.**

2 FOREACH 相关记录，(<I_1>，<I_2>，<var>，<subscripts>，<type>)∈**D**, 依次

IF<type> = 'FLOW' or <type> = 'OUTPUT' THEN

D=D - { (*，*，<var>，<subscripts>,INPUT) }

ELSE<type> = 'ANTI' THEN

D=D∪{ (<I_1'>，<I_2>，<var>，<subscripts>,ANTI)|

(<I_1'>，<I_1>，<var>，<subscripts>,INPUT)∈**D**}

D=D∪{ (<I_1'>，<I_2>，<var>，<subscripts>,ANTI) |

(<I_1>，<**I'**$_1$>，<var>，<subscripts>,INPUT)∈**D**}

ENDIF

3. 删除所有剩余的输入相关 **D=D -** { (*，*，*，*,INPUT) } ，向可视化工具提供整个修改的相关记录表。

3.4.1.3.2.4 通过可视化工具和过滤查询工具将跨循环数据相关图显示给程序员

以上得到的是一个总的跨循环数据相关图，包括关于所有变量的所有类型的数据相关。同时处理的子循环迭代空间是多维。而在可视化工具中我们向程序员显示的迭代空间实际上是多维迭代空间 $\mathbf{I_n}$ 在 L_1 和 L_2 二维的投影 $\mathbf{I_2}$：

$$\mathbf{I_2}=\{(x,y) \mid (i_1,...,i_{l1},..,i_{l2},...,i_n)\in \mathbf{I_n} \wedge i_{l1}=x \wedge i_{l2}=y\}$$

所以只要 $\mathbf{I_n}$ 有跨循环相关：$((I_1,...,I_n)$，$(J_1,...,J_n)$，<var>，<subscripts> ，<type>)，且

$$(I_{l1},I_{l2}) \neq (J_{l1},J_{l2})$$

则在 $\mathbf{I_2}$ 有跨循环相关：$((I_{l1},I_{l2})$ ，(J_{l1}, J_{l2})，<var>，<subscripts> ，<type>)。循环迭代数据相关图可视化工具需要显示的就是上述信息。

相关图过滤查询工具则为程序员提供了根据变量、相关类型过滤数据相关图的查询工具。

3.4.2 语义信息的可视化

为了使过程调用图、语句数据相关图、迭代空间数据相关图等程序语义信息易于为程序员理解，PROFPAT 集成了若干可视化工具。通过可视化图形，程序员能更直观地了解给定程序的语义关系**[58][59][68][69]**。图 3-5、图 3-7、图 3-8分别给出了 PROFPAT 中的语句相关图、过程调用图和迭代空间相关图的可视化图形。

程序的过程调用图是对过程之间调用和被调用关系的图形表示。PROFPAT 的过程调用图可视化工具有助于程序员更好地理解程序的跨过程语义。为了实现过程调用图在图形窗口中的可视化， 由调用图布局算法计算出几何坐标信息，附加在程序过程调用图数据结构中。调用图布局算法根据过程调用层次自顶向下地放置过程结点，使得被调用过程结点垂直位置总是位于调用过程结点之下。通过过程结点之间的自上而下的联线表示出过程调用关系。

数据相关图(DDG) 有助于程序员理解循环并行化的原理**[12][64]**。PROFPAT 集成了两类数据相关图可视化工具：语句相关图和迭代空间相关图。前者展示语句之间的数据相关性，后者展示跨循环的数据相关性。

将数据相关分析结果以语句相关图形表示的实现难点在于如何清晰地组织语句相关性。我们的语句相关图布局算法的实现策略是首先将语句结点按语句行号顺序垂直排成一列(之所以不采用类似控制流图的复杂布局，是因为考虑到实现效率以及信息的清晰)，然后通过带有方向箭头的曲线联接彼此相关的语句结点。不同着色的相关曲线表示出表示相关性类型的不同，不同曲度的曲线则避免图形上同色相关

边的相交。

为了程序员可视化分析循环及跨循环数据相关，我们构造一个局部数据结构来刻划迭代空间相关图**[58][59]**。通过模拟执行的方法将计算得到的跨循环数据相关填入该数据结构中。一旦用户选择了待分析循环，PROFPAT 首先向程序员交互地收集需要分析的循环迭代空间边界以及符号变量的数值，然后通过计算每个循环迭代所引用的变量集合和比较不同循环迭代对变量的引用集合是否相交来得出跨循环相关信息。循环迭代相关图采用的是二维图形显示，所以当分析多重循环时，每个图形结点实际凝聚表示一组循环迭代的集合(类似于投影)，而两个相关图形结点之间的迭代相关边则反映了它们代表的迭代集合之间存在的跨循环数据相关。

当屏幕中同时显示上百个语句结点或迭代结点对之间的相关边时就很难看清图形，为了避免过于复杂的数据相关图误导程序员对程序的直观理解，PROFPAT 中必须引入信息过滤机制来帮助程序员专注于他所关心的信息，隐藏他不关心的信息。为此，通过在两个语句对和迭代结点对之间的相关边实际上压缩概括不同变量相关对的集合，通过信息对话过程实现由程序员提供指定相关类型、相关语句、相关循环或相关程序段等的过滤方法。

```
PARAMETER(m=10,n=10)
        DO j1=1,ceil(m/2)
          DO j3=1,ceil(n/2)
              u1=floor(alog(float(m/(2*j1-1)))/alog(2.))
              u2=floor(alog(float(n/(2*j3-1)))/alog(2.))
              DO jj2=0,u1+u2
                 DO jj4=0,max(0,jj2-u1),min(jj2,u2)
                    j2=jj2-jj4
                    j4=jj4
                    i1=(2*j1-1)*2**j2
                    i2=(2*j3-1)*2**j4
                    a(2*i1,2*i2)=a(i1,2*i2)+a(2*i1,i2)
                 ENDDO
              ENDDO
          ENDDO
        ENDDO
```

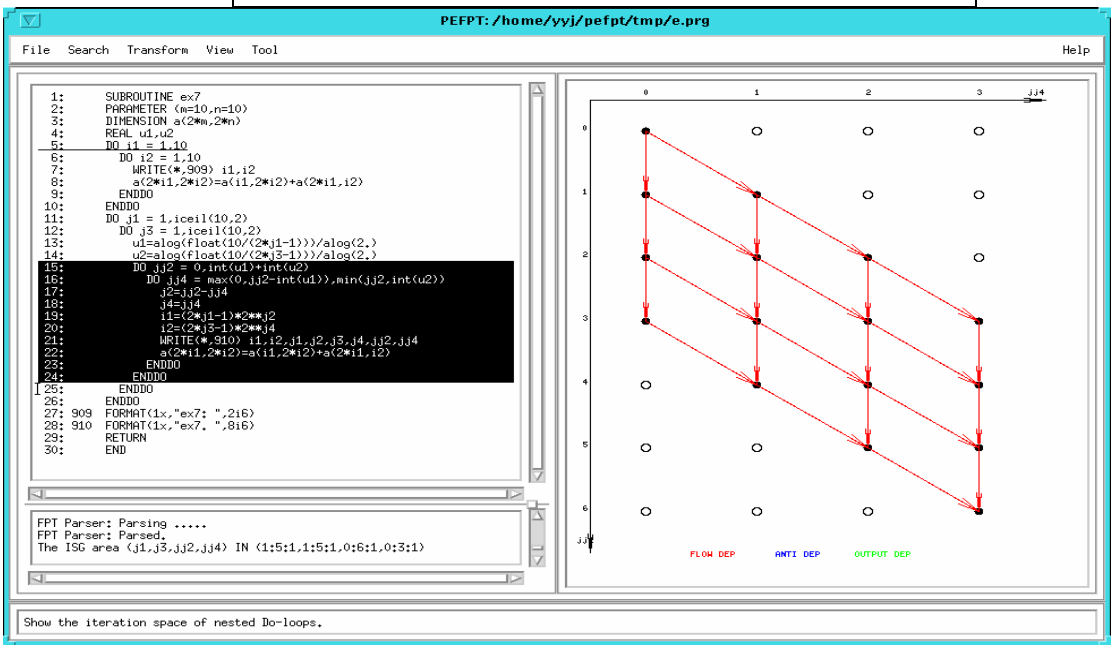


图 3-12、PROFPAT显示循环迭代数据流相关的可视化图形

以上是一个循环迭代数据相关图的程序例子，从它的循环迭代数据相关图（图 3-12）中我们可以看到二维循环迭代空间的数轴表示，内层循环为 X 轴方向。外层循环为 Y 轴方向，坐标点表示循环迭代空间在二维的投影集合，实心点表示确实执行到的循环迭代，空心点表示虽在程序员观察范围之内但并未执行到的循环迭代。三种不同颜色用于区分流相关、反相关和输出相关，而图 3-12中只显示了流相关，是因为过滤查询工具的作用。该相关图清楚标明在上述循环程序中内层循环 JJ4 是可以并行化的，而外层

循环 JJ2 则无法并行化。类似地，通过显示 JJ1，JJ3 的迭代相关图，可以看出 JJ1 和 JJ3 循环都是可并行化的。

```
PARAMETER(m=10,n=10)
DOALL j1=1,ceil(m/2)
  DOALL j3=1,ceil(n/2)
      u1=floor(alog(float(m/(2*j1-1)))/alog(2.))
      u2=floor(alog(float(n/(2*j3-1)))/alog(2.))
      DO jj2=0,u1+u2
        DOALL jj4=0,max(0,jj2-u1),min(jj2,u2)
          j2=jj2-jj4
          j4=jj4
          i1=(2*j1-1)*2**j2
          i2=(2*j3-1)*2**j4
          a(2*i1,2*i2)=a(i1,2*i2)+a(2*i1,i2)
        ENDDOALL
      ENDDO
  ENDDOALL
ENDDOALL
```

3.4.3 程序相关性分析和并行化变换

相关性分析是实现自动并行化编译工具和并行程序设计环境的关键功能。GCD 测试, Banerjee 测试**[12]**和扩展的 Banerjee 测试**[77]**可用来在编译时获得数据相关信息。但是程序中存在一些障碍，妨碍了精确的自动相关性测试，为了保证目标程序的正确性(与源程序计算结果一致)，并行化编译如果无法自动证明并行化变换的正确性，只好放弃使用该变换。自动并行化的这一保守性说明了自动并行化工具的固有弱点：即使实现了很强的变换技术，如数组私有化、么模变换技术等，也会由于自动分析时的信息不足或不完整而无法对实际程序应用。为此提出了许多针对性的信息分析技术，如符号相关性分析，跨过程常数传播，跨过程线性关系传播，指针别名分析等等，但每一项技术都有其适用范围和实现效率方面的局限性。如何通过交互手段弥补自动分析的不足，就成为提高程序并行化效果的重要方面。并行程序设计环境，如伊利诺伊大学的 Faust，莱思大学的 ParaScope，威斯康星大学的 ParaDyn，复旦大学和根特大学的 PEFPT/PROFPAT 纷纷开展了交互并行程序设计工具的研究。许多并行程序设计环境，如 ParaScope，Faust，PEFPT/PROFPAT 等等在编辑和浏览工具中集成了对语句数据相关图的可视化工具，但是除了 PEFPT/PROFPAT 之外，很少环境实现了循环迭代空间数据相关图的可视化工具。但是由于循环在程序计算量中举足轻重的重要性，以及循环并行度较高，因此循环变换是研究得最多、最彻底的一类程序变换。因此，循环迭代相关可视化工具对于交互理解和分析程序中的循环，决定采用合适的循环变换具有十分重要的作用：

而借助程序员交互提示弥补自动分析的不足，PROFPAT 能够更精确地分析数据相关性。一旦分析了数据相关性，并行化程序变换能自动对没有跨循环相关性的循环并行化，如果循环不能自动并行化，通过数据相关反馈信息，PROFPAT 会向程序员揭示妨碍并行化的原因。

基于数据相关性测试结果，PROFPAT 可以对程序应用多种自动并行化变换技术，如数组私有化**[43][72][73]**, 循环分布与合并**[74]**, 循环展开,循环分块, 循环交换**[61]**及么模变换**[11][24][66]**。程序员也可以提示系统选择合适的程序变换技术。

4 PROFPAT 的应用

本节说明了应用 PROFPAT 中交互工具的一般指导性步骤，介绍了 PROFPAT 一些主要交互功能的具体应用过程，最后报告了应用 PROFPAT 并行化 SPECfp95 测试包中三个自动并行化编译工具还不能并行化的串行程序的实验过程及实验结果。

4.1 用 PROFPAT 交互分析串行程序的一般应用步骤

一般而言，用 PROFPAT 分析串行程序的应用步骤如下：

1) 对给定输入数据使用程序计算量分析工具模拟执行串行程序，发现占据程序主要计算量的**子程序和循环**;
2) 对步骤 1 所发现的重要的待分析程序段(子程序和循环)，使用最大潜在并行性分析工具和循环并行性分析工具检测在给定输入数据时待分析程序段的潜在并行性;
3) 对步骤 1 和步骤 2 所发现的在给定输入数据时具有潜在循环级并行性的循环，使用自动并行化变换工具分析对任意输入数据是否可并行化;
4) 对无法自动并行化的循环, 使用数据相关性分析工具和各种语义关系可视化工具交互地分析该程序段对任意输入数据是否可并行化；
5) 借助语义信息的提示，考察合适的循环变换以便交互地分析和变换程序：

5.1) 如果数据相关图显示出程序段不包含跨循环数据相关，则不必修改循环体，该循环可以完全并行化为 DOALL 循环;

5.2) 如果数据相关图显示出循环中没有本质的跨循环数据流相关，使用可私有化数组分析工具发现在给定输入集合下的可私有化变量，然后验证这些变量对任意给定输入集合都是可私有化的。分析跨过程调用信息时可以借助过程调用图可视化工具直观地加以考察。应用指定数组私有化变换可以消除临时变量引起的非本质的相关性(反相关和输出相关)，从而可以施行并行化变换。

5.3) 如果循环迭代空间数据相关图显示出存在跨循环数据流相关，则通过交互的循环么模变换有时可以改变数据相关距离，从而使循环并行化;

5.4) 如果数据相关图显示没有循环级并行性，而只有语句级和任务级并行性，通过交互修改程序仍可以用并行程序段(parallel sections)利用这些并行性;

6) 对每个可能并行化但无法自动并行化的重要程序段，重复第 4 和第 5 步骤。

下面具体解释 PROFPAT 交互工具在数组私有化、相关性分析和么模并行化变换的初步应用。

4.1.1 交互相关性分析和并行化变换

相关性测试的目标是在保证目标程序的正确性(与源程序计算结果一致)的前提下验证实施程序并行化变换的正确性和可能性。当自动并行化编译如果无法自动证明并行化变换的正确性时，必然采取保守的假设，认为存在相关性，不能进行程序变换。例如：

```
k=ODD(k1)                          INTEGER FUNCTION ODD(K)
DO i=LOW,UP                        ODD=2*K+1
   a[4*i]=...                      RETURN
   ...=a[2*i+k]                    END
ENDDO
```

求解以上相关性问题，就是求解下列相关性联立方程：

$$4x=2y+k$$

如果不存在整数解，则一定无相关性。根据 GCD 测试原理，若等式

$$ax+by=c$$

有整数解，必须 gcd(a,b)|c。所以，如果 not gcd(a,b) | c，则无相关性。

现在对这个程序例子，如果不作跨过程符号分析，无法知道 not gcd(4,-2)|k 的事实，GCD 测试只能保守地判定相关性存在。

一般的说，在相关性联立方程 **AX=b 中，**如果 **A** 或 **b** 中出现符号变量，仍可进行带符号的相关性分析，但自动分析工具只有能证明无论符号变量取何值时都不产生相关性方能自动并行化。否则必须借助程序员交互提示自动分析工具该符号的取值条件，如图 4-1所示。

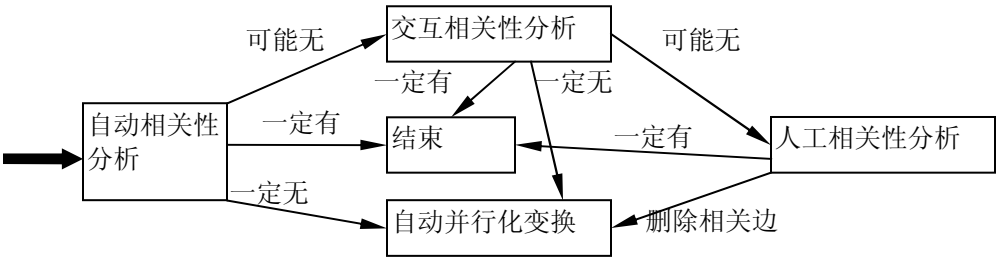


图 4-1、交互相关性分析流程

从程序员角度看，交互程序并行化变换是图 4-2这样的一个过程：

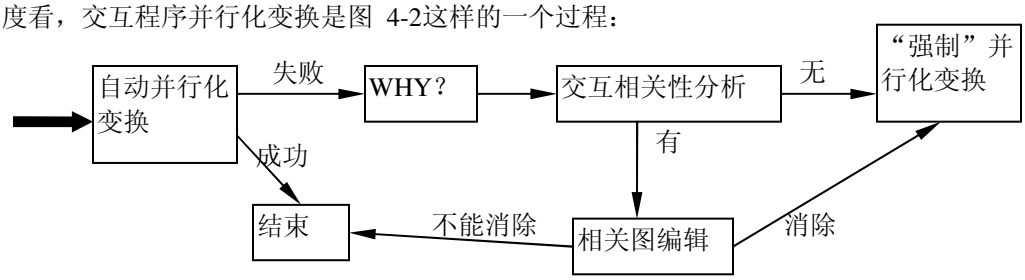


图 4-2、交互程序并行化变换流程

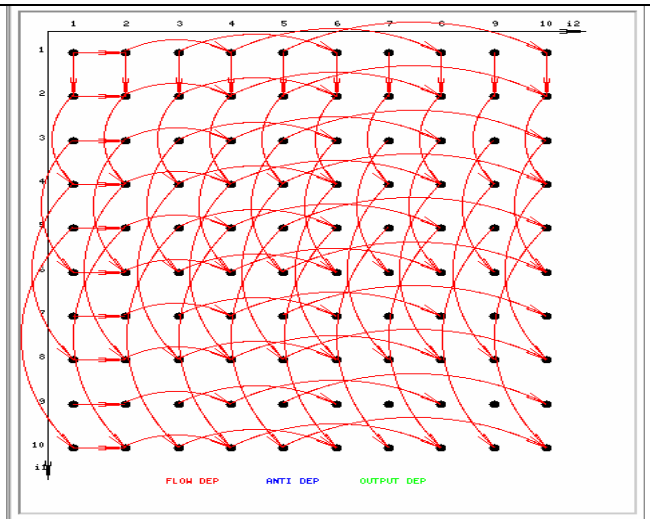
请看下面的例子，由于相关距离向量矩阵满秩，需要由程序员根据经验人工构造并行化变换。

表 4-1、循环交互并行化变换前后的程序及其迭代相关图示例

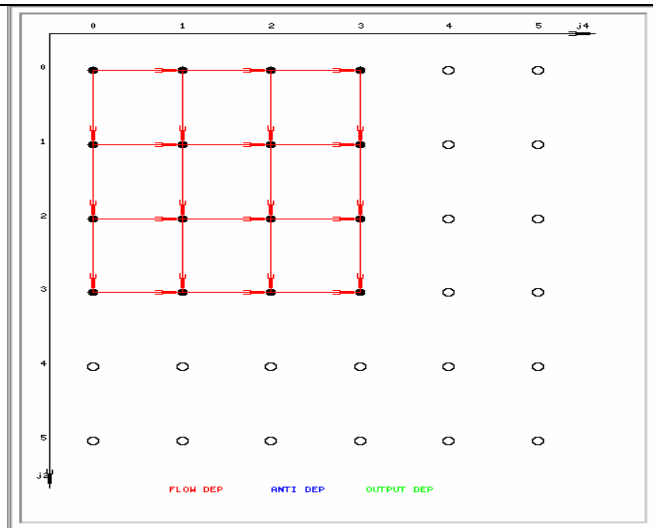
```
PARAMETER(m=10,n=10)
DO i1=1,m
    DO i2=1,n
        a(2*i1,2*i2)=a(i1,2*i2)+a(2*i1,i2)
    ENDDO
ENDDO
```

```
DOALL k1=1,ceil(m/2)
    DOALL k3=1,ceil(n/2)
        DO k2=0, floor(alog(float(m/(2*k1-1)))/alog(2.))
            DO k4=0, floor(alog(float(n/(2*k3-1)))/alog(2.))
                i1=(2*k1-1)*2**k2
                i2=(2*k3-1)*2**k4
                a(2*i1,2*i2)=a(i1,2*i2)+a(2*i1,i2)
            ENDDO
        ENDDO
    ENDDO
ENDDO
```

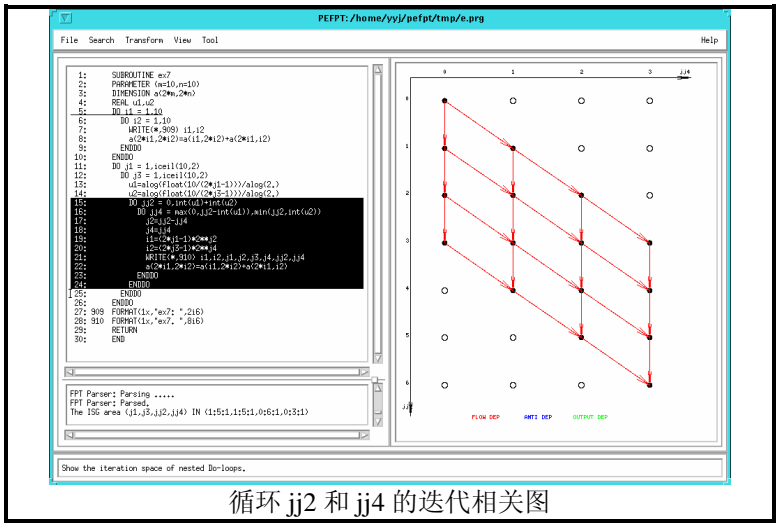
```
DOALL j1=1,ceil(m/2)
  DOALL j3=1,ceil(n/2)
      u1=floor(alog(float(m/(2*j1-1)))/alog(2.))
      u2=floor(alog(float(n/(2*j3-1)))/alog(2.))
      DO jj2=0,u1+u2
        DOALL jj4=0,max(0,jj2-u1),min(jj2,u2)
          j2=jj2-jj4
          j4=jj4
          i1=(2*j1-1)*2**j2
          i2=(2*j3-1)*2**j4
          a(2*i1,2*i2)=a(i1,2*i2)+a(2*i1,i2)
        ENDDO
      ENDDO
  ENDDO
ENDDO
```



循环 i1,i2 的迭代相关图



循环 k2,k4 的迭代相关图(k1=1,k3=1)



循环 jj2 和 jj4 的迭代相关图

4.1.2 交互数组私有化分析及并行化变换

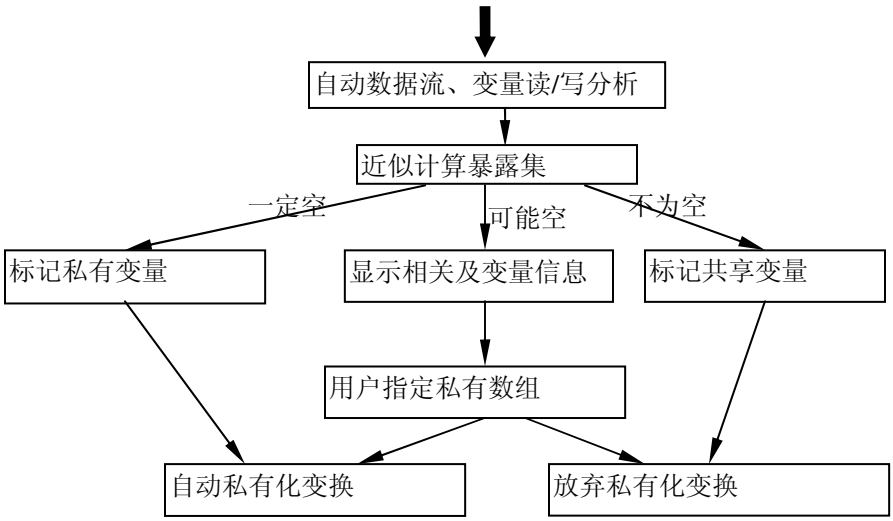


图 4-3、交互循环私有化变换流程

数组私有化变换技术的关键是识别可私有化数组，我们提出的相关-覆盖数组私有化方法可以较好地近似计算暴露集，原则上根据暴露集是否为空可以判定数组可私有化性。但是由于所有自动分析数组私有化方法是近似的：符号变量、不精确集合表示及数据流保守计算，使暴露集合计算存在不精确性。所以 PROFPAT 提供交互指定私有数组的功能，其使用流程如图 4-3所示。

为了交互数组私有化，当自动分析方法失效时，程序员首先观察循环迭代相关图，通过相关图过滤工具查询是否存在本质上的跨循环流相关，如果相关图中只有非本质的反相关和输出相关时，再逐个分析程序中的数组变量是否对所有输入皆可私有化，如果程序员认定某变量可私有化时便可以交互地修改变量信息，最后采用自动私有化变换工具生成代码。

例如，所示实例是截取自 Perfect 测试程序包**[18]**中的 NASI.F 的 ACTFOR 过程。

```
PARAMETER(NSP=10)
      DIMENSION XDT(2000),IND(2000)
      DO 240 I=2, NSP
         DO 235 J=1, I-1
            XD=RAND(0)
            XDT(J)=RAND(0)
            RSQ=RAND(0)
            RCUTS=RAND(0)
            IF (RSQ.LT.RCUTS)THEN
               IND(J)=1
            ELSE
               IND(J)=0
            ENDIF
235      CONTINUE
         L=0
         DO 236 J=1, I-1
            IF (.NOT.(IND(J).EQ.0)) THEN
               L=L+1
               IND(L)=J
            ENDIF
236      CONTINUE
         IF (.NOT.(L.EQ.0)) THEN
            DO 237 J=1, L
               XD=XDT(IND(J)-1)
237         CONTINUE
         ENDIF
240   CONTINUE
```

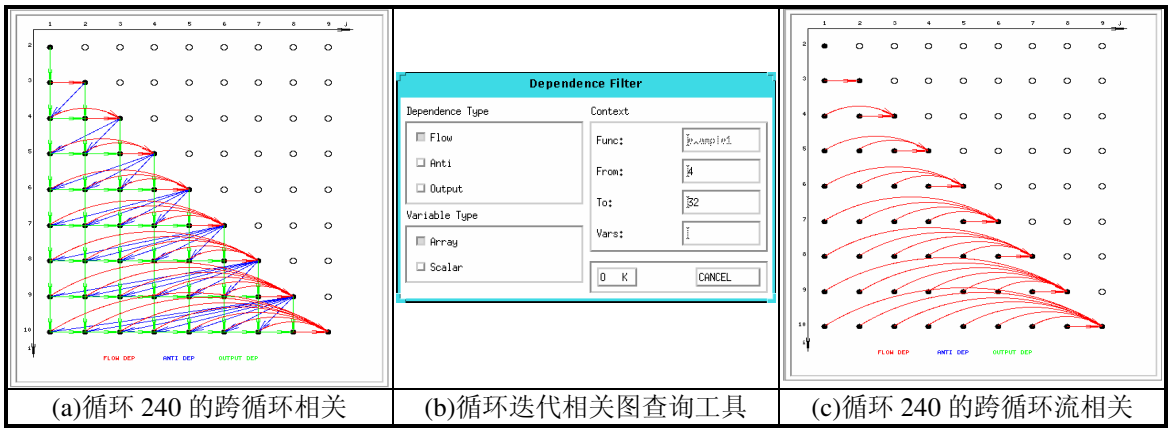
图 4-4、数组私有化示例

为了显示数组 XDT 在任意输入组合下的数据相关性，我们利用 NASI.F 提供的随机数发生函数产生乱数来替代 RSQ 和 RCUTS 的赋值表达式，以便能够执行到由不确定条件 RSQ.LT.RCUTS 控制的所有分支，分析下标数组 IND(J)取各种可能值时数组 XDT 是否可私有化。

DO 循环 240 的迭代相关图由 PROFPAT 的循环迭代相关图可视化工具显示，如表 4-2a 所示。其中存在许多跨循环相关边，但是在通过查询工具(如表 4-2b 所示)过滤掉数组私有化可以消除的反相关和输出相关后，我们发现外层循环中不再有任何跨循环相关了(如表 4-2c 所示)。因此，外层循环是可通过私有化加以并行化的。

也可以通过 PROFPAT 提供的相关性分析表工具分析 DO 循环 240 是可并行化的性质。首先通过相关表可视化工具显示循环中所有变量的数据相关，如表 4-3a 所示。表中每一条记录表示一对变量之间的一种数据相关。虽然从表 4-3a 中我们看到大量的相关记录，但是通过相关表格查询工具(如表 4-3b 所示)过滤掉不跨循环数据相关以及数组私有化可以消除的反相关和输出相关后，我们发现循环中不再有任何跨循环流相关了(如空表 4-3c 所示)。因此，如果外层循环是可私有化的，便可以并行化。

