6. 软件过程模型的概念模式

研究过程模型的一个很重要方面是确定组成软件开发过程的概念模式信息，这些信息属于许多不同的类型，如图5所示。

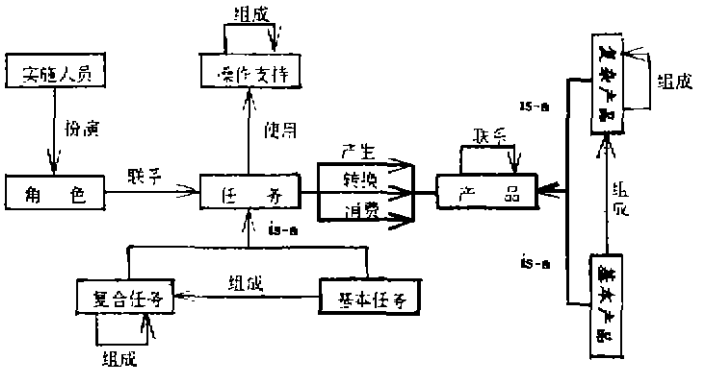

图5 过程模型的概念模式

一个软件过程是由任务(活动)集组成的，这些任务又可提精细化，分成若干子任务，这样一直下去，直到分解成基本任务为止。一般而言，高层任务相应于管理任务。在任务执行时，就会创建、消耗和修改产品。

产品表示了真实世界对象，是基本成份的聚集。在软件过程执行期间，产品的行为是由状态集来描述的。由于任务的开始依赖于工具使用情况及其产品的当前状态，因而这些产品的结构形成了处理产品基本成份的任务执行顺序。

在执行给定的任务时需要人力和物质资源，执行者代表负责任务管理的人或工具，他可以由能力集来描述。操作支持可以是在任务执行期间使用的一个工具集，也可以是用其知识来完成任务的开发人员。有两类基本任务：一类是简单任务，由单个人来执行，如编写一个函数；另一类是与多个开发人员密切相关的任务，这是由多个人(小组)来执行的，如会面碰头等。

每个任务至少与一个角色联系在一起，这个角色是在任务实施期间由一个或多个执行者扮演的。一个角色负责一个任务集，为此可以区别各种不同的角色，负责一项任务的管理人员、顾问、以及参加者等等。一个任务的管理人员负责任务的执行及所产生的产品质量。顾问是拥有完成任务所需知识的专家，参加者是实现任务产品的人员，一个角色可以与几个任务联系起来。

结束语 随着学术界和工业界对软件过程的研究和开发的关注，不仅提交了许多研究报告，而且还开发了一些支持软件过程的集成环境。作者对近年来各国同行在软件过程方面的工作进行了追踪、分析、研究和总结，并在此基础上提出了软件过程管理的系统模型 M-POID[42]，以 M-POID 为方法学开发了一个软件过程的集成环境 ISPE[43]。目前，正是软件过程的研究成果向产品转化的时期，不久就会有更多的软件过程支持产品出现，这将不仅有助于缩短软件产品的开发周期，而且还将提高软件产品的质量。

(参考文献共43篇略)

(上接第29页)

参考文献

[1] P. Beckman, Portable Parallel Programming in HPC++, In ICPP'96 Proc.

[2] A. S. Girmshaw, Easy-to-use Object-oriented Parallel Processing with Mentat, Computer, 26(5)1993

[3] P. A. G. Silivotti, A Tutorial for CC++, Technical Report CS-TR-94-02, California Institute of Technology, 1994

[4] J. Bannning, A Method for Determining the Side Effect of Procedure Calls, Ph. D. thesis, Stanford University, 1978

[5] M. Burke, etc., Interproceduaral Dependence Analysis and Parallelization, ACM SIGPLAN'86 Symposium on Compiler Conf. 12(7)1986

[6] M. B. Girkar, Functional Parallelism: Theoretical Foundation And Implementation, Ph. D. dissertation, University of Illinois at Urbana-Champaign, 1991

[7] Z. Li & P. Yew, Interprocedural Analysis for Parallel Programs, In Proc. Int. Conf. on Parallel Processing, 1988

