计算机科学1996 Vol. 23 №. 2

并行面向对象语言的研究与比较

张 宁 温冬婵
（清华大学计算机系 北京 100084）

摘 要 Concurrent programming is much more difficult than its sequential counterpart. Concurrent Object-Oriented Language (COOL) is widely believed to be able to reduce the difficulties of concurrent programming. In this paper, some most influential COOLs are introduced, analysed and compared.

关键词 Concurrent object-oriented language, Concurrent programming, Object-oriented programming, Programming language.

一、引 言

随着并行多机系统和网络分布计算平台日益流行，并发/并行程序设计技术得到广泛应用。并行程序设计远较顺序程序设计复杂，应用程序必须处理诸如任务和数据划分、进程通讯、同步和调度诸多繁琐细节，并且并行程序往往与硬件和操作系统密切相关，其设计极其困难，所开发出来的程序难调试、难维护、难移植。软件已成为并行系统应用发展的瓶颈，开发简单、高效、易用、好移植的并行程序设计语言，已成为当务之急。

对象模型具有天然的并行性，非常适合于描述并行系统；而且面向对象技术在软件工程上的巨大优势，如封装性、重用性和多态性等，也能大大降低并行程序设计的难度，提高并行程序的可维护性和可移植性。这种思想激发了对并行面向对象语言（简称COOL）的大量研究，并产生了许多并行面向对象语言。开发并行面向对象语言通常有以下三种途径：

1. 基于并行对象模型，研制全新的并行面向对象语言。大多数早期的并行面向对象语言都采用了这种方式，这类新语言以一种自然高效的方式支持并行面向对象程序设计。

2. 扩充已有的面向对象语言，使其支持并行面向对象程序设计。大多数这类语言使用下列技术的组合，在对象机制基础上引入并行机制：(1)所有的并行类都是某一系统并行类的子类，如 Eiffel 和 PRESTO。(2)引入新的关键字，修正符（Modifier）或编译指示（Director）来描述并行类，如 uC++ 和 CEiffel。(3)扩充原语言的语法和语义，以支持某种通用并行对象模型，如 ACT++ 和 Actalk。

这类语言根据其对宿主语言扩充的程度，又可分为两类：(a)扩充原语言的语法和语义，引入新的编译器和预编译器。(b)不修改语言本身，用类库的形式提供某种并行对象机制。这类扩充语言的宿主语言主要是 C++，Smalltalk，Eiffel 和 Actors。

二、关键技术分析与比较

1 进程管理

并行面向对象语言，以一种面向对象的方式，解决并行程序设计的三个基本问题：并行进程的描述、进程通信机制和进程同步机制。我们区分进程和对象两个概念：进程调用对象的方法。

1.1 **进程创建**。并行面向对象语言的进程创建有两种方式：显式和隐式。在显式方式中，进程与对象相互独立，语言提供显式进程创建机制，这通常是一些预定义的线程（Thread）或根对象（Root）类。用这种方式很容易在已有的 OO 语言上增加并行机制，而不必修改语言本身，如 PRESTO。在隐式方式中，进程和对象结合在一起，对象引用自动创建并行进程。这种方式更加自然灵活，但语言实现复杂，并且要处理并行和对象机制的冲突，即同步代码不可重用，这称为继承异常（Inheritance Anomaly），大多数并行 OO 语言采用隐式进程创建。

1.2 **进程终止**。进程可以显式或隐式终止。在隐式终止方式中，对象为每个收到的消息创建一个

进程，消息处理完后进程消亡，Actor 是这种方式的典型代表。隐式进程终止增大了并行性，减轻了程序员负担，但增加了进程创建/消亡开销，降低了程序的执行效率。

在显式终止方式中，进程在处理完一个消息后接着处理下一个消息。这减少了进程创建和消亡开销，但程序员必须显式终止进程。ABCL/1采用这种方式。

1.3 **进程激活**。进程可以在创建时激活，也可以在收到消息后才激活。第一种方式允许进程在没有消息时运行，这增加了活动并行线程，但会浪费系统资源，且难于实现。POOL-T 采用这种方式。