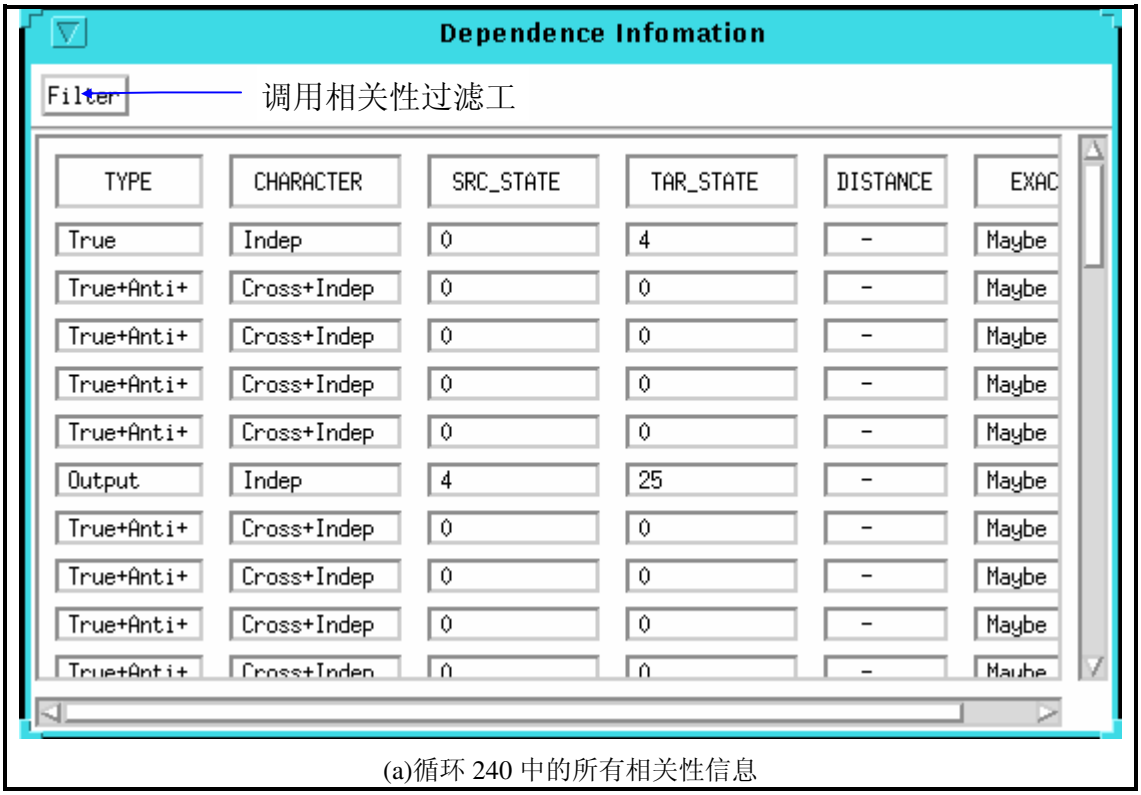
表 4-2、通过循环迭代相关图交互数组私有化分析



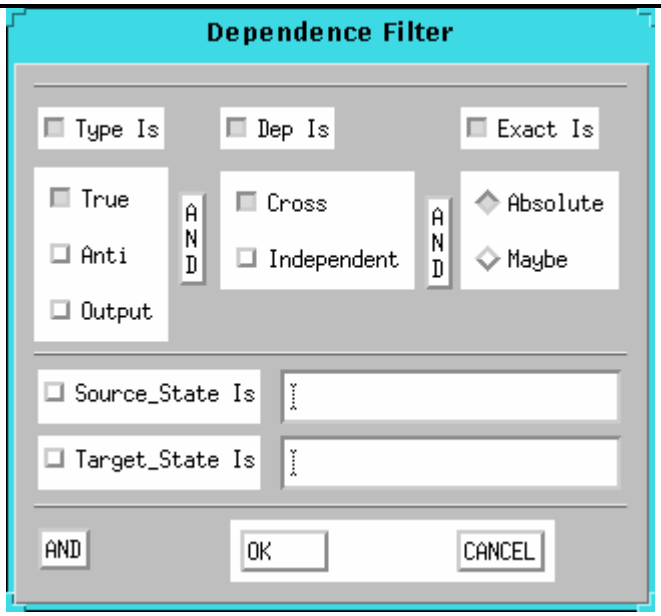
(a)循环 240 的跨循环相关 | (b)循环迭代相关图查询工具 | (c)循环 240 的跨循环流相关

通过上述交互分析发现循环 240 是可并行化的，也就是说其中的临时数组 XDT 和 IND 是可以私有化的。但是由于系统无法自动分析出上述可私有化数组(自动数据流分析结果如表 4-4a 所示)，在进行交互并行化变换时，程序员首先要指定这两个数组是私有变量而非共享变量(如表 4-4b 所示)，才可以使用系统的“强制”循环 240 并行化功能，如表 4-4c 所示。

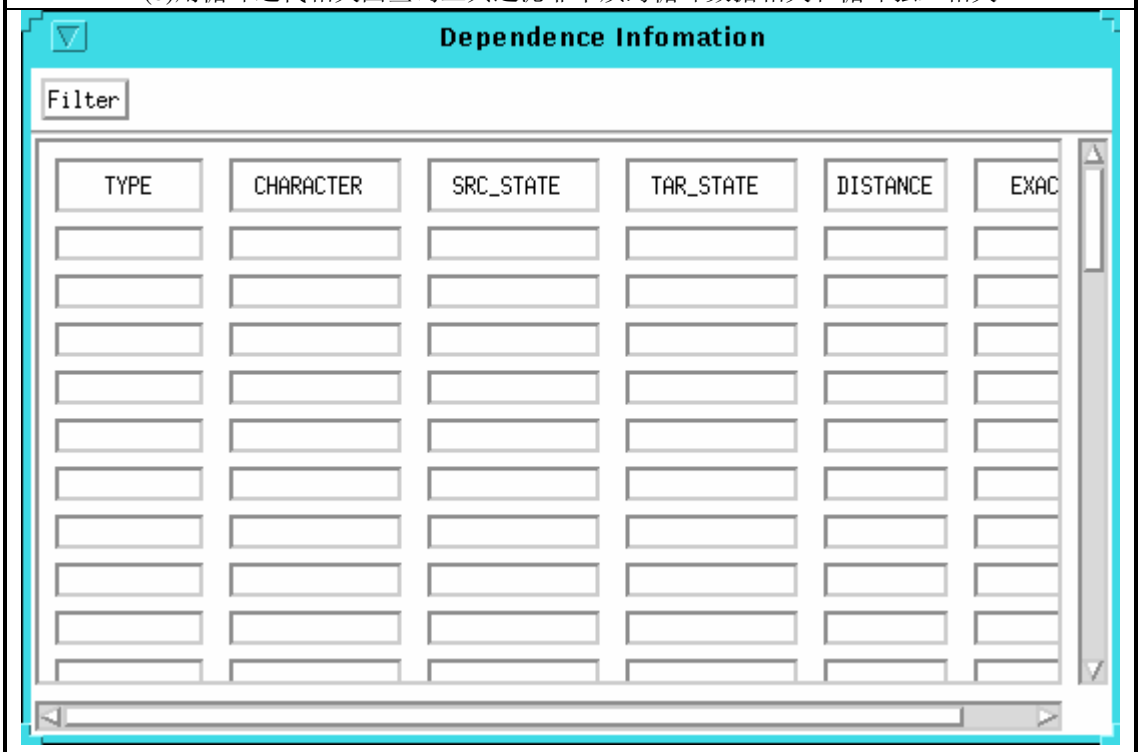
表 4-3、通过相关信息表交互查询工具进行数组私有化分析



(a)循环 240 中的所有相关性信息

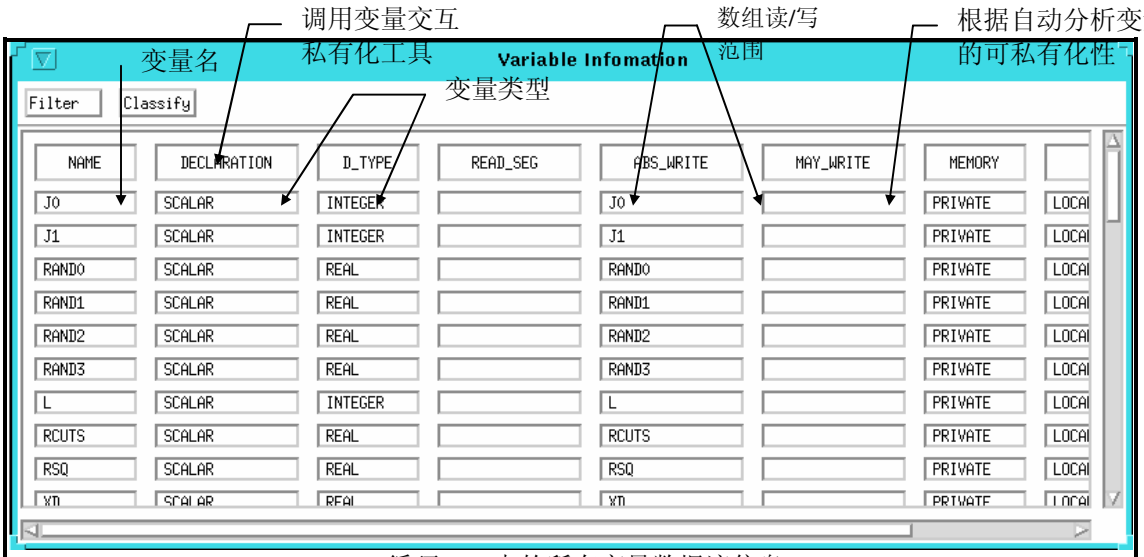


(b)用循环迭代相关图查询工具过滤非本质跨循环数据相关和循环独立相关

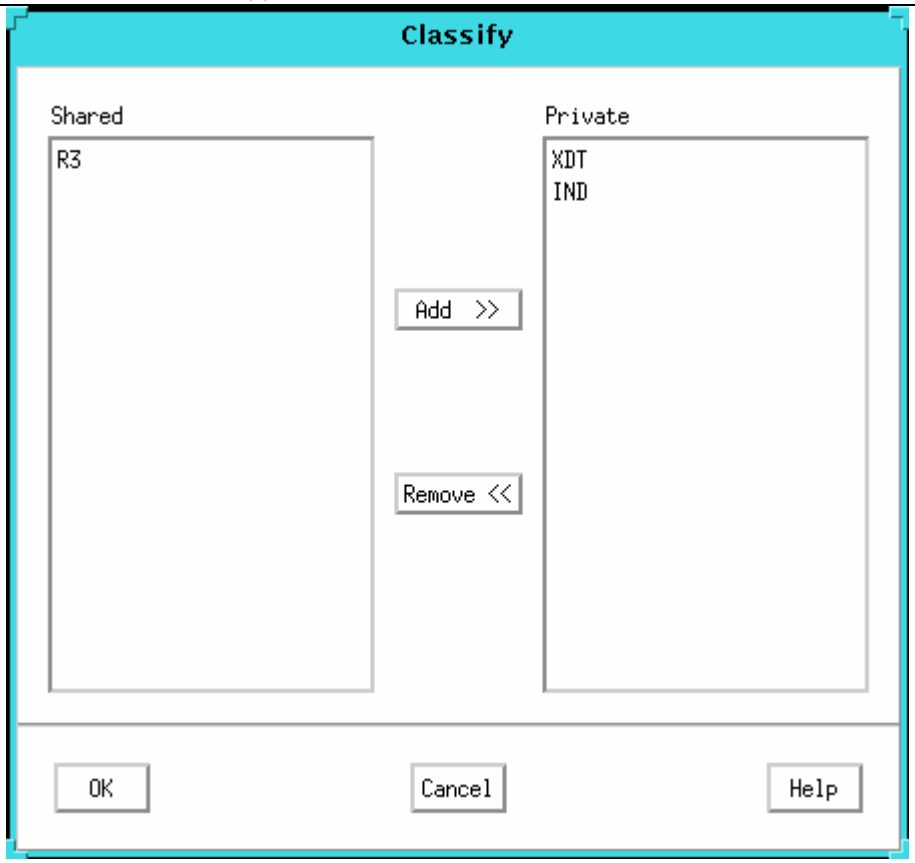


(c)相关性过滤后的相关信息表(为空)

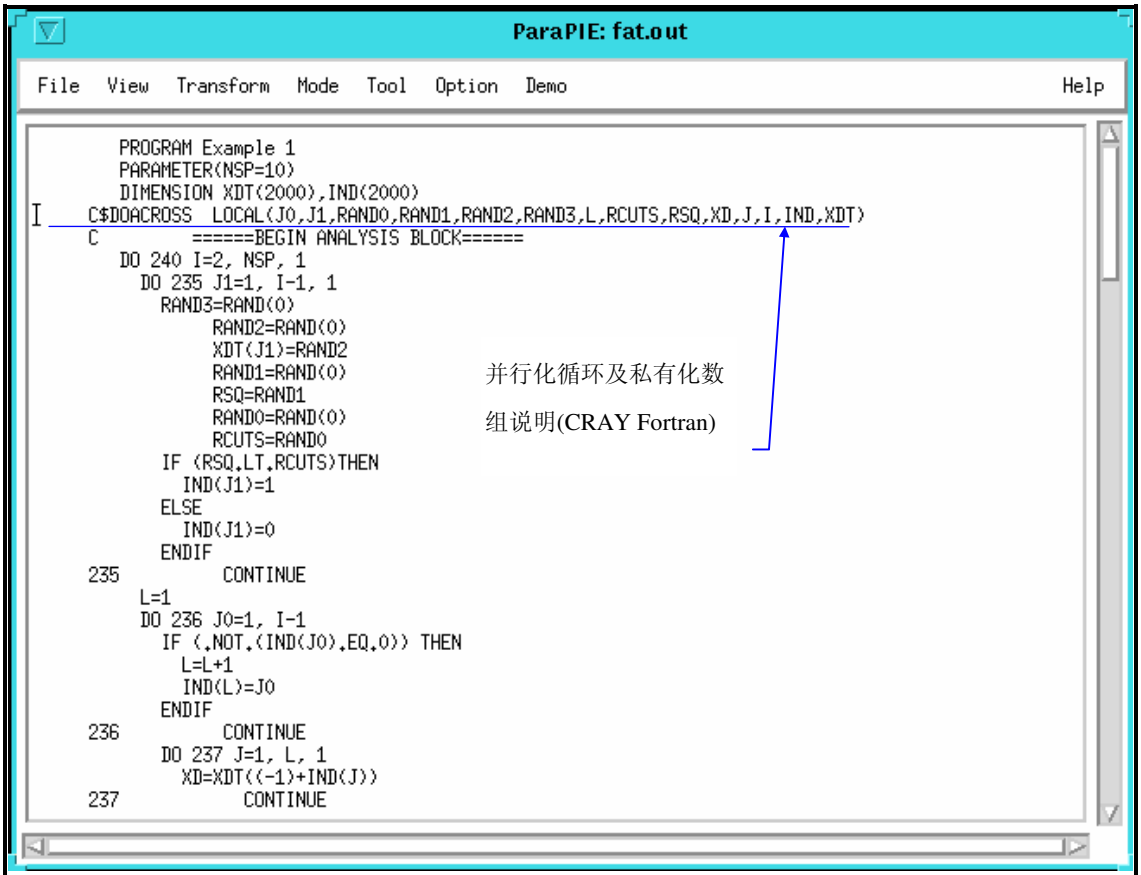
表 4-4、通过变量数据流信息查询表格和指定变量私有化工具进行数组私有化变换



(a)循环 240 中的所有变量数据流信息



(b)交互指定数组 XDT 和 IND 为私有数组(交互私有化)



(c)交互指定私有化数组后的“强制”自动并行化结果

4.1.3 交互循环么模并行化变换

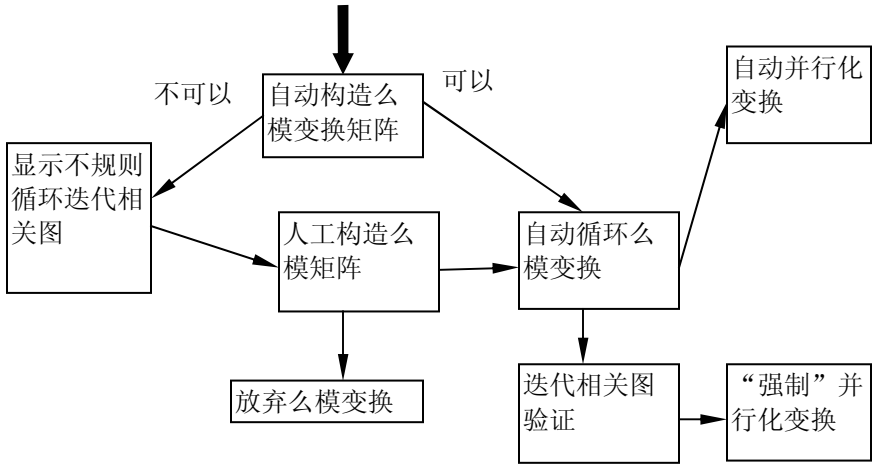


图 4-5、交互循环么模并行化变换流程

对于已知相关距离集合 **D** 的循环，只要存在么模矩阵 **U**，使得若$\forall$**d**$\in$**D**|**d>0**$\Rightarrow$**Ud>0**，则可以进行自动么模变换(后面将介绍当相关距离矩阵已知时如何自动寻找么模矩阵的方法)。否则，如果无法自动找到么模变换矩阵，则必须人工构造么模变换矩阵。PROFPAT 中的交互循环么模并行化流程如图 4-5所示。

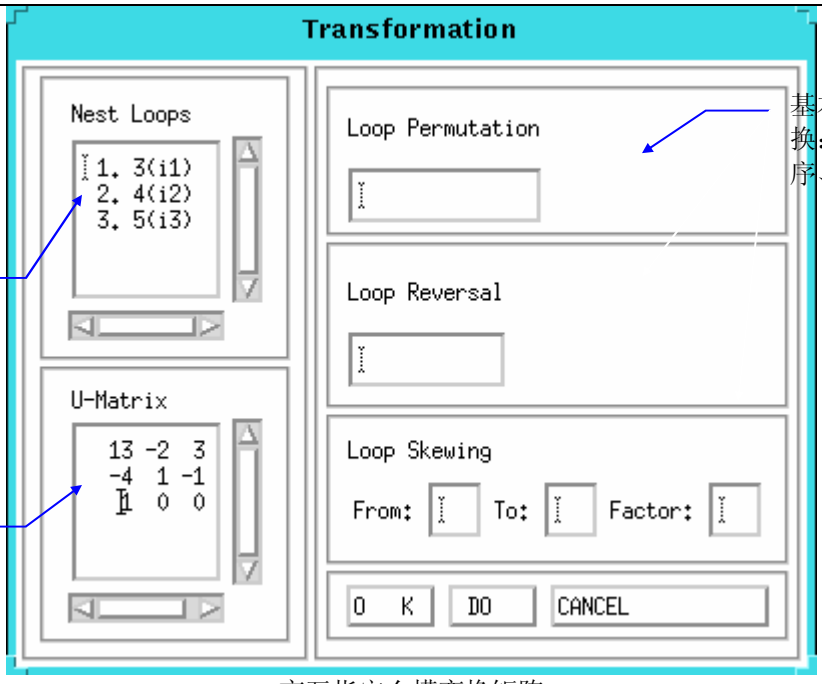
例如，表 4-5的循环程序是自动并行化编译工具无法分析和变换的，由于交互分析了循环迭代相关图[5]，程序员可以指定采用么模矩阵对该循环执行么模变换，并对目标程序分析其迭代相关图，通过循环迭代

[5] 已经通过相关图查询工具隐去了反相关和输出相关，并只显示关于数组 A 的相关性。

相关图验证该么模变换后的目标程序可并行化，程序员从而可以强制外层 j1 循环并行化。通过进一步分析程序员还可以强制内层 j3 循环并行化。

表 4-5、循环交互么模并行化变换前后的程序及其迭代相关图示例

```
DO i1=1,100
 DO i2=i1,100
  DO i3=i1+i2,200
      a(i1+i2,3*i2+3*i3,-3*i2+3*i3)=a(i1,i2,i3)
  ENDDO
 ENDDO
ENDDO
```



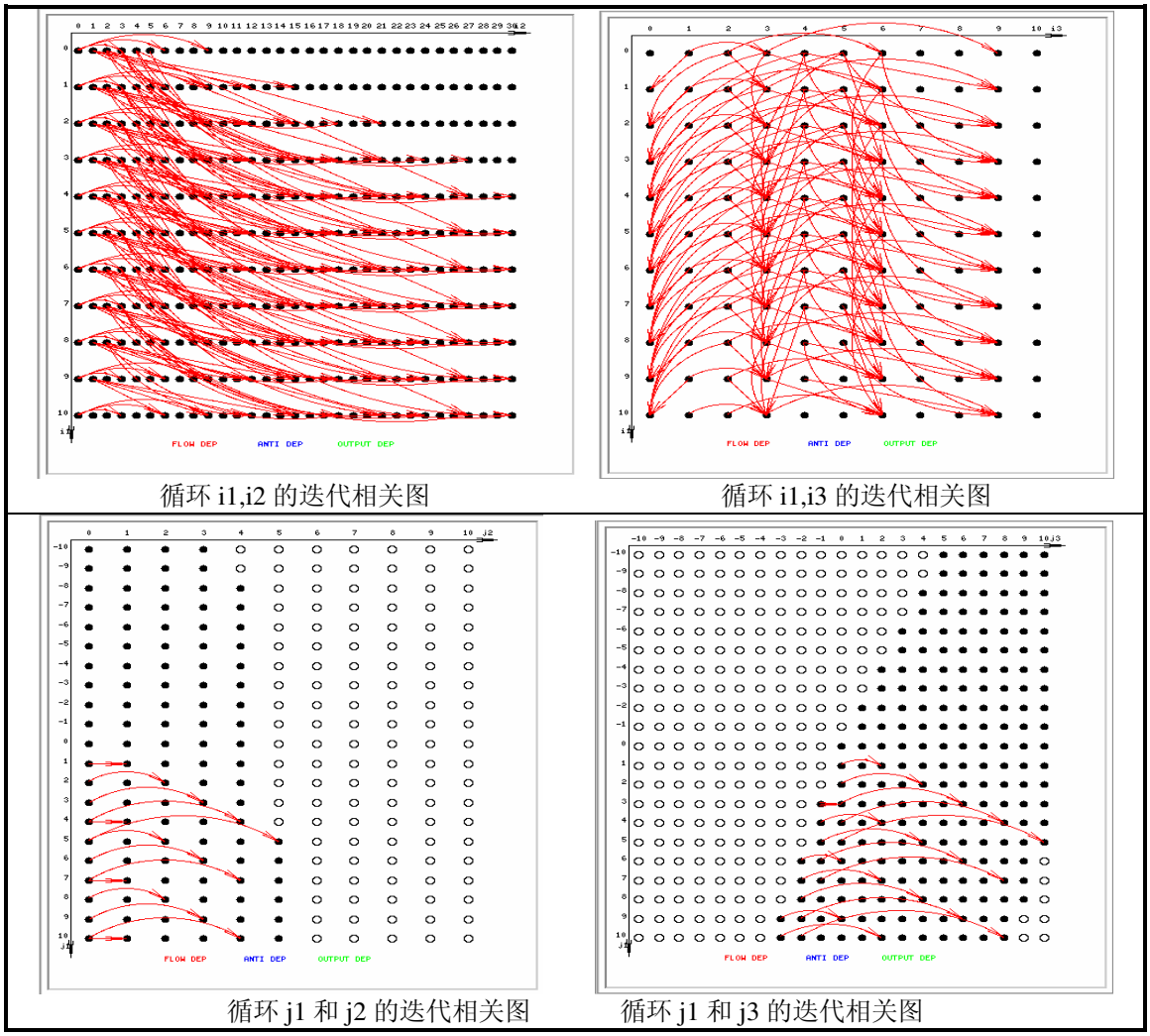
嵌套循环层次、行号及迭代变量

输入么模矩阵

基本么模变换：交换、反序、斜错

交互指定么模变换矩阵

```
 DOALL j1=17,1700
  DO j2=max(ceil((2-j1)/3.),ceil((-3*j1-400)/11.)),
1   min(min(min(-5,floor((200-j1)/3.)),
1   floor((700-j1)/2.)), floor((500-5*j1)/16.))
    DOALL j3=max(max(1,j1+3*j2-100),ceil((j1+2*j2-200)/5.)),
1           min(min(100,floor((-j2)/5.)),floor((j1+3*j2)/2.))
       i1=j3
       i2=j1+3*j2-j3
       i3=j1+2*j2-5*j3
       a(i1+i2,3*i2+3*i3,-3*i2+3*i3)=a(i1,i2,i3)
    ENDDOALL
  ENDDO
 ENDDOALL
```



循环 i1,i2 的迭代相关图　　循环 i1,i3 的迭代相关图

循环 j1 和 j2 的迭代相关图　　循环 j1 和 j3 的迭代相关图

由于本例中已知相关距离向量是循环迭代变量的线性表达式，且相关距离矩阵不满秩，所以外层并行化么模变换矩阵是可以自动计算的，本文将在后面第 6 章详细加以解释。

4.2　用 PROFPAT 分析困难的串行程序的实验

4.2.1 SPEC Cfp95 测试程序包

表 4-6、SPEC Cfp95 测试程序包的 10 个程序简介

程序序号	程序简要说明
101.tomcatv	Vectorized mesh generation.
102.swim	Shallow water equations.
103.su2cor	Monte-Carlo method.
104.hydro2d	Navier Stokes equations.
107.mgrid	3d potential field.
110.applu	Partial differential equations.
125.turb3d	Turbulence modeling.
141.apsi	Weather prediction.
145.fpppp	From Gaussian series of quantum chemistry benchmarks.
146.wave5	Maxwell's equations.

如表 4-6所示，SPECfp95 benchmarks 由 10 个串行 Fortran 程序组成，是衡量计算机性能及并行化编译效

果的通用测试程序包**[51][57]**。
除了七个程序 APPLU, HYDRO2D, MGRID, SU2COR, SWIM, TOMCATV, TURB3D 可以并行化外，141.APSI, 145.FPPPP and 146.WAVE5 这三个程序曾经被公认为是极难并行化或几乎无法由程序员手工并行化的**[51][57]**。先进的自动并行化编译工具（AFT 和 SUIF）已经能对其中 APPLU, HYDRO2D, MGRID, SU2COR, SWIM, TOMCATV, TURB3D，都取得了显著的效果，但是对于另外三个程序 APSI, FPPPP 和 WAVE5 却无能为力，无法并行化（加速比为 1）。那么能否通过 PROFPAT 的工具，如程序计算量分析工具、最大潜在并行度分析工具、相关性分析工具和数组私有化分析工具等，对这些程序取得一定的并行化效果呢？下面我们就用 PROFPAT 分析这几个困难的 SPECfp95 测试程序。

4.2.2 用 PROFPAT 分析困难的 SPECfp95 测试程序

我们试图用 PROFPAT 提供的程序计算量分析工具、最大潜在并行度分析工具、相关性分析工具和数组私有化分析工具等对 SPEC95 测试程序中的 APSI，FPPPP 和 WAVE5 这三个自动并行化难奏效的程序逐一加以人机交互地分析，终于发现它们事实上是可并行化的。

4.2.2.1 程序计算量分析

首先，通过程序计算量分析工具，我们识别出这些程序中最耗时的重要子程序和循环，如表 4-7所示。它们分别是 APSI 中的 RUN 子程序，FPPPP 中的 TWLDRV 子程序和 WAVE5 中的 PARMVR 子程序，分别占据各自串行程序执行时间的 99%, 60%,和 59%。而且，这些过程中最耗时的重要循环分别是 APSI 中的循环 20,30,40,50,60,70,100，FPPPP 中的循环 1001 和 WAVE5 中的循环 120。

表 4-7、对三个程序的程序计算量分析结果

程序名	APSI.F	FPPPP.F	WAVE5.F
主要程序过程	RUN	TWLDRV	PARMVR
程序计算量	99%	60%	59%
耗时循环标号	20,30,40,50,60,70,100	1001	120

下面用 PROFPAT 中的其他工具针对它们分别进行的分析。

4.2.2.2 并行性分析

通过最大潜在并行性分析工具得知，以上提到的重要子程序在各自程序的给定输入数据下都具有潜在并行性。但是必须注意的是，对给定输入下具有的并行性，程序员还需要仔细加以分析，只有确定对任意输入，它都具有并行性时才能采取并行化变换。
进一步通过循环级并行性分析，可知由于存在跨循环数据流相关，WAVE5 中的循环 PARMVR/120 不能完全并行化为 DOALL 循环，除非能够利用循环间的非循环迭代级并行性。
接下来对 FPPPP 和 APSI 分析是否对所有输入数据，这些子过程都可并行化。
通过相关性分析工具为每一个子过程构造数据相关图，帮助我们从中识别出所有阻碍并行化的相关边及导致跨循环相关的变量。为了消除这些相关性，我们借助数组私有化分析工具判定出在给定输入下这些变量是可私有化的。接下来我们需要进一步人工分析这些变量对所有输入数据是否都是可私有化的。
由于对程序 FPPPP 无法做出上述判断，所以我们设计了一个运行时刻判断并行性的算法，成功地对 FPPPP 并行化。
对程序 APSI 由于过程 RUN 中的主要循环包含了深层嵌套的子过程调用，所以借助 PROFPAT 的调用图

构造和浏览工具，我们交互获得了必要的跨过程信息，最终证明所有这些变量的可私有化性质，导致对RUN 过程中绝大多数主要循环的并行化。

以下是对这三个程序的并行化示例，所引用的程序段已经过一定简化，以利于说明主要问题。

4.2.2.2.1 WAVE5

从 DO 循环 PARMVR/120(如表 4-8a)中我们可以看到，由于这些归约数组为稀疏数组，该循环不能被现在的技术并行化。通过相关图可视化工具，我们发现 PARMVR/120 的循环体由三个分离的子相关图组成，三个语句相互之间可以同时执行。

进一步分析表明由于数组 CX，CY，CZ 不相交，所以可以对它们同时计算。通过循环分布产生三个循环，然后通过增加 Parallel Section，利用三个循环之间的循环间并行性，并行化结果如表 4-8b 所示。