[8] G. Zhu, L. Xie and Z. Sun, A New Mehtodology of Data Dependence Analysis for Parallelizing C++, ACM Sigplan Notice, 30(12)1995

[9] G. Zhu, L. Xie and Z. Sun, PRG: Procedure Reference Analysis in Extracting Non-Data Parallelism, Proc. of ICA3PP'96, 1996

OO 并行编程系统的研究现状与发展

The State and Advance of Object-Oriented Parallel Programming System

杜建成 陈道蓄 谢 立

(南京大学计算机科学系 南京 210093)

摘 要 This paper presents the approaches to develop object-oriented parallel programming system (OOPPS). Explicit language components or basic classes for describing parallelism, synchronization and communication can be provided. On the other hand, the implicit task parallelism, as well as data parallelism within a task, can be developed automatically by the system. However, combining both approaches will be beneficial for developing easy-to-use OOPPS with high efficiency, portability, and application adaptability.

关键词 Object oriented, Parallelism, Synchronization, Communication, Task parallelism, Data parallelism.

1 引言

并行程序的设计、调试与维护相对于串行程序来说更为复杂和困难，于是人们开始寻求用 OO 技术来开发并行系统。OO 编程系统提供信息的封装性、继承性和多态性。封装性支持高度的模块化，减少误操作的机会，增强程序可读性；继承性利于提高系统的开发效率，减少代码冗余；多态性给程序员提供了更大的灵活性。

在 OO 系统中，对象具有信息封装性，适合于作为描述松散耦合系统的基本部件和并行执行的自然单位。基于此，我们可以将并行性归为两类：对象间的并行性和对象内的并行性。前者属于任务级并行性，后者侧重于数据并行性的开发。

在考察 OO 系统中两个任务 A、B 的可并行问题时，有这样几种情况：①A 与 B 两个任务在执行过程中无信息的交流，即 A、B 可完全并行执行，称之为第一类并行性。②A 与 B 在执行过程中的某一时刻有信息交流，其余时间并行执行，称之为第二类并行性。③A 任务在执行的起点需要 B 任务的信息，而 B 任务只有执行到终点才能产生这些信息，这时 A、B 只能串行执行。在开发第二类并行性的时候，须要有效地解决通讯和同步问题。因此，设计一个 OO 并行编程系统(包括语言、库、预处理器和编译器)要解决并行性的表示与识别、通讯与同步以及调度问题。通常，解决这些问题有两条途径：

• 显式途径：在语言级提供描述手段，由用户显式定义可并行成分以及所需的通讯与同步。显式方式较容易实现，能产生质量较高的并行代码，但增加了程序员的负担，对程序员要求高。

• 隐式途径：由系统预处理器和编译器自动识别可并行部分，并提供所需的通讯与同步。隐式方式减轻了对程序员的要求，提高了开发效率，但编译器设计过于复杂，产生的代码质量往往不理想。

近年来，出现了综合这两种方式的并行程序开发系统，如 Mentat[2] 系统，用户负责数据和计算的分解，系统完成通讯、同步以及任务的调度工作。让用户和编译器各自去做自己擅长的工作，有利于提高并行程序的质量以及开发效率。

2 显式并行设施

如果采用显式方案设计并行编程系统，需要在语言级提供描述并行、同步和通讯的设施以及方便编程的并行类库。目前很多 OO 并行编程语言都是在串行语言(如 C++)基础上增加这类并行设施扩充而成的，如 CC++[3]，Mentat[2]，HPC++[1]等等。提供这些设施有两种途径：一是修改宿主 OO 语言的编

译器，在语言里增加新的命令。这种方案给用户提供了更多的灵活性，可以描述较小粒度的并行性。二是提供描述并行、同步和通讯的基类。用户可以利用OO的继承机制，利用这些基类(由语言实现者在库函数中提供)派生出描述问题域的构件。这种方法无须修改编译器，移植方便。

