第33卷 第7期 计算机研究与发展 Vol. 33 No. 7
1996年 7月 COMPUTER RESEARCH & DEVELOPMENT July 1996

并行化技术与工具

金国华 陈福接
(国防科学技术大学计算机研究所 长沙 410073)

摘 要 程序并行化工具由于它能有效地解决多种并行机结构间的代码可移植性和大大地减轻用户使用并行机的困难，已成为当今并行处理领域的一个热门研究课题。相信随着对并行机系统越来越广泛的使用，它还将会得到不断的发展和完善。本文着重介绍并行化关键技术和工具系统的研究历史与现状，并就这一研究课题今后的发展趋势提出一些看法。

关键词 并行化技术，并行化工具，数据相关性分析，程序重构

PARALLELIZING TECHNIQUES AND TOOLS

Jin Guohua and Chen Fuhie
Computer Institute, University of Science & Technology for National Defence, Changsha 410073

Abstract Program parallelizing tools have become one of the popular research projects in the field of parallel processing as they can effectively solve the code portability between different parallel architectures and significantly reduce the difficulty of using parallel computers. With the more and more widespread application of parallel processing systems, they will also be developed and perfected continuously. In this paper, we emphatically introduce the research situation of parallelizing techniques and tools, and then propose some viewpoints about the trend of their development in the future.

Keywords Parallelizing technique, parallelizing tool, data dependence analysis, program restructuring.

1 并行化研究的背景和意义

追求更高的劳动效率、更好的生活质量，促进社会的不断向前发展，人类为此不仅使用了各种各样作为体力劳动增强器的机器，也越来越普遍地使用了作为脑力劳动的辅佐的计算机，计算机已应用到人类社会的各个领域。同样是源于对高性能的追求，人们早在计算机发展的初期，就认识到了并行处理的重要性，而且在计算机发展过程中也一直试图在系统的各个层次上开发并行性。然而由于应用范围与技术条件的限制，早期成果并不显著。近年来，高科技领域对计算机性能提出了越来越高的要求，而物理上的极限和工程实现上的问题使单机速度

原稿收到日期：1994-12-07；修改稿收到日期：1996-02-08。本课题得到国家自然科学基金资助。**金国华**，博士，1995年至1996年在美国明尼达大学作访问学者，目前主要研究巨型机FORTRAN语言及C语言并行化、并行编译等。**陈福接**，教授，主要研究领域：计算机体系结构。

很难满足应用要求，并行机结构引起了普遍关注。大规模、超大规模集成电路，特别是微处理器技术的逐渐成熟为并行计算机结构提供了技术-物质基础，使并行机系统的发展和应用成为可能。事实上并行计算机正在取代高性能单处理机系统而成为下一代超高速计算机系统的代表机型。目前大部分超级计算机和小型超级计算机都已经采用了并行体系结构。

然而，并行机结构的迅速发展并没有得到软件的相应支持，并行机系统的性能普遍没有得到充分发挥。据言，一台超级计算机的有效性能通常只有其峰值性能的 5%～25%[41]。人们希望并行计算机能尽快地得到软件的有效支持，使它能够更高效地运行，能更方便地为人们所用。并行化的研究为此提供了一条很重要的途径。

在当前缺乏能被用户普遍接受的并行程序设计语言标准的情况下，开发并行化工具能有效地解决多种并行机结构间的代码可移植性问题，避免了用户为使用新的并行机结构重写其所有的应用程序。由于重写将带来新的程序错误和效率问题，这种代码的可移植性在目前并行机结构发展迅猛的形势下是非常需要的。

并行程序比串行程序更难开发、调试、维护、理解和高效。要使并行程序高效运行，就必须有效地利用目标并行机的结构特性(多功能部件，流水线，向量寄存器，cache，local memory，互连网络结构等)，然而大多数用户(尤其在应用界)并不希望也没有兴趣花费很多额外的时间去了解结构细节，开发并行程序。他们更希望借助于并行化工具来免去或大大减轻他们在这方面的劳动。

各个应用领域都存在着大量的经过许多人长时间设计、使用和测试的大型程序。由于缺乏必要的文档，通常，从事并行化改写工作的程序员并不完全了解这些代码的全貌。改写过程又往往会引入大量的错误。利用并行化工具只需花费很少的人力就可以并行化现有程序，使其能在并行计算机上得到正确而又高效的运行。

2 并行化研究的历史与现状

2.1 关键技术：数据相关性分析和程序重构

2.1.1 数据相关性分析

有关数据相关性分析的工作早在 60 年代就已经开始[16]。然而这方面成就突出的应该算是 Kuck，Banerjee，Wolfe 等人所做的工作[36,10-11,63]。Banerjee 于 1976 年在他短短的 36 页硕士论文中率先提出了用于判别循环中语句间相关关系的 gcd 测试和界限测试法(后被称为 Banerjee 不等式测试法)。此后，Banerjee，Wolfe，Burke，Triolet，Callahan，Li 等人又分别在他们的博士论文和后续发表的一些论文和著作中提出了方向向量[63]、多级相关性测试[17]等一系列重要概念，讨论并给出了多维下标表达式情况的相关性求解方法[13,35]和过程间相关性分析方法[17,56,20,45,9]，进一步充实了早期的相关性分析工作，从而为循环结构的向量化和并行化奠定了很好的理论基础，也为开发过程间并行性创造了一定的条件。

相关性分析按精度分类可以分精确和近似两种方法，精确测试法给出的是相关关系存在的充要条件，所以采用精确测试法可以准确地判别相关关系的存在与否；而近似测试法判别的是存在相关的必要条件，采用近似测试法可以确定相关关系的不存在，但在不能证实不相关的情况下，为保证向量化和并行化的正确性，只能作出保守估计，假设相关关系的存在。

另外，相关性分析也可以按数组访问的下标线性度分成线性下标相关性分析和非线性下标相关性分析两种，或根据分析范围的不同分过程内相关性分析和过程间相关性分析两种。下面我们分别介绍它们的研究现状。

(1) 线性下标情况

对于单层循环的一维线性数组访问，目前已经有有效的精确测试方法可以运用[11,13]，然而对于多层循环的一维线性数组访问尤其是多维线性数组访问，除部分特殊情况外[13]，大多数相关性分析算法仍然只能给出近似的答案。