这个程序段的并行化使 WAVE5 取得了一定的加速比。

表 4-8、挖掘循环120/PARMVR中的非循环并行性

```
DO 120 L=L3,L4
      CX(IJ(L))=CX(IJ(L))+XMULT*W1(L)
      CY(IJ(L))=CY(IJ(L))+YMULT*W1(L)
      CZ(IJ(L))=CZ(IJ(L))+ZMULT*W1(L)
120  CONTINUE
```

```
PSECTION BEGIN
PRIVATE L
SECTION BEGIN
DO L = L3,L4
    CX(IJ(L))=CX(IJ(L))+XMULT(L)*W1(L)
ENDDO
SECTION END
SECTION BEGIN
DO L = L3,L4
                                        CY(IJ(L))=CY(IJ(L))+YMULT(L)*W1(L)
                                        ENDDO
                                        SECTION END
                                        SECTION BEGIN
                                        DO L = L3,L4
                                        CZ(IJ(L))=CZ(IJ(L))+ZMULT(L)*W1(L)
                                        ENDDO
                                        SECTION END
                                        PSECTION END
```

另外，利用并行 DO 循环 100、130 同串行 DO 循环 140 之间数据重用的性质，开发多处理机间 cache 的数据局部性，有效地优化了串行循环 140 对 cache 的利用，提高了并行加速比，在我们的系统作 cache 优化前其并行执行时间为 154 秒，而 cache 优化后并行执行时间为 108 秒，加速比分别为 1.32 和 1.88 本文将在$7.2详细探讨这个问题。

4.2.2.2.2 FPPPP

在程序 FPPPP 中，DO 循环 TWLDRV/1001 中的变量 fq0 至 fq5 都引用了上一迭代中定义的值，经过动态数据流分析之后(见第$7.1节)，我们并行化了该循环，把 FPPPP 的加速比从 0. 76 提高至 2.47。

4.2.2.2.3 APSI

由于程序 APSI 的主要循环体中包含深层次嵌套的子过程调用，必须通过跨过程分析工具获取必要的过程间信息。在表 4-9右部所示的程序段中，分析跨过程信息可以得出 MOD(N,IFAC[2+I]).EQ.0, for

1≤I≤NF。这个信息在分析在表 4-9左部所示的程序时是至关重要的。 借助该信息可知, 在循环首次迭代时，通过典型的跨过程符号向前替代，可以分析出 IP 总是 N 的因子，亦即 IP*L1.EQ.N。通过此信息，足以证明该循环的数组是首覆盖的，从而符合相关-覆盖方法提出的数组私有化判定准则，对所有输入数据，该循环通过数组私有化可以并行化(见第 5 章)。类似地，可以证明过程 RUN 中的其他主要循环也可以并行化。

```
NA=1                                    NL=N,NF=0,TEMP0=0
L2=N                                    REPEAT
DO K=1,NF                                 NTRY=NTRY+2
    KH=NF-K1                              WHILE(Nl-Ntry*(Nl/Ntry).EQ.0&Temp0.NE.1)
    IP=IFAC(KH+3)                           NF=NF+1
    L1=L2/IP                                IFAC(2+NF)=NTRY
    IDO=N/L2                                NL=NL/NTRY
    IDL1=IDO*L1                             IF (NTRY.EQ.2).AND.(NF.NE.1)) THEN
    NA=1-NA                                   DO 106 I=2,NF,1
    ...                                         IFAC(4+NF-I)=IFAC(3+NF-I)
    IF (IDO.EQ.1) NA=1-NA               106     CONTINUE
    IF(NA.NE.0)THEN                           IFAC(3) = 2
    CALL RADFG(IDO,IP,L1,IDL1,C,CH,...)     ENDIF
    ELSE                                    IF (NL .EQ. 1) THEN
    CALL RADFG(IDO,IP,L1,IDL1,CH,C,...)       TEMP0 =1
    ENDIF                                     GOTO 160
    L2=L1                                   ENDIF
ENDDO                                     ENDWHILE
                                        160   UNTIL(TEMP0.EQ.1)
```

表 4-9、分析APSI

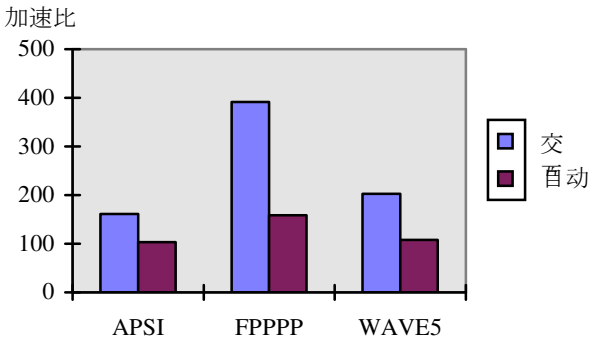
4.2.3 实验结果

借助 PROFPAT 的工具进行人机交互辅助分析和变换，我们最终对 WAVE5、FPPPP 和 APSI 都取得了明显的加速效果，见表 4-10。

为了说明并行程序设计交互环境 PROFPAT 的应用效果，我们选取硬件环境（具有 4 个 MIPS R10000 处理器的 SGI Origin 2000 并行计算机）作为并行化实验条件，同时选取并行化编译研究通常采用的测试程序包 SPEC95，通过分别将串行程序、并行化后的并行程序在上述环境下执行，根据得到的实际执行时间绘制了表 4-10。

表 4-10、利用PROFPAT对SPEC95测试程序包的并行化效果比较

测试程序名	串行程序 执行时间	并行程序 执行时间	4 机 加速比
APSI	161.2	103.0	1.57
FPPPP	391.3	158.4	2.47
WAVE5	202.8	108.0	1.88



在利用 PROFPAT 分析过程中，还发现了一些值得在自动并行化编译器中实现的实用技术，如提高可私有化数组识别率的相关-覆盖分析数组私有化方法**[72][73]**，利用过程繁衍进行跨过程线性关系传播和数据相关性分析的技术**[25]**，对已知相关距离的多重循环计算合法么模变换矩阵的方法**[67][70]**，结合归约识别增强幺模变换的方法，通过划分数据相关图利用非循环级并行性的 Parallel Section 方法**[56]**，动态数据流分析方法**[42]**，利用循环间数据重用的 cache 优化方法**[26]**等等。我们将把这些适合自动化的技术结合到下一代自动并行化编译器的研制过程中，以取得更好的效果。

5. 数组私有化新技术：相关覆盖方法

数组私有化是并行化编译中的重要技术，本节提出了一个数组私有化的新方法，相关-覆盖方法。它将相关性分析技术和数据覆盖技术有机地结合在一起，具有了效率高、处理范围广、易于扩充的优点。其私有化判定能力超过了基于相关性和数据覆盖的方法，可以处理一些其它方法不能处理的问题。通过扩充基本的相关-覆盖方法，使其不仅能够处理无条件控制结构的单重循环，也能处理有复杂条件控制结构的多重循环。本节从理论上证明了相关-覆盖方法的私有化判定能力超过了目前最流行的两种数组私有化方法。

5.1 数组私有化技术的重要性

当前随着并行计算机的大量使用，并行化编译的重要性日趋显著。进入九十年代，并行编译的研究取得了很大的进展 。若干并行编译的关键技术被发现 ，数组私有化正是其中之一**[18][28]**。九五年后，几个新一代并行编译系统，如伊利诺依大学的 Polaris**[17]**、斯坦福大学的 SUIF[45]**[60]**和复旦大学的 AFT**[76][77]**，相继问世，这些系统的并行化能力大大高于传统的并行编译系统。测试结果表明，对几个流行的测试软件而言，传统的并行编译系统一般仅能有效加速其中 20%左右的程序，而由于新一代并行编译系统都实现了数组私有化技术**[17]** [45]**[76]**，可以有效加速 50%左右的程序**[76]**。

数组私有化主要处理串行程序中作为临时变量的数组。在传统串行程序设计中往往大量重用临时变量，以减少存储空间。然而使用临时变量会引起所谓的存储相关问题，阻碍并行化的实现。数组私有化技术识别出哪些数组是临时变量，将这些临时数组转化(私有化)为并行程序的私有数组，从而消除存储相关，将原来无法并行化的程序变换为并行程序。

一般的数组私有化识别问题本质上是不可计算的，因此所有数组私有化计算方法都只能是整体近似的或部分精确的。数组私有化的难精确求解性决定了实用的数组私有化方法应该具有计算效率高、处理范围广、易于扩充的优点。

5.2 基本的相关-覆盖方法

本节将介绍的相关-覆盖方法就是一个适用于自动并行化系统的数组私有化实用方法。

已有的数组私有化计算方法过分依赖于非私有化本质信息的获得(如完全精确的数据相关或者数据暴露集)，既引入了不必要的计算量，降低了计算效率，又将数组私有化方法的应用范围限制于可以计算这些不必要信息的场合。如何抓住数组私有化本质，避免计算不必要的信息，提高计算效率和应用范围，是相关-覆盖方法首先要解决的问题。过去的数组私有化方法过分依赖于数组元素集合的内部表示(如正则段、不等式组)，而不同的表示方法并不是数组私有化方法的本质。相关-覆盖方法把数组私有化判定原则从具体数组表示方法中独立出来，适用于各种数组表示。

在 AFT 系统中，我们实现了相关-覆盖方法。由于在需要并行化的实际程序中普遍存在复杂的条件控制结构、多层嵌套循环以及其他一些难于计算的复杂因素，我们还将探讨如何将相关-覆盖方法提出的数组私有化判定准则加以适当扩展，在不改变方法基本框架的同时，通过避开一些难计算的因素，减少不必要的中间计算，在 AFT 实现后对测试程序获得很高的应用效果。我们还将从理论上证明，相关-覆盖方法的判定能力超过了 Polaris 和 SUIF 两个系统使用的方法。

在相关-覆盖方法中，我们引进了两个重要概念：反相关写集（Non-Antidependent Write)和写自覆盖集(Self Covered Write)。

5.2.1 利用反相关写集近似计算暴露集

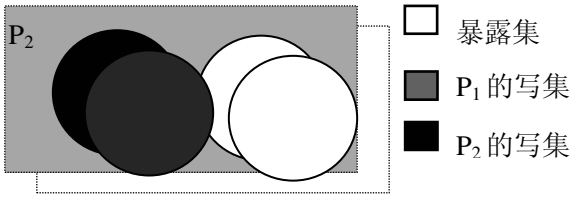


图 5-1、暴露集计算原理

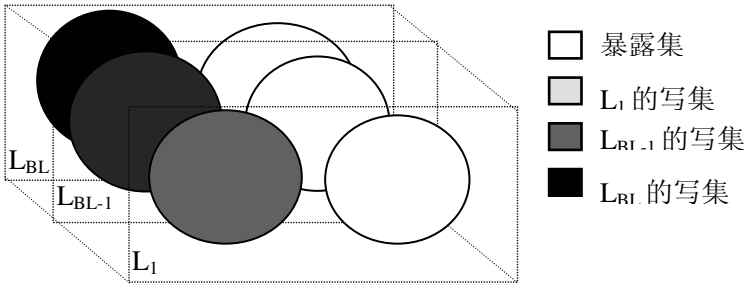


图 5-2、循环迭代的暴露集计算示意

求解暴露集是不可计算的问题，现有的求解暴露集的方法都是近似的。在 Z.Li 在文献[43]中提出一种逐循环迭代计算数组暴露集的方法。如图 5-1所示，对 $P=P_1;P_2$，有 $UE_P = UE_{P1} + (UE_{P2} - Set_{P1}(W_{P1}))$。于是，对于循环 $L=L_1;L_2;...;L_{BL}$，其中 L_k 表示第 k 次循环迭代，BL 为 L 的迭代次数。对上述计算方法加以扩充，如图 5-2所示，我们知道数组在循环 L 中的暴露集 UE_L 可由循环迭代的暴露集 UE_{Lk} 依次(k=BL,BL-1,...,1)合并计算部分和 PUE_{Lk} 而得：

$PUE_{Lk} = UE_{Lk} + (PUE_{Lk+1} - Set_{Lk}(W_{Lk}))$，(k=BL,BL-1,...,1)，

$PUE_{LBL} = \{\}$，$UE_L = PUE_{L1}$。

为了求解暴露集，上述方法中必需通过对每个循环迭代的暴露集和写集建立数据流迭代方程近似计算。而在下面介绍的相关-覆盖方法中为化简暴露集的近似计算，提出了无反相关(NAD)写集的概念，可以一次高效地处理整个循环。

定义 5-1：给定程序段 P 中的一个读引用 r，称 P 中的写引用 w 是关于 r 的无反相关写，如果 r 与 w 在 P 中无反相关。P 中所有关于 r 的无反相关写构成 P 中关于 r 的无反相关写集。直观地说，如果读写(r,w)之间没有反相关，那么一定可以把写 w 交换到在读 r 之前执行。因此，如果 P 中只有一个读 r，那么 P 中关于 r 的无反相关写集所引用的数组元素一定是非暴露的[1]。因此，我们有下面的定理可以用无反相关写集来近似地求解暴露集。

定理 5-1(暴露集近似计算)：$UE_P \subseteq \bigcup_{r\in R_P} (Set_P(r) - Set_P(NaW_P^r))$。

证明：假设定理不成立，那么一定存在 A(j)使得：

(1) $A(j) \notin \bigcup_{r\in R_P} (Set_P(r) - Set_P(NaW_P^r))$，但

(2) $A(j) \in UE_P$。由此可知存在 $r \in R_P$ 使得下述两条件成立：

(3) $A(j) \in Set_P(r)$

(4) 在 r 之前 A(j)没有被写过。由(1)和(3)得

(5) $A(j) \in Set_P(NaW_P^r)$，即存在 $w \in W_P$，使得

(6) $A(j) \in Set_P(w)$，且

(7) r 和 w 在 P 中没有反相关。但由(3),(4),(6)又知，

[1] 不然，若存在一个被某个无反相关写 w 引用的数组元素是暴露的，则在写 w 之前必存在一个读 r 引用了该元素，于是(r,w)之间必存在反相关，这与前提矛盾。

(8) r 和 w 在 P 中有反相关。(7)与(8)矛盾，证毕。

依据上述定理的计算虽然是近似的，但在常见的情况下可以得到精确的结果。如在表 2-10a 串行程序中，I 循环体的暴露集为 $UE_{I\text{循环体}}$={A(1)}。循环 I 的循环体内只有一个写 w 为 A(J)，一个读 r 为 A(J-1)，(r,w)间无反相关。因此有 $\boldsymbol{R}_{I\text{循环体}}$={A(J-1)}，$NaW^{r}_{I\text{循环体}}$=$\boldsymbol{W}_{I\text{循环体}}$={A(J)}。读集 $Set_{I\text{循环体}}(\boldsymbol{R}_{I\text{循环体}})$=A (1: M-1)，写集 $Set_{I\text{循环体}}(\boldsymbol{W}_{I\text{循环体}})$= A(2:M)。 $Set_{I\text{循环体}}(r)-Set_{I\text{循环体}}(NaW^{r}_{I\text{循环体}})$=A(1:M-1) -A(2:M)={A(1)}=$UE_{I\text{循环体}}$。可见在这个例子中我们的方法是精确的。

由基本准则的推论和**暴露集近似计算**定理我们得到：如果在循环 L 中，对任意 2≤k≤BL 和任意 r∈ $\boldsymbol{R}_{L_k}$，均有 $Set_{L_k}(r)\subseteq Set_{L_k}(NaW^{r}_{L_k})$，那么数组是可私有化的。因为 BL 可能很大或不确定，对每个 i 计算 $NaW^{r}_{L_i}$ 是不现实的，但根据经典的相关性测试方法可以很容易求得 NaW^{r}_{Lv}。因此我们有更实用的判定准则。

判定准则 1：在循环 L 中，若对任意 r∈ $\boldsymbol{R}_{Lv}$ 均有 $Set_{Lv}(r)\subseteq Set_{Lv}(NaW^{r}_{Lv})$，那么数组是可私有化的。

5.2.2 利用自覆盖写集避免计算复杂读引用

和绝大多数的私有化方法一样，判定准则 1 只考虑读写的覆盖关系。这里读是中心，当某些写的下标很复杂使得 Set 算子无法作用时，允许忽略这些写。但当读的下标很复杂，如例 3 的串行程序的第 7 行读引用，使 Set 算子无法作用时，上述判定方法就不再适用。为了处理复杂的读下标，我们介绍一个不对复杂的读引用使用 Set 算子，只考虑写-写覆盖关系的判定方法。

(a)串行程序		(b)并行程序	
1.	DIMENSION A(100)	1.	DIMENSION A(100)
2.	DO I=1,N	1.	DOALL I=1,N
3.	DO J=1,I	1.	LOCAL PA(100)
4.	A(J)=...	1.	DO J=1,100
5.	ENDDO	1.	PA(J)=A(J)
6.	DO J=1,F(I)	1.	ENDDO
7.	...=A(G(I,J))	1.	DO J=1,I
8.	ENDDO	1.	PA(J)=...
9.	ENDDO	1.	ENDDO
		1.	DO J=1,F(I)
		1.	...=PA(G(I,J))
		1.	ENDDO
		1.	IF (I.EQ.N) THEN
		1.	DO J=1,I
		1.	=PA(J)
		1.	ENDDO
		1.	ENDIF
		1.	ENDDOALL

图 5-3、自覆盖写集与数组私有化识别

定义 5-2：如果对 1≤i≤BL-1，都有 $Set_{L_i}(W)\subseteq Set_{L_{i+1}}(W)$，称数组在循环 L 中是写自覆盖的。

写自覆盖是我们提出的相关-覆盖方法中的又一重要概念，它表示在循环的某个迭代定义的任意数组元素都将在下一个迭代中被重新写过。如何判定写自覆盖的方法超出了本节的范围，但写自覆盖在大多数场合下都不难由下标表达式与循环界限求得。在图 5-3a 中，在 I 循环的每个迭代中，写引用的集合 $Set_{Li}(W)$=A(1:I)，显然 $Set_{Li}(W)$=A(1:I)⊂(A(1:I+1)= $Set_{Li+1}(W)$，所以 A 在循环 I 中是写自覆盖的。应用写自覆盖和无反相关写集，我们得到了只考虑写—写覆盖关系的判定准则。

判定准则 2：如果数组在循环 L 中是写自覆盖的，并且对任意 r∈ $\boldsymbol{R}_{Lv}$，均有 $Set_{Lv}(W_{Lv})\subseteq Set_{Lv}(NaW^{r}_{Lv})$，那么数组是可私有化的。

证明：根据定理 1 和基本准则 1，只要证

$(\bigcup_{j=1}^{k-1} Set_{L_j}(W))\cap \bigcup_{r\in R_{Lk}} (Set_{L_k}(r)-Set_{L_k}(NaW^{r}_{L_k}))=\varnothing$，即可。

$(\bigcup_{j=1}^{k-1} Set_{L_j}(W))\cap(\bigcup_{r\in R_{Lk}} (Set_{L_k}(r)-Set_{L_k}(NaW^{r}_{L_k}))$

$\subseteq Set_{L_k}(W)\cap(\bigcup_{r\in R_{Lk}} (Set_{L_k}(r)-Set_{L_k}(NaW^{r}_{L_k}))$ (根据写自覆盖的性质)

$= \bigcup_{r \in R_{L_k}} (Set_{L_k}(r) \cap (Set_{L_k}(W) - Set_{L_k}(NaW^r_{L_k}))$

=∅ （根据判定准则的条件），证毕。

根据判定准则 2，可知图 5-3a 的程序也是可以私有化的。

5.2.3 初值复制和终值复制

在私有化变换时，一个可私有化数组是否需要终值复制可以由数组的数据流分析判定。如果一个数组的值在循环之后不再被使用，那就不需要终值复制。如果需要终值复制并且数组是写自覆盖的，那么终值就是最后一次迭代的值。如果不是写自覆盖的，那么需要对每个迭代 L_i 求 $Last_i(W) = Set_{L_i}(W) - \bigcup_{j=i+1}^{BL} Set_{L_j}(W)$，这只能对某些特殊情况求解。当求出 $Last_i(W)$ 后，每个迭代要对 $Last_i(W)$ 表示的所有数组元素进行终值复制。

对于初值复制而言，集合 $Set_{L_{body}}(r) - Set_{L_{body}}(NaW^r_{L_{body}})$ 表示的数组元素需要初值复制，当该集合无法计算是对数组的每个元素都需要初值复制，如图 5-3b。

```
1.    DO I=2, NSP
2.      DO J=1, I-1
3.        XDT(J)=...
4.      ENDDO
5.      L=0
6.      DO J=1, I-1
7.        IF (.NOT.(IND(J).EQ.0)) THEN
8.          L=L+1
9.        ENDIF
10.         ...
11.     ENDDO
12.     IF (.NOT.(L.EQ.0)) THEN
13.       DO J=1, L
14.         XD=XDT(IND(J))
15.       ENDDO
16.     ENDIF
17.   ENDDO
```

图 5-4、截取自 Perfect 测试程序 NASI 的子程序 ACTFOR

图 5-4所示的这个数组私有化的实例是不能应用基于线性关系的相关性分析方法加以识别的，因为第 14 行的读引用中，IND(J)和 L 都不能表示为线性约束条件，同时，由于读引用的元素集不能表示成数组区域的集合，所以也不能用基于集合覆盖的数据流方法加以识别。由于第 3 行的写引用中数组 XDT 在循环 I 中满足写自覆盖的性质，应用我们相关－覆盖方法的判定准则 2，不需要计算读应用集就可以精确判定该程序可以数组私有化。

5.3 扩充相关-覆盖方法的适用范围

相关-覆盖方法的基本准则适用于判定循环体只包含顺序语句的单层循环的私有化条件。而实际程序中普遍存在各种含有复杂嵌套结构的循环，使得程序自动并行化编译中无法实际应用这些准则。所以必须对相关覆盖方法就复杂控制结构加以扩充。

由于任意复杂控制流都可以通过结构化方法将其转化为只含顺序语句，IF-THEN-ELSE 和循环语句(DO, WHILE, REPEAT 循环)。对私有化而言，REPEAT 可以看成为顺序语句，WHILE 语句可以看成为 IF 语句，这样复杂控制结构只剩下 IF 语句和 DO 循环语句。

重要的是，当循环中含有嵌套控制结构时，NaW^r_{Lv} 和 $Set_{Lv}(NaW^r_{Lv})$ 的计算方法将会发生变化。应该指出由于我们的判定准则是近似的，因此如想提高判定的精度，原则上可以有很多改进的余地。但任何的改进都是以提高时-空复杂性为代价的。我们提出的方案是平衡了各种因素后提出的折衷方案。

5.3.1 处理嵌套 IF 语句

5.3.1.1 引入确定写算子合并IF 语句分支的数据流信息

IF 语句有 THEN 和 ELSE 两个分支，且每个分支的写引用集一般不相同。每执行一次 IF 语句只有一个分支被执行，因而从 IF 语句整体看引起了写的不确定性。写的不确定性将数组的引用区域分为一定写区域和可能写区域。一定写区域是指在程序的所有可能执行方式下，这个区域的数组都一定被写过，可能写区域是指在程序的所有可能执行方式下被写数组区域的总和。对 IF 语句来说，我们仍用以前描述的 Set_P 算子计算可能写区域，并引入下面的 $DSet_P$ 算子计算一定写区域。 $DSet_P$ 算子的定义是递归的：

$$DSet_P(W_S)=\begin{cases} Set_P(W_S) & \text{S contains no IF} \\ DSet_P(DSet_{S-then}(W_{S-then})\cap DSet_{S-else}(W_{S-else})) & \text{S is IF} \\ DSet_P(W_{S/S'})\cup Dset_P(W_{S'}) & S'\subset S, S' \text{ is IF} \end{cases}$$

其中，程序段 S 是 P 的一部分，S'⊂S 表示程序段 S'是程序段 S 的一部分，符号 S/S'表示在程序段 S 中排除子程序段 S'的其余部分。

值得注意的是，由于 IF 语句的某个分支实际上可能永远不被执行，因此无论是 Set_P 算子还是 $DSet_P$ 算子的计算都是保守的。

现在我们可以将上节的判定准则推广到含 IF 语句的嵌套循环：

判定准则 I：L 是循环，r∈ R_{Lv}，如果 $Set_{Lv}(r)\subseteq DSet^r_{Lv}(NaW^r_{Lv})$，那么 r 不妨碍私有化。

判定准则 II：L 是循环，r∈ R_{Lv}，如果数组在循环 L 中是写自覆盖的且 $Set_{Lv}(W_{Lv})\subseteq DSet^r_{Lv}(NaW^r_{Lv})$，那么 r 不妨碍私有化。

5.3.1.2 利用IF 条件表达式的性质简化计算

在上述的判定准则中，IF 语句的条件表达式的性质没有被充分利用。下面我们看以下 IF 条件表达式性质可以怎样对简化判定准则起作用。

分析 r 是否影响数组私有化时，如果读引用 r 包含在 IF 某个分支 IF(exp)中，可以假定存在一次循环迭代执行到此分支（否则 r 就不需要判定）。由于另一 IF 分支 IF(.not.exp)的写引用 w 在本次迭代中不可能同 r 有反相关，在我们的相关一覆盖方法中可以忽略不必计算。这在一定程度上简化了反相关写集的计算。

如果分析不包括 r 的 IF 语句，根据 DSet 算子的上述定义，当 IF 条件表达式 exp 为循环不变量时，即使无法确定 exp 值，由于总有一个 IF 分支没有执行，则 DSet 算子可以简化为空。因此我们忽略所有循环不变量控制的 IF 语句中的确定写计算。

5.3.2 处理嵌套 DO 循环

我们处理嵌套 DO 循环的出发点是在整体信息无效时充分利用部分信息的有效性，这与整个方法的思路是一致的。

5.3.2.1 简化嵌套循环中的写集计算

上面提出的准则对含有嵌套的 DO 语句的循环仍然适用。但对于两个循环 $L'\subset L$，我们既可以将 L' 中的写引用 w 看成为一个整体，也可以将它在 L' 每个迭代中的实例 $w(L'_1), w(L'_2), \cdots, w(L'_{tc})$ 分别对待。

对每个读引用 r，r 与 w 在 Li 中没有反相关当且仅当 r 与 $w(L'_1), w(L'_2), \cdots, w(L'_{tc})$ 在 L_i 中都没有反相关，因此部分写更容易满足无反相关的条件。还有当 Set 算子无法作用到整体写上时，它对部分写可能仍然有效。因此我们有必要考虑部分写。

但如果所有循环的所有部分写都单独考虑会产生太大的工作量，为此我们只考虑满足下面定

义的首覆盖的特殊情况。

定义 5-3：w 是数组 A 在循环 L 上的写引用，如果对 2≤i≤tc，都有 $Set_{L_1}(w) \supseteq Set_{L_i}(w)$，称 w 在循环 L 中是首覆盖(First Covered)的。

对于首覆盖的数组整体写引用我们只要考虑第一次迭代中的部分写即可。假设 w 为循环 L 中的写引用，L 中包含 w 的 n 个嵌套循环为 $L^1 \subset L^2 \subset \cdots \subset L^n$，w 在其中 m 个循环 $L^{i_1}, L^{i_2}, \ldots, L^{i_m}$ 中是首覆盖的，我们用 $w(L_1^{i_1}, L_1^{i_2}, \ldots, L_1^{i_m})$ 表示这个部分写，那么在判定 L 的私有化过程中，我们就用 $w(L_1^{i_1}, L_1^{i_2}, \ldots, L_1^{i_m})$ 来代替 w。

5.3.2.2 处理嵌套循环中的读引用

处理了写引用，我们还要处理读引用。由于一对变量如果在外层循环体内没有相关，那么在内层循环一定没有相关。因此对于两个循环 LP⊂L，如果它们都包含 r，那么 LP 中的一个写引用 w 只有下列三中可能，1)既不属于 NaW_{LP}^r 也不属于 NaW_{Lv}^r，2)既属于 NaW_{LP}^r 也属于 NaW_{Lv}^r，3)属于 NaW_{LP}^r 但不属于 NaW_{Lv}^r。对前两种情况，我们不必特殊考虑。为处理第三种情况，可以将判定准则 I 中对 $Set_{Lv}(r) \subseteq DSet_{Lv}(NaW_{Lv}^r)$ 的判定，用 $AUE_{Lv}(r) = Set_{Lv}(Set_{LP}(r) - DSet_{LP}(NaW_{LP}^r - NaW_{Lv}^r)) - DSet_{Lv}(NaW_{Lv}^r)$ 是否为空来代替。我们称 $AUE_{Lv}(r)$ 为近似暴露集，因为对外层循环 Lv 而言，它概括了 r 在内层循环 LP 的暴露集信息。

借助近似暴露集，下面的判定准则 I 可以处理嵌套循环：

判定准则 I：LP⊂L 是嵌套循环，r∈ $R_{Lv} \cap R_{LP}$，如果 $AUE_{Lv}(r) = \varnothing$，那么 r 不妨碍私有化。

由于相关性测试可以一次测试出在所有嵌套循环中是否有相关，而 Set 算子总是逐层计算的，因此如果避免一些重复计算，上述的计算就不会增加计算复杂性。

5.3.2.3 用输出相关性质过滤不必私有化判定的数组

到目前为止，我们的判定准则都是针对某一层循环的，而实际上许多层循环都需要进行私有化判定。但很多情况下，我们并不需要对每层循环分别使用判定准则：因为需要私有化的临时数组都是一些在各循环体内被反复使用的临时变量，这时一定会出现跨循环的输出相关，因此我们可以通过判定某层循环是否有跨循环的输出相关，来决定该循环是否需要使用判定准则来进行私有化判定，如果对某层循环不存在跨循环的输出相关则可以跳过对它的私有化判定。此外每次使用判定准则后，输出相关性的中间计算结果可以保留，以便重用。

算法 5-1:数组私有化识别算法

ALGORITHM: Determine if a read reference to array holds privatization in a loop.
INPUTS:
- r -- the read references to be tested
- A -- the array being referred by r
- L -- the complex loop whose body encloses r, which has M nests: $L^M \subset ... L^1=L$ and L^M is the innermost loop nest that encloses r, denoted as $r \subset body(L^M) \subset L^M \subset ... L^1=L$.