2.1 并行标识设施

描述并行佬为，可屹有以下几种方法：

(1)以对象作为并行的单位，每个对象被赋予一个或多个进程(线程)。如在Mentat[2]中，Mentat对象被定义成并行执行的自然单位。任何一个类，如果它包含的计算量很大，或者包含一些费时的操作(如I/O)，或者包含了与其它对象的共享信息，都可以被定义成Mentat类。一个Mentat类的实例的方法可以与程序中的其它计算并行执行。Java语言提供了多线程机制，它提供了一个称为Thread的类，该类以及该类的派生类的对象均为线程，线程可以与系统中其它的成分并行执行。在ABCL/1中，每个对象原则上都可以被并行执行，只要它所需要的信息能够被及时提供。

(2)将进程或线程定义成语言的一个构件。在Presto中，采用了这种方案，定义了一个Thread类。该类的实例代表一个进程(线程)，该类一旦被创建，便可以启动一个或多个对象的操作，例如：

```
Stack *s=new Stack(100); //创建对象 s
Thread *t=new Thread(" Pusher",TID);
        //创建线程 t
t->Start(S,S->Push,43);
        //线程 t 执行 s 的操作
```

(3)语句级并行标识设施。在CC++[3]中，用*par*和*parfor*这两个关键字来表达并行性，前者表明一个语句块中各个语句可并行执行，后者表明一个循环中的各个迭代可以并行执行。在HPC++[1]中，用#praga HPC-INDEPENT来指明循环中的各次迭代是独立的。

2.2 同步设施

为了使互相合作的进程(线程)之间能够协调一致地工作，需要提供同步设施。

(1)锁：这是一种比较简单的同步设施。在PARC++中定义了两类锁BlockLock和SpinLock，它们提供了以下几个方法：lock，trylock和unlock，这两个锁的不同之处在于：当一个锁已被占用了，BlockLock将释放所占有的资源而SpinLock则一直占有cpu，直到该锁可以被使用。Java中也提供了类似的机制，它用一个关键字synchronized来对方法或其中的一段代码进行标记，表明同一时刻只能有一个进程(线程)进入该区域。

(2)monitor：这是一种比较高级的同步机制。在CC++，PARC++，PRESTO中都提供了monitor，它可以由系统提供的一个基类来实现。一个monitor一次只允许一个进程(线程)在其中运行，在monitor中执行的进程(线程)通过条件变量进行同步。monitor中维持着两个队列：就绪队列和条件队列。就绪队列中存放着准备进入monitor的进程(线程)集合，而条件队列中存放着等待某一条件的进程(线程)的集合。方法signal用来唤醒某一等待进程(线程)，方法wait用来将因某条件得不到满足而不能继续运行下去的进程(线程)放入相应的条件等待队列。进程(线程)调用方法enter进入monitor，调用方法leave离开monitor。用户可以利用这种形式的monitor作为基类，根据具体需求派生相应的子类，实现有效的同步。

2.3 通讯设施

CC++[3]中，采用一个称为异步通道(一个先进先出的消息传送缓冲区)的机构来管理通讯。联系着这个缓冲区有两个操作：nonblockingsend()和blockingreceive()，前者用来将一个消息放入缓冲区，CC++将它设计成原子函数，不允许被中断；后者用来从缓冲区中取走一个消息，如果没有消息可被移走，系统允许执行这个操作的进程(线程)被挂起。CC++用Achannel类来实现异步通道。

在PARC++中用mailbox类来实现通讯，它在Achannel的基础上进行了功能扩充，同时提供了同步消息发送和异步消息发送功能。

2.4 数据并行类

数据并行类允许一个类的方法并行地在一组对象上执行。在PC++中，从同一个类派生出来的对象的结构化集合用一个特殊的类Collection来定义。一个Collection类的构造函数形式如下：

Collection Class<elementtype>C (Template Class T,<Collection-size>,<align-function>)(<element_constructor>)

其中Template Class的构造函数形式为：

TemplateClass T(<size of target-template>,

<processor-array>,<distribution-function>)

TemplateClass 所完成的功能是将模板 template 分布到各个处理节点上,而 align-function 的功能是将 Collection 中各个成员对象映射到 template 上。一个 CollectionClass 包括两类方法:全局方法和 MethodofElement 函数,是对成员对象进行操作的,它们供全局方法调用。系统可以定义一个基本的 Collection 类库,用户利用 OO 的继承机制,定义自己需要的 Collection 类,通过 Collection 能较容易地实现数据并行计算。

