第8卷第6期 软件学报 Vol. 8, No. 6
1997年6月 JOURNAL OF SOFTWARE Jun. 1997

CCPP：一个并发C++语言的设计与实现*

温冬婵 王鼎兴 张 宁

（清华大学计算机系 北京 100084）

摘要 并发面向对象语言COOL(concurrent object-oriented language)是一种有效的并发程序设计语言.本文提出了一种并发对象模型(Concurrent C++, CCPP)及其语言.在CCPP并发对象模型中,所有对象都是并发对象,对象间的通讯都采用异步消息发送方式.对象间的同步采用"需要时等待"策略,同一对象内并发线程间的同步用路径表达式描述.CCPP语言是对C++语言的扩充,采用预编译方法实现.CCPP语言允许并发/顺序代码重用.

关键词 并发面向对象语言,并发程序设计,C++.

中图分类号 TP312CC

目前,基于共享存储的并行多机系统已经商业化、实用化.这类系统往往具有很高的峰值计算速度,提供了巨大的并行计算潜力.然而,这类系统的计算能力往往得不到充分利用,其主要原因是软件技术——并发程序设计技术远远落后于硬件的发展.并发程序设计技术具有描述并发系统的强大能力,可开发硬件系统的并行性,获得良好的加速比,在诸如操作系统、数据库、用户界面等诸多领域取得了广泛的成功.然而,并发程序设计在顺序程序设计的基础上又增加了一个复杂度,程序必须处理并发进程的创建、同步、通信等诸多细节,使并发程序设计极为困难,开发出来的程序难以验证、修改.不同的并发系统往往使用完全不同的并发机制,使得为一个系统开发的程序很难移植到另一个系统上.软件已成为并行多机系统应用及发展的瓶颈.

为了降低并发程序设计的难度,研究人员提出了各种并发程序设计模型,如函数型、逻辑型、过程型和面向对象型.多年的研究与实践证明,面向对象技术可以大大降低并发程序设计的难度.在面向对象模型中,对象是封装了数据和行为的实体,该实体可以是主动的,也可以是被动的,还可以兼而有之,消息传递是对象间相互作用的唯一方式.这种模型具有天然的并发性.面向对象技术的基本概念——对象和并行程序设计的基本概念——进程在一定意义上具有相似性:都有访问受限的局部数据;都有封装的行为;都以消息传递(Message Passing)方式相互通讯.这种相似性,促使研究人员把并发程序设计技术与面向对象技术结

---

* 本文研究得到国家自然科学基金资助.作者温冬婵,女,1946年生,副教授,主要研究领域为逻辑程序设计,面向对象语言以及并行处理技术.王鼎兴,1937年生,教授,博士导师,系主任,主要研究领域为计算机并行,分布处理技术,面向人工智能的体系结构,并行编译技术.张宁,1970年生,硕士生,主要研究领域为并行/分布处理技术,面向对象语言及并行编译技术.

本文通讯联系人:温冬婵,北京 100084,清华大学计算机系

本文1996-05-29收到修改稿

合起来,形成所谓的并发面向对象程序设计(Concurrent Object-Oriented Programming)技术:它既有面向对象技术在软件工程上的巨大优势,如代码重用,又有并发程序设计技术能开发系统并行性的优点.近年来,并发面向对象技术的研究取得了巨大进展,涌现出了一大批并发面向对象语言COOL(concurrent object-oriented language),如Concurrent Smalltalk[1], Distributed Eiffel[2], CEiffel[3], Concurrent C++[4], uC++[5], PCC[6], CooC[7], ABCL[8], ACT++[9], Presto[10]等.有关并发面向对象语言研究的历史与现状见文献[11~17]等.

然而,虽然并发面向对象语言的研究取得了巨大进展,并发OO语言却并没有得到广泛应用,其主要原因是:

(1) 对象机制与并发机制相互独立

在很多并发OO语言中,对象机制与并发机制相互独立.语言在OO机制之外再构造一个独立的并发机制,模仿传统的并发程序设计模式,提供某种较低级的并发程序设计手段.由于对象与并发相互独立,并发程序设计的难度并未降低.这类语言严格说来不能称为并发面向对象语言.