早期提出的一些方法[63,17,31]，为了提高效率，在处理多维线性数组访问的相关性问题上，或采用数组下标逐维测试方法或通过线性化方法将多维相关性分析问题转化为单维相关性问题。一般来说，在不出现耦合下标[58]的简单情况下，逐维测试法可以获得较为精确的结果，线性化方法则由于在丢番图方程中引入过多的变量精度较差；然而在另一方面，当出现耦合下标时，逐维测试法因孤立地对待数组的每一维下标，结果往往过于保守，线性化方法虽然能改善部分情况的测试精度，但多数情况下效果也并不明显。

Banerjee 的多维 GCD 测试法[13]和 Z. Li 的 λ-多维联立测试法[46]对相关性分析的精度有了一定的改善(根据 Li 的实验结果，应用 λ-联立测试法，从 Eispack 库中检测出的不存在相关的数组引用比逐维测试法增加 10.4%)，表现在多维 GCD 测试法能用于判别和求解同时满足所有丢番图方程的整数通解(不考虑约束条件)，λ-多维联立测试法可以判别同时满足所有丢番图方程和约束条件的实数解(不考虑整数解)。

从理论上来说，更为精确的相关性分析信息可以通过整数规划或线性规划来求得。近年来，在针对相关性分析的求解特性(大量小型方程组求解)，应用和改进 LP、IP 算法方面得到了较多的研究。Triolet 在他的相关性分析算法中[56]使用了 Foirier-Motzkin 消元法(一种基于线性不等式成对比较的线性规划方法，可以获得实数解)。Wallace 的约束矩阵测试法[61]采用了修改后的单纯形算法。Lu 和 Pugh 提出了用于相关性分析的整数规划算法。大多数算法使分析的精度有了不同程度的提高，但效率问题仍然是这类算法的关键问题。要获得一维多变量尤其是多维多变量数组访问情况下的精确相关性信息，算法往往具有指数时间复杂性[32]。

我们认为精确而又高效的相关性分析算法应该是多种算法的有效合成，Maydan 和 Goff 等人在这方面的研究成果[49,32]就是两个很好的例子。

(2) 非线性下标情况

和线性下标相对应，非线性下标指的是数组的下标是循环控制变量的非线性表达式。非线性下标可以部分地出现于数组下标的某些维，构成部分线性访问模式，也可以出现于数组下标的全部维，构成非线性访问模式。

目前大部分相关性测试方法仍然局限于对线性下标的分析。对于部分线性和非线性的访问模式可以通过使用重构技术，和向前替换、归纳变量替换和常数传递等，部分地将其转化为线性下标。但根据沈志宇等人在 Illinois 大学对 6 个库程序的分析结果[58]，转换后的程序中部分线性和非线性的访问模式仍然占有全部数组访问的 13.45%和 33.95%。

近年来出现了一些符号处理方法[47,55]，但它们或分析代价太大[47]或处理情况相对简单[55]，效果并不理想。要获得精确的相关性分析结果还需要有更为有效的符号处理方法。

(3) 含过程调用情况

含过程调用情况下的相关性分析(即过程间相关性分析)，以开发调用过程和被调用过程中的各种潜在并行性为目的，包含有三个方面的内容：别名分析、过程间常数传递和过程访问信息计算。

① 别名分析　使用别名分析可以提高相关性分析的精度。一般来说，别名分三类：最简单的别名关系由显式说明引起(FORTRAN 中的 EQUIVALENCE 语句和 C 中的 UNION 语句都具有这种功能)，我们称它为显式别名；另外，别名也可以通过复制参数产生(一个实参同时传给两个或多个不同的形参，或全局变量作为实参传给形参)，即所谓的参数别名，或由指针引起，被称作指针别名。这里我们主要介绍别名参数的研究工作。

传统的别名分析方法以变量名作为基本分析单位，不考虑数组变量的下标细节。它们对非循环调用图的处理非常简单。通过遍历调用图的每个过程结点标出程序中的所有别名。对于循环调用图情况(即含递归调用)，Banning[15]提出了检测别名的迭代算法，Cooper[24]又将别名分析划分为检测别名引入和追踪别名传递两部分，提出了别名传递的数据流分析框架。但由于忽视了数组变量的下标细节，这种变量名分析方法往往精度较差，不能满足并行化的要求。

② 过程间常数传递　和别名分析一样，使用过程间常数传递可以提高被调用过程的相关性分析精度，开发被调用过程中更多的并行性。

简单的过程间常数传递可以通过过程内的局部常数传递[62]和对检测出的不变形参的常量替换的反复迭代来实现，但这种方法的效率较低，迭代过程常常参杂有许多不必要的工作。Callahan 的过程间常数传递方法[19]通过引入跳跃函数 J_s^f(s 为调用点，f 为被调用过程的一个形参)，压缩过程体信息，将上述迭代过程转换成了对等式

$$f=A_{s=P}\cdot_Q J_s^f$$

的迭代求解，从而可以节省许多迭代时间，提高了常数传递效率。

③ 过程访问信息计算　相关性分析的一个重要组成部分是计算语句的 IN，OUT 集合。一般说来，在忽略别名影响和过程调用的情况下，IN、OUT 集合的计算较为简单。当语句包含过程调用时，编译无法直接确定所访问变量的集合，这时编译必须进行过程间访问分析，否则只能基于调用点所获得的信息作出保守假设。

访问信息计算的最简单方法是通过分析每个过程 P，确定对 P 的一次调用会引起对 P 的哪些参数的修改。这

里包括了对 P 中直接修改的和 P 中通过 CALL 语句间接修改的所有参数的分析。如果我们用 REF(P)表示过程 P 的所有参数访问对集合(访问对由参数和参数的访问类型构成)；LREF(P)表示过程 P 直接访问的所有参数对集合；MAP(e,s)表示调用点 $e=P\to Q$ 处(P 调用 Q)形参到实参的映射函数，那么采用这种方法的计算可描述为：

$$REF(P)=LREF(P)$$

$$REF(P)=REF(P)\cup[\cup_{e=P\to Q}MAP(e,REF(Q))]$$

对于非循环调用图，计算过程比较简单，只要对调用图作深度优先遍历，对每个过程 P 分析其直接修改的和其孩子(已分析)修改的参数。对于循环调用图，整个计算过程需要迭代进行。虽然迭代过程总能终止，但由于 MAP 函数的出现，算法不够高效[25]。