3 隐式并行性的自动分析

隐式并行化(即自动并行编译)的主要任务是识别和实现串行程序中潜在的可并行性,为此首先要进行依赖关系分析,包括语句间依赖关系分析和过程间依赖关系分析。目前已有不少依赖关系分析技术,其中许多是从 Fortran 程序的并行化工作中发展出来的,但对 OO 语言的并行化同样适用。OO 语言的一些特点(如继承机制,对象作为参数传递,多态等)也给并行性的分析带来了新的影响,须要发展新的方法和技术处理它们,文[8]提出用冲突集和方法引用向量来分析对象内和对象间的并行性的方法,文[9]提出用 PRG 图(Procedure Reference Graph)作为工具来分析存在对象参数情况下过程间依赖关系分析问题。总的来说,这方面的工作开展尚在起步阶段。

3.1 语句间依赖关系分析

分析语句间的依赖关系(Interstatement Dependence),尤其是循环体内语句间的依赖关系是自动并行编译的基础。语句间的依赖关系分析技术基本上可以分为两类,一类是近似测试法,如 GCD 法、Banerjee 不等式法,这些测试方法给出了存在依赖关系的必要条件,即通过这种测试方法可以断定不存在依赖关系,但是如果不能肯定不存在依赖关系,则假定存在依赖关系。另一类是精确测试法,如 Delta 测试法、λ 测试法、多维 GCD 法,这些测试方法给出了存在依赖关系的充要条件,但算法也相对复杂。在实际运用中,可以根据具体情况灵活采用这两种方法。无论是哪一种方法,依赖关系测试都可以归结为解满足一组线性约束条件的一组线性丢番图方程,精确测试法求出方程组的整数通解,并且检查是否有满足所有约束条件的解,而近似测试法检查方程组或方程是否有整数解,然后测试该方程组或方程满足约束条件的解存在的必要条件。

程序的依赖结构可以用依赖关系图来描述。依赖关系图是有向图 G=(V,E),对于描述语句间的依赖关系图,每个节点代表语句,节点之间的联线代表不同类型的依赖关系。

3.2 过程间依赖关系分析

OO 程序中存在着对象间的大量相互调用,为了开发对象方法间甚至方法内的潜在并行性,须要利用过程间依赖关系(Interprocedural Dependence)分析技术。这种分析技术可以解决别名问题,过程间常数传递问题以及过程访问信息的表示和计算。进行别名分析时,如果分析对象是标量,则比较简单;如果是数组,还应考虑数组下标。文[4]给出了检测别名的迭代算法和过程间常数传递的分析方法。一般可将过程访问信息分析分为流敏感信息分析和流不敏感信息分析。流敏感分析考虑过程体内的控制流,它考察的是一种必然情况;流不敏感分析不考虑过程体内的控制流,它考察的是一种可能情况。在进行分析时,如果变量是数组,还应考虑下标情况,这就涉及到区域分析技术。

3.2.1 区域分析 首先用合适的形式表达和描述区域,在选择表达方式时,要考虑表达的精确性、区域合并的能力以及检测区域相交的能力。

(1)原子和原子映象[7]。这种方法不仅记录了过程中数组访问的每一维线性下标表达式,还保存了循环控制变量的界限(用循环不变量和外层循环控制变量)。一个原子包含了数组访问的局部信息,可由两维数组 A 表示,A 的第 i 行对应数组的第 i 维,如果数组访问的第 i 维表达的是循环控制变量的线性组合,则第 i 行的第 j 列包含了表达式中第 j 个循环控制变量的系数。每一行还有一项表明该维表达式是否为线性。原子映象是原子的扩展,与原子相比,它还包括了循环控制变量的界限,这一信息被连接到原子的末尾。利用这些信息,就可以进行依赖关系的分析,该方法的缺点是对访问信息的合并操作比较困难。