大多数并行 OO 语言采用第二种方式，即有消息时才激活进程。这种方式会增加系统运行开销。Actors 和 Concurrent Smalltalk 采用这种方式。

1.4 **进程粒度**。不同的并行粒度适用于不同的硬件配置和通讯开销。细粒度并行通常允许一个对象中有多个活动线程，这增大了并行性，但也增加了通信和同步开销；粗粒度并行通常对应于一个对象只有一个活动线程，或多个对象共享一个线程。粗粒度并行适合于通信开销较大的网络系统。许多语言支持多种粒度并行。Actors 允许细粒度并行，ABCL/1支持中粒度并行，Argus 采用粗粒度并行。

表1 并行 O-O 语言的进程管理技术

进程管理			
语言	创建 终止	激 活	粒度
ABCL/1	隐式 回答后继续	消息接收时	中
ABCL/R	隐式 回答后继续	消息接收时	中
Actor	隐式 回答时	消息接收时	细
Argus	显式 回答时终止	保护程序的入口调用	粗
Concurrent Smalltalk	隐式 回答后继续	消息接收时	中
COOL	隐式 回答时终止	唤醒并行函数	中到大
Eiffel	显式 回答时终止	唤醒进程对象	中
Emerald	隐式 回答后继续	创建时	粗
Gnu C++	隐式 回答后继续	创建时	粗
Hybrid	显式 回答后继续	对象与线程创建时	粗
Nexus	显式 回答时终止	消息接收时	粗
Parmacs	静态 回答后继续	程序开始时	中
POOL-T	隐式 回答后继续	创建时	中
Presto	显式 回答时终止	线程对象创建时	大

2 通信与同步

2.1 **消息类型**。在并行面向对象语言中，通信通过对象间的消息传递完成，有三种类型的通信方式：同步、异步和未来(Future)。

(1)同步通信方式类似于远程过程调用，最易实现。但由于消息发送者和接收者必须等待会聚(Rendezvous)，因而会浪费系统资源。同步通信系统可靠，易于验证，POOL-T 就采用了这种方式。

(2)异步通信方式消除了等待会聚，消息发送者不必等待结果返回就继续执行，因而增加了并行性。但异步通信方式可靠性差，难于调试和验证。此外，系统必须提供某种显式同步机制，以确保程序的正确性。

(3)未来通信方式介于同步与异步通信方式之间。消息发送总是立即返回，不必等待，因而增加了并行性。但返回的不是消息结果，而是一个未来变量(Future Variable)，用以保存消息返回值。当发送者需要消息返回结果时，则进入与接收者的同步点。系统自动检测未来变量中是否已存有消息返回结果。如果有，则继续执行；否则发送者阻塞，直至结果返回。未来通信方式增加了系统执行开销。

2.2 **消息接收**。对象可以显式或隐式地接收消息。隐式方式表示系统自动进行消息接收，因而程序员不必也不能干预消息接收的次序，不能实现消息优先级，显式方式需要程序员干预消息接收和处理过程，可实现消息优先级，因而更灵活。但显式消息接收增加了程序员负担，并降低了系统运行效率。

2.3 **同步**。正确的同步将使并行进程相互协作，产生所期望的结果。过多的同步会降低系统并行性并增加了运行开销，过少的同步则会使程序具有不确定性，难以验证和调试。

一个对象在任何时候能够响应哪些消息，是由对象当时的状态决定的，这称之为同步约束。在继承时，子类增加新的方式或修改父类方法都将破坏父类方法间的同步约束，从而使父类方法难以使用。这破坏了面向对象程序设计的最大优点：封装和继承。这种继承与同步的冲突，是并行面向对象语言设计与实现中最难解决的问题。目前对此问题尚无好的广泛接受的方法。解决继承异常的基本原则是：同步代码与方法代码分离(Seperation)和同步约束分散到各个方法(Localization)。