(2) 继承与并发相互冲突

对象与并发的结合并不象初看起来那么简单.一个严重的问题就是并发、同步和继承的冲突,即所谓的"继承冲突"(Inheritance Anomaly 等).[18,19]并发对象在收到多个消息时,响应哪些消息是由该对象的状态决定的.这种约束称为并发对象的同步约束,并发对象内表示这种同步约束关系的代码称为同步代码(Synchronization Code).当子类继承父类代码时,子类增加新的方法或修改父类原有的方法都将破坏父类方法间的这种同步约束关系,从而使子类对父类代码的继承非常困难.子类需了解父类的实现细节,并可能要改写大部分的父类方法.这破坏了面向对象方法的最大优点:信息隐藏与代码重用.解决继承异常的基本原则是把对象的同步代码和方法代码分开,分别继承.目前对继承异常尚无好的解决方法.

(3) 顺序/并发代码不能相互重用

虽然顺序代码与并发代码在语法和语义上都极为相似,然而顺序代码却不能在并发环境中执行,并发代码也不能被顺序解释.

本文提出了一种并发对象模型及其语言CCPP的设计与实现,渴望能很好地解决上述问题,并进一步促进对并发OO技术的理解.

1 CCPP并发对象模型与语言

1.1 CCPP并发对象模型

对象是封装了数据和代码的自主实体,对象间通过消息传递相互作用.因而对象具有天然的并发性.在CCPP语言中,我们不引入新的并发概念,而是赋予对象一种并发语义.在CCPP模型中,所有的对象都是并发对象.多个并发对象可以同时运行,同一对象内亦可有多个并发线程.对象间的消息传递都采用异步消息发送方式和"需要时等待"(Wait-When-Necessary)的同步策略,即消息发送对象(Client Object)不必等待消息返回即可继续执行;当需要访问此消息结果时,系统自动检测结果是否已返回.如果尚未返回,则执行线程阻塞.

CCPP 模型允许对象间和对象内两级并行:异步消息发送引发并发线程,允许一个对象内有多并发线程则降低了并发粒度,"需要时等待"的同步策略则进一步增大了并发计算的可能性.

在 CCPP 并发对象模型中,我们不区分其它并发面向对象语言中常出现的"顺序对象"(即C++对象)和"并发对象"(Concurrent Object),"主动对象"(Active Object)和"被动对象"(Passive Object),"同步对象"(Synchronous Object)和"异步对象"(Asynchronous Object),以及"原子对象"(Atomic Object)等概念,所有的CCPP对象都是并发、异步和被动的.通过设置不同的path和thread参数(见下文),CCPP对象也可成为上述任何一种对象.

在CCPP并发对象模型中,对象由界面(interface)、实现(implementation)和并发线程(concurrent thread)3部分组成,其中的实现部分又分为数据域、方法和同步约束3部分,如图1所示.

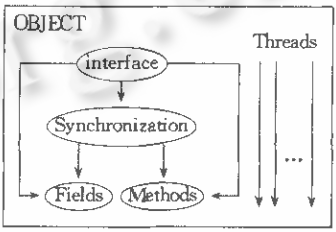

图1 CCPP并发对象

对象是CCPP程序的基本构筑单位.CCPP程序由对象构成,程序的执行由向一个根对象(root object)发送某一消息开始.对象在响应消息时创建新的并发对象并向它们发送异步消息,从而引起一连串的并发任务,如图2所示.