OUTPUT:
- Privatizable(r,A,L) -- determines whether the r holds the privatization of A in L

PREDEFINED OPERATIONS:
- Set(P,X) -- compute the $Set_P(X)$.
- DSet(P,X) -- compute the $DSet_P(X)$.
- NADW(r,P) -- compute the non anti-dependence write set with respect to r in P.
- LCOD(L) -- test if loop L carries output-dependence among write references of A.
- SCW(A,L) -- test if the write reference set to A is self-covered.in L.
- FC(w,L) -- test if the write reference(s) w is first covered in loop L.

IMPLEMENTATION:

```
begin
    if not LCOD(L) return false ; ignore the nest without loop carried output dependence
    for each w∈NADW(r, L), and w ⊂ body(L^S) ⊂ L^S ⊂ ... L^1 = L,
        DSet(body(L^S),w) = w
        for s = S,1,-1
          if FC(w,L^s) = true
                DSet(L^s,w)=DSet(L^s_1,w)
          else
                DSet(L^s,w) = ⋃ DSet(body(L^s_i),w)
                              1≤i≤tc
          end if
        end for
    end for
    Privatizable = false
    if subscripts in r is computable                    ; criterion I can be used
        AUE = Set(body(L^M),r)
        Privatizable = (AUE=Φ)
    endif
    m = M
    while m>0 and not Privatizable
        m = m - 1
        if SCW(A,L^m) and Set(L^m,W)⊆DSet(L^m,NADW(r,L^m)) ; criterion II success
            Privatizable = true
        else if subscripts in r is computable                ; criterion I success
            AUE = Set(L^m,AUE) - DSet(L^m, NADW(r, L^m))
            Privatizable = ( AUE = Φ )
        endif
    end while
    return Privatizable
end
```

5.3.3 处理所有控制结构的私有化判定算法

综合以上所述推广的判定准则，我们提出私有化判定算法(算法 5-1)。

5.3.4 截取自 APSI 的较复杂实例

```
      SUBROUTINE DCTDXD(...,SAVEX,...)
      DIMENSION HELP(NX),SAVEX(2*NX+15),...
      DO 100 J=1,NY
        ...
        CALL RFFTF1(NX, HELP, SAVEX,
     SAVEX(NX+1),
                          SAVEX(2*NX+1)
        ...
  100   CONTINUE
      END
      SUBROUTINE RFFTF1 (N,C,CH,WA,IFAC)
      DIMENSION CH(*),C(*),WA(*),IFAC(*)
      NF=IFAC(2)
      NA=1
      L2=N
      DO 111 K=1,NF
         KH=NF-K1
         IP=IFAC(KH+3)
         L1=L2/IP
         IDO=N/L2
         IDL1=IDO*L1
         NA=1-NA
         ...
         SWITCH ( IP )
         CASE 4:
            IF (NA .NE. 0) THEN
            CALL RADF4 (IDO,L1,CH,C,...)
            ELSE
            CALL RADF4 (IDO,L1,C,CH,...)
            ENDIF
         CASE 2:
            IF (NA .NE. 0) THEN
            CALL RADF2 (IDO,L1,CH,C,WA(IW))
            ELSE
            CALL RADF2 (IDO,L1,C,CH,WA(IW))
            ENDIF
         CASE 3:
            IF (NA .NE. 0) THEN
            CALL RADF3 (IDO,L1,CH,C,...)
            ELSE
            CALL RADF3 (IDO,L1,C,CH,...)
            ENDIF
         CASE 5:
            IF (NA .NE. 0) THEN
            CALL RADF5 (IDO,L1,CH,C,...)
            ELSE
            CALL RADF5 (IDO,L1,C,CH,...)
            ENDIF
         DEFAULT:
            IF (IDO .EQ. 1) NA = 1-NA
            IF (NA .NE. 0) THEN
              CALL RADFG
     (IDO,IP,L1,IDL1,CH,CH,CH,
     *                          C,C,WA(IW))
            NA = 0
            ELSE
              CALL RADFG (IDO,IP,L1,IDL1,C,C,C,
     *                          CH,CH,WA(IW))
            NA = 1
            ENDIF
         ENDSWITCH
         L2=L1
  111   CONTINUE
        ...
      END

      SUBROUTINE RADF2 (IDO,L1,CC,CH,WA1)
      DIMENSION CH(IDO,2,L1), CC(IDO,L1,2),
     WA1(*)
        DO 101 K=1,L1
          CH(1,1,K) = CC(1,K,1)+CC(1,K,2)
          CH(IDO,2,K) = CC(1,K,1)-CC(1,K,2)
        101 CONTINUE
        IF (IDO-2) 107,105,102
        ....
        107 CONTINUE
        RETURN
        END

          SUBROUTINE RFFTF1x (N,C,CH,WA,IFAC)
          DIMENSION CH(*),C(*),WA(*),IFAC(*)
          NF=IFAC(2)
          NA=1
          L2=N
          KH=NF-K1
          IP=IFAC(KH+3)
          L1=N/IP
          IDO=1
```

```
      IDL1=L1                                          ENDSWITCH
      NA=0                                             L2=L1
      ...                                              DO K=2,NF
      SWITCH ( IP )                                      ...
      CASE 4:                                          ENDDO
        CALL RADF4x(1,N/4,C,CH,...)                    ...
      CASE 2:                                          END
        CALL RADF2x(1,N/2,C,CH,WA(IW))
      CASE 3:                                    SUBROUTINE RADF2x(IDO,L1,CC,CH,WA1)
        CALL RADF3x(1,N/3,C,CH,...)              DIMENSION CH(1,2,L1), CC(1,L1,2), WA1(*)
      CASE 5:                                    DO 101 K=1,L1
        CALL RADF5x(1,N/5,C,CH,...)                CH(1,1,K) = CC(1,K,1)+CC(1,K,2)
      DEFAULT:                                     CH(1,2,K) = CC(1,K,1)-CC(1,K,2)
       CALL RADFGx(1,IP,N/IP,N/IP,CH,CH,CH       101 CONTINUE
     *             C,C,WA(IW))                   RETURN
        NA = 0                                     END
```

图 5-5、截取自 SPECfp95 测试程序 APSI 的实例

图 5-5实例把截取自 SPECfp95 测试程序 APSI.F 的程序段改写整理成易读的形式。在 APSI.F 中占据主要计算量的是标号 20,30,40,50,60,70 和 100 的循环。对子程序 DCTDXD 中的循环 J 并行化的困难在于如何计算子程序 RFFTF1 中的数组 CH 的写区域和暴露区域，所以必须分析象 RFFTF1 子程序中 K 循环这样复杂的循环。

如上程序中所述 K 循环的分支语句将控制流简化为五种情况，分别调用五个子程序：RADF2, RADF3, RADF4, RADF5, RADFG。但是如果只考虑 J 循环第一次循环迭代，则可以简化 K 循环的分析。许多常数可以传播到子程序中，从而消除那些不会执行到的 IF 分支，使子程序 RFFTF1, RADF2, RADF3, RADF4, RADF5, RADFG 在 J 循环首次迭代时简化为 RFFTF1x, RADF2x, RADF3x, RADF4x, RADF5x, RADFGx。

如果 IP=2，在 J 循环首次迭代中 NA 等于 1，使 RADF2x 中消去了 IF 分支，从而 CH 数组显然是肯定写，其写引用区域为 CH(1:N/2*2)。如果 MOD(N,2).EQ.0，则 N/2*2=N。在 J 循环其它迭代中，由于 NA 可能等于 0，CH 数组既可能被读引用，也可能被写引用。根据 RADF2 的数组说明，CH 的引用区域不超过 CH(1:IDO*2*L1)。向前替代后 IDO*2*L1 简化为 N/L2*2*L2/2≤N。

由于 MOD(N,IP).EQ.0 可以根据 IFAC 数组的初始化程序段得出，所以 N/IP*IP=N。类似于 RADF2，对于其它子程序 RADF3, RADF4, RADF5, RADFG，我们都可以分析出 J 循环首次迭代时，DSet(L, NADW(r,L))=CH(1:N/IP*IP) = CH(1:N)⊇ CH(1:N/L2*IP*L2/IP)，一定包含 J 其它迭代的读引用集 Set(L,R)和写引用集 DSet(L,W)。因此 CH 数组的确定写引用集是首覆盖的，并且用于替代整体写集计算的首覆盖部分写集包含所有迭代的读引用集。应用相关—覆盖方法的判定准则 I，我们知道该实例的数组 SAVEX 在 J 循环是可以私有化的。

5.4 相关工作比较

1988 年 Feautrier 首先提出了数组扩张的方法[29]，开数组私有化研究之先河。九十年代以来，随着伊利诺依大学的实验性研究工作的巨大进展和Ω测试的提出，数组私有化方法的研究达到了高潮，发表了大量关于数组私有化的论文[43][45][48]。按流行的分类方法，数组私有化方法可以分为数据流方法和相关性方法。数据流方法一般采用传统的标量数据流分

析框架来进行数组的数据流分析，在分析中采用数据覆盖技术[45]。相关性方法一般只能处理没有 IF 语句的程序，它要求数组下标表达式和循环界限都是线性函数，它在循环迭代空间中利用线性约束条件，求解最近相关即真相关，并根据真相关来判定数组是否可私有化[48]。一般说来，数据流方法是不精确的但处理范围较广、处理效率高；相关性方法在它的处理范围内是精确的，但超出范围后就不能使用且处理效率低。近年来，这两种方法有相互结合的趋势，但它们或者是在数据流方法中用线性约束来表示集合，或者是将相关性方法推广到某些特殊的 IF 语句，因此在本质上没有太大的变换。

本节的方法没有采用数据流的框架，但使用了数据流方法中的覆盖技术，没有计算真相关，但利用了传统的相关性信息。虽然本节的方法是不精确的，但它在某些相关性方法的模型下是精确的。由于我们的方法可以直接计算任意一层循环的可私有化数组，而不是逐循环迭代计算，因此在实际计算时效率大大提高。我们的判定准则设法绕开了一些无法计算的因素，例如在读引用无法作用 Set 算子时只考虑写引用，而避开了对复杂读引用的直接计算。因此我们的方法可以判定出其它方法无法判定的可私有化数组。对于一些复杂情况的改进只需对个别数组表示集合和算子的定义加以修改，而整个判定框架不变。

下面我们将相关-覆盖方法同两个典型的方法相比较。一个是 Polaris[17]中的方法，它是典型的数据流方法；另一个是 SUIF[36][60]中的终写树(LWT)方法，它是典型的相关性方法。Polaris 和 SUIF 是当今国际上最著名的系统，它们的私有化方法有很强的代表性。

Polaris 的数组私有化方法是由内向外逐层计算暴露集的过程，当一个循环体的暴露集为空时，这层循环的数组可私有化。它以控制流图为基础，对每个读引用在控制流图上找其之前的写引用集，并求读元素集和这些写元素集的差。对于多个分支，则对每个分支的结果进行相应的交、并运算，对 DO 循环则将该循环所有循环体的暴露集之并作为整个循环的暴露集。这种方法对 IF 语句和顺序语句的处理同相关-覆盖方法相同，但对循环的处理弱于相关-覆盖方法，因为按控制流顺序找到的写引用同读引用之间总是没有反相关，所以它的计算弱于 $AUE_{Lv}(r)$。由此可以看出，相关-覆盖方法的能力强于基于数据覆盖的 Polaris 的方法。特别是对循环的处理，由于相关-覆盖方法使用了无反相关写的概念，因此可以得到比较准确的结果。

SUIF 的采用了求终写树(LWT)的技术。这种技术要求，对程序的每个写引用 w 而言，包含 w 的每个循环 L 都具有下述性质：或者 L 的循环参数既不出现在 w 中，也不出现在包含 w 的其它循环的循环界限中；或者 w 在 L 中是非自相干的(None Self-Interfere)，即对 i≠j，$Set_{L_i}(w) \cap Set_{L_j}(w) = \varnothing$。我们分别称终写树模型中的上述两类循环为 w 的无关循环和非自相干循环。终写树方法非常复杂，其核心是处理一对读写，然后通过将每对读写的分析结果近似的综合，来判定数组是否可私有化。

现在我们用 LWT 来分析一对读写的情况。注意到只有无关循环才需要私有化。对 w 的无关循环 L 而言，因为对 i≠j，$Set_{L_i}(w) = Set_{L_j}(w)$，显然 L 是写自覆盖的而且 w 是首覆盖的。

下面的定理 5-2给出了循环 L 中数组可私有化的充要条件。

定理 5-2：如程序中仅有一对数组的读写对(r,w)，L 为包含(r,w)的无关循环，包含 w 的嵌套循环为 $L^1 \subset L^2 \subset \cdots \subset L^n \subset L$，其中 $L^{i_1}, L^{i_2}, \ldots, L^{i_m}$ 为无关循环，那么数组可私有化的充要条件为：对 2≤i≤tc，r 在 L_i 中与 $w(L_1^{i_1}, L_1^{i_2}, \ldots, L_1^{i_m})$ 无反相关。

证明：由于 L 是写自覆盖的，w 在无关循环中是首覆盖的，如果对 2≤i≤tc，r 在 L_i 中与 $w(L_1^{i_1}, L_1^{i_2}, \ldots, L_1^{i_m})$ 无反相关，那么由判定准则 II 我们可知 r 不妨碍私有化。因此循环是私有化的。

反之，如果对 2≤i≤tc，r 在 L_i 中与 $w(L_1^{i_1}, L_1^{i_2}, \ldots, L_1^{i_m})$ 有反相关，那么必存在 A(j)使得 A(j)∈ $Set_{L_i}(r) \cap Set_{L_i}(w(L_1^{i_1}, L_1^{i_2}, \ldots, L_1^{i_m}))$ 且 A(j)先读。由于 L 的循环体中除 $L^{i_1}, L^{i_2}, \ldots, L^{i_m}$ 外

均为非自相干循环，因此在 L 的循环体中 $w(L_1^{i_1}, L_1^{i_2}, \ldots, L_1^{i_m})$ 必为对 A(j)的第一个写，所以 A(j)在 L_i 一定是暴露的。又因为 L 是无关的，因此 A(j)在 L_{i-1} 中必然被写，所以数组在 L 中不是可私有化的。证毕。

定理 5-2说明了，在适用终写树方法的模型下，如果使用精确的相关性测试[2]，那么相关-覆盖方法也可以得出精确结论。但我们的方法比终写树方法要简单高效得多。

[2] 终写树模型要求数组下标和循环界限都是线性的，这时Ω测试可以得出精确的相关性。

6 自动么模变换技术

么模变换是一类范围很广的循环变换，人们通常研究的一些基本循环变换(如斜错、交换、反序等)都是它的特例。么模变换具有很适合对循环并行化的性质：如保持整数循环迭代空间体积和维数不变，使规范循环仍变换为规范循环等等。但是对于已知 n 维距离向量矩阵的多重串行循环，过去的并行化编译研究还缺乏寻找使循环外层并行化的么模矩阵的可行算法。本节介绍了多重串行循环并行化的么模变换方法，不仅从理论上证明满足外层并行化要求的合法么模矩阵是存在的，而且通过构造性证明给出一个计算外层并行化么模变换矩阵的可行算法，并提出了扩大其适用范围的有效途径。

6.1 循环么模变换的实用意义

并行化编译是串行程序并行化的有效工具，它结合相关性分析发现串行程序中有并行性的程序段并施以合法的并行化变换，从而使此程序段能并行执行**[63]**。因此，有效的相关性分析**[12]**和自动并行化变换方法**[64]**是并行化编译成功的关键。串行程序的循环具有很大的并行性，因此循环并行化变换自然成为并行化变换研究的重点，人们发现了许多有效的循环变换手段**[11][24][61][62]**，么模变换便是其中之一。

么模变换是一类范围很广的循环变换，人们为了发掘程序中的并行性而通常研究的一些基本循环变换(斜错(skewing) **[62]**、交换(interchange)**[61]** 、反序(reversal)等)都是它的特例。么模变换具有保持整数循环迭代空间体积和维数不变的性质，不仅如此，步长为 1 的规范循环在么模变换后仍是步长为 1 的规范循环，变换后程序书写格式仍较为规整。么模变换是适合对循环并行化变换的一种重要手段。

6.2 循环么模变换的主要困难

但是，使用么模变换进行循环自动并行化的困难在于如何对给定多重串行循环，自动找出使循环并行化的恰当的么模变换矩阵。现有的大多数自动并行化编译器**[76]**没有实现一般的自动并行化么模变换, 就是因为无法对多重循环自动计算能使之并行化的么模变换矩阵。而利用交互并行程序设计环境**[58]**提供的迭代空间相关图可视化工具**[59]**对循环程序的分析经验表明：即使由人来寻找恰好能对循环并行化的么模变换矩阵也是很困难的。本文研究的算法目标就是对给定多重串行循环，自动找出使循环并行化的恰当的么模变换矩阵。

对简单的二重循环，U.Banerjee 用算法说明了如何根据源循环中的跨循环常数相关距离，寻找合法的外层或内层并行化么模变换**[11]**。他的对二重循环内层并行化方法可以扩充至多重循环内层并行化，但他对二重循环的外层并行化方法无法直接处理多重循环外层并行化的情况。本文提出了一种确定使多重串行循环外层并行化的么模变换矩阵的算法，解决了相关距离已知时多重嵌套循环并行化的问题。

另外，当多重循环的常数距离矩阵满秩时，E.H.D'Hollander 的迭代划分和求代表迭代点的方法能够增加一个额外的外层并行循环，压缩内层串行循环的相关迭代链，进一步发掘外层并行化循环迭代**[24]**。我们的方法完全可以同 D'Hollander 的方法结合在一起应用。

6.3 寻找循环并行化么模矩阵

随着迭代空间的变换，循环的跨循环相关距离向量也一起变换, 因而有些循环变换可使目标循环并行化。对相关距离向量已知的循环，寻找合法的并行化变换一般有两种方法：侧重于使内层循环并行的 wavefront

方法和侧重于使最外层循环并行的 partition 方法。Wavefront 方法基本思想是根据相关约束，将迭代空间划分为一系列 wavefront 集合，每个 wavefront 中的循环迭代可以并行执行，跨循环相关只存在于不同 wavefront 的迭代之间。如果迭代空间必须划分为至少两个 wavefront，就必须构造一个串行外层循环，依次执行每个 wavefront，而 wavefront 的执行则是并行的内层循环。所有 wavefront 的个数不大于循环迭代的关键串行路径长度的 wavefront 变换就是最优 wavefront 变换。Partition 方法基本思想是根据相关约束，将迭代空间划分为一系列 partition 集合，每个 partition 中的循环迭代存在相关链，不能并行执行，但是跨循环相关的迭代只存在于同一 partition 中，不同 partition 的迭代之间不存在跨循环相关。这样，如果迭代空间可以划分为两个以上的 partition，就可以构造一个并行最外层循环，每个迭代执行一个 partition，而一个 partition 中的循环迭代则串行执行。每个 partition 都无法进一步分解的 partition 变换就是最优 partition 变换。

表 6-1、二重循环迭代空间幺模变换

例子	**D**	**U**	**UD**	边界条件
DO i_1=5,100 DO i_2=16,80 $a(i_1,i_2)=a(i_1-2,i_2-4)+a(i_1-3,i_2-6)$ ENDDO ENDDO	$\begin{bmatrix}2 & 3\\4 & 6\end{bmatrix}$	$\begin{bmatrix}2 & -1\\1 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\2 & 3\end{bmatrix}$	LL_1=-70 UU_1=184 $LL_2=\max\{5, (\lceil 16+j_1)/2\rceil\}$ $UU_2=\min\{100, \lfloor(80+j_1)/2\rfloor\}$
DO i1=5,100 DO i2=5,100 a(i1,i2)=a(i1,i2-1)+a(i1-2,i2+3) +a(i1-3,i2+7) ENDDO ENDDO	$\begin{bmatrix}0 & 2 & 3\\1 & -3 & -7\end{bmatrix}$	$\begin{bmatrix}3 & 1\\1 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 3 & 2\\0 & 2 & 3\end{bmatrix}$	LL1=20 UU1=400 LL2=max{5, $(\lceil -100+j1)/3\rceil\}$ UU2=min{100, $\lfloor(-5+j1)/3\rfloor\}$

U.Banerjee 说明了如何根据源循环中的跨循环常数相关距离，对二重循环寻找合法并行化幺模变换[1]。如表 6-1，第一个例子具有平行（线性相关）的距离向量 $\mathbf{d}_1^T$=(2,4)和 $\mathbf{d}_2^T$=(3,6)组成距离向量矩阵 **D** 的列。通过计算行变换幺模矩阵 **U**，将源串行循环变换为合法的最外层并行循环(**UD** 的首行为 **0**)，这是一种 partition 技术。第二个例子具有三个距离向量，距离向量矩阵 **D** 行满秩[6]，可找到 **U** 阵，使内层循环并行化(**UD** 的首行元素全大于零)，即一种 wavefront 技术。

上述方法是否可以扩充至 n(>2)重循环的情况呢？Banerjee 文[1]中对循环外层并行化的幺模矩阵计算方法仅仅适用于二重循环，对 n>2 重循环并不适用。本文探讨的 n 重循环幺模并行化算法是一个发掘最多可并行外层循环的幺模变换，使得 n-rank(**D**)个外层循环一定能够并行化，而且 rank(**D**)-1 个内层循环也能并行化。下面的中心就是论证在 n 维迭代空间中，只要所有 m 个相关距离向量都是常数：

- 如果给定 n×m 的距离向量矩阵 **D** 的秩 rank(**D**)<n，就可以找到一个合法的行变换幺模矩阵 **U**，使得 $\mathbf{UD}=[\mathbf{0} \mid \mathbf{D'}^T]^T$，**D'**是 rank(**D**)×m 的合法距离向量矩阵且为行满秩。
- 如果 **D'**为行满秩矩阵，可以找出一个幺模变换 **U'**，使除了最外重循环[7]外，所有内层循环可并行化。

如果 **D** 为满秩可逆矩阵，行列式|**D**|>1，就还可以用 E.H.D'Hollander 的迭代划分和求代表迭代点的方法，通过增加一个外层并行循环，使所有内层串行循环的相关迭代链是紧凑的[2]。所以在用我们的方法进行内层并行化之前，如果 rank(**D'**)=m 且|**D'**|>1，可以利用 D'Hollander 的方法发掘出可能的并行化循环(共|**D'**|

[6] 这里，对 n×m 矩阵 **A**，只要 rank(**A**)= n 即称为行满秩，rank(**A**)= m 即称为列满秩，行、列均满秩则为满秩可逆阵。
[7] 实际上是第 n-rank(**D**)+1 层循环。如果 rank(**D**)=0，则不存在需要进一步并行化的串行循环。

个并行循环迭代)，然后再对其余内层循环并行化，最后结果将得到 n-rank(**D**)+1 个并行外层循环，一个串行循环和 rank(**D**)-1 个并行的内层循环。

6.3.1 使外层循环并行化

在所有 m 个相关距离向量都是常数的 n 维迭代空间中，如果给定 n×m 的距离向量矩阵 **D** 的秩 rank(**D**)<n，只要找到一个合法的行变换幺模矩阵 **U**，使得 **UD**=[**0** | **D'**^{T}]^T，**D'是** rank(**D**)×m 的合法距离向量矩阵且为行满秩，就可以对外层 n-rank(**D**)重循环并行化。

循环并行化变换必须保证在原来迭代空间中存在跨循环相关的两个循环迭代保持串行迭代执行顺序，亦即要求距离向量的合法性。

定义 6-1 (合法的距离向量)：n 维相关距离向量 **d** 为合法(valid)的，如果它是按字典序大于 **0** 向量的整数向量：**0**$\angle_n$**d**。相关距离矩阵 **D** 是合法的，如果组成它的每个距离向量 **d**$_1$,...,**d**$_m$ 都是合法的。

引理 6-1 (一定合法的变换矩阵充要条件)：对 n x n 阶幺模矩阵 **U**：如果任意合法的 n x m 阶距离向量矩阵 **D**，**D'=UD** 也是合法的距离向量矩阵，当且仅当 **U** 是下三角矩阵，并且对角线上的所有元素等于 1。

证明：(⇐)因为 **D** 是合法距离向量矩阵，$d_{11}\geq 0, d_{12}\geq 0, ..., d_{1m}\geq 0$；又 **U** 是下三角幺模矩阵，并且对角线上的所有元素等于 1。对任意 **d'**$_i$=**Ud**$_i$, $d'_{1i}= d_{1i} \geq 0$。若 $d'_{1i}>0$，则 **d'**$_i$ 为合法距离向量。若 $d'_{1i}=0$, 则 $d_{1i}=0$，考查 $d'_{2i}=u_{21}*d_{1i}+d_{2i}= d_{2i}\geq 0$, 若 $d'_{2i}>0$，则 **d'**$_i$ 为合法距离向量。若 $d'_{2i}=0$, 则 $d_{2i}=0$，考查 $d'_{3i}=u_{31}*d_{1i}+u_{32}*d_{2i}+d_{3i}= d_{3i}\geq 0$, ...依此类推，若 $d'_{1i}=d'_{2i}=....=d'_{k-1,i}=0$，则 $d_{1i}=d_{2i}=...=d_{k-1,i}=0$，$d'_{k,i}= \sum_{j=1}^{k} u_{kj}*d_{ji}=d_{ki} \geq 0$，...，综上所述，除非 $d'_{1i}=d'_{2i}=....=d'_{n,i}=0$，否则必有一个 k，使 $d'_{ki}>0$。所以 **d'**$_i$ 合法。

(⇒)(1) 对 j>1，若 $u_{1\,j}<0$，取 **d**=[...,0,d_j=1,0...]^T，则 $d'_1=u_{1\,j}<0$，**d** 合法但 **d**'不合法；若 $u_{1\,j}>0$，取 **d**=[d_1=1,...,0,d_j= -|u_{11}|/u_{1j} -1,0...,0]^T，则 $d'_1= u_{11}-|u_{11}| /u_{1j}*u_{1\,j} -u_{1\,j}<0$，**d** 合法但 **d**'不合法。故 $u_{12}=...=u_{1n}=0$，...。对 j>k，若 $u_{1j}=...=u_{k-1\,j}=0$, 但 $u_{kj}<0$，取 **d**=[...,0,d_k=1,0...]^T，则 $d'_1=u_{kj}<0$，**d** 合法但 **d**'不合法；若 $u_{k-1\,j}>0$，取 **d**=[...0,d_k=1,-| u_{kk} | /$u_{k\,j}$-1,0...]^T，则 $d'_1=u_{kk}-|u_{kk}|/u_{1j*}u_{kj}-u_{kj}<0$，**d** 合法但 **d**'不合法。故 $u_{kk+1}=...=u_{kn}=0$。所以 U 一定是下三角阵。**(2)** 因为 U 是满秩下三角阵，所以 $u_{kk}\neq 0$。若 $u_{kk}<0$, 取 **d**=[...0,d_k=1,0...]^T，则 **d**'=[...,0,$d'_k=u_{kk}<0$,...]^T，**d** 合法但 **d**'不合法，故 **U** 对角线元素大于 0。因为 **U** 是幺模矩阵，所以其下三角阵对角线的乘积等于行列式±1，所以这些对角线元素都等于 1。证毕。

由**引理 6-1**可知，三种基本幺模变换中，反序阵对角线元素不全大于 0，交换阵不是下三角阵，所以它们不能保证对所有合法相关距离向量变换结果仍然合法。而只有下三角行斜错变换才能保证变换的合法性。那么，对某个相关距离矩阵 **D**, rank(**D**)<n，是否有可能找到合法的幺模变换矩阵 **U**，使得 **UD**=[**0** | **D'**^{T}]^T，**D'是** rank(**D**)×m 的合法距离向量矩阵且为满秩？下面的**定理 6-1**给出了肯定的回答。

定理 6-1(合法的消元幺模矩阵存在性)：在 n 维迭代空间中，只要所有 m 个相关距离向量都是常数，给定 n×m 的距离向量矩阵 **D** 的秩 s=rank(**D**)<n，一定存在一个合法的行变换幺模矩阵 **U**，使得 **UD**=[**0** | **D'**^{T}]^T，**D'是** rank(**D**)×m 的合法距离向量矩阵且为行满秩。

$$\mathbf{D}=\begin{array}{c} \\ e_1 \\ ... \\ e_n \end{array}\begin{array}{c} \begin{array}{ccc} d_1 & ... & d_m \end{array} \\ \begin{bmatrix} d_{1\,1} & ... & d_{1\,m} \\ ... & ... & ... \\ d_{n\,1} & ... & d_{n\,m} \end{bmatrix} \end{array}$$

算法 6-1: 计算外层并行化幺模矩阵

构造性证明:

首先，将 n 维相关距离向量按列横排的 **D**=[**d**$_1$,...,**d**$_m$]表示为行向量的按列纵排 **D**=[**e**$_1$,...,**e**$_n$]^T，其中每个 **e**$_i$ 都是 m 维行向量。这是以下计算的关键。寻找 **U**，使 **UD** 为[**0**,...,**0**,**e**$_{n-s+1}$,...,**e**$_n$]^T 的步骤如下：

1.(初始化)s=rank(**D**), t=1，z=0，E_0={}，E_1={**e**$_1$}，E_2={ **e**$_2$,...,**e**$_n$}，T=0 ，UU={}

/* t 记录已经处理的线性无关 **e** 向量个数，z 记录已变换为 **0** 向量的个数,
T 记录中间幺模变换个数, */

2.IF z=n-s GOTO 4。

3.For $E_0=\{\mathbf{e}_1=\mathbf{0},...,\mathbf{e}_z=\mathbf{0}\}$，$E_1=\{\mathbf{e}_{z+1},...,\mathbf{e}_{z+t}\}$, $E_2=\{\mathbf{e}_{z+t+1},...,\mathbf{e}_n\}$，**D** 为合法的相关距离矩阵。