常用的方法有三种：同步函数，卫士方法和方法集。

(1)同步函数，对象中有一个表达对象方法间同步约束关系的同步函数，如 Eiffel 中的 live() 函数，

子类继承父类方法时，必须重写同步函数。

(2)卫士方法。每个方法都有一个卫士命令(Guard)，只有在卫士命令得到满足时才可以引用方法。

(3)方法集。定义对象在每个状态下可以响应的方法集和对象的状态转换关系。在某一状态下，只有对应方法集中的方法可以被调用。典型的例子是Actors，对象每执行完一个方法后用become语句指定下一个可以响应的消息集。

上述方法，没有一种可以完全解决继承异常问题。当前一种流行的做法是混合使用上述三种方法，并研究同步代码的继承机制。

表2 并行O-O语言的进程通讯与同步

语言	通信特征			
	消息类型			接受
	同步	异步	未来	
ABCL/1	X	X	X	隐式
ABCL/R	X	X		隐式
Actor	X	X	X	隐式
Argus	X	X	X	显式
Concurrent Smalltalk	X	X	X	隐式
COOL	X	X	X	N/A
Eiffel	X	X	X	显式
Emerald		X		隐式
Gnu C++	X	X	X	隐式
HybrID	X			隐式
Nexus	X	X	X	显式
Parmacs	X	X	X	隐式
POOL-T	X			隐式
Presto	X	X		N/A

三、几种并行面向对象语言简介

1 **Actors**。不是一个语言，而是一个并行面向对象计算模型。Actors中的基本单位是Actor。Actor可以动态创生。每个Actor都有自己的语义集。Actor由两部分组成：一个信箱和一个行为。每个信箱与一个消息队列相联系(In-coming message queue)。每当有消息时，Actor执行一个脚本(Script)。如果脚本能识别该消息，就接收它，否则拒绝它。Actor在接收消息后，可以：1)向一个熟人(Acquaintances)，即本Actor知道信箱的Actor，发送消息；2)向自己发送消息，这通常是创建自己的一个拷贝；3)创建一个Actor以处理该消息。

脚本还使用Become原语指定一个替代行为(Replacement)，即一个新的Actor，它拥有同样的信箱名，能够接收下一条消息。Actor可以通过在响应新收到的消息之前就指定替代行为来增加并行性，未来变量被看作Actor，当需要其值时，就向它发送一个求值消息。

Actor将它不能响应的消息委托给一个叫作代理(Proxy)的Actor处理。它通常含有响应此消息的特殊信息，包括例外处理机制。继续Actor(Continuation Actor)处理同步函数调用时将阻塞，直至所期望的同步事件出现。

2 **Presto**。由华盛顿大学开发，是对C++的扩充，用C++类如thread线程对象和同步对象，支持并行程序设计，PRESTO适用于共享存储系统。

线程对象包含一个程序记数器和一个执行栈。线程对象可以动态创建，激活和联合(Join)。为降低线程创建的开销，PRESTO线程可重复使用。运行支持系统创建一个调度线程scheduler以管理可重用线程，每个物理处理器由一个processor对象表征，它从scheduler对象处获得可执行线程。消息发送对象以同步或异步方式调用操作，对象实现者决定对象以顺序或并行方式响应消息。

PRESTO提供了Spin Lock和Blocking Lock，以及管程(Monitor)、条件变量等对象用于同步。PRESTO不支持方法重载，不进行参数类型检查，方法调用也不返回结果。

3 **POOL-T**(Parallel Object-Oriented Language-T)。对象在创建时激活，每个对象中都有一个自主活动线程。对象包括说明和实现两部分。一个根单元(Root Unit)，充作主程序和程序执行起点。POOL-T是一个纯OO语言，认为所有数据都是对象，这简化了POOL-T实现，但降低了它的执行效率。