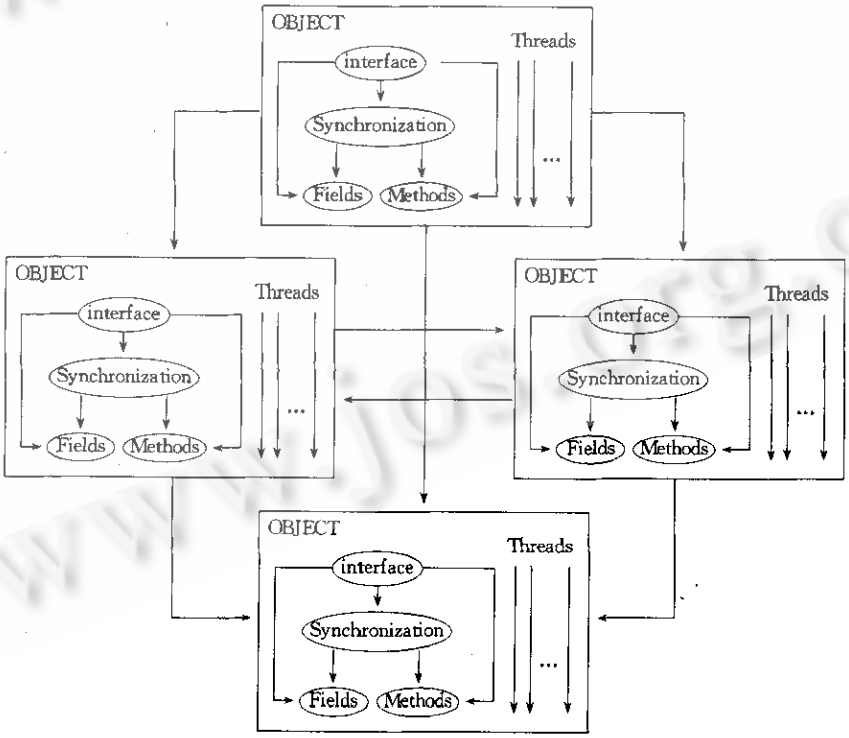

图2 CCPP程序模式

CCPP程序中无全局变量.除了main()函数及其局部变量外,所有的变量都是某一对象的数据域,所有的函数都是某一对象的方法.为了让已开发的C++程序通过CCPP编译,可为每一个C++程序构造一个Program对象,令所有的全局变量为该对象的public

数据域,所有的全局函数为该对象的public方法.程序的执行由向program对象发送main消息开始.

1.2 CCPP语言设计原则

CCPP语言是对C++语言的扩充.所以选择C++语言,是因为(1) C++语言是目前应用最广泛的面向对象语言;(2) 绝大多数并行系统都支持C/C++语言;(3) 我们在前一阶段的研究工作中已完成一个C++编译器,熟悉C++语言的编译技术.

在设计CCPP语言时,我们遵循以下原则:(1) 对语言的扩充应尽可能的小;(2) 语言概念应清晰统一、简单明了;(3) 应能实现串/并代码重用.

基于上述模型和原则,CCPP语言对C++语言作了如下扩充:

(1) 引入path和thread 2个关键字,用以描述并发对象的同步约束关系和对象内所允许的最大并发线程个数.

(2) 引入CONCURRENCY和RUNTIMESYSTEM 2个系统类,用于CCPP语言的实现和扩展.

1.3 CCPP语言

CCPP语言基于CCPP并发对象模型,以对象作为程序的基本构筑单位,无全局变量.CCPP语言与C++语言在语法上的唯一区别在于在类定义中引入了path和thread说明.

1.3.1 Path说明

Path语句用路径表达式(Path Expression[20~22])描述并发对象内并发线程间的同步约束关系,其语法为:

path语句 ::= path 路径表达式
路径表达式 = 域名 | 方法名
路径表达式 = 路径表达式, 路径表达式
路径表达式 = 路径表达式; 路径表达式
路径表达式 = n :(路径表达式)
路径表达式 = [路径表达式]

路径表达式由类的域名,方法名和路径操作符组成.令A,B为类的域名或方法名,则

A,B 表示A、B可并发执行;
A;B 表示A、B必须顺序执行;
n:(A) 表示最多可有n个A并发执行;
[A] 表示可以有任意多个A并发执行;

Path说明是CCPP类定义的可选项.一个类中可以有多个path语句,以任意次序出现.但每个path语句必须在一行内结束.凡未在path语句中出现的方法,其执行不受任何限制,即可以和任意多个其它方法并发执行.一个没有path说明的类是一个完全并发类,即对其方法和域的引用不受任何限制.这样,一个C++类在CCPP中即为一个完全并发类.Path语句可以为子类继承和修改.

在实现上,path语句采用C++注释形式,如下所示:

```
#ifdef_CCPP
```

```
#define path path
#else
#define path //
#endif
```

由于 path 语句的缺省含义和实现方法,C++类可以由 CCPP 编译器编译,CCPP 类也可以由 C++编译器编译.

1.3.2 Thread 语句

Thread 语句定义一个并发对象内最大允许的并发线程个数. Thread 语句的语法为

Thread 语句 = thread 整数

一个消息是否为某个并发对象所响应,取决于 2 个条件:

(1) 该消息是否满足 path 语句所表述的同步约束关系;

(2) 对象是否有空闲线程执行该请求.

对于满足上述条件的消息,并发对象创建一个并发线程执行该响应消息.

Thread 语句是 CCPP 类定义的可选项,缺省时 thread 数为 0. 当 thread 数为 0 时,该对象就退化成为一个顺序对象:对象是被动的代码和数据的封装体,对象内无执行线程,消息发送对象必须用自己的执行线程,按过程调用的方式执行该请求. 当 thread 数为 1 时,对象就成为所谓的原子对象:对象内只有 1 个执行线程,1 次响应 1 个消息. 当 thread 数很大时,对象为每一个异步消息创建 1 个并发线程. 增大 thread 数,即可增大对象的并发度,使对象可以同时响应更多的消息,获得更多的执行机会. 合理分配各对象的 thread 数,可以调整系统的负载,达到最佳的加速比. Thread 语句可以为子类继承和修改. 在多重继承时,thread 数最大的父类 thread 说明为子类的 thread 说明. Thread 语句的实现技术与 path 语句相同,都采用 C++注释形式. 这样就可实现 C++代码与 CCPP 代码的相互重用.