(2)数组访问线性化[5]。线性化方法仅针对被存储的访问信息,不影响代码中的实际的数组访问。例如,A[i+j]=B[i+j],线性化之后便成为 MEM[M

$*(i-1)+j+bA]=MEM[M*(i-1)+j+bB]$，A，B均为M×N的数组，bA，bB分别表示A，B在存储器中的起始地址，这样就解决了别名问题，这种方法对访问区域的合并是取最小下限和最大上限，在很多情况下比较保守。

(3)线性不等式方法。这种方法用线性不等式来表示的k维凸形空间来描述数组访问区域，每一个区域被定义为一个二元对(A,σ)，A是区域所属的数组的名字，σ是形如$1\leqslant\Omega_k\leqslant k$的线性不等式的集合，$\Omega_k$表示A中所访问的元素在第k维的变化范围。

(4)RRSD方法。也是用一个数组名和一组表达式来描述一块数据访问区域，每个表达式对应数组的一维，RRSD方法具有高效的合并和相交操作，但是所描述的区域有限。

(5)DAD方法。用数组访问空间的正交界和对角界来描述被访问区域，正交界是每一维的上下界，对角界是每两维的和差界限，其一般形式为：

$\alpha\leqslant X_i\leqslant\beta \quad i\in[1,k]$

$\alpha\leqslant X_i-X_j\leqslant\beta \quad \forall i,j\in[1,k],i\neq j$

$\alpha\leqslant X_i+X_j\leqslant\beta \quad \forall i,j\in[1,k],i\neq j$

3.2.2 过程分析工具 在开发过程间的并行性时，往往要进行全局分析，选择合适的中间形式作为分析工具。

(1)PSG法。Callahan利用PSG图(Program Summary Graph)来获取流敏感信息(flow-sensitive information)。Callahan定义了两类流敏感信息：KILL集和USE集，分别表示一定被修改的变量集和被写之前被使用的变量集。PSG图代表程序的抽象，概括了过程间的控制流信息。PSG包含了四类节点：进入点、调用点、返回点、退出点。对于每个过程的每个形参都对应着一个进入点和一个退出点；对于调用点处的每一个实参，都对应着一个调用点和一个返回点。调用点和进入点之间以及退出点和返回点之间的联线表示形参与实参之间的结合。另外，从进入点和返回点也有一些边连到调用点和退出点，这些反映了过程的控制流结构。对于计算KILL集和USE集来说，PSG图是一个有效的数据结构。

(2)HTG分析法[8]。PDG采用统一的方式来处理数据依赖和控制依赖，是一种广为采用的分析程序中依赖关系的方法，但是它比较适合于识别数据并行性，用它检测任务并行性并不理想。HTG(Hierarchical Task Graph)是一个有向无环图HTG=(HV,HE)，对于HV中的每一个节点x，它属于下列类型之一：

• 起始节点：起始节点没有入弧，从它出发可达HV中的每一节点。

• 终止节点：终止节点没有出弧，从HV中的每一节点均可达终止节点。

• 简单节点：代表一个没有子任务的任务。

• 复合节点：包含着子任务的节点，每个复合节点对应着一个子图HTG(x)=(HV(x),HE(y))。

• 循环节点：代表一个循环任务，同复合节点类似，它也对应着一个子图。

HTG中弧的构造，是以程序的控制流图为基础的，文[6]给出了构造弧的算法。

在HTG中，节点x的执行条件是：①迫使x点得到执行的控制条件得到满足。②若$y\delta_d x$，则或者y已完成执行，或者已知y不会被执行。在HTG中，执行条件得到满足的各节点均可并行执行。由于HTG图中包含了控制流和数据流信息，可以方便地进行控制依赖和数据依赖的分析，挖掘任务并行性。

4 程序的变换与优化

为了消除或削弱依赖关系，开发和增强并行性，在保证程序语义不变的前提下，需要对程序做一些变换与优化。目前这方面的工作主要集中在对循环的变换上。

(1)标量和数组的私有化(privatization) 是指在每一个处理节点上，都有一份标量或数组的拷贝，私有化可以消除循环携带依赖，减少通讯量。

(2)循环剥离(loop peeling) 将若干个迭代从循环体的首部或尾部移走，目的是可以消除依赖或便于循环合并。

(3)循环交换(loop interchange) 改变紧嵌套循环中两个循环的位置，例如把可以并行化的循环交换到最外层，增加可并行迭代体的粒度。循环变换并不总是合法的，可交换的条件为：当一个深度为d的紧嵌套循环$V=(v_1,v_2,\cdots,v_p,\cdots,v_q,\cdots,v_d)$，循环p，q交换之后，依赖向量为$V'=(v_1,v_2,\cdots,v_q,\cdots,v_p,\cdots,v_d)$只要V'按字典序为正，即其第一个非0分