IF rank($E_1+\{\mathbf{e}_{z+t+1}\}$)=t+1 THEN

$E_1=E_1+\{\mathbf{e}_{z+t+1}\}$，$E_2=E_2-\{\mathbf{e}_{z+t+1}\}$

t=t+1

GOTO 2

ELSE

$\mathbf{e}_{z+t+1}$ 与 $\mathbf{e}_{z+1},...,\mathbf{e}_{z+t}$ 线性相关，存在非零整数 $a_1,...,a_t,a_{t+1}$，而且 $a_{t+1}>0$，$\gcd(a_1,...,a_t,a_{t+1})=1$

使 $a_1\mathbf{e}_{z+1}+...+a_t\mathbf{e}_{z+t}= a_{t+1}\,\mathbf{e}_{z+t+1}$. **(1)**

/* 这些系数可以通过以下方法计算：设 D_{sub} 为 D 的 t 阶满秩子矩阵(从 E_1 的 t 个 m 维 行向量中选取无关的 t 个列，D_{inc} 为 e_{z+t+1} 对应于这些列的 t 维子行向量。令 $a_{t+1}=|\det(D_{sub})|>0$，则 $[a_1,...,a_t]=a_{t+1}D_{inc}D_{sub}^{-1}$，由此得到的所有整系数再分别除以其 最大公约数，从而保证 $\gcd(a_1,...,a_t,a_{t+1})=1$。*/

3.1 计算下三角斜错矩阵

WHILE $a_{t+1}\neq 1$ DO /*辗转相除法*/

FOR i=1,...,t,

$b_i=\lfloor a_i/a_{t+1}\rfloor$ **(2)**

$a_i=a_i \bmod a_{t+1}$ **(3)**

ENDFOR /* $a_{t+1}>a_i\geq 0$, 对所有 i=1,...,t */ **(4)**

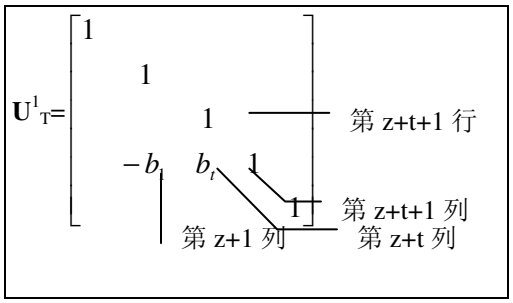
T=T+1

UU=UU+$\{\mathbf{U}^1{}_T\}$

$\mathbf{D}=\mathbf{U}^1{}_T\mathbf{D}$

/* $\mathbf{e}_{z+t+1}= - b_1\mathbf{e}_{z+1} -... - b_t\mathbf{e}_{z+t} + \mathbf{e}_{z+t+1}$. **(5)**

由**引理 6-1**，若 **D** 是合法的，则 $\mathbf{U}^1{}_T\mathbf{D}$ 是合法的*/



3.2 计算交换矩阵

T=T+1

UU=UU+$\{\mathbf{U}^2{}_T\}$

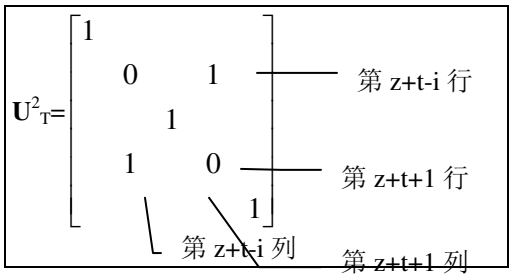
$\mathbf{D}=\mathbf{U}^2{}_T\mathbf{D}$。

/* 其中 i 为令 $a_{t-i}>0$ 的最小非负整数，

$a_{t-i+1}= ...=a_t=0$。 **(6)**

若 **D** 是合法的，则 $\mathbf{U}^2{}_T\mathbf{D}$ 是合法的：

因为若 **D** 是合法的，若交换 $\mathbf{e}_{z+t-i}$ ， $\mathbf{e}_{z+t+1}$ 不合法，则必须存在 j 使 $d_{1j}=...=d_{z+t-i-1,j}=0$, $d_{z+t-i,j}>0$ 且 $d_{z+t+1,j}<0$，但是由**(1)**、**(4)**、**(6)**式可知 $d_{z+t+1,j}=a_{z+t-i}*d_{z+t-i,j}/a_{z+t+1}>0$，矛盾。*/



3.3 计算下三角斜错矩阵

/*为了使**(1)**不变，修正系数*/

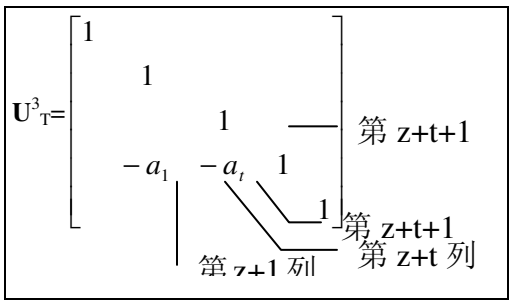
$a_1= - a_1$, ..., $a_{t-i-1}=-a_{t-i-1}$,$a_{t-i}=a_{t+1}$, $a_{t+1}=a_{t-i}$

/* 由**(2)**、**(3)**、**(4)**、**(5)**式可知**(1)**式对新的 $a_1,...,a_t,a_{t+1}$ 和 $\mathbf{e}_{z+t+1}, \mathbf{e}_{z+1},...,\mathbf{e}_{z+t}$ 仍成立 */

ENDWHILE /* 因为 $\gcd(a_1,...,a_t,a_{t+1})=1$，故 此循环一定会终止*/

/*消元*/ T=T+1，UU=UU+$\{\mathbf{U}^3{}_T\}$， $\mathbf{D}=\mathbf{U}^3{}_T\mathbf{D}$

/* 由**引理 6-1**，若 **D** 是合法的，则 $\mathbf{U}^3{}_T\mathbf{D}$ 是合法的*/



3.4 计算循环置换矩阵：

/* 置换 **0** 向量 */

T=T+1，

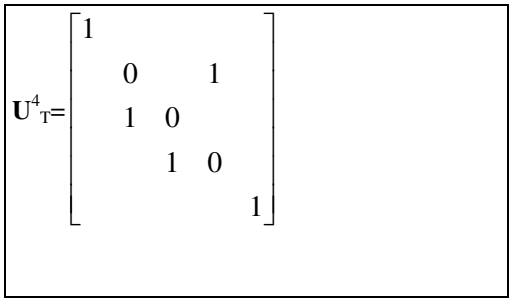
UU=UU+$\{\mathbf{U}^4{}_T\}$，

$\mathbf{D}=\mathbf{U}^4{}_T\mathbf{D}$ /*因为 $\mathbf{e}_{z+t+1}=\mathbf{0}$，若 **D** 是合法的，则$[\mathbf{0}^T, \mathbf{e}_1{}^T,...,\mathbf{e}_t{}^T,\mathbf{0}^T]^T$ 是合法的，所以 $\mathbf{U}^4{}_T\mathbf{D}=[\mathbf{0}^T,\mathbf{e}_1{}^T,...,\mathbf{e}_t{}^T]^T$ 也是合法的*/

$E_0 = E_0+\{\mathbf{0}\}$

$E_2=E_2-\{\mathbf{e}_{z+t-1}\}$

z=z+1



ENDIF
ENDFOR

4. $\mathbf{U}=\mathbf{U}_T\mathbf{U}_{T-1}...\mathbf{U}_1$ 即为所求幺模矩阵，$\mathbf{U}_T$, $\mathbf{U}_{T-1}$，...，$\mathbf{U}_1\in$UU。 证毕。

推论 1：若距离向量个数少于迭代空间维数(m<n)，必有幺模变换使最外至少 n-m 层循环并行化。

由于上述算法中 **U** 的计算只涉及行变换，所以有下列推论。

推论 2：若 **U** 是根据**定理 6-1**对给定距离向量 $\mathbf{d}_1,...,\mathbf{d}_m$ 计算出的合法幺模变换矩阵，则只要在原迭代空间中合法的距离向量 **d** 属于以 $\mathbf{d}_{j1},...,\mathbf{d}_{js}$ 为基向量张成的 s 维子空间，在变换后新迭代空间中 **Ud** 仍旧是合法的，且也能最外 n-s 层并行化(**Ud** 前 n-s 个分量为 0)。

证明：因为距离向量 **d** 仍落在原基向量 $\mathbf{d}_{j1},...,\mathbf{d}_{js}$ 张成的 s 维整数子空间中，所以存在一系列非全零实数[8] $a_1,...,a_s$，使 $\mathbf{d}= a_1\mathbf{d}_{j1}+...+a_s\mathbf{d}_{js}$。$\mathbf{Ud}=\mathbf{U}(a_1\mathbf{d}_{j1}+...+a_s\mathbf{d}_{js})= a_1\mathbf{U}\mathbf{d}_{j1}+...+a_s\mathbf{U}\mathbf{d}_{js}$。因为所有 $\mathbf{d}_{j1},...,\mathbf{d}_{js}$ 的前 n-s 个分量为 0，所以 **Ud** 前 n-s 个分量为 0。得证。

定理 6-2：算法 6-1 计算得到的幺模变换是外层并行层次最多的幺模变换，即不能再有其它幺模变换产生更多可并行化外层循环。

证明: 若有多于 n-rank(**D**)的行可变换为 **0**，则 rank(**UD**)<rank(**D**)，而幺模变换应不改变矩阵秩，矛盾。得证。

6.3.2 使内层循环并行化

定理 6-3：在 n 维迭代空间中，只要所有 m 个相关距离向量都是常数，给定 n×m 的距离向量矩阵 **D** 的秩 s=rank(**D**)=n，一定存在一个合法的行变换幺模矩阵 **U**，使得 **UD 是** n×m 的合法距离向量矩阵且所有距离向量的第 1 个元素大于零(**UD** 的首行元素大于 0)。

算法 6-2：计算内层并行化幺模矩阵

证明：

若存在一系列非负整数 $u_1,...,u_{n-1}$，$d_{nj}+\sum_{i=1}^{n-1}u_i d_{ij}>0$ for j=1..m，则幺模矩阵：

$\mathbf{U}_T=\begin{bmatrix} u_1 & ... & u_{n-1} & 1 \\ & & 1 & \\ & ... & & \\ 1 & & & \end{bmatrix}$ 即为所求。令 k=1, $u_{n-1}=\max\{0,\ \left\lceil \frac{-d_{n,j}}{d_{n-1,j}} \right\rceil \mid 1\le j\le m, d_{1\,j}=...=d_{n-2\,j}=0,\ d_{n-1\,j}>0\ \}$

,...., k=n-1, $u_1=\max\{\left\lceil -\frac{-d_{n,j}+\sum_{k=1}^{n-2}u_{n-k}d_{n-k,j}}{d_{1,j}} \right\rceil \mid 1\le j\le m,\ d_{1\,j}>0\ \}$。证毕。

由算法 6-1，我们对 **D** 作幺模 $\mathbf{U}_1$ 变换得到 $\mathbf{U}_1\mathbf{D}=[\mathbf{0}^T|\mathbf{D'}^T]^T$，**D'**为 s=rank(**D**)阶行满秩阵，现在根据算法 6-2 对 **D'**得到的 $\mathbf{U}_s$ 作 $\mathbf{U}_2=\begin{bmatrix} I_{n-sxn-s} & 0_{n-sxs} \\ 0_{sxn-s} & U_{sxs} \end{bmatrix}$，则 $\mathbf{U}_2\mathbf{U}_1\mathbf{D}$ 完成进一步对内层循环并行化的任务。

6.4 计算循环迭代空间边界

n 重循环迭代空间 I 为$\{(i_1,...,i_n)\in\mathbf{Z}^n \mid L_1\le i_1\le U_1,\ L_2(i_1)\le i_2\le U_2(i_1),\ ...,\ L_n(i_1,...,i_{n-1})\le i_n\le U_n(i_1,...,i_{n-1})\}$,变换后的

[8] 可以证明为有理数

迭代空间 J 为$\{(j_1,...,j_n)\in \mathbf{Z}^n \mid LL_1\le j_1\le UU_1$，$LL_2(j_1)\le j_2\le UU_2(j_1)$，...，$LL_n(j_1,...,i_{n-1})\le j_n\le UU_n(j_1,...,j_{n-1})\}$。由于绝大多数循环使用线性边界表达式，所以可以将 I 空间边界条件表示为 **Ai≤b** 的形式，其中 **A** 是一个 2n×n 的整系数矩阵，**b** 是一个 2n×1 的整系数向量: $L_k=-b_{2k}+\sum_{j=1}^{k-1}a_{2k,j}i_j$, $a_{2*k\,k}$= -1, $U_k=b_{2k-1}-\sum_{j=1}^{k-1}a_{2k,j}i_j$, $a_{2*k-1\,k}$= 1，k=1,...,n。因为 **i=U**$^{-1}$**j**，所以在 J 空间中边界条件变为 **AU**$^{-1}$**j≤b**，只要用整数 Fourier-Motzkin 投影消元算法就可以计算出 LL_k 和 UU_k 的线性表达式。

算法 6-3(Fourier-Motzkin)：给定边界条件 **Ai≤b**，及算法 6-1和算法 6-2得到的幺模阵 $\mathbf{U}_1, \mathbf{U}_2$，计算 J 空间中 LL_k 和 UU_k 的线性表达式。

1. k=n, **C=[A(U**$_1$**U**$_2$**)**$^{-1}$**| b]=[AU**$_2^{-1}$**U**$_1^{-1}$ **| b]**
2. if k=1 goto 4
3. /* 消元 **C** 的第 k 列 */
   L={i | $c_{i\,k}$<0 }, U={i | $c_{i\,k}$>0 }
/*因为上下界必存在，所以|L|, |U|≥1*/
   for i ∈ L∪U
      g = gcd($c_{i\,1}$, ..., $c_{i\,k}$)
      for t=1,k
         $c_{i\,t}$= $c_{i\,t}$/g
      endfor
      $c_{i\,n+1}$=$\lfloor c_{i\,n+1}/g \rfloor$
   endfor
   dim **C'**(|L|*|U|,n+1) = **0**
   /* 新的系数矩阵 C' */
   s = 1
   for i ∈ L
      for j ∈ U
      LL_k= max(LL_k, $\lceil -c_{i\,n+1}+\sum_{t=1}^{k-1} c_{i\,t}*\mathbf{j}_t \ /c_{i\,k} \rceil$)
      UU_k= min(UU_k, $\lfloor c_{j\,n+1}+\sum_{t=1}^{k-1} -c_{j\,t}*\mathbf{j}_t \ /c_{j\,k} \rfloor$)
      g=gcd($c_{j\,k}$,$c_{i\,k}$)
      for t=1,k-1
         c'$_{s\,t}$ = ($c_{j\,k}$*$c_{i\,t}$ - $c_{i\,k}$*$c_{j\,t}$)/g
      endfor
      c'$_{s\,n+1}$ = ($c_{j\,k}$*$c_{j\,n+1}$-$c_{i\,k}$*$c_{i\,n+1}$)/g
      s = s+1
    endfor
   endfor
   k=k-1, **C=C'**
   goto 2
4. L={i | c_{i1}<0 }, U={i | $c_{i\,1}$>0 }
   for i ∈ L
      for j ∈ U
         LL_1= max(LL_1, $\lceil -c_{i\,n+1}/c_{i1} \rceil$)
         UU_1= min(UU_1,$\lfloor c_{j\,n+1}/c_{j1} \rfloor$)
      endfor
    endfor
end

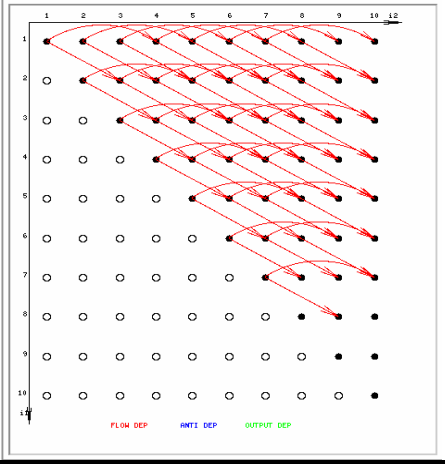
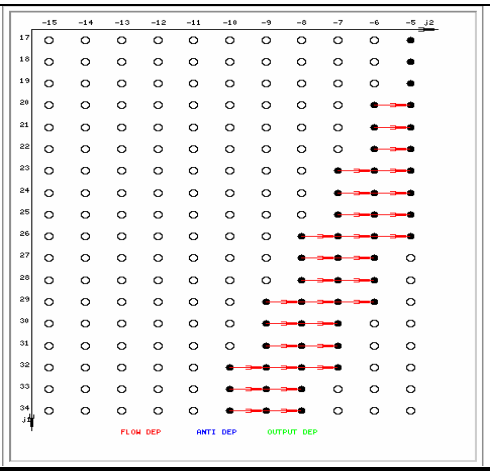
如果在原循环边界条件常数项 **b** 中出现循环不变符号变量，则通过符号表达式运算仍可应用本算法求出新循环的边界条件。

6.5 多重循环迭代空间幺模变换应用实例

表 6-2、多重循环循环幺模变换实例

	(a)原循环	(b)并行化后循环

程序	DO i1=1,100
	  DO i2=i1,100
	  DO i3=i1+i2,200
	  a(i1+1,i2+3,i3+2)= ...
	  ...=a(i1,i2+1,i3+5)+a(i1+1,i2,i3)
	  ENDDO
	  ENDDO
	ENDDO
		DOALL j1=17,1700
		  DO j2=max(ceil((2-j1)/3),ceil((-3*j1-400)/11)),
		  min(-5,floor((200-j1)/3),
		  floor((700-j1)/2),
		  floor((500-5*j1)/16))
		  DOALL j3=max(1, j1+3*j2-100,
		  ceil((j1+2*j2-200)/5)),
		  min(100, floor((-j2)/5),
		  floor((j1+3*j2)/2))
		  i1=j3
		  i2=j1+3*j2-j3
		  i3=j1+2*j2-5*j3
		  a(i1+1,i2+3,i3+2)= ...
		  ...=a(i1,i2+1,i3+5)+a(i1+1,i2,i3)
		  ENDDOALL
		  ENDDO
		ENDDOALL

对应的循环迭代相关图		

分析如表 6-2a 的循环，现根据算法 6-1求得幺模矩阵 U_1，使循环外层并行化，因为得到的 **D**'满秩可逆，|**D**'|=1，所以不需要再根据 D'Hollander 方法进一步分析 partition。但内层尚待使用算法 6-2并行化，得到幺模矩阵 U_2，从而完成了幺模矩阵的计算，如表 6-3。然后根据**算法 6-3**求得循环边界表达式(表 6-4)，这样就完成了幺模变换(表 6-2b)。由于所有算法都是自动的，所以不难在自动并行化变换工具中实现。

表 6-3、计算多重循环幺模变换矩阵结果

算法	距离向量矩阵 **D**	幺模矩阵 **U**	**UD**
外层并行化	$\begin{bmatrix}1 & 0\\ 2 & 3\\ -3 & 2\end{bmatrix}$	$\begin{bmatrix}13 & -2 & 3\\ 1 & 0 & 0\\ -4 & 1 & -1\end{bmatrix}$	$\begin{bmatrix}0 & 0\\ 1 & 0\\ 1 & 1\end{bmatrix}$
内层并行化	$\begin{bmatrix}0 & 0\\ 1 & 0\\ 1 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\ 1 & 1\\ 1 & 0\end{bmatrix}$

结果么模变换	$\begin{bmatrix}1&0\\2&3\\-3&2\end{bmatrix}$	$\begin{bmatrix}13&-2&3\\-4&1&-1\\1&0&0\end{bmatrix}$	$\begin{bmatrix}0&0\\1&1\\1&0\end{bmatrix}$

表 6-4、计算多重循环幺模变换边界条件

符号	A	b	$C_1=[AU_2^{-1}U_1^{-1}\mid b]$	C_2	C_3
边界矩阵	$\begin{bmatrix}1&0&0\\-1&0&0\\1&-1&0\\-1&1&0\\0&0&1\\1&1&-1\end{bmatrix}$	$\begin{bmatrix}100\\-1\\0\\100\\200\\0\end{bmatrix}$	$\begin{bmatrix}0&0&1&100\\0&0&-1&-1\\-1&-3&2&0\\1&3&-1&100\\1&2&-5&200\\0&1&5&0\end{bmatrix}$	$\begin{bmatrix}0&0&0&99\\-1&-3&0&-2\\0&1&0&-5\\1&3&0&200\\5&16&0&500\\1&2&0&700\\-3&-11&0&400\\1&3&0&200\end{bmatrix}$	$\begin{bmatrix}-1&0&0&-17\\0&0&0&198\\-1&0&0&1468\\1&0&0&2096\\-3&0&0&345\\2&0&0&3400\\7&0&0&11900\\5&0&0&8500\end{bmatrix}$
循环边界	L1=1 U1=100 L2=i1 U2=100 L3=i1+i2 U3=200		LL3=max{1, j1+3j2-100 , $\lceil$(j1+2j2-200)/5$\rceil$ } UU3=min{100, $\lfloor$-j2/5$\rfloor$, $\lfloor$(j1+3j2)/2$\rfloor$}	LL2=max{$\lceil$(2-j1)/3$\rceil$,$\lceil$(-3j1-400)/11$\rceil$} UU2=min{-5, $\lfloor$(200-j1)/3$\rfloor$, $\lfloor$(700-j1)/2$\rfloor$, $\lfloor$(500-5j1)/16$\rfloor$}	LL1=17 UU1=1700

下面大致解释算法 6-1对上例的计算过程：

为了消除 D=$\begin{bmatrix}1&0\\2&3\\-3&2\end{bmatrix}$ 的第 3 行,$\mathbf{e}_1$=[1 0],$\mathbf{e}_2$=[2 3],$\mathbf{e}_3$=[-3 2],E_0={}, E_1={$\mathbf{e}_1$,$\mathbf{e}_2$}, E_2={$\mathbf{e}_3$},

计算初始系数[a_1,a_2,a_3]：使 $a_3\mathbf{e}_3=a_1\mathbf{e}_1+a_2\mathbf{e}_2$。令 $D_{sub}=\begin{bmatrix}1&0\\2&3\end{bmatrix}$，$D_{inc}$=[-3 2]，$D_{sub}^{-1}=\begin{bmatrix}1&0\\-2/3&1/3\end{bmatrix}$，

a_3=|det(D_{sub})|=3，[a_1, a_2]= $a_3D_{inc}D_{sub}^{-1}$=[-13, 2]；消元变换步骤如下：

1 $3e_3=-13e_1+2e_2$
2 $3(5e_1+e_3)=2e_1+2e_2$
3 $2e_2=-2e_1+3(5e_1+e_3)$
4 $2(e_1-(5e_1+e_3)+e_2)=(5e_1+e_3)$
5 $(5e_1+e_3)=2(e_1-(5e_1+e_3)+e_2)$
6 $-2(e_1-(5e_1+e_3)+e_2)+(5e_1+e_3)$

T	6	5	4	3	2	1
$a_3\mathbf{e}_3=a_1\mathbf{e}_1+a_2\mathbf{e}_2$	结束	$e_3= 2e_2$	$2e_3= e_2$	$2e_3= -2e_1+3e_2$	$3e_3= 2e_1+2e_2$	$3e_3= -13e_1+2e_2$
合法变换	循环置换	$0= e_3 -2e_2$	交换 e_2,e_3	$e_3= e_1- e_2+e_3$	交换 e_2,e_3	$e_3= 5e_1+e_3$
U_T	$\begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}$	$\begin{bmatrix}1&0&0\\0&1&0\\0&-2&1\end{bmatrix}$	$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}$	$\begin{bmatrix}1&0&0\\0&1&0\\1&-1&1\end{bmatrix}$	$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}$	$\begin{bmatrix}1&0&0\\0&1&0\\5&0&1\end{bmatrix}$

6.6 相关工作

迭代空间标记划分集合(Partitioning and Labelling(PL))方法**[24]**也利用么模变换矩阵来确保循环迭代执行的词典序，并且也提供了代码生成算法。但是与它相比。使用本文提出的 UTF 方法能够发掘更多的并行性，产生较少的循环嵌套；但是也需要对迭代空间边界作更多的计算。

为了说明问题，我们就以表 6-5的程序为例比较两种方法产生的代码：

表 6-5、比较PL方法与UTF方法实例

```
C The original program
      DO i1=1,10
        DO i2=1,10
          DO i3=1,10
            a(i1+1,i2+3,i3+2)=a(i1,i2+1,i3+5)+a(i1+1,i2,i3)
          ENDDO
        ENDDO
      ENDDO
```

```
C Code generated with Partitioning and Labelling method
      io1=1
      DOALL io2=1,3
        DOALL io3=1,10
          DO i1=io1,10
            y1=(i1-io1)/1
            i_m1=1+mod(mod((io2+2*y1)-1,3)+3,3)
            DO i2=i_m1,10,3
              y2=(i2-io2-2*y1)/3
              i_m2=1+mod(mod((io3-3*y1+2*y2)-1,10)+10,10)
              DO i3=i_m2,10,10
                a(i1+1,i2+3,i3+2)=a(i1,i2+1,i3+5)+a(i1+1,i2,i3)
              ENDDO
            ENDDO
          ENDDO
        ENDDO
      ENDDO
```

```
C Code Generated with the method in the paper
     DOALL j1=-4,158
       DO j2=max(1,ceil((j1-28)/13.)),min(10,floor((17+j1)/13.))
         DO j3=max(ceil((1-j1+j2)/3.),ceil((1-j1+5*j2)/2.)),
    1          min(floor((10-j1+j2)/3.),floor((10-j1+5*j2)/2.))
            i1=j2
            i2=j1-j2+3*j3
            i3=j1-5*j2+2*j3
            a(i1+1,i2+3,i3+2)=a(i1,i2+1,i3+5)+a(i1+1,i2,i3)
         ENDDO
       ENDDO
     ENDDO
```

从表 6-5中可以看到：

- 为了并行化原来的 3 层嵌套循环，PL 方法产生了 5 层嵌套循环(包括两个额外的外层嵌套循环)，而 UTF 方法仍旧产生 3 层嵌套循环，同时使最外层循环并行化；
- PL 方法共产生了 3x10= 30 个并行任务(迭代空间划分集 partition), 而每个任务所串行执行的循环体迭代数最多为 34；而 UTF 方法共产生了 158-(-4)+1=163 个并行任务，每个任务所串行执行的循环体迭代数最多为 8，因此，如果循环体粒度较大而忽略两个并行循环的调度时间的差异，那么使用 UTF 方法所取得的最大加速比是使用 PL 方法的加速比的 34/8=4.25 倍；
- UTF 方法需要更多的 max/min, ceil/floor 函数来动态计算迭代空间的边界; 而 PL 方法则对每个内层循环迭代计算其代表迭代点。

6.7 扩充幺模变换适用范围

6.7.1 非常数相关距离

讨论到目前为止，自动应用上述幺模变换方法计算的前提是相关距离是常数，这样才能自动计算合法的幺模变换矩阵。事实上，上述循环并行化变换是一般的方法，并不一定要求常数相关距离的条件。根据**定理 6-1**的推论 2，只要所有合法距离向量属于由若干常数相关距离向量张成的子空间，就可以通过对常数距离向量的并行化幺模变换，将外层循环并行化。

因此，如果非常数相关距离向量的每个常数相关距离实例都在一组常数相关距离向量为基底张成的子空间中，就可以根据这组常数距离向量基底，计算外层并行化幺模变换矩阵。形式地说明上述思想，我们有下列定理：

定理 6-4：如在迭代空间 I 中给定非常数相关距离向量 $\mathbf{F(i)}=\begin{bmatrix} f_1(\mathbf{i}) \\ . \\ . \\ . \\ f_n(\mathbf{i}) \end{bmatrix}$，整函数向量 **F(i)**的每个整函数分量 $f_k(\mathbf{i})$都能表示为最小整数函数[9]同类项(不一定为线性函数)$g_1(\mathbf{i}),\ldots,g_m(\mathbf{i})$的整系数线性组合，其中 d_{ij} 为整系数，且每列系数构成一个合法相关距离基向量。

$$\begin{cases} f_1(i) = d_{11}g_1(i) + \ldots + d_{1m}g_m(i) \\ \quad\ldots \\ f_n(i) = d_{n1}g_1(i) + \ldots + d_{nm}g_m(i) \end{cases} \quad (*)$$

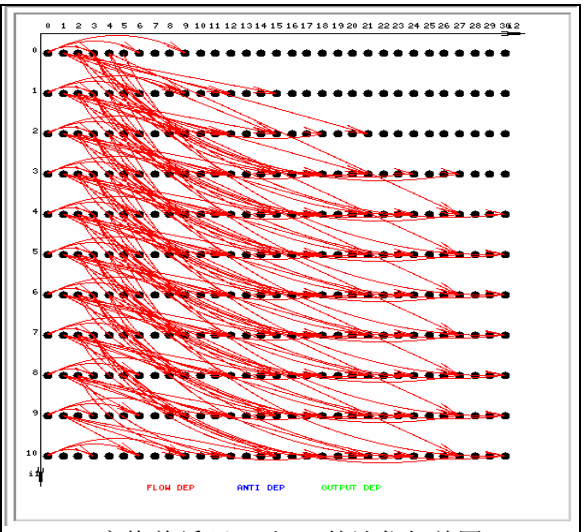
证明：因为所有整函数向量 **F(i)**的常数相关距离向量实例一定属于由一组整数基向量张成的子空间，所以只要找到最小个数的基向量 $\mathbf{d}_1,\ldots,\mathbf{d}_m$，就可以保证(*)成立，但 $\mathbf{d}_1,\ldots,\mathbf{d}_m$ 不一定全为合法距离向量。不失一般性，设 $\mathbf{d}_j$ 不合法，存在 i≥1，使 $d_{1j}=\ldots=d_{i-1j}=0$，且 $d_{ij}<0$。令 $g_j(\mathbf{i})=-g_j(\mathbf{i})$，$d_{ij}=-d_{ij}$，则(*)仍成立且 $\mathbf{d}_j$ 已合法。证毕。