Cooper[23,26]进一步将访问信息计算问题分成全局变量访问和形参访问两部分：

$$REF(P)=LREF^{+}(P)\cup[\cup_{e=P\to Q}GREF(Q)]$$

$$LREF^{+}(P)=LREF(P)\cup[\cup_{e=P\to Q}MAP(e,RMOD(Q))]$$

其中 LREF+(P)表示 P 中直接访问的和通过传地址方式传递到其它过程被访问的参数对集合；RMOD(P)表示由 P 直接访问的或通过 P 中过程调用访问的 P 的形参；GREF(P)表示全局参数访问对集合，RMOD(P)可以通过计算 map 矩阵的自反传递闭包 map* 来确定[26]，从而有效地避免了 MAP 函数的计算。此后 Cooper 又提出了 binding 图概念[27]，简化了形参访问信息的计算，使简单访问信息计算问题在线性时间内能得到求解。然而 Cooper 方法的不足在于它和其它简单访问信息计算方法一样，仅仅计算了变量名访问信息。这对于数组来说是不够的，要确定更详细的数组访问信息，需要有更为精确的数组区域访问分析。

Li 的原子和原子映象法[44,45]所做的区域分析最为全面，它不仅记录了过程中每一数组访问的每一线性下标表达式，也保存了所有循环变量的界限(用循环不变量和外层循环变量表示)。存储这些信息的数据结构是原子和原子映象。原子包含了数组访问的局部信息，每个原子由一个二维数组 A 来表示，A 的第 i 行第 j 列(A_{ij})包含了第 i 维下标表达式中第 j 个循环变量的系数(循环变量按最外层到最内层依次排列)，A_{i0}代表常数项。另外，对应数组的每一行还有一位标志，当且仅当该维表达式是线性时，置该标志。原子映象在原子的基础上又增加了每个循环变量的界限信息。采用原子和原子映象法进行过程间相关性分析，首先将被调用过程中每个数组访问的原子映象转换成调用点处的原子映象，然后使用标准的相关性测试方法对这些原子映象做相关性分析。由于记录的这些信息本质上等价于源代码，这种方法获得的访问信息最为精确，但问题是无法对访问信息作合并操作，子程序中每个数组的每次访问都需要对应有一个原子和原子映象，存储空间消耗很大，这个问题在其它的一些方法中是以信息精度为代价来解决的。

Burke 和 Cytron 的线性化方法[17]将所有不同的数组访问转化为对同一单维数组不同区域的访问，保存的是线性化后的数组访问信息和循环界限信息。如，语句

$$A(i,j)=(A(i,j-1)+A(i,j+1)+A(i-1,j)+A(i+1,j))/4$$

中的 A 数组访问将被表示为 MEM[$N\times(i-1)+j+INIT_A$]，MEM[$N\times(i-1)+(j-1)+INIT_A$]，MEM[$N\times(i-1)+(j+1)+INIT_A$]，MEM[$N\times(i-2)+j+INIT_A$]和 MEM[$N\times i+j+INIT_A$]，这里 $INIT_A$ 指数组 A 在存储器中的启始地址。线性化后的访问信息可以单独保存，也可以合并后保存(两个访问区域的合并是取最小下限和最大上限)。单独保存获得的访问信息精度较高，但存在和 Li 方法同样的问题，合并可以节省存储空间，但结果往往过于保守。

Triolet 提议用线性不等式表示的 K 维凸形空间来描述一个数组的访问区域[56,57]，其算法分为两步：

(1) 用自顶向下的分析方法计算所有过程的执行环境；

(2) 用自底向上的分析方法计算所有过程的访问区域。精确地说，过程的执行环境定义了过程中每一局部变量的取值范围，是一个线性不等式集合。数组的访问区域由二元序偶(A,σ)来表示，其中，A 是区域所属的数组名字，σ 为形如 $1\leqslant\varphi_k\leqslant u$ 的线性不等式集合，φ_k 对应 A 中所访问元素在第 k 维的变化范围。数组的局部访问区域是单个语句访问的区域(其中的不等式可能含局部变量)，数组的全局区域是整个过程访问的区域，由过程中所有局部区域合并而成(不含任何局部变量)。过程调用语句的数组访问区域由被调用过程的数组全局区域在调用点处的执行环境下转换得到。相关性测试就是测试表示访问区域和执行环境的线性不等式组的一致性。这里，对线性不等式组的求解代价很高，理论上的时空复杂性是指数级的[12]。另外，一系列的合并操作也增加了结果区域的复杂

性(可能产生和合并的原始访问数同样多的不等式)。这是 Triolet 不等式方法的不足之处。

如果说前面的方法是试图以效率为代价换取最高精度的话，那么 Callahan[20]提出的受限规则域描述方法维(RRSD)实际上采用的是一种以损失部分精度来换取效率的策略。在[20]里，一个 RRSD 被定义为由一个数组名和一组表达式组成，每个表达式对应数组的一维，记录了数组访问空间的这一维中被访问的元素。表达式的取值有三种(以第 j 维为例)：① $i_j=\perp$ 表示任意值；② $i_j=\alpha$ 表示不变量；③ $i_j=\alpha\pm i_k(k\neq j)$ 表示相对位移量。使用 RRSD 方法可以描述数组空间的点、行、对角线、平面和其它子区域。区域间的合并操作是逐维进行的。两个相同常数 α 的合并结果是常数 α；两个相同表达式 $\alpha\pm i_k$ 的合并结果是表达式 $\alpha\pm i_k$；其它情况的合并结果取任意值 $\perp$。相关性分析表现为对区域的求交，如交集非空，则存在相关；否则不存在相关。两区域(r_1 和 r_2)间的相交关系由下列条件决定：

(1) 相应元素至少有一个具有 $\perp$ 形式；

(2) 一个具有 $\alpha\pm i_k$ 形式，另一个不是 i_k 的函数；

(3) 二者都具有 $\alpha+i_k$ 或 $\alpha-i_k$ 形式且相同的常数 α；

(4) 二者为相同的常数 α。

如果 r_1，r_2 中至少有一对相应元素不满足上述任何条件，则两规则域互不相交，否则交集非空。RRSD 方法具有高效的合并，求交操作，但所能描述的区域很有限，任何过程只要访问了任意两个不同行，列或对角线都被认为访问了整个数组。