量不为负，交换合法。

(4)循环反转(loop reversal) 改变循环进行的方向，经常和别的迭代空间变换技术相结合，因为它改变了依赖向量。

(5)条块分割(strip mining) 可以用来调整并行操作的粒度。

(6)循环收割(loop shrinking) 当一个循环当中存在着依赖，阻碍了迭代的并行执行，只要依赖距离大于1，依然可以开发出一定程度的并行性。考虑一个紧嵌套循环，假设数组的依赖距离均为常数，且最小依赖距离为k，可以k为步长，将迭代空间划分成若干块，每块内部可并行执行，块与块之间可以串行执行。

(7)循环分布(loop distribution) 将一个单一的循环分成若干个子循环，每个子循环有和原循环相同的迭代空间，它的循环体是原循环体的一个子集。循环分布可用来产生紧嵌套循环，所产生的子循环往往具有更少的依赖性。循环分布不能对循环体中的语句作任意划分，分布是有条件的，其算法比较复杂。

其它诸如循环合并(loop fusion)，循环归一(loop coalescing)等变换对程序的并行性都有一定的意义，这里不再一一介绍。

结束语 设计一个OO并行编程系统，本身是一件很复杂的工作，一般应该使它达到四个目标：高效率、易使用、移植方便以及广泛的应用适应性。

为了实现这些目标，有很多问题需要进一步研究，例如：(1)如何确定用户和系统之间的任务界面；(2)OO继承机制以及对象作为参数对并行性分析的影响；(3)如何利于并行性识别设计OO程序中间形式描述；(4)任务的粒度控制和优化问题，包括权衡通讯与并行；(5)通讯和同步的优化问题；(6)移植问题，不仅要考虑代码的移植，还应考虑性能的移植问题；(7)在开发数据并行性时，如何解决数据的自动分布问题。

(下转第13页)

(上接第63页)

除了吞吐量的提高，PF_ATM协议栈还有其它一些优点。它为建立点到多点连接提供了一个简单接口，这个特征对电视会议系统等应用来说是诱人的。基于Socket的接口还使得应用进程能够在一个连接的生存期内改变QOS参数，改变流控约定等等，这对实现交互式的动态电视会议系统也是十分有用的。

结束语 为了充分利用ATM的高带宽、服务质量约定等特性，我们提出了一个新的ATM专用协议栈，给出了该协议栈的语义描述，并通过模拟一个视频会议系统验证了该协议栈的优越性。实验模拟结果表明，这种新的协议结构有效地利用了ATM的优良特性，尤其适合多媒体数据传输应用。

鸣谢 本文的写作和实验模拟得到上海邮电管理局科技处的大力协助，在此表示感谢。

参考文献

[1] M. Laubach, Classical IP and ARP over ATM, Internet RFC1577, Jan. 1994

[2] B. Manning and R. Colella, DNS NSAP Resource Records, Internet RFC1706, Oct. 1994

[3] ATM Forum LANE_SWG, "LAN Emulation Over ATM Specification" V1.0, Jan, 1995

[4] 4.4BSD Programmer's Reference Manual, Apr. 1994

[5] D. Saha, et al., A Video-conferencing tested on ATM: Design, Implementation, and Optimizations, Proc. IEEE Conf. on Multimedia Computing and Systems, Apr. 1995

[6] R. Black and S. Crosby, Experience and Results from the Implementation of an ATM Socket Family, Technical Report of Cambridge University Computer Laboratory, Oct. 1993

[7] 方震、龙浩、吴时霖，ATM局域网仿真体系结构的研究，《小型微型计算机系统》

[8] 方震、龙浩、吴时霖，ATM网络的流量控制，《计算机科学》，№.3，1997