1.4 一个例子

作为一个简单的例子,考虑下述读/写者问题:

多个线程并发访问一个共享数据,允许同时读,但必须互斥写,其 CCPP 实现如下:

```
Class Buffer {
    private
        DATA data;
    public
        DATA read( ) { return data;}
        void write(DATA item) { data=item; }
    path 1:([read],write)
    thread 5
}
```

这只是一个简单的例子,程序中所给出的算法可能导致写者永远等待的情况.

2 CCPP 语言实现技术

CCPP 语言的实现采用预编译技术,一个 CCPP 程序先通过 CCPP 预编译器生成C++

代码，再由C++编译器生成可执行代码，如图3所示.

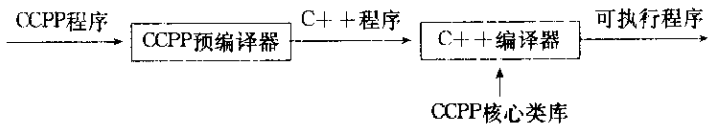

图3 CCPP编译流程

下面分别介绍CCPP预编译的几项关键技术.

2.1 CONCURRENCY和RUNTIMESYSTEM类

CONCURRENCY和RUNTIMESYSTEM类的引入是为了CCPP语言的实现与扩展. 在CCPP并发对象模型中，每个对象都要维护3个队列：消息队列MessageQueue、返回队列ReplyQueue和空闲线程队列IdleThreadQueue. 对象将收到的消息挂在消息队列上，将消息返回的结果挂在消息发送对象的返回队列上，将所有的空闲并发线程都挂在空闲线程队列上，供重复使用，以节省线程创建、同步和消亡的开销. 所有这些队列及其操作都在CONCURRENCY类中定义. CCPP编译器认为所有类(除CONCURRENCY类外)都是CONCURRENCY类的子类，自动在所有类定义的继承部分插入"virtual CONCURRENCY'. CONCURRENCY类中还定义了一个静态RUNTIMESYSTEM类指针，指向在系统启动时创建的一个RUNTIMESYSTEM对象. 用户不能直接调用RUNTIMESYSTEM对象的方法. RUNTIMESYSTEM对象的引入是为了CCPP语言的实现、移植和扩展，例如：

(1) 可以用RUNTIMESYSTEM对象隐藏与系统密切相关的实现细节，以提高语言的可移植性.

(2) 在分布式系统中，可由RUNTIMESYSTEM对象维护一个系统对象表，建立一个类似于分布共享虚存(DSM)的分布共享虚拟对象空间，从而使CCPP语言支持分布存储的并行多机系统和网络平台.

(3) 可以构造不同的类以实现不同的负载平衡、对象迁移和处理器调度算法，通过让RUNTIMESYSTEM类继承这些不同的类以适应不同的算法.

2.2 异步消息发送(Asyncronous Message Sending)

在CCPP模型中，对象间的消息传递是通过线程实现的，它们的执行是异步的. 对象对自己的数据域和方法的引用则采用过程调用的方式，并且不受同步约束关系的限制.

CCPP对某一client对象的某一方法中的形如server.method( )的语句作如下处理：

(1) 如果client对象使用此消息发送的返回值，即将server.method( )的返回值赋予某一变量或作为实参用于某一方法调用，则生成一个消息返回节点，挂在client对象的消息返回队列上.

(2) 如果server对象的空闲线程队列中有空闲线程，并且method消息满足server对象的path语句所表述的同步约束关系，则从server对象的空闲线程队列中取一空闲线程执行对应的method代码；否则，将该消息挂在server对象的消息队列上.

(3) 在client对象中使用此消息返回值的地方，插入代码检测结果是否已返回. 如果已返回，则继续执行. 否则，如果此时method消息满足server对象的path语句所表述的同步约束关系，则用client对象的当前执行线程执行相应的method方法代码，并除去client对

象消息返回队列的相应节点. 否则,执行线程阻塞.