为了更精确地描述数组访问的区域，又能执行高效的合并和求交操作，Balasundaram 提出了数据访问描述(DAD)方法[9]。DAD 用简化区域来描述一个数组的被访问部分，简化区域由数组访问空间的正交界和对角界表示。正交界是每一维的上、下界，对角界是每两维间的和、差界限。n 维数组访问空间的简化区域以如下不等式形式存储：

(1) 正交边界　　$\alpha\leqslant X_i\leqslant\beta$，　　$1\leqslant i\leqslant n$

(2) 正对角边界　　$\alpha\leqslant X_i-X_j\leqslant\beta$，　　$1\leqslant i,\ j\leqslant n,\ i\neq j$

(3) 反对角边界　　$\alpha\leqslant X_i+X_j\leqslant\beta$，　　$1\leqslant i,\ j\leqslant n,\ i\neq j$

简化区域的合并是取每一对对应正交界或对角界的最小下界和最大上界，简化区域间的相交关系通过比较对应边界的上、下界来判别。如果存在一对对应的正交边界或对角边界，其中一个的上界小于另一个的下界，则区域不相交，否则相交。和其它方法相比，DAD 方法具有合并，求交操作简单的优点，而且相关性计算也比较简单，所描述的简化区域也非常一般，可以包括各种规则形状，精度上要比 RRSD 方法更好一些。

图 1 给出了上述各种访问信息计算方法对数组元素访问的不同描述。从中我们不难发现，Li 的原子、原子映象法和 Burke 的线性化方法最为精确(这里线性化方法没有做合并区域操作)，Triolet 的线性不等式方法和 Balasundaram 的 DAD 方法其次，Callahan 的 RRSD 方法在这种情况下和传统的按名分析方法一样不精确。

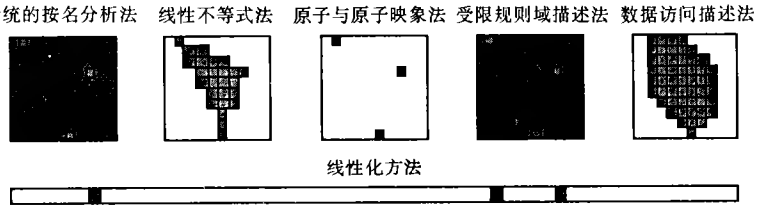

图 1　不同方法对数组访问 A[1,2],A[4,8],A[10,6]的不同描述

2.1.2 程序重构(program restructuring)

以相关性分析为基础，对程序作语义等价的(自动或手工)重构(也称改写 rewriting[42]或转换 transformation[37])提高程序在目标机系统上的实际运行性能是程序并行化和向量化的最终目的。

向量化程序重构经过多年的研究，目前已基本成熟，然而并行化程序重构并非如此。与向量化相比，一方面，并行化程序重构需要面对更加复杂多样的程序结构，以更为复杂的全局数据流分析，过程内、过程间相关性分析为基础，开发更多、更大粒度的并行性；另一方面并行化程序重构需要更多地考虑并行机的各种结构和机器特

性，使目标机性能能得到充分的发挥。所以程序的并行化改写要比程序的向量化改写更为复杂。目前，这方面的工作主要还停留在对单个重构技术的研究上，具体表现在以下几个方面(这里我们仅例举一部分)：

(1) 消除、打开相关

属于这类重构技术的主要有：私有化(privatization)，标量扩展(scalar expansion)，结点分割(node splitting)，下标集分割(index set splitting)和循环剥离(loop peeling)等。它们或引入额外的存储空间，或分割循环下标集，来消除或打开输出相关、反相关和跨迭代相关，促进其它重构技术的应用和并行性的开发。

① 私有化[29] 一个数组或标量变量使该变量局部于每个循环迭代，用于消除跨迭代相关。

② 标量扩展[37,52] 将一个标量变量转换为一个一维数组，用于消除可能阻止并行性开发的输出相关和反相关。

③ 结点分割[37,53] 主要用于存在含反相关的相关圈的场合，通过将一反相关的源复制到一个新引入的临时数组，并换名相关源为这个新数组，将反相关打开，促成后续的循环分布开发出循环中的并行性。

④ 下标集分割[11,63] 将一个循环的迭代空间分成两个相邻的部分，用于打开相关圈或打开跨某一迭代的相关。

⑤ 循环剥离[69] 将循环的前 k 个迭代或后 k 个迭代分离出来，可用来打开和循环的前 k 个迭代或后 k 个迭代有关的相关。

(2) 调整执行序开发并行性

这类重构主要通过改变迭代内或迭代间的语句执行序来展示或增加循环级并行性，或优化数据局部性。属于这类重构技术的有：循环分布(loop distribution)，循环交换(loop interchange)，循环扭曲(loop skewing)，循环融合(loop fusion)，循环对准(loop alignment)，周期压缩(cycle shrinking)和语句交换(statement interchange)等。

① 循环分布[36,31] 通过划分循环中的语句将一个循环转变成为多个具有相同循环头的循环，用于从那些必须串行执行的语句中抽取出可以并行执行的语句。

② 循环交换[63,2] 将两个相邻的循环进行交换，改变迭代空间的遍历次序，可用于开发并行性，调整并行循环的粒度和优化数据局部性。

③ 循环扭曲[64,65] 以展现波前并行性为目的，通过“扭曲”迭代空间改变内层循环的相关距离，使所有相关在循环交换后都能转变为外层循环携带相关，从而实现内层循环并行化。循环扭曲本身不能直接开发出任何并行性，它的意义在于辅助循环交换完成并行性开发。

④ 循环融合[27] 是循环分布的逆变换，它通过融合两个相邻的循环来增加并行区域的粒度和提高数据的重用性。

⑤ 循环对准[20] 通过将一个或多个语句实例从一个迭代移到另一个迭代使外层循环携带相关转化为内层循环携带相关或循环独立相关。

⑥ 周期压缩[53] 通常用于存在流相关圈(仅由流相关组成的圈)的循环结构，抽取出隐含在其中的并行性。转换后的循环结构由外层串行循环和内层并行循环组成。周期压缩只有当所有相关距离都大于 1 时才能开发出并行性，这时的折减因子 $\lambda>1$。