如果上述整系数 d_{ij} 构成 m 个合法的常数距离向量 $\mathbf{d}_1,\ldots,\mathbf{d}_m$（由定理 6-4总能得到)，则在距离矩阵 **D** 中只要用 $\mathbf{d}_1,\ldots,\mathbf{d}_m$ 替代原来的距离向量 **F(i)**，就可以最终得到常数距离矩阵 $\mathbf{D}_1$。然后对 $\mathbf{D}_1$ 用算法 6-1求得外层并行化幺模变换矩阵 **U**，根据**定理 6-1**推论 2，只要 **F(i)**是合法的距离向量，**UF(i)**也是合法的且外层同样是可并行化的。

[9] 实际可以证明是有理数函数

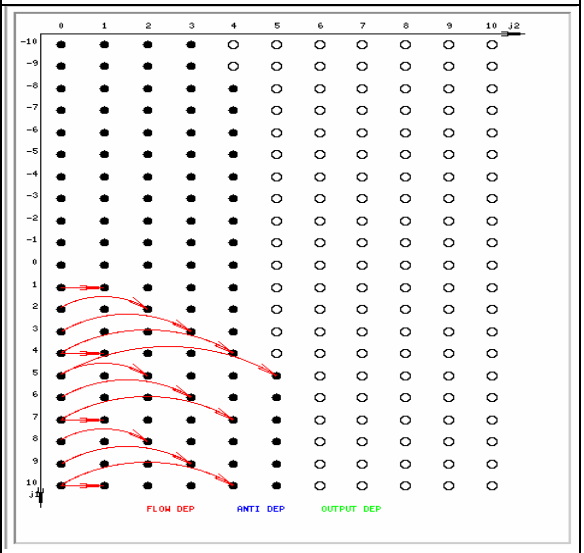
表 6-6、非常数相关距离的么模并行化变换实例

```
DO i1=0,n1
  DO i2=0,n2
    DO i3=0,n3
      a(i1+4*i2+3*i3,6*i3,i1-2*i2+3*i3)=...
      ...=a(i1+i2,i2+i3,i3+i1)
    ENDDO
  ENDDO
ENDDO
```



变换前循环 i1 和 i2 的迭代相关图

```
DOALL j1=-2*n2,13*n1+3*n3
  DO j2=max(0,ceil((-3*n3+j1)/13)),
1          min(n1,floor((2*n2+j1)/13))
    DO j3=max(ceil((-j1+j2)/3),
1                ceil((-j1+5*j2)/2)),
1           min(floor((n2-j1+j2)/3),
1           floor((n3-j1+5*j2)/2))
      i1=j2
      i2=j1-j2+3*j3
      i3=j1-5*j2+2*j3
      a(i1+4*i2+3*i3,6*i3,i1-2*i2+3*i3)=...
      ...=a(i1+i2,i2+i3,i3+i1)
    ENDDO
  ENDDO
ENDDO
```



变换后循环 j1 和 j2 的迭代相关图

表 6-6的例子说明了非常数相关距离的循环也可以通过么模变换使外层并行化。取 $g_1(\mathbf{i})=j$，$g_2(\mathbf{i})=k$，则得到两个常数距离向量(表 6-7)，正好构成了前面所示的常数距离向量例子。因此，只要用同样的外层并行化么模变换，就可以使这个循环外层并行化。么模变换前后的循环迭代空间数据相关图由 PROFPAT 可视化工具相程序员显示了目标循环的可并行化性质(表 6-6)。

表 6-7、分析非常数相关距离向量

建立相关联立方程	$i^1+4*j^1+3*k^1 = i^2(i^1,j^1,k^1)+j^2(i^1,j^1,k^1)$ $6*k^1 = j^2(i^1,j^1,k^1)+k^2(i^1,j^1,k^1)$ $i^1-2*j^1+3*k^1 = k^2(i^1,j^1,k^1)+i^2(i^1,j^1,k^1)$
把 $\mathbf{i}^2$ 表示为 $\mathbf{i}^1$ 的显函数	$i^1+j^1 = i^2(i^1,j^1,k^1)$ $3*j^1+3*k^1 = j^2(i^1,j^1,k^1)$ $-3*j^1+3*k^1 = k^2(i^1,j^1,k^1)$
计算可变相关距离向量	$1*j^1 = i^2(i^1,j^1,k^1)-i^1$ $2*j^1+3*k^1 = j^2(i^1,j^1,k^1)-j^1$ $-3*j^1+2*k^1 = k^2(i^1,j^1,k^1)-k^1$

结果	$F(i^1,j^1,k^1)=\begin{bmatrix} j^1 \\ 2j^1+3k^1 \\ -3j^1+2k^1 \end{bmatrix}$ $D_1=\begin{bmatrix} 1 & 0 \\ 2 & 3 \\ -3 & 2 \end{bmatrix}$

所以，对相关距离向量为已知常数的一般 n 维循环，本节提出了外层和内层并行化的幺模变换方法，不仅证明了满足外层并行化要求的合法幺模矩阵是存在的，而且通过构造性证明给出了计算使最多层数外层循环并行化的合法幺模矩阵的算法。这种并行化方法可以扩充于非完全嵌套循环和非常数相关距离循环。本节提出的所有算法都可在自动并行化编译时实现。

6.7.2 结合归约识别技术增强幺模变换

除了缺乏自动计算幺模矩阵的方法外，幺模变换在过去的并行编译系统中所起作用不大的另一个主要原因是没能同归约识别技术相结合。幺模变换要求各相关都有确定的相关距离，但当循环中出现归约语句时，由于大多数归约引起的相关是杂乱无章的，根本不存在相关距离，致使现有的幺模变换无法应用。即使归约相关的相关距离确定的情况下，如果将归约所带来的相关等同于一般的相关，将使幺模变换的有效性大为下降，以致产生出低效的并行代码。在此基础上我们提出有效地结合归约识别和幺模变换的方法，可以开发出更大的并行度。

所有的并行变换要保证每一相关在变换下的不变性，也就是说，假设存在数据相关 $S_1 \angle S_2$，那么变换后 S_1 必在 S_2 之前执行，但对归约相关不必保持这种不变性，因为按任意顺序执行归约语句都不会改变最终结果。所以，我们在进行幺模变换时可以忽略并行化外层循环的归约相关，但必须同时保证归约的不变性，即归约语句在幺模变换后仍然是归约语句，并称这种幺模变换为增强幺模变换。增强幺模变换时要保证变换后归约语句不被同时执行，只要对变换后的程序进行归约并行化即可。下面我们通过一个定理证明增强幺模变换保证归约语句的不变性，从而保证变换的合法性。

定理 6-5：对任意一层循环，该循环中具有归约形式的单句经增强幺模变换仍为具有归约形式的单句，反之亦然。

证明：对循环 L 中的任意具有归约形式的单句 S：v=v ⊕ e。

1) 若 v 为标量，经增强幺模变换之后，单句 S 的形式仍为 v=v⊕ e’。

2) 若 v 形式为 A(**f**(**i**))数组元素。经增强幺模变换后，$\mathbf{i}=\mathbf{j}\mathbf{U}^{-1}$。变换之后的归约单句为 **b**(**g**(**j**))=b(**g**(**j**))⊕ e’，其中 **g**(**j**)=**f**(**i**)，显然它仍具有归约形式。

所以原循环中的具有归约形式的单句经增强幺模变换仍具有归约形式，同时由于增强幺模变换是一个可逆变换，故命题成立。

虽然原来对某层循环有效或无效的归约语句，增强幺模变换后不一定对转换后的该层循环有效或无效，幺模变换对归约相关距离向量的改变可以由下列定理计算。

引理 6-2（数组归约相关计算）：在 n 重循环中，相关语句 $A(S_1(\mathbf{i}),...,S_k(\mathbf{i}))= A(S_1(\mathbf{i}),...,S_k(\mathbf{i}))\oplus e(\mathbf{i})$产生的归约相关距离 **d** 为满足齐次方程 **Fd=0** 的最小非零正向量。其中 $\mathbf{i}=(i_1,..., i_n)$为循环的迭代下标变量，$S_1(\mathbf{i}),...,S_k(\mathbf{i})$为 k 维数组 A 的线性下标表达式，e(**i**)为不含数组 A 引用的任意归约表达式。**F** 为一个 k×n 阶矩阵，其第 l 行向量由 $S_l(\mathbf{i})=\sum_{m=1}^{n} f_{lm} i_m$ 定义。

证明：$S_1(\mathbf{i}),...,S_k(\mathbf{i})$为 k 维数组 A 的线性下标表达式，若相关语句 $A(S_1(\mathbf{i}),...,S_k(\mathbf{i}))= A(S_1(\mathbf{i}),...,S_k(\mathbf{i}))\oplus e(\mathbf{i})$引起跨循环流相关，则相关联立方程为：$\mathbf{F}\mathbf{i}_1=\mathbf{F}\mathbf{i}_2$。因为 $\mathbf{d}=\mathbf{i}_2-\mathbf{i}_1$，所以 $\mathbf{F}\mathbf{d}=\mathbf{F}(\mathbf{i}_2-\mathbf{i}_1)=\mathbf{0}$，又因为是跨循环流相关，所以按词典序 **d>0**，且 **d** 最小。证毕。

例如，在 $\mathbf{i}=(i_1,i_2)$为下标变量的二重循环中，相关语句 $A(i_1+i_2)= A(i_1+i_2)\oplus e(i_1,i_2)$产生跨循环流相关，则相

关向量 **d**=(d_1,d_2)可以这样计算：

$$\begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}d_1\\ d_2\end{bmatrix}=\begin{bmatrix}0 & 0\end{bmatrix}，\ d_1>0 \text{ 或 } d_1=0 \text{ 且 } d_2>0$$

显然，取 **d**=$\begin{bmatrix}d_1\\ d_2\end{bmatrix}=\begin{bmatrix}1\\ -1\end{bmatrix}$为满足上式的最小合法距离向量。

定理 6-6（幺模变换后的归约相关计算）：n 重循环中的归约语句 S: A(S_1(**i**),...,S_k(**i**))= A(S_1(**i**),...,S_k(**i**))⊕e(**i**)
如果产生跨循环流相关 **d**, 在幺模变换 **U** 后归约语句 S'产生跨循环流相关 **d'=Ud**。

证明：因为幺模变换 **U** 后，**j=Ui, i=U^{-1}j**。所以归约语句 S'为:

$$A(S_1(\mathbf{U^{-1}j}),...,S_k(\mathbf{U^{-1}j}))= A(S_1(\mathbf{U^{-1}j}),...,S_k(\mathbf{U^{-1}j}))\oplus e(\mathbf{U^{-1}j})$$

原相关联立方程 **Fi**$_1$=**Fi**$_2$ 变换为为相关联立方程：**F U^{-1}j** $_1$=**F U^{-1}j** $_2$，所以：

d'= j$_2$**- j**$_1$= **U^{-1}i**$_2$**- U^{-1}i**$_1$=**U^{-1}(i**$_2$**- i**$_1$)= **U^{-1}d**。证毕。

下面给出对算法 6-2的改进算法，以便适当选择内层么模变换矩阵使用增强么模变换，也可以对内层循环消除归约相关。

算法 6-4: 寻找消除内层归约相关的内层并行化么模矩阵 U：
输入：在 n 维迭代空间中，给定 n×(m+r)的距离向量矩阵 **D** 的秩 s=rank(**D**)=n，其中 m≥n,r≥0,分别是非归约相关距离向量和归约相关距离向量数。
输出：找到这样一个合法的行变换幺模矩阵 **U**，使得循环内层可并行且完全消除归约。
步骤：

令 k=1, $u_{n-1}=\max\{0, \lceil \left|\frac{-d_{n,j}}{d_{n-1,j}}\right| \rceil \mid 1\le j\le m+r, d_{1\,j}=...=d_{n-2\,j}=0, d_{n-1\,j}>0\}$

,...., k=n-1, $u_1=\max\{\lceil -\frac{-d_{n,j}+\sum_{k=1}^{n-2}u_{n-k}d_{n-k,j}}{d_{1,j}} \mid 1\le j\le m+r, d_{1\,j}>0\}$。

则这些非负整数 $u_1,...,u_{n-1}$，满足 $d_{nj}+\sum_{i=1}^{n-1}u_i d_{ij}>0$,j=1..m+r，所以幺模矩阵

U$_T$=$\begin{bmatrix}u_1 & ... & u_{n-1} & 1\\ & & 1 & \\ & ... & & \\ 1 & & & \end{bmatrix}$ 即为所求。算法结束。

6.7.2.1 总结对归约相关增强么模并行化的算法：

设任意 n 重嵌套循环相关距离向量矩阵 **D**=[**D**$_1$|**D**$_2$]，其中 **D**$_1$ 和 **D**$_2$ 分别是非归约相关距离向量矩阵和归约相关距离向量矩阵。
忽略外层循环的归约相关，我们首先由算法 6-1对非归约相关距离向量矩阵 **D$_1$** 作幺模 **U**$_1$ 变换得到 **U**$_1$**D**$_1$=[**0**^{T}|**D$_1$'**^{T}]^T, **D**$_1$'为 s=rank(**D$_1$**)阶行满秩阵，现在根据**算法 6-4**对包含归约相关距离向量的矩阵 **D'=U$_1$D** 计算得到的 **U**$_s$ 作

U$_2$=$\begin{bmatrix}I_{n-sxn-s} & 0_{n-sxs}\\ 0_{sxn-s} & U_{sxs}\end{bmatrix}$，则 **U$_2U_1$D** 完成进一步对内层循环并行化的任务。若么模变换之后外层可并行化，则检验所有的归约语句，是否对外层有效(存在跨循环流相关），有效的归约变量需做归约变形，若都无效，则外层可直接并行化。

6.7.2.2 对归约相关增强么模并行化的实例

表 6-8、增强么模变换实例

计算说明	程序
根据**引理 6-2**，计算包括 b,c 归约相关在内，所有跨循环相关： $D=\begin{bmatrix}1&0&1&1\\2&3&-1&0\\-3&2&-5&0\end{bmatrix}$ $[\ a_1\ \ a_2\ b\ c]$	DO i1=1,10 DO i2=1,10 DO i3=1,10 a(i1+1,i2+3,i3+2)=a(i1,i2+1,i3+5)+a(i1+1,i2,i3) b(i1+i2,5*i1+i3)= b(i1+i2,5*i1+i3)+i1 c(i2,i3)=c(i2,i3)+(i2+i3) ENDDO ENDDO ENDDO
忽略 b,c 引起的归约相关： $D_1=\begin{bmatrix}1&0\\2&3\\-3&2\end{bmatrix}$ 进行外层并行化么模变换 $U_1=\begin{bmatrix}13&-2&3\\1&0&0\\-4&1&-1\end{bmatrix}$， $D'_1=\begin{bmatrix}0&0\\1&0\\1&1\end{bmatrix}$	DOALL j1=-4,158 DO j2=max(1,ceil((j1-28)/13.)),min(10,floor((17+j1)/13.)) DO j3=max(ceil((1-j1+j2)/3.),ceil((1-j1+5*j2)/2.)), 1 min(floor((10-j1+j2)/3.),floor((10-j1+5*j2)/2.)) i1=j2 i2=j1-j2+3*j3 i3=j1-5*j2+2*j3 a(i1+1,i2+3,i3+2)=a(i1,i2+1,i3+5)+a(i1+1,i2,i3) Critical_Section_B b(i1+i2,5*i1+i3)= b(i1+i2,5*i1+i3)+I1 End_ Section Critical_Section_C c(i2,i3)=c(i2,i3)+(i2+i3) End_Section ENDDO ENDDO ENDDOALL
忽略归约相关， 进行内层并行化么模变换 $U_2=\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}U_1$ $=\begin{bmatrix}13&-2&3\\-4&1&-1\\1&0&0\end{bmatrix}$， $D'=\begin{bmatrix}0&0&0&13\\1&1&-4&1\\0&1&1&1\end{bmatrix}$。 $[\ a_1\ \ a_2\ b\ c]$。 最内层循环对 b 的归约相关仍旧存在。	DOALL j1=-4,158 DO j2= max(ceil((2-j1)/3.),ceil((6-j1)/2.),ceil((-5-4*j1)/13.)), 1 min(floor((20-j1)/3.),floor((60-j1)/2.), floor((49-4*j1)/13.)) DOALL j3=max(1,j1+j2-10,ceil((j1+2*j2-10)/2.)), 1 min(10,j1+j2-1,floor((j1+2*j2-1)/5.)) i1=j3 i2=j1+3*j2-j3 i3=j1+2*j2-5*j3 a(i1+1,i2+3,i3+2)=a(i1,i2+1,i3+5)+a(i1+1,i2,i3) Critical_Section_B b(i1+i2,5*i1+i3)= b(i1+i2,5*i1+i3)+i1 End_section Critical_Section_C c(i2,i3)=c(i2,i3)+(i2+i3) End _Section ENDDOALL ENDDO ENDDOALL

消除内层 b 的归约相关的么模变换(增加一次斜错变换）： $U_3=\begin{bmatrix}1&0&0\\0&1&1\\0&0&1\end{bmatrix}U_2$ $=\begin{bmatrix}13&-2&3\\-3&1&-1\\1&0&0\end{bmatrix}$， $D'=\begin{bmatrix}0&0&0&13\\2&1&1&-3\\1&0&1&1\end{bmatrix}$。 [a_1 a_2 b c] 所以还要对 c 归约变形	DOALL j1=-4,158 DO j2=max(ceil((5-j1)/3.),ceil((-3*j1-3)/13.),ceil((8-j1)/2.)), 1 min(floor((50-j1)/3.),floor((80-j1)/2.), floor((66-3*j1)/13.)) DOALL j3=max(1,ceil((j1+3*j2-10)/4.),ceil((j1+2*j2-10)/7.)), 1 min(10,floor((j1+3*j2-1)/4.), floor((j1+2*j2-1)/7.)) i1=j3 i2=j1+3*j2-4*j3 i3=j1+2*j2-7*j3 a(i1+1,i2+3,i3+2)=a(i1,i2+1,i3+5)+a(i1+1,i2,i3) b(i1+i2,5*i1+i3)= b(i1+i2,5*i1+i3)+i1 Critical_Section_C c(i2,i3)=c(i2,i3)+(i2+i3) End_Section ENDDOALL ENDDO ENDDOALL

以上为一个消除归约的么模变换的实际例子。表 6-8a 为一个三层循环，循环中第二句为外层循环的归约语句，我们在外层么模并行化变换之前忽略了它的影响，采用么模矩阵$\begin{bmatrix}13&-2&3\\1&0&0\\-4&1&-1\end{bmatrix}$，如表 6-8b 所示。原来算法 6-2的内层并行化么模矩阵$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}$不能消除由第二句引起的归约，如表 6-8c 所示，在采用了消除内层归约**算法 6-4**之后，么模矩阵将修改为$\begin{bmatrix}1&0&0\\0&1&1\\0&1&0\end{bmatrix}$，我们可以从表 6-8d 中看到，内层归约相关现象被完全消除。我们注意到，在牺牲部分并行度的情况下，利用归约对某一层循环是否有效这一概念，实现了归约消除。

6.7.3 非完全嵌套循环

对上述么模变换另一个有意思的扩展工作是考虑如何对非完全嵌套循环进行么模变换。最简单的方法是通过在最内层循环体中增加 IF 捍卫条件把外层循环语句嵌入内层循环体中，但是这种方法的缺点是显然的，它增加了额外的计算，引入了不必要的跨循环数据相关。

另外值得注意的是，上述所谓 NBBL(Non-basic-to-basic conversion)变形并不是对所有非完全嵌套循环都适用的，只有满足一定合法条件的 NBBL 变形才是正确的，例如表 6-10所示，原循环中的语句实例 $S_1[4]$ 在变形后的循环中得不到执行。

表 6-9、对非完全嵌套循环变形为完全嵌套循环

DO
  S1
  DO
  S2
  ENDDO
  S3
ENDDO
	DO
	  DO
	  IF (first iteration)
	  S=S1;S2
	  ELSE IF (last iteration)
	  S=S2;S3
	  ELSE
	  S=S2
	  ENDIF
	  ENDDO
	ENDDO

(a)非完全嵌套循环	(b)完全嵌套循环

表 6-10、不合法的NBBL

DO I=1,4
S1: H_1(I)
  DO j=max(-i+3,i-1),2
S2: H_2(I,J)
  ENDDO
ENDDO
	DO I=1,4
	  DO j=max(-i+3,i-1),2
	  IF J.EQ.MAX(-I+3,I-1) H_1(I)
	  H_2(I,J)
	  ENDDO
	ENDDO

(a) 非完全嵌套循环	(b)完全嵌套循环

Jinling Xue 介绍了一种通过 NBBL 有条件地将非完全嵌套循环转化为完全嵌套循环的方法**[65]**。非形式地说，NBBL 条件就是在将外层嵌套循环 L_k 中的语句 S_1 向内层嵌套循环 L_{k+1} 下沉时必须保证由 $L_{k+1},...,L_n$ 边界条件交上 S_1 的 IF 捍卫条件所定义的循环迭代空间非空。

由 NBBL 条件可以看出，并非所有外层循环语句可以嵌入最内层循环，如表 6-10就是一例。Jinling Xue 方法的主要贡献就是扩展了非完全嵌套循环迭代空间的定义，提出了对原循环迭代空间增加额外的一维来表示循环体中的各个语句。这样就可以用多个完全嵌套循环的迭代空间来表示非完全嵌套循环迭代空间。这时，对非完全嵌套循环应用 UTF 变换 U，等价于对各个对应于每个语句的完全嵌套循环的迭代空间应用扩张的么模变换(把循环体中的串行语句看作额外一层不变形的循环，则么模变换 $\begin{bmatrix} U & 0 \\ 0 & 1 \end{bmatrix}$ 可以对扩张的循环迭代空间变换)。这样在代码生成时将可以产生不带有 IF 捍卫条件的非完全嵌套循环体。

表 6-11、对非完全嵌套循环么模变换

	DO I=1,4
	  DO J=1,3
	  IF (J.EQ.1) H1(I)
	  H2(I,J)
	  ENDDO
	ENDDO
	DO I'=2,7
	DO J'=max(1,I'-4), min(I'-1,3)
	  I=I'-J'
	  J=J'
	  IF (J.EQ.1) H1(I)
	  H2(I,J)
	  ENDDO
	ENDDO
		DO I'=2,7
		  DO J'=max(1,I'-4),1
		  I=I'-J'
		  J=J'
		  H1(I)
		  ENDDO
		  DO J'=max(1,I'-4),min(I'-1,3)
		  I=I'-J'
		  J=J'
		  H2(I,J)
		  ENDDO
		ENDDO

DO I=1,4
S1: H1(I)
  DO J=1,3
S2: H2(I,J)
  ENDDO
ENDDO		

(a)原循环	(b)NBBL 变形及直接么模变换 $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	(c) 分别对各语句么模变换

例如，表 6-11首先对原循环进行 NBBL 变形，然后如果直接对循环体执行么模变换，则循环体内仍存在 IF 语句，而若分别对不同循环语句进行么模变换，就可以消除 IF 捍卫条件。

以上扩充的么模变换方法应用前提是数据相关和么模矩阵都必须已知，然而也没有给出计算么模变换矩阵的方法。因此，如果结合本文中的并行化么模变换矩阵计算方法，非完全嵌套循环也可以自动加以并行化变换。

7 其他一些编译新技术

7.1 动态数据流分析

在静态编译时，由于缺乏足够的信息，程序中一些潜在的并行性无法得到识别。若将分析推迟到动态执行时，我们便能很精确地求得程序的并行性。在以往的动态测试中，多数将焦点集中在对循环中数组动态相关性的分析。由于数组下标过于复杂或依赖于输入参数，无法在静态时对其进行精确表示，那么根据在动态时精确得到的下标值就可以确定两数组是否相关。

为了说明动态数据流分析方法，我们用表 7-1a 所示的程序例子来说明。在进行并行化变换之前，通过插入状态变量分析数据流信息，根据运行时的输入数据对循环动态得出能否并行化的判断，选择执行并行化版本还是串行版本。

表 7-1、 动态数据流分析示意

DO I=1,100
  IF(*f*(A[I])) THEN
s1: Y=...
  ENDIF
  ...
s2: X=Y
  ...
  IF(*g*(A[I])) THEN
s3: X=B
  ENDIF
  ...
s4: ...=X
  ...
ENDDO
	PARALLELABLE=1
	DOALL I=1,100
	  LOCAL X',Y'
	  Y'=0, X'=0
	  IF(*f*(A[I])) THEN
	  IF(Y'.EQ.0) THEN
	  Y'=1
	  ENDIF
	  ENDIF
	  IF(Y'.EQ.1) THEN
	  X'=1
	  ELSE
	  X'=0
	  ENDIF
		IF(*g*(A[I])) THEN
		  IF(X'.EQ.0) THEN
		  X'=1
		  ENDIF
		  ENDIF
		  IF(X'.EQ.0) THEN
		  PARALLELABLE=0
		  BREAK
		  ENDIF
		ENDDOALL
		IF(PARALLELABLE) THEN
		  PARALLEL SECTION
		ELSE
		  SERIAL SECTION
		ENDIF

首先观察表 7-1a 变量 Y，在动态执行某次迭代时若 f(A[I])成立，则执行 s1，循环迭代对 Y 的第一次访问为写；若某迭代中 f(A[I])不成立，则说明该迭代 S2 对 Y 的读为暴露的，循环迭代对 Y 的第一次访问将为读。传统的动态测试认为它有可能构成跨循环流相关，从而妨碍了循环的可并行性。实际上，当某次迭代中 f(A[I])不成立，尽管 S2 中的 Y 暴露而致使它在该迭代先读，但此时若 g(A[i])为真，由于 S3 又对 X 赋了值，所以 S2 中对 X 的赋值是无效的，该迭代对 Y 的先读就不妨碍循环的可并行性。所以该循环能否并行化可以通过动态执行数据流分析加以判断。

为了验证这些暴露点是否引起跨循环的流相关，我们进行了如下的动态数据流分析。设静态时分析不出私有性的那些标量组成原始测试集 PSS。动态数据流分析对每个 PSS 元素引入了两个状态：浑浊态与纯净态。浑浊态表示该变量具有在该迭代以前定义的值，或它引用了具有浑浊态的变量。纯净态表示该变量的值是在本迭代定义的。同时，我们将赋值语句实例也分为两类：PSS 实例与非 PSS 实例。PSS 实例说明该实例的写为一 PSS 元素，非 PSS 实例说明该实例的写不是 PSS 元素。

动态数据流分析描述如下：

1. 在迭代的开始，将 PSS 中的每一元素的状态初始化为浑浊态。
2. 对迭代中的每个对 PSS 进行读写的赋值语句实例，我们有：

   a 如果非 PSS 实例读浑浊的 PSS 元素，则该循环不可并行化。

   b 如果 PSS 实例没有读任何浑浊 PSS 元素，则将该 PSS 实例的写置为纯净态。

   c 如果 PSS 实例读浑浊的 PSS 元素，则将该 PSS 实例的写置为浑浊态。

假设在表 7-1a 中，集合{x, y}为动态数据流分析中的 PSS，表 7-1b 给出动态数据流分析的代码。变量 PARALLELIZABLE 为该循环是否可能并行化的标记，它为一个共享变量，虽然它具有输出相关，但由于我们不关心输出的顺序，且输出值同一，所有它不影响分析代码的并行性。常数 0 表示浑浊态，而 1 表示纯净态。

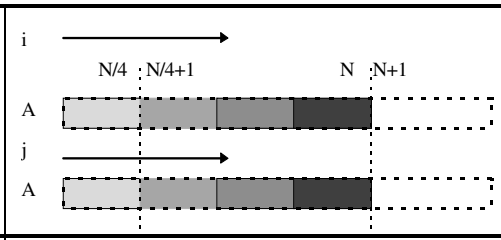
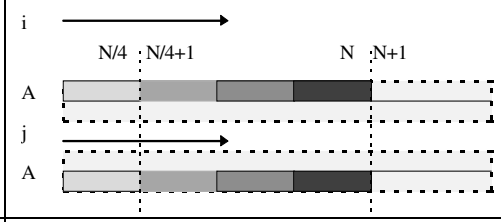
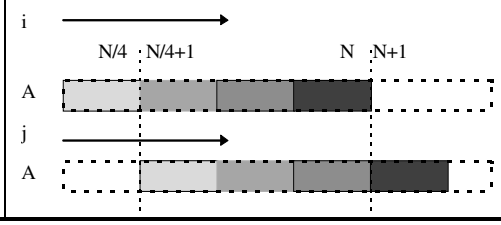
7.2 循环间 cache 优化

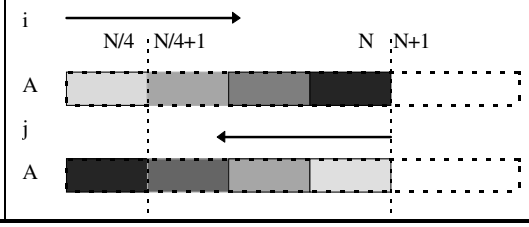
不失一般性，本节主要讨论缺省 Blocking 调度下，并行循环与并行循环之间、并行循环与串行循环之间数据重用的开发和利用，以提高 cache 的命中率。