POOL-T的消息传递是同步的，点到点的。消息显式接收。消息按到达的顺序存储在一个队列中，每当对象执行answer语句时，它响应消息队列中名字出现在answer语句中的第一条消息。Answer语句与Ada的Select语句类似。在完成回答后，接收者可能执行某个后处理代码段(Post Processing Section)。

4 **Concurrent Smalltalk**。既支持标准Smalltalk方法调用，也支持异步方法调用。标准Smalltalk方法调用是同步的，用原子对象(Atomic Object)实现。异步方法调用用同步方法调用和

CBox 对象实现。异步方法调用总是立即返回一个 CBox 对象。当需要返回结果时，发送者向 CBox 对象发送求值消息获得返回值。如果结果尚未返回，则发送者等待。

5 COOL，由斯坦福大学开发，是对 C++ 语言的扩充，允许在共享存储环境中进行中粗粒度并行。COOL 允许三层并行性：对象内并行（每个方法都可以异步执行）、对象间并行（不同对象的方法可并行执行）和方法内并行（同一方法内的不同函数调用可并行执行）。

对象的方法既可以是异步的，又可以是互斥的。互斥方法可以实现函数级同步，并且比 monitor 更灵活，允许更大的并行性。

COOL 提供未来变量、Spin Lock 和 Blocking Lock 以实现细粒度同步。和 POOL-T 一样，COOL 不支持继承、友元函数（Friend Function），虚函数（Virtual Function）和方法重载。

6 ABCL/1（Actor Based Concurrent Language），基于 Actors 模型，对象执行脚本（Script）。脚本定义了对象行为、可响应的消息集和响应方式。独立对象可以并行执行。但在对象内部，消息被顺序处理。对象在创建时处于休眠状态（Dorment），直到被消息唤醒。对象可以用一个脚本模式选择所响应的消息，并把当前不能响应的消息放在消息队列尾，供以后处理。处理完消息后，对象执行一个 Select 结构，回到休眠状态。

通常消息传递是异步的，点到点的。消息中含有消息标志、参数、发送者名和返回地址。对象有两个消息队列：普通消息和快件消息（Express Messege）队列。快件消息可以中断普通消息处理。

ABCL/1支持三种消息传递：过去型（Past），现在型（Now）和未来型（Future）。过去型和未来型消息传递是异步的，现在型是同步的。ABCL/1也支持广播通信。

四、结论

对并行面向对象语言的研究已进行了很多年，并产生了大量的新语言，但还没有一个得到真正广泛的应用。究其主要原因是对象机制与并行机制的结合，远比想象的要困难。根据 Bertrand Meyer 对纯面向对象语言的定义：(1)模块结构；(2)数据抽象（对象是抽象数据类型的实现）；(3)自动存储管理（系统自动回收无用对象）；(4)类（所有的非简单类型都是类）；(5)继承；(6)多态和动态联编；(7)多重继承和重复继承。目前没有一种并行面向对象语言满足所有这七条要求。并行面向对象语言距离实用化，还有很长的一段路要走。（参考文献共15篇略）

（上接第79页）

五、结束语

虽然 CORBA1.1 文档定义了足够多的 IDL 语言和 API，但它并未考虑从 ORB 到 ORB 的通信标准。例如在 DME 中只能用 DCE 的 RPC 完成 ORB 之间的通信。为了弥补这一不足，OMG 最近正致力于开发（到1994年初）一个向上兼容的2.0修改版本，以期实现从 ORB 到 ORB 的通信标准。这样，任何两个 CORBA 的 ORB 实现，即便是独立实现的，仍可保证有互操作性，而且可共享名字域。其它问题，象本地事务处理、保密协议、服务质量控制和多重预测扩充，也将是 ORB 下一版本所追求的目标。象任何一种技术一样，CORBA 将没有止境。

参考文献

[1]Richard Mark Soley, Role of Object Technology in Distributed Systems, OMG, Inc. 1994

[2]David Chappell, The OSF Distributed Management Enviroment, Chappell and Associates, 1994