⑦ 语句交换[1,20] 或更一般地，语句重排序(statement reordering)，改变循环中的语句序。可以用来将向后相关(backward dependence)转变为向前相关(forward dependence)促成循环分布转换，也可以用来优化 DOACROSS 循环[53]。

(3) 优化多级存储

这类重构的目的是调整一个循环在计算和存储访问间的平衡，更好地利用多级存储和功能流水线。属于这类重构技术的有：条化(strip mining)，标量置换(scalar replacement)，循环展开(loop unrolling)和展开与合并(unroll and jam)等。

① 条化[48,52] 将步距为 1 的一个循环转换为一个由步距大于 1 的外层循环和完成步内迭代的内层循环组成的嵌套循环。它可以用来开发并行性(类似于周期压缩)，也可以用来优化向量寄存器和 cache 的使用(根据向量寄存器长度和 cache 容量决定新步距的大小)。对于多重嵌套循环，它还可以和循环交换一起使用，调整迭代执行序，增加数据访问的局部性和数据的可重用性。和条化类似的重构技术还有块化(blocking)([38]中的块化等同于

条化，[43]中的块化实际上是条化加循环交换)，迭代空间片化(tilmg)[66-67](片化是条化加循环交换)和[68]中的 sectioning 和 combing(前者等同于条化，后者等同于片化)等。

② 标量置换[22]　将具有一致相关的数组访问替换成标量临时变量以便分配到寄存器。它通过减少要求的存储访问数来改善程序的性能。

③ 循环展开[70]　通过展开一个循环的循环体，减少循环开销，增加标量替换的潜在机会，减少存储器和寄存器间的数据传输。

④ 展开与合并[22]　通过展开嵌套循环中一个外层循环的循环体和合并所得到的内层循环，可以增加标量替换的潜在机会，改善流水线利用率(在出现递归时)，改善 cache 性能，提高 cache 命中率。

2.2 并行化系统

一些典型的并行化系统见表 1 所示。

最早的向量化和并行化工具[38,51](以向量化为主)PARAFRASE 由 Illinois 大学于 70 年代末研制成功，它支持基本的数据相关性分析(采用 GCD 测试法和 Banerjee 不等式测试法[10])，执行大量的程序重构改善程序的并行性。在 PARAFRASE 中，程序并行化过程分三阶段：结构无关的优化，结构相关的优化和机器相关的优化，对程序的重构按预先规定的次序(要做的重构和重构间的次序事先由用户在 Passlist 中说明)逐遍进行(每遍完成一个重构)，重构后的相关性信息，通过重新分析程序得到。用户和系统间的信息交流主要依靠用户在源程序中插入断言和命令将信息传给系统和系统通过 transformation record 记录转换过程中的有关信息供用户分析来实现。PARAFRASE 的实现对后来的向量化和并行化工具的研究产生了深远的影响。

PARAFRASE Ⅱ 是目前 Illinois 大学正在开发的一个多语种源对源重构编译器，根据研究计划[55]和 PARAFRASE 相比，PARAFRASE Ⅱ 将进一步提高相关性分析精度(通过符号相关性测试)，增加过程间访问信息的计算[20]，别名分析和过程间常数传递的能力[19]，提供更多的并行化重构技术(周期压缩等)。此外，PARAFRASE Ⅱ 还将具有循环自动调度(auto-scheduling)功能，并通过利用 FAUST 环境[53]提供交互并行化和图形显示的能力。

PFC(Parallel Fortran Converter)是继 PARAFRASE 之后，Rice 大学于 1979 年开始开发的，目前还在不断完善的一个向量化并行化工具[2,20,40]。PFC 的早期目标是完成自动源对源向量化功能，将串行 FORTRAN 程序转换为语义等价的向量 FORTRAN 程序(ANSI FORTRAN 8X)。近年来，该项目主要致力于串行程序的自动并行化问题的研究，在原来的基础上进一步完善和扩充了数据相关性分析[6,13,17,63]，过程间副作用分析[25]，过程间常数传递[19]和过程间规则域分析[20,40]等功能，提高了相关性分析的精度。PFC 的分析结果主要用于开发循环级并行性，也可以以 ASCII 码文件形式输出供 Rice 大学的其它两个并行化工具 PROOL 和 PED 使用。

PTOOL 是应 Los Alamos 国家实验室研究人员的要求由 Rice 大学自 1985 年开始研制的一个半自动并行程序设计工具[5]，工具由两部分组成：PSERVE 和 PQUERY，它们分别运行于不同的机器，PSERVE 是早期向量化 PFC 版本的修改版，完成相关图的构造并将它存入数据库(数据库由两个文件组成，一个是相关图文件，另一个是保存循环结构信息和定义引用链信息的文件)；PQUERY 使用 PSERVE 构造的数据库和用户进行交互对话，PQUERY 在屏幕上显示实际源程序代码，用户选择要分析的循环，PQUERY 识别出指定区域的全局变量(PTOOL 将变量分为全局和局部两类)，显示出可能阻止并行执行的任何相关并同时给出简单的解释和提示信息。另外，为了更好地分析大型科学程序，PTOOL 提供了简单的相关过滤机制使用户可以迅速地从大量的相关关系中挑选出他们感兴趣的那一部分相关。然而作为程序设计支持系统，PTOOL 的交互还不够充分，它不能在程序改动后马上重新显示相关，要重新显示就要求对整个程序再次调用 PSERVE，这样做的代价很大。

PED(ParaScope editor)是 Rice 大学以 PFC、PTOOL 和 R^n[①]的研究工作为基础，目前正在开发的一个交互并行程序设计工具[8,35,50]。使用 PED，用户和工具各尽其责，用户选择出需要考虑的循环(PED 只开发循环级并行性)，工具计算并显示对应的相关性信息，用户确认相关信息的正确性，选择需要执行的循环重构，工具为用户提供专家性意见，执行复杂的程序重构。作为 ParaScope 并行程序设计环境[21]的一个重要组成部分，最终完成的

① R^n 是 Rice 大学于 1982 年开始研制的一个串行程序设计环境[4,25]，由模块编辑器、合成编辑器、程序编译器和模块编译器等部分组成。通过编辑编译过程中对过程间信息的收集和记录能有效地支持过程间分析和优化，帮助程序员产生整个程序的高质量机器代码，R^n 是第一个将过程间分析和优化用于程序编译的系统。