当程序并行执行时，缺省 Blocking 调度将可并行执行的循环按循环变量连续等分到相当于处理机个数的几个进程中，其中一个为主进程，其余的为副进程，每个进程交给一个处理机完成，各进程访问到的数据保存在执行该进程的处理机的局部 cache 中。而对于串行执行的循环，缺省调度将它划分到主进程中，串行循环访问到的数据保存在执行主进程的处理机的局部 cache 中。例如：一个将要运行在由四个处理机组成的并行机上的循环 200 次的 DO 循环，若它是可以并行执行的，缺省调度将 1-50 次循环划分到主进程，由一个处理机完成，将 51-100、101-150、151-200 次循环分别划分到三个副进程，每个副进程由一个处理机完成。若该 DO 循环不能并行执行，则缺省调度将 200 次循环都划分到主进程。

7.2.1 并行循环之间的数据重用

表 7-2、并行循环之间的数据重用例子

例 1	DOALL I=1,N A(I)=..... ENDDO DOALL J=1,N .....=A(J)..... ENDDO	
例 2	DOALL I=1,N ...=A(2*I-1)..... ENDDO DOALL J=1,N A(2*J)=..... ENDDO	
例 3	DOALL I=1,N .....=A(I)..... ENDDO DOALL J=1,N A(J+N/4)=..... ENDDO	

例 4	DOALL I=1,N A(I)=..... ENDDO DOALL J=1,N ....=A(N-J+1)..... ENDDO	

在缺省调度下，两个并行循环之间的数据重用有时可以得到很好的利用，但有时存在于两个并行循环之间的数据重用却得不到很好的利用。

如表 7-2中的四个程序例子：假设多处理机系统由四个处理机组成，则缺省调度下，并行循环在各相应进程中引用的 A 数组元素分别如表 7-2所示。

例 1 和例 2 很好利用了并行循环间的数据重用：在例 1 中，两个并行执行的循环需要引用的 A 数组区间完全重合且按相同方向的次序访问，它们之间对 A 数组区间 A(1:N)的重用得到了完全的利用。在例 2 中，如果 cache 的行长不小于 2，则两个并行循环之间存在对 A 数组区间 A(1:2*N)中下标为偶数的数组元素的重用，这部分重用在缺省调度下也能得到完全的利用。

例 3 和例 4 则没能利用并行循环间的数据重用：在例 3 中，第一个并行循环需访问的 A 数组区间 A(1:N)的后 3/4 部分与第二个并行循环需访问的 A 数组区间 A((N/4)+1:N+N/4)的前 3/4 部分重叠，两个并行循环对 A 数组区间 A((N/4)+1:N)的重用在缺省调度下完全得不到利用。但是，如果让第一个循环在循环变量从 1 变化到 3*N/4 时访问它被第二个循环重用的 A 数组区间 A((N/4)+1:N)，在循环变量从(3*N/4)+1 变化到 N 时访问不被第二个循环重用的 A 数组区间 A(1:N/4)，或者让第二个循环在循环变量从 1 变化到 N/4 时访问 A 数组区间 A((N+1:N+N/4)，在循环变量从(N/4)+1 变化到 N 时访问 A 数组区间 A((N/4)+1:N)，则两个并行循环对 A 数组区间 A((N/4)+1:N)的重用就能得到完全的利用。

在例 4 中，两个并行循环按相反方向的次序访问 A 数组区间 A(1:N)，它们对该数组区间的重用在缺省调度下也完全得不到利用。如果让其中一个循环改变访问 A 数组区间的方向，使它们按相同方向的次序访问 A 数组区间，则它们对 A 数组区间 A(1:N)的重用就能得到完全的利用。

表 7-3、待分析变换的并行循环

```
DOALL   I=1,N,1
      .....A(a1*I+b1).....
ENDDO
...
DOALL   J=1,M,1
      .....A(a2*J+b2).....
ENDDO
```

一般地，对循环的下界和步长均规范化为 1 的两个并行循环，如果存在循环间数据重用，它们重用的数据是一维数组，且其元素的下标是循环变量的线性函数，我们可以分以下三步通过循环变换使它们对该数组的重用全部得到利用。

1. 如果 $a_1*a_2<0$，两个并行循环按相反方向的次序访问 A 数组，这时，我们对其中一个循环实施循环颠倒(loop reversal)变换，使它们按相同方向的次序访问 A 数组。

   若 $a_1<0$，我们选择对 i 循环实施 loop reversal 变换，即在 i 循环的循环体中，用 N-I'+1 替换循

环变量 I。设替换后 A 数组元素的下标为 $a'_1*I'+b'_1$，则

$b'_1=b_1+a_1*(N+1)$

$a'_1=-a_1$

若 $a_2<0$，对 j 循环实施 loop reversal 变换，即在 j 循环的循环体中，用 M-j'+1 替换循环变量 j。设替换后 A 数组元素的下标为 $a'_2*j'+b'_2$，则

$b'_2=b_2+a_2*(M+1)$

$a'_2=-a_2$

。进入下一步骤。

2. 现在 $a_1*a_2>0$，两个并行循环按相同方向的次序访问 A 数组。设 I 循环在循环变量 $I=K_1$ 时开始访问的 A 数组区间被 J 循环重用，设 J 循环在循环变量 $J= K_2$ 时开始访问的 A 数组区间重用 I 循环引用的 A 数组区间。由 $a_1+ b_1= a_2K_2+ b_2$ 及 $a_2+ b_2= a_1K_1+ b_1$ 可得：

$K_2= \max(1,\lfloor(a_1+b_1-b_2)/a_2\rfloor)$

$K_1= \max(1,\lfloor(a_2+b_2-b_1)/a_1\rfloor)$

如果 $K_1>N$ 或 $K_2>M$，两个并行循环之间不存在对 A 数组的循环间数据重用。当 $K_1<N$ 且 $K_2<M$ 时，第一个循环在循环变量 I 从 K_1 开始访问的 A 数组区间与第二个循环在循环变量 J 从 K_2 开始访问的 A 数组区间存在数据重用。两个并行循环对 A 数组区间的重用可归纳为三种情况：

(a) $K_1=K_2=1$

(b) $1=K_2<K_1$

(c) $1=K_1<K_2$

定义函数 sdmod(x,y)为：sdmod(x,y)= IF x≤y THEN x ELSE x-y ENDIF

若(a)，两个并行循环都先访问重用的 A 数组区间，不再需要做这步循环变换。

若(b)，J 循环在 J 从 1 变化到 M 时，先访问重用的 A 数组区间。对 I 循环进行循环变换，即在 I 循环的循环体中用 sdmod(I'+K_1-1,N)替换循环变量 I，使 I'从 1 变化到 N 时，也先访问重用的 A 数组区间，后访问不被重用的 A 数组区间。若(c)，I 循环在 I 从 1 变化到 N 时，先访问重用的 A 数组区间。对 J 循环进行循环变换，即在 J 循环的循环体中用 sdmod(J'+K_2-1,M)替换循环变量 J，使 J'从 1 变化到 M 时，也先访问重用的 A 数组区间，后访问不被重用的 A 数组区间。如果$|a_1|*N=|a_2|*M$，两个并行循环访问同样长度的 A 数组区间，经过上述的循环变换后，存在于两个并行循环之间的对 A 数组区间的重用可以得到完全的利用，结束。否则，进行下一步变换。

3. 如果$|a_1|*N\neq|a_2|*M$，两个并行循环访问不同长度的 A 数组区间，经过上述的循环变换后，存在于两个并行循环之间的对 A 数组区间的重用在缺省调度下只有一部分能得到利用。要使它们对 A 数组区间的重用全部得到利用，必须把访问较长 A 数组区间的那个循环分割成两个循环，变换后的结果如表 7-4所示。调度循环并行执行会带来一定的开销，是否将访问较长 A 数组区间的循环进行分割须权衡分割后增加的开销和可获得的 cache 命中率的提高。

表 7-4、循环分割

DOALL I=1,int(a_2*M/ a_1) ... ENDDO DOALL I=int(a_2*M/ a_1)+1,N ... ENDDO DOALL J=1,M ..... ENDDO	DOALL I=1,N ... ENDDO DOALL J=1,int(a_1*N/ a_2) ... ENDDO DOALL J= int(a_1*N/ a_2)+1,M ... ENDDO
$\|a_1\|$*N>$\|a_2\|$*M	$\|a_1\|$*N<$\|a_2\|$*M

7.2.2 并行循环与串行循环之间的数据重用

并行循环与串行循环之间的数据重用在由四个处理机组成的多处理机系统中一般只有 1/4 能得到利用，如表 7-5所示的程序段：

表 7-5、测试程序包spec95中wave5的一段程序

```
      DO 100 L=l3,l4
         QMLT(L)=QMULT
100   IF (X(L).LT.0.) QMLT(L)=0.
      DO 130 L=L3,L4
         ......
         IJ(L)=I(L)+NX2*(J-1)
         W1(L)=FXC*FYC
         W2(L)=FX(L)*FYC
         W3(L)=FXC*FY(L)
         W4(L)=FX(L)*FY(L)
130   CONTINUE
      DO 140 L=L3,L4
         Q(IJ(L))=Q(IJ(L))+W1(L)*QMLT(L)
         Q(IJ(L)+1)=Q(IJ(L)+1)+W2(L)*QMLT(L)
         Q(IJ(L)+NX2)=Q(IJ(L)+NX2)+W3(L)*QMLT(L)
         Q(IJ(L)+NX2+1)=Q(IJ(L)+NX2+1)+W4(L)*QMLT(L)
140   CONTINUE
```

在这段程序中，循环 100 和循环 130 经过相关性测试后，都能并行执行，而循环 140 由于存在跨循环的数据相关，只能串行执行(该循环由于存在稀疏矩阵，因此归约也无效)。我们对循环 140 的执行时间进行测试后发现：循环 140 在程序并行执行时的执行时间要比在程序串行执行时的执行时间长。经过分析，我们认为造成这种现象的原因是：我们所使用的多处理机系统由四个处理机组成，程序串行执行时，整个程序划分为一个进程，循环 100、循环 130、循环 140 都由同一处理机执行，循环 100、循环 130 对数组 QMLT、IJ、W1、W2、W3、W4 的定义都保留在该处理机的局部 cache 中，循环 140 对这几个数组的引用都能 cache 命中；而程序并行执行时，循环 100 和循环 130 在缺省调度下按循环变量 L 连续被划分为长度相等的四段，分别交给四个进程，每个进程由一个处理机来完成，这四段是：

L3+((L4-L3+1)/4)*i～L3+((L4-L3+1)/4)*(i+1)-1　　(i=0,1,2,3)

设 i=0 这一段划分到主进程，由处理机 cpu_0 完成，其余三段划分到副进程，分别由处理机 cpu_1、cpu_2、cpu_3 完成，则循环 100 和循环 130 执行完后，数组 QMLT、IJ、W1、W2、W3、W4 驻留在处理机

cpu_i 局部 cache 中的数组区间为[L3 + ((L4 - L3 + 1) / 4) * i : L3 + ((L4 - L3 + 1) / 4) * (i + 1) -1] (i = 0,1,2,3)。当执行循环 140 时，由于该循环只能串行执行，整个循环被划分到主进程，由处理机 cpu_0 完成，因此，循环 140 对数组 QMLT、IJ、W1、W2、W3、W4 的引用只有下标从 L3 到 L3+(L4-L3+1)/4-1 的元素能 cache 命中，它们驻留在处理机 cpu_0 的局部 cache 中，占需要引用的数组元素的 1/4，而对其余 3/4 的数组元素的引用都 cache 失效，它们分别驻留在其它三个处理机的局部 cache 中。cache 失效增加了循环 140 的执行时间，因而循环 140 在程序并行执行时的执行时间要比在程序串行执行时的执行时间长。

从上面的分析我们知道，要使程序并行执行时与并行循环存在数据重用的串行循环的执行时间缩短，必须提高串行循环引用重用数据的 cache 命中率。而并行循环定义的数组元素分段保存在各个处理机的局部 cache 中，为此我们提出开发和利用并行循环与串行循环之间数据重用的优化方法，即根据循环间数据重用信息以及并行循环在缺省调度下的分段情况将串行循环划分为相当于处理机个数的几段，由于串行循环存在跨循环的数据相关，各段间是不能并行执行的，因此需要在串行循环各段间引入同步变量，使串行循环各段在同步机制的控制下并行执行。将我们的优化方法用于表 7-5的程序段，需把串行循环 140 按循环变量 L 连续划分为四段，在每段之间引入一个同步变量，用以控制一个段只有在它的前一段执行完后才开始执行，本例引入 A(0)、A(1)、A(2)、A(3)、A(4) 共五个同步变量。最后将外层循环标识为可并行执行，如表 7-6所示。

表 7-6、串行循环划分

```
DOACROSS    I=0,3
            DO    L=L3+((L4-L3+1)/4)*I , L3+((L4-L3+1)/4)*(I+1)-1
            .....
            ENDDO
ENDDOACROSS
```

在缺省调度下将按外层循环的循环变量 I 把它划分到四个进程中去，分别交给四个处理机执行，因此，内层循环对数组 QMLT、IJ、W1、W2、W3、W4 的元素的引用都能在执行它的处理机的局部 cache 内找到，提高了 cache 的命中率。我们在 SGI 的 ORIGIN2000（4 个 196MHz 的 R10000 CPU，1024M 内存，2M cache） 上对 SPEC95 中的 wave5 作了测试。在我们的系统作 cache 优化前其并行执行时间为 154 秒，而 cache 优化后并行执行时间为 108 秒，加速比分别为 1.32 和 1.88，因此我们提出的方法是实际有效的。

当并行循环访问到的数组区间与串行循环访问到的数组区间重叠得不好，或并行循环与串行循环按相反次序访问重用的数组区间时，需要把这里优化方法和循环变换方法结合起来考虑，才能更充分地开发和利用并行循环与串行循环之间的数据重用。

8 并行程序设计环境交互功能的扩充

通过并行程序设计交互环境 PROFPAT 在程序并行化方面的应用，我们发现 PROFPAT 基本达到了最初定下的设计目标，但是，要使 PROFPAT 真正成为并行程序设计的实用环境，其功能上仍有许多可以进一步改进和扩充的余地，包括及时引入最先进的自动并行化技术，通过信息的组织、查询、解释、过滤，将自动并行化技术更好地同人机交互结合起来。

8.1 引入更先进的自动化技术

从程序员的角度看，最好程序的并行化任务能够完全自动化，如果能够做到这样理想的自动并行化编译，就不再需要交互程序设计环境了。而事实上，对于复杂的应用程序而言，自动并行化技术还远远不能达到这样理想的境界；反过来说，如果没有已经实现的自动并行化分析和变换技术，就绝对不能在程序设计环境交互环境中开发出实用和有效的各种并行化工具了。所以说自动技术的完善和提高是改进程序设计交互环境的重要基础条件。

跨过程的信息传播，指针和别名分析等等都是可以在改进的 PROFPAT 中加以考虑的有效自动化技术。由于如何表示、归纳程序中数组信息和指针信息都是程序分析，尤其是自动分析的难点，所以在最新的自动并行化编译研究中都正在寻找高效和精确的分析技术，一旦这些技术得以实现，无疑将对基于并行化编译的并行程序设计交互环境产生积极的作用。

8.2 改善人机交互

如果说自动并行化技术是并行程序设计交互环境的基础，那么人机交互就是并行程序设计交互环境的优势。由于在复杂程序并行化任务面前，程序员的作用是不可或缺的，所以，如何有效地把系统自动和交互分析的程序信息展示给程序员，引导程序员通过交互操作选择正确的程序变换更好地发掘隐藏在串行程序中的并行性，是改进和提高并行程序设计交互环境的关键。

8.2.1 程序信息的有效组织、联系和过滤

要把系统自动和交互分析的程序信息有效地展示给程序员，中间表示和外部表示信息的有效组织是中心环节。从逻辑上说，程序的各种信息是相互关联的，它们之间存在整体-局部，一般-特殊，一一对应，或者一对多等结构和数量关系。例如，过程与子过程之间，循环语句与嵌套循环语句之间，循环与循环迭代空间之间，数据相关与语句之间，数据相关与循环之间，数据相关与变量信息之间，变量信息与过程之间，变量信息与循环之间，等等。PROFPAT 通过过程调用图、循环迭代相关图、语句相关图、变量相关表、变量读写流信息表、程序计算量分析表、程序潜在最大并行性表等等结构化信息显式地表示和组织信息，在应用实践中帮助了并行化程序员有效地理解和分析程序。

通过更有效地组织信息，不仅要使信息组织得有助于理解和分析程序，而且要使信息组织得有助于浏览和检索信息。因为不同类型的信息之间互相有联系，所以需要在信息之间控制导航。类似超文本的导航机制帮助程序员知道怎样查找和获取信息。

例如，在 PROFPAT 程序浏览窗口中，光标位置能告诉程序员他正在访问程序的哪个部分。我们通过实现查找过程名和语句行号的功能来帮助程序员移动光标来查看程序的其他部分，也可以通过 DO 循环表

中的遍历来导航查看程序的不同 DO 循环。在图形窗口中，我们可以通过窗口滚动、鼠标单击图形敏感区域来遍历图形的不同部分信息或者其他相关信息。例如，当单击过程调用图结点时，程序窗口能够自动将光标移动至相应过程的第一条代码行处，单击语句相关图中的相关边时，程序窗口能够加亮显示产生相关的对应语句行，在消息反馈窗口中显示相关类型、变量和相关向量等文字信息。

8.2.2 程序变换的有效提示或解释

在自动并行化时，有些中间结果揭示了自动分析和变换的依据，但是在程序并行化完成后就丢失了，不能为程序员利用。在 PROFPAT 中我们试图通过记录系统自动推算的过程来解释某些自动分析和变换的结果。它能告诉程序员系统为什么采纳或者拒绝一个变换。例如，当进行自动并行化变换时，由于跨循环数据相关导致某些 DO 循环不能被并行化。阻碍并行化的这些数据相关可以显示在程序窗口下面的正文信息窗口中。

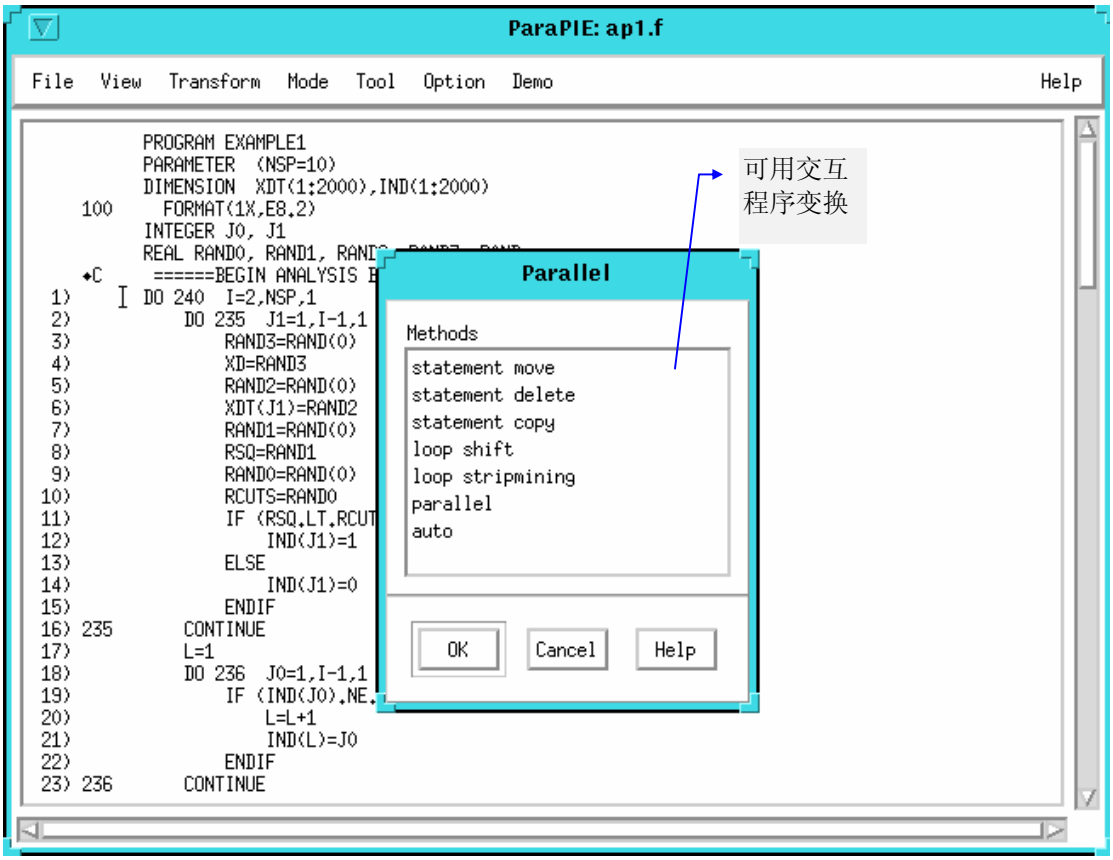


图 8-1、可用的交互程序变换

某些语义信息可以用来解释另一些语义信息。例如，循环迭代空间可以用来解释跨循环数据相关，程序执行时间统计和潜在并行性分析信息可以用来解释并行化的效果。

随着自动并行化研究的不断深入，新的程序变换技术会更多地涌现出来，因此，随着更细致的分析及变换，更多的变换选择会摆在程序员面前。通过有效的提示和解释影响可应用的程序变换（如图 8-1）或程序变换的可应用性，程序员在并行化程序时会更加有的放矢。当然，这需要将并行程序设计环境与自动并行化技术的研究更好的结合在一起。

8.2.3 增强程序信息可视化功能

8.2.3.1 多视图的可视化信息

为了有效标注解释图形的丰富信息内容，不仅要求“全”，更要求“直观”，因此不能一味将所有信息都一股脑儿地显示给程序员，使之举足无措，正如前面信息过滤的目的一样，应该允许信息通过多种角度提供给程序员。例如，我们已经实现的相关变量表、语句相关图和循环迭代相关图就是互为补充的表格或图形，分别从不同角度将影响并行化的数据相关信息提供给程序员。显然可视化视图越丰富，程序员的选择余地就越大。

8.2.3.2 三维动画

程序的数据相关图会受到并行化变换的影响而改变，如果使用 3 维图形和动画技术可视地模拟这一改变过程，将大大有助于程序员对程序的理解。

例如，在即将开展的 JavaPIE 研究中，我们准备改进二维循环迭代数据相关图的显示，使三重循环迭代空间可以由三维图形直接显示，而且能通过程序员交互旋转视角，帮助程序员发现程序中有意义的相关模式，为改进循环并行化技术提供进一步研究的有力工具。

在 JavaPIE 中我们已经实现了一个三维循环迭代空间数据相关图可视化原型工具 JavaIter3D，不仅实现了上述三维显示、交互修正视角的要求，而且还通过有向图分层算法给迭代图着色，并利用 Java 语言**[55]**的多线程技术以动画方式将可优先调度的循环迭代集合逐个突显出来，从而帮助程序员分析出 N 个迭代的多重循环在最大并行化后最长的串行执行路径迭代个数 L（循环最大加速比可达到 N/L）。

8.3 辅助程序的动态分析和预测

在实验中我们发现，程序的动态行为，如计算量和并行度的分析在很大程度上依赖于输入数据。而用于测试程序的输入数据往往人为缩小了数据的变化范围，导致用实际输入数据运行程序时未必就能达到测试时的加速比和并行度，因为程序并行化必须考虑所有输入组合下都正确的变换，所以往往达不到理想的并行度。当输入数据集合发生变化时，如果有工具能帮助发现程序动态行为的改变规律，则更有助于选择适合实际数据的并行化方案。

程序中的模式能为有经验的程序员识别和加以利用，指导选择适当的程序变换。所以在环境中不仅集成更多的程序理解工具是很有意义的，而且记录分析不同变换对不同程序段性能的优化，对于更好地预测在特定模式下，不同程序变换的应用效果是十分重要的。为此，我们在 PROFPAT 中已经实现了用户操作事件记录工具，它能动态记录一次程序分析变换周期的所有用户交互事件，具有可重播的功能，这样就可以帮助记录程序员的决策过程，如果在实现一种变换效果记录工具，与事件记录工具结合，就为分析程序模式提供了保存原始记录的辅助工具。

8.4 JavaPIE：在 Internet 上延伸并行编程交互环境研究

作为有效的并行程序分析环境，PROFPAT/PEFPT 的实践活动取得了很好的应用效果，例如被公认为难以并行化的三个 SPEC95 测试程序也通过交互分析找出了影响并行化的症结，得到了一定的并行化结果。

但是作为理想中的实用并行化编译技术开放研究平台，PEFPT/PROFPAT 尚存在一定的差距，主要体现在以下几个方面：

●缺乏充分的软件移植性、伸缩性和开放性: 即使使用 C/Motif 开发，在不同 UNIX 开发平台上实际上仍存在着许多不一致性，增加了软件移植难度。我们的 AFT/PROFPAT 系统在经过 HP9000，SUN Sparc, IBM RS6000, DEC Alpha AXP, Siemens RM600 等一系列平台移植后才逐渐趋于稳定，而且因为基于不同并行化编译工具实现了不同的程序中间表示，这也使并行编程环境缺乏功能相互移植的足够开放性。

●信息文档组织欠缺与松散：参与系统设计的人员之间、系统各个时期的文档之间缺乏组织，系统文档与用户说明之间缺乏一致性和可维护性，容易导致重要功能缺乏必要说明，以及重要说明文档缺乏检索支持等缺憾。

●不能及时更新以满足需求变化，跟不上合作研究共享技术的需要：由于工具集成于一个环境中，不同编程环境（如 PROFPAT 同 ParaPIE）之间新添的功能，即使由同一小组开发也必须重新编码才能互相移植，其他研究小组开发的新工具或技术由于我们缺乏对其内部实现结构的足够了解，也就更难加入不断更新的环境中了。

因此，为了达到更理想的移植性、开放性、及时共享性，我们准备在 Internet 上设计一个编程环境 JavaPIE(Java Parallel programming Internet Environment), 利用 Internet 上的 Java 语言**[55]**实现部分交互功能，并有机地延伸出来，为蓬勃地开展并行编程交互环境国际化合作研究开辟更为广阔的前景。

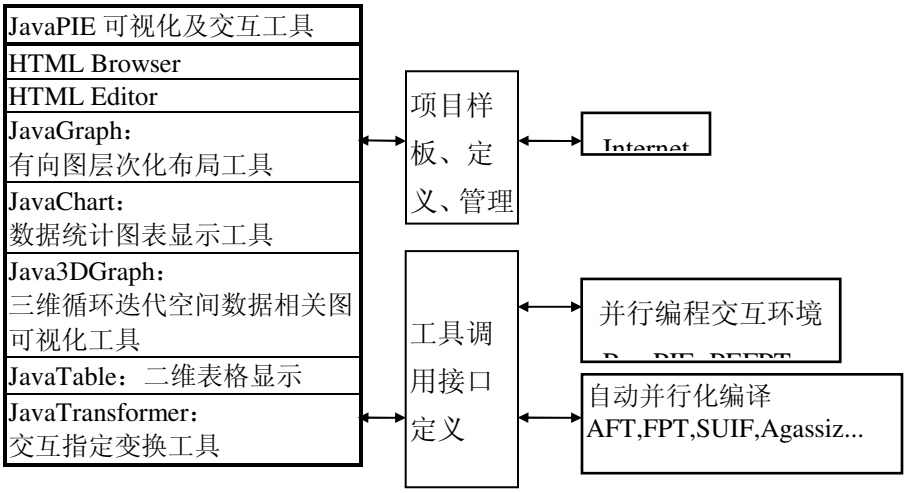


图 8-2、JavaPIE的系统结构

JavaPIE 的目标是利用 Internet 更好地开展并行编程交互环境的国际合作研究，因此如何将 Internet 的网络资源同 PROFPAT 的交互功能相结合，对不同层次和耦合度的合作研究项目组提供安全可靠而又及时的信息交流，是设计 JavaPIE 时必须考虑的决定性因素。

表 8-1、JavaPIE系统的主要工具和数据集

JavaPIE 主要工具	功能描述	对应 FPT/PROFPAT 支持
HTML 编辑器	编辑串行源程序正文	程序编辑器、代码生成器、程序对照工具
HTML 编辑器	编辑项目文档与样板脚本	项目管理器、宏操作记录
HTML 浏览器	获取 Internet 信息	无
JavaTable	观察变量信息表	数据流分析器、变量过滤器
JavaTable	观察数据相关信息表	数据相关性分析器、数据相关过滤器
JavaGraph	显示数据相关图	数据相关图分析器：操作、语句、任务级

JavaGraph	显示控制流图	结构化变换工具
JavaGraph	显示过程调用图	调用图分析器
Java3DGraph	显示循环迭代空间数据相关图	循环迭代空间数据相关图分析器
JavaChart	循环最大并行性数据	最大并行度分析器
JavaChart	过程及循环计算量数据	程序计算量分析器
JavaTransformer	选择程序段及变换	并行化变换工具

从 JavaPIE 的系统结构（图 8-2）和主要工具（表 8-1）我们可以看到 JavaPIE 与 PROFPAT 的继承调用关系。由于 JavaPIE 克服了 PROFPAT 依赖于程序内部中间表示的固有缺陷，所以能够方便地融合不同并行化编译工具和并行编程环境的功能，通过项目管理以超文本 HTML 有效地组织文档和描述实验过程和可重复实验结果，为同行交流程序并行化研究的心得提供了共同的平台。JavaGraph、Java3DGraph、JavaChart 、JavaTable、JavaTransformer，可以使用 Java Applet 实现并嵌入到 html 页中。由于统计图、扩展表格、以及菜单式交互程序设计可以借鉴比较一般的 Java 应用接口工具包 awt 加以实现，这里不加赘述。对于 JavaGraph，由于过程调用图，语句相关图，数据相关图等是程序语义信息的可视化图形显示的重要功能，所以必须设计一个能够清楚显示带圈有向图的图形布局算法，并允许用户交互拖曳调整图形布局，以达到直观、直接显示程序语义信息的目标。此外，如果把该算法用于循环迭代空间数据相关图，就可以分析出循环最大并行性（得到的最大层次号即循环最大并行化调度后最长串行执行路径长度。）下面就简要介绍 JavaGraph 这个重要算法的实现原理：