PED 将具有源程序编辑，过程内和过程间相关性分析(符号分析、别名分析、过程间常数传递、规则域描述方法)，同步分析，分析结果的显示(将进一步扩充 PTOOL 的相关过滤机制)，各种程序重构和程序编辑或重构后对源程序和分析结果的快速增值修改等功能。

表 1 典型并行化系统一览表

序号	系统名称	研制机构	研制带头人	研制时间	主 要 功 能
1	PARAFRAST	University of Illinois at Urbana Champaign, U. S. A	D. Kuck	—70 年代末	重构按预定次序逐遍进行，用户和系统间的信息交流以批方式进行
2	PARAFRASE Ⅱ	University of Illinois at Urbana Champaign, U. S. A	D. Kuck, C. D. Polychronopoulos	80 年代末—	面向多语种，符号相关性测试，过程访问信息的计算，别名分析，过程间常数传递，循环自调度，借助 FAUST 交互
3	PFC	Rice University, U. S. A	K. Kennedy	1979—	自动并行化，过程间副作用分析，过程间常数传递，过程间规则域分析
4	PTOOL	Rice University, U. S. A	K. Kennedy	1985—	分离式半自动并行程序设计工具，简单的相关过滤机制
5	PED	Rice University, U. S. A	K. Kennedy	1989—	交互并行化，源程序编辑，过程间相关性分析(符号分析、别名分析、过程间常数传递、规则域描述方法)，相关性增值修改，丰富的循环重构
6	PTRAN	IBM T. J. Watson Research Center, U. S. A	F. Allen, M. Burke, R. Cytron	1987—	自动并行化，常数传递，线性递归变量的识别，过程间相关性分析(别名分析，过程间常数传递和过程间访问区域分析)，循环级、任务级并行性开发
7	PAT	Georgia Institute of Technology, U. S. A	K. Smith	80 年代末—	交互并行化，只针对单赋值语句分析，不计算距离、方向向量，简单的重构，简单的相关性增值修改，能插入同步
8	SIGMACS	University of Illinois at Urbana Champaign, U. S. A	D. Kuck	80 年代末—	交互并行化，显示过程调用图和进程图，计划支持语句插入，删除后的相关性增值修改
9	SUPERB	Bonn University, Germany	H. Zima	1985—1989	面向分布存储多机系统，根据用户对数据划分的说明产生带 send 和 receive 的结点程序
10	KDPASTE	国防科大计算机研究所软件工程教研室	吴健安	1988—	面向少量高性能向量单机构成的向量多机，优先向量化，开发少量大粒度并行性，交互并行化
11	FAT	复旦大学计算中心	朱传琪	1989—	自动并行化，可对并行程序进行分析

PTRAN(Parallel TRANslator)自动并行化工具由 IBM T. J Watson 研究中心近年来研制[7,18,28]，它具有很强的程序分析功能，可以完成控制流、数据流和控制相关性分析，常数传递，线性递归变量的识别(包括互相定义的

线性递归变量)以及过程内和过程间的相关性分析(别名分析、过程间常数传递和过程间访问区域的分析)。PTRAN 可以开发循环级和任务级并行性，但能做的程序重构非常简单。

PAT(interactive Fortran Parallelizing Assistant Tool)是 Georgia 工学院开发的一个交互源对源并行化转换工具[59,60]。其相关性分析仅局限于循环中对每个变量只写一次的 Fortran 程序。PAT 只采用简单的相关性测试方法，不计算距离和方向向量，不能做像循环交换和循环扭曲这样的循环级转换，不进行符号分析和过程间分析，但 PAT 能分析含一般并行结构的程序，支持赋值语句的插入、删除、复制、对整、移动和循环并行化，并对相关信息作相应的增值修改，也能插入同步来保证特定的相关。

SIGMACS 是 Illinois 大学 FAUST 程序设计环境下的一个交互可编程并行化工具[33,35]，它计算显示过程调用图和进程图，执行交互式程序转换并计划支持语句插入和删除后对相关性信息的自动修改。

SUPERB(SUprenum ParallelizER Bonn)是 Bonn 大学开发的一个集 MIMD 并行化和 SIMD 并行化(即向量化)于一体的交互系统[69]，用于交互地将串行程序转换为可运行于 SUPRENUM 分布存储多处理机系统的数据并行 SPMD 程序。SUPERB 对程序作控制流分析、数据流分析、常数传递和数据相关性分析，根据用户对数据划分的说明产生带有必要 send 和 receive 操作的结点程序。

国内对并行化工具的研究起步较晚，目前已有的其它系统主要有 KD-PASTE 和 FAT。KD-PASTE(PArallelization SofTware Environment)是由国防科大软件工程教研室研制的一个并行化系统，它以少量(2～8 台)高性能向量单机构成的向量多机系统为背景，优先考虑向量单机性能的发挥，开发少量大粒度并行性[71,72]。FAT(Fortran Automatic Transformer)是复旦大学计算中心研制的一个自动并行性分析系统，它具有全局数据流、控制流分析和数据相关性分析功能，能自动分析标准 Fortran 程序，将它改写为语义等价的并行向量程序，也可以对已有并行程序进行分析[73]。

3　并行化研究的发展趋势

纵观并行化研究的历史与现状，我们认为今后并行化技术的研究将主要体现在以下几个方面：

(1) 复杂下标的相关性分析和过程间相关性分析

精确的相关性分析是提高并行性检测能力的基础。复杂下标的数据相关性分析和过程间相关性分析，包括过程间数据流分析、别名分析、过程间常数传递和过程间数组访区域的分析，将成为研究的重点。

(2) 技术间的有效协调

每个重构技术都有各自的应用条件和背景，注意各技术间的有效协调应用对开发并行性具有重要作用。

(3) 不规则并行性开发

除规则并行性外，实际程序中也存在着大量的不规则并行性，不规则并行性的开发不可忽视。

(4) 并行性的多层次开发

并行性的来源是多方面的，按粒度大小顺序可分为：问题级、算法级、程序级和操作级并行性。程序级并行性又可进一步分为：任务级、循环级和语句级并行性，只有有效协调地开发多个层次上的并行性，才能保证并行机系统的充分利用。

(5) 结构相关的重构技术

并行机系统结构各不相同，针对目标机结构，研究更为有效的程序重构技术对提高机器的实际运行性能非常关键。

此外，我们认为未来的并行化工具应具有如下特点：

(1) 增值相关性分析

程序重构和修改后进行增值相关性分析，有效改善工具的运行效率。

(2) 更高的智能

工具将具有更高的智能，能自动有效地完成程序的并行化过程，也能启发和帮助用户完成程序并行性的开发。

(3) 友好的人机界面

友好的人机交互界面使人和机器能各尽所能，取长补短，协同完成代码的改写，产生能高效运行的并行代码。

(4) 工具的集成

并行化工具作为并行程序设计环境的一个重要组成部分，应能和环境中的其它配套工具，如编辑工具、执行分析工具、可视化工具、调试工具和目标代码生成工具等有效配合，共同完成并行程序设计任务。

参 考 文 献

[1] Allen J R. Dependence analysis for subscripted variables and its application to program transformation. Ph. D. thesis, Rice University, 1983.

[2] Allen J R, Kennedy K. Automatic loop interchange. In: *Proceedings of the SIGPLAN'84 Symposium on Compiler Construction*, June 1984.

[3] Allen J R, Kennedy K. PFC: A program to convert Fortran to parallel form. In: *Supercomputers: Design and Applications*. IEEE Computer Society Press, 1984.

[4] Allen J R, Kennedy K. Programming environments for supercomputers. Rice University, TR85-18, Mar, 1985.

[5] Allen J R, Baumgartner D, Kennedy K, *et al*. PTOOL: A semi-automatic parallel programming assistant. In: *Proceedings of the* 1986 *International Conference on Parallel Processing*, Aug 1986.

[6] Allen J R, Kennedy K. Automatic translation of Fortran Programs to vector form, *ACM Transactions on Programming Languages and Systems*, Oct 1987, 9(4).

[7] Allen F, Burke M, Charles P, *et al*. An overview of the PTRAN analysis system for multiprocessing. In: *Proceedings of the* 1987 *International Conference on Supercomputing*, 1987.

[8] Barasundaram V, Kennedy K, Kremer U, *et al*. The ParaScope editor: An interactive parallel programming tool. In: *Proceedings of Supercomputing* '89, Nov 1989.

[9] Balasundaram V. A mechanism for keeping useful internal information in parallel programming tools: The data access descriptor. *Journal of Parallel and Distributed Computing*, 1990, 9.

[10] Banerjee U. Data dependence in ordinary programs. M S thesis, University of Illinois at Urbana-Champaign, Nov 1976.

[11] Banerjee U. Speedup of ordinary programs. Ph D thesis, University of Illinois at Urbana-Champaign, 1979.

[12] Banerjee U. A direct parallelization of CALL statements A review. University of Illinois at Urbana-Champaign, CSRD Rpt. No 576, Apr 1986.

[13] Banerjee U. Dependence analysis for supercomputing. Kluwer Academic Publishers, 1988.

[14] Banerjee U. A theory of loop permutation. In: Gelernter D, *et al* ed. *Languages and Compilers for Parallel Computing*, The MIT Press, 1990.

[15] Banning J. A method for determining the side effects of procedure calls. Ph D thesis, Stanford University, Aug 1987.

[16] Bernstein A. Analysis of programs for parallel processing. *IEEE Transactions on Electronic Computers*, Oct 1966, 15(5).

[17] Burke M, Cytron R. Interprocedural dependence analysis and parallelhaztion. In: *Proceedings of SIGPLAN* '86 *Symposium on Compiler Construction*, 1986.

[18] Burke M, Cytron R, Ferrante J, *et al*. Automatic discovery of parallelism: A tool and an experiment. In: *Proceedings of SIGPLAN Symposium on Parallel Programming: Experience with Applications, Languages, and Systems*, 1988.

[19] Callahan D, Cooper K, Kennedey K, Torczon L. Interprocedural constant propagation. *Journal of the ACM*, 1986.

[20] Callahan D. A global approach to detection of parallelism, Ph D thesis, Rice University, 1987.

[21] Callahan D, Cooper K, Hood R, *et al*. ParaScope: A parallel programming environment. *The International Journal of Supercomputer Applications*, Winter 1988, 2(4).

[22] Callaham D, Carr S, Kennedy K. Improving register allocation for subscripted variables. In: *Proceedings of the ACM SIGPLAN* '90 *Conference on Programming Language Design and Implementation*, June 1990.

[23] Cooper K, Kennedy K. Efficient computation of flow insensitive interprocedural summary information, *SIGPLAN Notices*, 1984, 19(6).

[24] Cooper K. Analyzing aliases of reference formal parameters. In: *Proceedings of the 12th ACM Symposium on Principle of Programming Laguages*, Jan 1985.

[25] Cooper K, Kennedy K, Torczon L. The impact of interprocedural analysis and optimization in the R^n programming environment. *ACM Transactions on Programming Languages and Systems*, Oct 1987, 8(4).

[26] Cooper K, Kennedy K. Efficient computation of flow insensitive interprocedural summary information—— A correction. *SIGPLAN Notices*, Apr 1988, 23(4).

[27] Cooper K, Kennedy K. Interprocedural side-effect analysis in linear time. *SIGPLAN Notices*, July 1988, 23(7).

[28] Cytron R, Ferrante J, Sarkar V. Experiences using control dependence in PTRAN. In: Gelernter D, *et al*, eds. *Languages and Compilers for Parallel Computing*, The MIT Press, 1990.

[29] Eigenmann R, Hoeflinger J, Jaxon G, *et al*. Restructuring Fortran programs for Cedar, In: *Proceedings of* 1991 *International Conference on Parallel Processing*, Aug 1991.

[30] Eigenmann R, Blume W. An effectiveness study of parallelizing compiler techniques, In: *Proceedings of* 1991 *International Conference on Parallel Processing*, Aug 1991.

[31] Girkar M, Polychronopoulos C. Compiling issues for supercomputers, In: *Proceedings of Supercomputing* '88, Nov 1988.

[32] Goff G, Kennedy K, Tseng C. Practical dependence testing. In: *Proceedings of the ACMSIGPLAN* '91 *Conference on Programming Language Design and Implementation*, June 1991.

[33] Guarna V, Gannon D, Gaur Y, *et al*. Faust: An environment for programming parallel scientific applications. In: *Proceedings of Supercomputing* '88, Nov 1988.

[34] Kennedy K, M^{c}Kinley K. Loop distribution with arbitrary control flow. In: *Proceedings of Supercomputing* '90, New York, Nov 1990.