INPUT: graphwidht,graphheight,Graph = <N,E>，N 为结点集合，E 为边集合。

OUTPUT: {node 的 x,y 坐标| node in N}

ALGORITHM

1. 根据边集 E 建立结点集 N

   N = {}, for e=<n1,n2> in E do N = N+{n1,n2} end

2. 根据深度优先树为结点编号(dfn: N→Z)

3. 强连通分支划分，为结点编号强连通分支中最小的 dfn (sccn: N→Z)

4. 垂直分层布局:

```
N1 = {}，maxLevel = 0，
repeat
    for n in N do
        cnt(n) = count{n1|<n1,n> in E and not n1 in N1 and scc(n1)<>scc(n)}
        if cnt(n)=0 then level(n)=maxLevel
    end
    N1=N1+{n|level(n)=maxLevel}
    maxLevel++
until N1=N
//以下是可选的步骤，通过增加虚拟结点避免跨层次边、点交叉
for e=<n_1,n_2> in E and delta=level(n_2)-level(n_1)>=2 do
    N = N+{nn_k|  level(nn_k)=j(k)=level(n_1)+k, k=1,...,delta-1}
    E = E-{<n_1,n_2>}+{<n_1,nn_1>,...,<nn_delta-1,n_2>}
end
```

5. 坐标计算:

for level from 0 to maxLevel do maxPosition(level) = 0 end

for n in N do position(n)=maxPosition(level(n))++ end

for n in N do

x(n)=graphwidth*(position(n)+0.5)/maxPosition(level(n))

y(n)=graphheight*(level(n)+0.5)/maxLevel;

end

图 8-3、有向图分层布局算法描述

下面介绍三个 JavaPIE 应用的原型，一个是结合程序过程及循环计算量分析的过程调用-循环嵌套图显示 JavaProfile（JavaGraph+JavaChart)(图 8-4)，一个是操作级和语句级数据相关图显示 JavaSCCLayout（JavaGraph+JavaGraph)（图 8-5），最后一个是 3 维循环迭代空间数据相关图显示 JavaIter3D（图 8-6）。

- 从 Linpack.f 测试程序得到的程序计算量数据反映在 JavaGraph 过程调用图的调用边上，表示子过程或嵌套子循环执行时间占上一级过程或循环执行时间的百分比，通过单击分层调用图的不同结点，在 JavaChart 中立即刷新统计饼图以反映新的整体-局部调用关系。此外，由于 JavaGraph 支持对强连通图的处理（强连通分支结点布局在同一水平层中），所以不仅可以显示 Fortran77 的非递归过程调用图，也可以显示 C 或 Fortran90 的递归过程调用图。

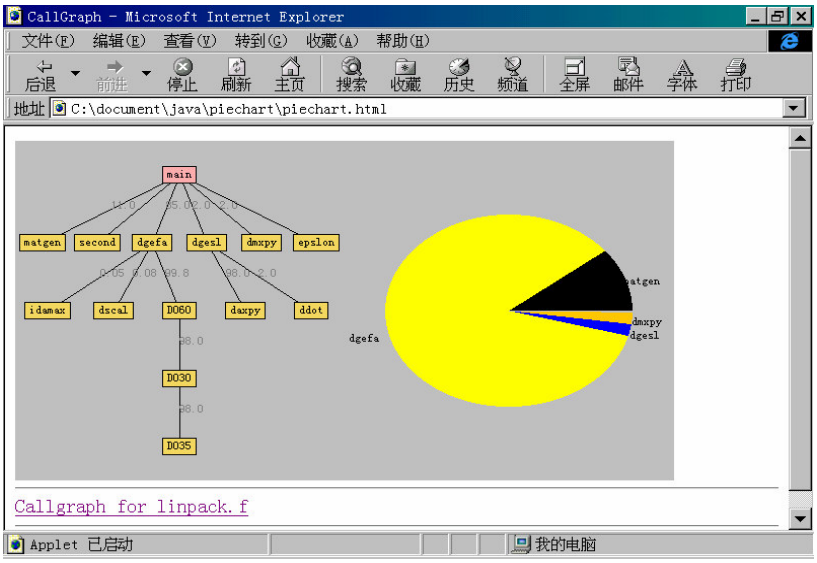


图 8-4、JavaPIE过程调用图及程序计算量统计图应用样板

- 再看下面例子：

(1) DO I=2, N

(2) A(i)=B(i)+C(i)

(3) D(i)=(A(i-1)+A(i+1))/2

(4) ENDDO

如果分析语句级数据相关图，则会显示流相关 $S2^{f}\rightarrow S3$ 和反相关 $S2^{a}\rightarrow S3$ 构成一个圈，而通过观察操作级数据相关图，则发现反相关 $S2^{a}\rightarrow S3$ 并不引起相关圈。

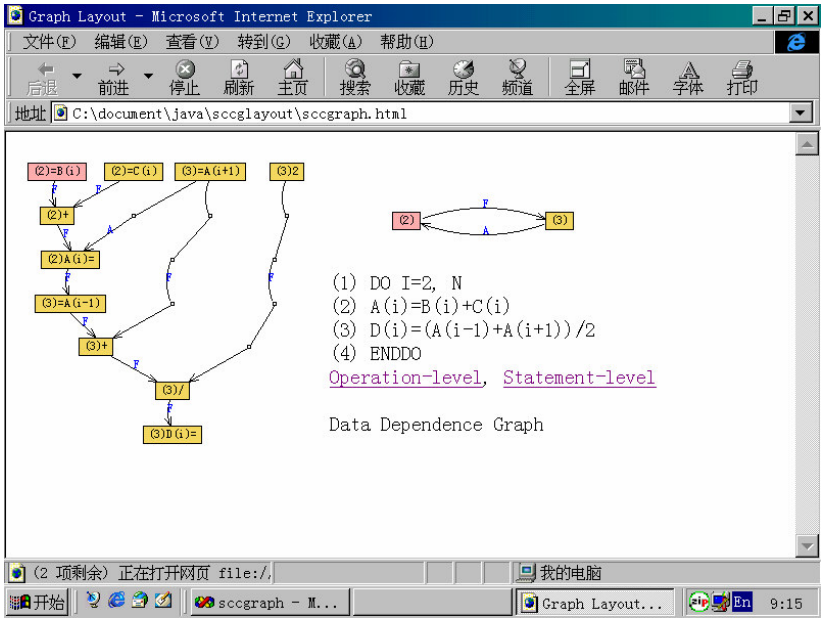
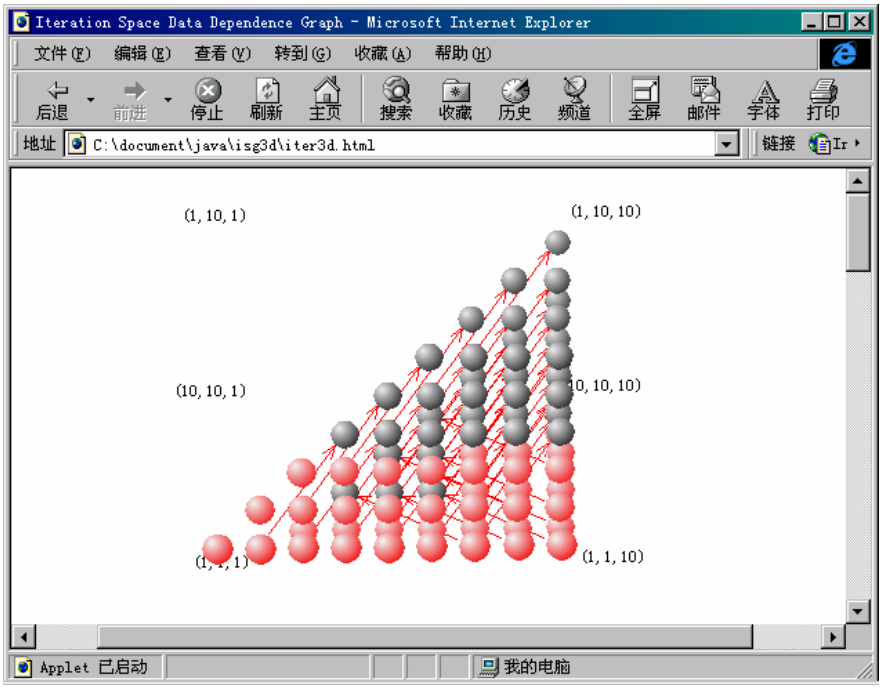
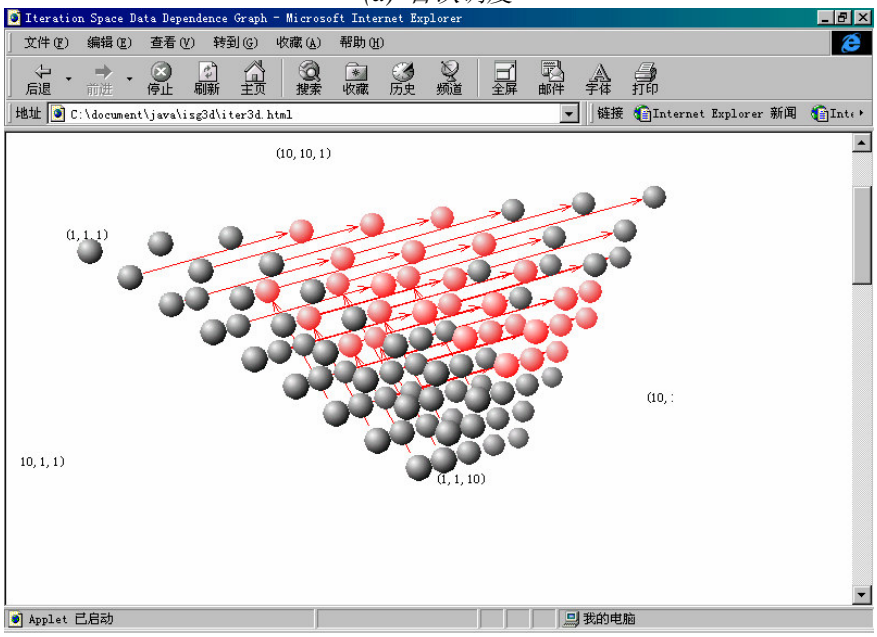


图 8-5、JavaPIE操作级和语句级数据相关图应用样板

- JavaIter3D 显示表 6-2a 的三重循环的三维循环迭代空间数据相关图：选取<i1,i2,i3>的子空间<1:10,1:10,1:10>加以观察，不在观察范围的迭代点没有绘出（也可以用白球显示），其中以圆球表示循环迭代点，黑球(cold)表示迭代空间中尚未执行的迭代，红球(hot)表示迭代空间中当前调度（第一次）可并行执行的迭代点。与 PROFPAT 中的二维循环迭代相关图一样，红色的相关边表示流相关，绿色代表输出相关，兰色代表反相关，存在跨循环相关边的两个循环迭代显然不能在同一次调度下并行执行。



(a) 首次调度



(b) 下一次调度

图 8-6、JavaPIE三维循环迭代空间相关图:转换视角显示不同调度任务可并行执行的迭代点。

在过去短短的几年中，我们同国际合作伙伴的信息交流已经从代价高昂的人员互访，软盘或磁带携带数据的脱机方式改变成通过 Email 交流信息、通过 FTP 传递数据、通过 HTTP 建立交互媒介的联机方式。与此同时，PROFPAT 系统也日见成熟，通过 Java 在 Internet 上的扩展（图 8-7），我们将通过 JavaPIE 同国内外的研究伙伴建立了更为密切的联系。

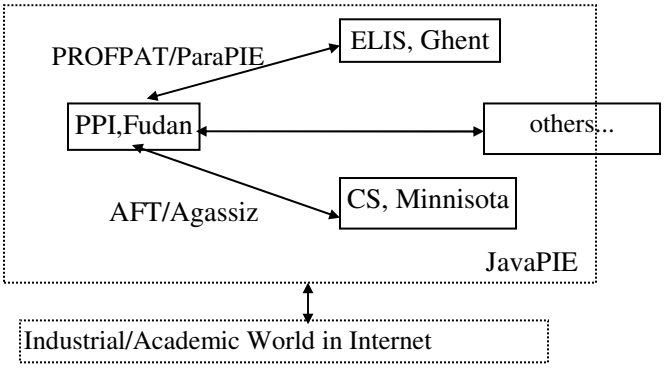


图 8-7、JavaPIE的现状和前景

9 结论

本文从理论上深入探讨了并行程序设计交互环境在帮助程序员分析和利用潜藏于串行程序的并行性的原理，详细介绍了并行程序设计环境 PROFPAT 中集成的主要设计功能和主要工具的实现方法，并结合交互相关性分析、交互循环并行化、交互数组私有化分析和变换、交互幺模变换等程序并行化分析、变换的初步应用实践，说明了这些工具和功能的应用方法。

将并行程序设计环境应用于分析难于自动并行化的 SPECfp95 测试程序，我们取得了较好的应用效果和技术发现：

- 本文通过介绍通过 PROFPAT 分析难以自动并行化的三个复杂的 SPEC95 测试程序 WAVE5, FPPPP 和 APSI 的实验过程和实验结果，进一步说明了并行程序设计环境在分析利用程序并行性提高计算性能方面的实际作用，证实了并行程序设计环境的有效性。
- 结合分析测试包程序的行为，并行程序设计交互环境还帮助我们找到了一些可以应用于自动并行化编译的有效改进技术，如：相关-覆盖数组私有化方法、多层嵌套循环幺模并行化变换方法、结合归约识别的增强幺模变换技术、非循环级并行性发掘、动态数据流分析、循环间 cache 优化等等。本文不仅从理论上阐述了这些新技术的原理，还概括了应用这些技术的自动算法。

我们相信，通过实现这些新的自动并行化技术，未来的自动并行化编译的可用性和并行化效果将得到加强，同时，通过改进和扩充并行程序设计交互环境的功能，更多的并行化技术等待着人们去发现。

10 后记

在朱传琪教授的悉心指导下，在复旦大学并行处理研究所从事的博士学习和研究工作始终使我感到受益菲浅。朱教授对学术问题造诣很高，洞察研究领域的深奥环节，指导我抓住并行程序设计交互环境和并行化编译研究的方向和要害。因此，每当我感到学习困难的时候，第一个想到需要请教的就是朱老师，而总能获得他的耐心指导。朱老师为我创造了难得的学习机会，使我得以与国际同行进行学术交流。朱老师不仅对我的学习和研究帮助极大，而且也照顾和关心我的各方面发展，使我丝毫不感到学习的枯燥和压力，从而获得了深入钻研的勇气和内在的求知动力。能够在朱老师的指导下进行博士学习，是我的幸运。

在学习过程中, 我还得到了 Erik D'Hollander 教授, 臧斌宇副教授的亲切指导, 张福波博士, 陈彤, 王琦, 施武, 丁永华, 李剑慧等老师的耐心帮助以及课题组其他同学的密切合作和相互切磋, 没有他们的指导、帮助和协作, 我很难顺利完成博士阶段的学习，在此由衷地表示感谢。

最后，我还要感激我的亲人和朋友从各方面对我的鼎力扶持，特别是父母在二十余年来对我培养、教诲和启发，鼓励我积极面对各种人生挫折的考验，挑战和磨练自己的意志，使我得以顺利成长。

在此，我无以为报，今后唯有以积极投入的精神加倍努力，不辜负大家对我的殷切厚望。谨以此文献给所有关心和爱护我的人们。

参考文献

[1] Adams J.C., W.S.Brainerd, J.T. Martin, B.T. Smith, J.L.Wagener, "Fortran 90 Handbook", New York: McGraw-Hill ,1992.

[2] Aho,A.V., Sethi,R., Ullman,J.D., *Compilers, Principles, Techniques, and Tools*, Addison-Wesley, 1986.

[3] Allen,F., Burke,M., Charles,P., Cytron,R., Ferrante,J., "An overview of the PTRAN analysis system for multiprocessing", Proc. of ICS'87, Athens, 1987.

[4] Allen,J.R., Baumgartner,D., Kennedy,K., Porterfield,A., "PTOOL: A semi-automatic parallel programming assistant", Proc. of ICPP'86, January 1986.

[5] Allen,J.R., Kennedy,K., "Automatic Transformation of Fortran programs to vector form", ACM Trans. on Programming Languanges and Systems, 9(4):491-542,October,1987.

[6] Allen,J.R., Kennedy,K., "PFC: A programs to convert Fortran to parallel form", Supercomputers: Design and Applications, pp186-205, IEEE Computer Society Press,1984.

[7] Anderson J.M., and Lam M.S., "Global Optimizations for Parallelism and Locality on Scalable Parallel Machines",pp112-125, ACM SIGPLAN '93 Conference on Programming Language Design and Implementation, Albuquerque, N.M.,Jun, 1993.

[8] Appelbe,B., Smith,K., McDowell,C., "Start/Pat: A Parallel Programming Toolkit", pp29-38,July 1989.

[9] Bailey,D., E.Barszcz, J.Barton, D.Browning, R.Carter, L.Dagum, R.Fatoohi, S.Fineberg, P.Fredrickson, T.Lasinski, R.Schreiber, H.Simon, V.Venkatakrishnan, S.Weeratunga, "The NAS Parallel Benchmarks", RNR Technical Report RNR094-007, March 1994.

[10] Balasundaram,V., Kennedy,K., Kremer,U., McKinley,K., Subhlok,J., "The ParaScope Editor: An Interactive Parallel Programming Tool", pp540-550,1989..

[11] Banerjee,U., "Unimodular transformations of double loops",in Advances in Languages and Compilers for Parallel Processing, pp.30-44,Cambridge, MA: MIT Press, 1991.

[12] Banerjee,U., *Dependence Analysis for Supercomputing*, Kluwer Academic Publishers, 1988.

[13] Banerjee,U., Eigenmann,R., Nicolau,A., Padua,D., "Automatic Program Parallelization ",Proc. of the IEEE, 81(2):211-243, Feb 1993.

[14] Bell,G., "Scalable Parallel Computers:Alternatives,Issues,and Challenges",Int'l J. Parallel Programming, vol. 22,pp3-46,1994.

[15] Bell,G., "Ultracomputers, A Teraflop Before Its Time",Communications of the ACM, vol. 35, pp27-47,1992.

[16] Berry, M., D.Chen, P.Koss, D.Kuck, L.Pointer, S.Lo, Y.Pang, R.Roloff, A. Sameh, E. Clementi, S.Chin, D.Schneider, G.Fox, P.Messina, D.Walker, C.Hsiung, J.Schwarzmeier, K.Lue, S.Orsag, F.Seidl, O.Johnson, G.Swanson, R.Goodrum, and J.Martin, "The Perfect Club Benchmarks: Effective Performance Evalution of Supercomputers", International Journal of Supercomputer Applications, Vol. 3, No.3, Fall 1989. pp.5-40.

[17] Blume,B., Eigenmann,R., Faigin,K., Grout,J., Hoeflinger,J., Padua,D., Petersen,P., Pottenger, B.,

Raughwerger,L., Tu,P., Weatherford,S., "Polaris: The Next Generation in Parallelizing Compilers", Proceedings of the 7th Workshop on Languages and Compilers for Parallel Computing, August 1994.

[18] Blume,W., Eigenmann,R., "Performance Analysis of Parallelizing Compilers on the Perfect Benchmarks Programs", IEEE Trans. on Para. & Distri. Sys., 3(6):643-656, Nov. 1992.

[19] Chandy,K.M., Kesselman,C., "Parallel Computing in 2001", IEEE Software, pp11-20, Nov. 1991.

[20] Chen,T., Zang,B., Zhu,C.Q., "A new method for array privatization ", in Proc.of HPCC'94, pp 43-50, Aug. 1994.

[21] Cooper,K., Hall,M., Kennedy,K., "A Methodology for Procedure Cloning", J. Comp. Lang,19(2):105-117,1993.

[22] Cooper,K., Kennedy,K., Torczon,L., "The impact of interprocedural analysis and optimization in the Rn programming environment", ACM Trans. on Programming Languages and Systems, pp491-523,October 1986.

[23] Cooper,K.et al,"The ParaScope Parallel Programming Environment",Proceedings of the IEEE, 81(2),Feb. 1993.

[24] D'Hollander,E.H., "Partitioning and labeling of loops by unimodular transformations", IEEE Trans. on Parallel and Distributed Systems, 3(4):465-476,1992.

[25] Ding,Y., Chen,T., Zang,B., Zhu,C.Q., "Cloning and Its Implementaion", J. Software,1996.

[26] Ding, Y., Yuan Q., Zang B., Zhu C.Q., "Cross-loop Reuse Techniques for cache Optimizations on Multiprocessors", accepted by J. of Software, 1997.

[27] Eggers,S., "Simulation Analysis of Data Sharing in Shared Memory Multiprocessors", Ph.D.thesis, University of California, Apr.1989.

[28] Eigenmann,R., Hoeflinger,J., Li,Z., Padua,D., "Experience in the Automatic Parallelization of Four Perfect Benchmark Programs", Proceedings of the 4th Workshop on Languages and Compilers for Parallel Computing, August 1991. pp.141-154.

[29] Feautrier,P., "Array Expansion", Proceedings of the 1988 ACM International Conference on Supercomputing, pp429-441, July, 1988.

[30] Flynn,M., "Some Computer Organizations and Their Effectiveness", *IEEE Transactions on Computers*, September, 21(9):948--960,1972.

[31] Flynn,M., "Very High Speed Computing Systems", Proc. of IEEE, 54(2):1901-1909,1966.

[32] Fox,G., et al, "Fortran D language specification", TR90-141, Rice University,.Dec. 1990.

[33] Geist,A., Beguelin,A., Dongarra,J., Jiang,W.C., Manchek,R., Sunderam,V., *PVM: Parallel Virtual Machine*,The MIT Press, 1994.

[34] Gross T., Steenkiste,P., "Structred Dataflow Analysis for Arrays and its Use in Optimizing Compiler", Software-Practice and Experience. 20(2):1333-155,Feb.1990 .

[35] Guarna,V.A.Jr., Gannon, D. et al, "Faust: An Integrated Environment for Parallel Programming",IEEE Software, July 1989, pp20-26.

[36] Hall,M., Murphy,B., Amarasinghe,S., Liao,S.-W., and Lam,M., "Interprocedural Analysis for

Parallelization", Proceedings of the 8th International Workshop on Languages and Compilers for Parallel Computing, August 1995.

[37] Huson, T. Macke, B. Leasure, and M. Wolfe, "The KAP/S-1: An advanced source-to-source vectorizer for S-1 Mark IIa supercomputer", Proceedings of the 1986 International Conference on Parallel Processing, August 1986. pp.833-835.

[38] Hwang,K., Briggs,F.A., *Computer Architecture and Parallel Processing*, McGraw-Hill, NY, 1984.

[39] Kennedy K., "Compiler technology for machine independent parallel programming", Int'l J. of Parallel Programming, 1994, 22(1).

[40] Kumar,M., "Parallelism in Computation-intensive applications", IEEE Trans. on Computers, 37(9),1988.

[41] Leasure,B., ed., "PCF Fortran: Language Definition, version 3.1", The Parallel Computing Forum, Champaign, IL, 1990.

[42] Li,J., Zang,B., Zhu,C.Q., "Runtime Dataflow Analysis", Technical Report of PPI, Fudan Univ.,1997.

[43] Li,Z., "Array Privatization for Parallel Execuation of Loops", Proc. Int. Symp on Computer Architecture, pp313-322, 1992.

[44] Li,Z.,Yew,P.C., Zhu,C.Q. "Data Dependence Analysis on Multi-dimensional Array References", ICS'89, pp.215-224,June 1989.

[45] Maydan,D., Amarasinghe,S., Lam,M., "Array Data-Flow Analysis and Its Use in Array Privatization", Proceedings of 20th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, Jan 1993.

[46] McKinley,K., "Evaluation Automatic Parallelization for Efficient Excecution on Shared Memory Multiprocessors",in Proc. ICS'94,pp54-64.

[47] Polychronopoulos,C., et al, "'"Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing, and Scheduling Programs on Multiprocessors"", ICPP'89.

[48] Pugh,W., "A Practical Algorithm for Exact Array Dependence Test", Communications of the ACM, Vol. 8,pp102-114, Aug 199

[49] Rauchwerger,L., Amato,N., Padua,D., "Run-time Methods for Parallelizing Partially Parallel Loops",Proc. the 1995 ICS, pp137-146, Barcelona, July 1995.

[50] Rauchwerger,L., Padua,D., "The PRIVATIZING DOALL Test: A Run-Time Technique for DOALL Loop Identification",Proc. of the 1994 Int. Conf. on Supercomputing, pp33-43,July 1994.

[51] Reilly,J., "A Brief Introduction to The SPEC CPU95 Benchmarks", SPEC Newsletter, Sep 1995.

[52] Shi,W., Zhu,C.Q., "An Implementation of Maximum Potential Parallelism Analyzing Method", J. Computer Engineering, Vol 18, No 2,pp9-15 1996.

[53] Singh, J.P., Hennessey J L., "An empirical investigation of the effectiveness and limitations of automatic parallelization", Proc. Int'l Symp. Shared Memory Multiprocessing, Tokyo, Apr. 1991.

[54] Stanford Compiler Group,The SUIF Parallelizing Compiler Guide,TR 1993.

[55] SUN Microsystem Corp., Java API Document, 1996.

[56] Tang W.; Shi W., Zang B., Zhu C.Q., "Exploiting loop parallelism with redundant execution",

Journal of Computer Science and Technology,12 (2):105-12, China, March 1997.

[57] Internet document: http://www.specbench.org/osg/cpu95/cfp95.

[58] Wang,Q., Yu,Y., D'Hollander,E.H., "Visualizing the Iteration Space in PEFPT",In Herzberger B., Sloots P.(Eds.) Proceedings High Perf. Computing and Networking,Viena, Springer Verlag, Apr. 1997, pp908-915.

[59] Wang,Q., Yu,Y., D'Hollander,E.H., "Interactive Programming using PEFPT", Proc. Seminar Parallel Computing: Software,Architecture and Operating Systems, pp123-129,Oct. 1996.

[60] Wilson,R., French,R., Wilson,C., Amarasinghe,S., Anderson,J., Tjian,S., Liao,S.-W. Tseng,C.-W, Hall,M., Lam,M., and Hennessy,J., "SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers", ACM SIGPLAN Notices, Vol. 29, No.12, December 1994. pp31-37.

[61] Wolfe,M., "Advanced Loop Interchange",Proc. 1986 Int'l Conf. on Parallel Processing,pp536-543.

[62] Wolfe,M., "Loop Skewing: The Wavefront Method Revisited", Proc. 1986 Int'l Journal on Parallel Programming, 15(4):279-293.

[63] Wolfe,M., *High Performance Compilers for Parallel Computing*, Addison-Wesley Publishing Company, 1996.

[64] Wolfe,M., *Optimizing Supercompilers for Supercomputers*, Ph.D. Thesis, University of Illinois, 1982.

[65] Xue,J., "Unimodular transformations of non-perfectly nested loops", Journal of Parallel Computing, 22(12):1621-45, Feb 1997.

[66] Yu,Y., Shi,W. Wang,Q., Zang,B., Ding,Y., Li,J., Wang,C., Zhu,C.Q., "Design and Application of Some Interactive Parallel Programming Tools in ParaPIE", Technical Report of PPI, Fudan Univ., 1997.

[67] Yu,Y., Shi,W., Zang,B., Zhu,C.Q., "Computing the Unimodular Transforming Matrix to Parallelize a Nested Sequential Loop with Const Dependence Distance ", Technical Report of PPI, Fudan Univ., Dec 1997.

[68] Yu,Y., Wang,Q., Shi,W., Zhu,C.Q., "The Parallel Programming Interactive Environment", Technical Report of PPI, Fudan Univ., 1995.

[69] Yu,Y., Wang,Q., Zang,B., Shi,W., Zhu,C.Q., D'Hollander,E.H., " Interactively Transforming Difficult Loops with Iteration Space Visualizer in PEFPT ", extended abstract, PPI, Nov. 1997.

[70] Yu,Y., Wang,Q., Zang,B., Shi,W., Zhu,C.Q., D'Hollander,E.H., " Interactively Studying the Unimodular Loop Parallelizing Transformations using PEFPT Programming Environment ", Int'l Seminar of Parallel Processing, Gent, Bel., Jan 1998.

[71] Yu,Y., Wang,Q., Zhu,C.Q., "The Design of a Parallel Programming Environment for FPT", Proc. Topic in Knowledge and Information Technology, pp69-80 Gent, Bel., Sept.1996.

[72] Zang,B., Yu,Y., Chen,T., Zhu,C.Q., "An Array Privatization Scheme Based on Data Dependence Test and Data Coverage", in Proc. Topic in Knowledge and Information Technology, pp 95-104, Sept. 1996.

[73] Zang,B., Yu,Y., Chen,T., Zhu,C.Q., "Dependence-Coverage: An Effective Method for Array

Privatization”, Technical Report of PPI, 1997.

[74] Zang,B., Zhu,C.Q., "最小循环分布和最大循环合并", 中国青年计算机研究新进展, pp 76-80, 1993.

[75] Zhang,F.B., “The FPT Programming Environment”, PhD Thesis, University of Gent, Bel.,1996

[76] Zhu,C.Q., Zang,B., Chen,T., "An Automatic Parallelizer ".Journal of Software(China), Mar 1996, Vol.7, No.3, pp180-186.

[77] Zhu,C.Q., Zang,B., Chen,T., "The Design Considerations and Test Results of AFT: A New Generation Parallelizing Compiler", Technical Report of PPI, Fudan Univ., 1996.

[78] Zima,H., Bast,H., Cerndt,M., “Superb: A tool for semi-automatic MIMD/SIMD parallelization.”, Parallel Computing, vol. 6, pp1-18, 1988.

B. 索引（按笔画顺序）