[35] Kennedy K, M^{c}Kinley K, Tseng C. Interactive parallel programming using the ParaScope editor. *IEEE Transactions on Parallel and Distributed Systems*, July 1991, 2(3).

[36] Kuck D. *The structure of computers and computations*, John Wiley and Sons, New York, 1978.

[37] Kuck D, Kuhn R, Padua D, *et al*. Dependence graphs and compiler optimizations, In: *Conference Record of the 8th ACM Symposium on the Principles of Programming Languages*, Jan 1981.

[38] Kuck D, Kuhn R, Leasure B, *et al*. The structure of an advanced retargetable vectorizer. In: Hwang K., Ed. *Tutorial on Supercomputers: Designs and Applications*, IEEE Press, 1984.

[39] Kuck D, Davidson E, Lawrie D, *et al*. Parallel supercomputing today and the CEDAR approach. *Science*, 231, 1986.

[40] Havlak P, Kennedy K. An implementation of interprocedural bounded regular section analysis. *IEEE Transactions on Parallel and Distributed Systems*, July 1991, 2(3).

[41] Hwang K. Advanced parallel processing and supercomputer architectures. In: *Proceedings of the IEEE*, Oct 1987.

[42] Lamport L. The parallel execution of DO loops. *Communications of the ACM*, Feb 1974, 17(2).

[43] Lam M, Rothberg E, Wolf M. The cache performance and optimizations of blocked algorithms. In: *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr 1991.

[44] Li Z. Interprocedural analysis and program restructuring for parallel programs. University of Illinois at Urbana Champaign, CSRD Technical Report No. 720, Jan 1988.

[45] Li Z. Intraprocedural and interprecedural data dependence analysis for parallel computing, Ph D thesis, University of Illinois at Urbana Champaign, Aug 1989.

[46] Li Z. An efficient data dependence analysis for parallelizing compilers, *IEEE Transactions on Parallel and Distributed Systems*, Jan 1990, 1(1).

[47] Lichnewsky A, Thomasset F. Introducing symbolic problem solving techniques in the dependence testing phases of a vectorizer. In: *Proceedings of the* 1988 *International Conference on Supercomputing*, 1988.

[48] Loveman D. Program improvement by source to source transformations. *Journal of the ACM*, Jan 1977, 17(2).

[49] Maydan D, Hennessy J, Lam M. Efficient and exact data dependence analysis. In: *Proceedings of the ACM SIGPLAN* '91 *Conference on programming Language Design and Implementation*, June 1991.

[50] M^{c}Kinley K. Automatic and interactive parallelization, Ph. D. thesis, Rice University, Apr 1992.

[51] Padua K, Kuck D, Lawrie D. High-speed multiprocessors and compilation techniques, *IEEE Transactions on Computers*,

Sept 1980, 29(9).

[52] Padua D, Wolfe M. Advanced compiler optimizations for supercomputers, *Communication of the ACM*, Dec 1986, 29(12).

[53] Polychronopoulos C. *Parallel Programming and Compilers*. Kluwer Academic publishers, 1988.

[54] Polychronopoulos C, Girkar M, Haghighat M, *et al*. Parafrase-2: An environment for parallelizing, partitioning, synchronizing, and scheduling programs on multiprocessors. In: *Proceedings of International Conference on Parallel Processing*, Aug 1989.

[55] Polychronopoulos C, Girkar M, Haghighat M, *et al*. The structure of Parafrase-2: An advanced parallelizing compiler for C and Fortran. In: Gelernter D, *et al*, eds. *Languages and Compilers for Parallel Computing*, the MIT Press, 1990.

[56] Triolet R. Interprocedural analysis for program restructuring with Parafrase University of Illinois at Urbana-Champaign, CSRD Rpt. No. 538, Dec 1985.

[57] Triolet R. Direct parallelization of CALL statements. In: *Proceedings of SIGPLAN'86 Symposium on Compiler Construction*, 1986.

[58] Shen Z, Li Z, and P. C. Yew. An empirical study of Fortran programs for parallelizing compilers. *IEEE Transactions on Parallel and Distribrted Systems*, July 1990, 1(3).

[59] Smith K, Appelbe W. PAT -- An interactive Fortran parallelizing assistant tool. In: *Proceedings of the* 1988 *International Conference on Parallel Processing*, St. Charles, IL, Aug 1988.

[60] Smith K, Appelbe B, Stirewalt K. Incremental dependence analysis for interactive parallelization. In: *Proceedings of the* 1990 *ACM International Conference on Supercomputing*, Amsterdam, The Netherlands, June 1990.

[61] Wallace D. Dependence of multi-dimensional array references. In: *Proceedings of the 2nd International Conference on Supercomputing*, July 1988.

[62] Wegman M, Zadeck F. Constant propagation with conditional branches. In: *Proceedings of the 12th ACM Symposium on Principle of Programming Languages*, Jan 1985.

[63] Wolfe M. Optimizing supercompilers for supercomputers. Ph D thesis, University of Illinois at Urbana-Champaign, Oct 1982.

[64] Wolfe M. Loop skewing: The wavefront method revisited. *International Journal of Parallel Programming*, 1986, 15(4).

[65] Wolfe M. *Optimizing Supercompilers for Supercomputers*. Pitman Publishing, London: 1989.

[66] Wolfe M. Iteration space tiling for memory hierarchies. In: Garry Rodrigue, editor. *Parallel Processing for Scientific Computing*, 1989.

[67] Wolfe M. More iteration space tiling. In: *Proceedings of Supercomputing* '89, Nov 1989.

[68] Wolfe M. Data dependence and Program restructuring. *The Journal of Supercomputing*, 1990, 4.

[69] Zima H, Bast H, Gerndt M. SUPERB: A tool for semi-automatic MIMD/SIMD parallelization. *Parallel Computing*, 1988, 6.

[70] Zima H, Chapman B. *Supercompilers for Parallel and Vector Computers*. Addison-Wesley Publishing Company, 1990.

[71] 并行化软件环境 KD-PASTE(V1.0). 研制报告, 国防科大计算机研究所软件工程室, 1991, 9.

[72] 并行化软件环境 KD-PASTE(V1.0). 技术报告, 国防科大计算机研究所软件工程室, 1991, 9.

[73] 程序并行化. 复旦大学计算中心并行处理研究组, 1992.