如果几个消息发送的返回结果被赋予同一变量,则该变量中保存最后一次消息发送的返回值,执行线程的同步也依据最后一次消息发送的结果. 先前的消息发送并不中止,只是其结果不再返回.

2.3 消息返回

server 对象在完成某一 client 对象的形如 server. method( )的消息请求后,作如下处理:

(1) 如果此消息需要返回,则将其结果送回 client 对象的消息返回队列的相应节点. 如果 client 对象此时有线程因等待此结果而阻塞,则激活该线程.

(2) 如果 server 对象的消息队列有可执行的消息,则用本线程执行该消息. 否则,把本线程挂在 server 对象的空闲线程队列上.

3 结　论

在 CCPP 模型中,所有对象都是并发对象,对象间的消息传递都采用异步消息发送方式. 并发对象间的同步采用"需要时等待"策略,同一对象内并发线程间的同步用路径表达式描述. 系统提供对象间和对象内 2 级并行,并采用"惰性计算"(lazy computation)模型进一步扩大并发计算的可能性. 对象本身是对程序数据和功能的自然划分,是相对独立的并发执行单位. 对象间的消息传递以一种简单的方式隐藏了并发任务的创建、同步和消亡等大量繁琐的细节,大大降低了并发程序设计的难度. CCPP 模型避免了其它并发面向对象语言模型中通常出现的庞杂的概念和概念间的冲突,实现了顺序/并发代码的相互重用,是一种简单、高效,易移植的并发面向对象程序设计语言模型.

目前,在由多台相同工作站通过网络构成的机群系统上,CCPP 语言已完成设计. 进一步的工作包括实现优化的 CCPP 预编译器以提高 CCPP 程序的执行效率;扩充 RUNTIMESYSTEM类以使CCPP 语言支持分布式系统和网络平台.

参考文献

1 Yokote Y, Tokoro M. The design and implementation of concurrent smalltalk. In: OOPSLA'86 Proceedsings, ACM, New York, 1986. 331～340.

2 Gunaseelan L, LeBlanc R J. Distributed eiffel: a language for programming multi-granular distributed objects. In: Proceedings of the Fourth International Conference on Computer Languages, IEEE, 1992.

3 Lohr K P. Concurrency annotations for reusable software. Commun. of ACM, Sept. 1993, **36(9)**: 80～89.

4 Gehani N H, Roome W D. Concurrent C++: concurrent programming with class(es). S-P&E16, 12, Dec. 1988.

5 Buhr P A, Glen Ditchfield R A Stroobosscher, Younger B M. uC++: concurrency in the object-oriented language C++. Software-Practice and Experience, Feb. 1992, **22(2)**: 137～173.

6 清华大学计算机系. 面向对象语言的并行编译技术. 技术报告(No: 863-306-02-05a), 1994.

7 Trehan R, Sawashima N, Morishita A *et al*. Concurrent object oriented 'C' (CooC). ACM SIGPLAN Not, Feb. 1993, **28(2)**: 45～51.

8 Yonezawa A. ABCL: an object-oriented concurrent system. MIT Press, 1990.

9 Kafura D, Lee K. ACT++: building a concurrent C++ with ACTORS. JOOP, May/June, 1990. 47～55.

10 Bershad B N, Lazowska E D, Levy H M. PRESTO: a system for object-oriented parallel programming.

Software-Practice and Expreerience, 1988,18(8):713~723.

11 Yonezawa A, Tokoro M. Object-oriented concurrent programming. MIT Press, 1987.

12 Agha G, Wegnber P, Yonezawa A. In research directions in object-based concurrency. MIT Press, 1993.

13 Agha G A, Hewitt C, Wegner P *et al*. Proceedings of OOPSLA/ECOOP'90 workshop on object-based concurrent programming. ACM OOPS Messenger 2, 2, April 1991.

14 Agha G A, Wegner P and Yonezawa A. Proceedings of ACM sigplan workshop on object-based concurrent programming. ACM SIGPLAN Not. 24, 4, April 1989.

15 Tokoro M, Nierstrasa O and Wegner P. Proceedings of workshop on object-based concurrent computing. ECOOP'91, LNCS 612, Springer, 1992.

16 Caromel D. Toward a method of object-oriented concurrent programming. Commun. of ACM, Sept. 1993,36(9):90~101.

17 Meyer B. Systematic concurrent object-oriented programming. Commun. of ACM, Sept. 1993,36(9):56~79.

18 Matsuoka S, Wakita K, Yonezawa A. Inheritance anomaly in object-oriented concurrent programming languages. In: Agha G, Wegnber P, Yonezawa A eds. Research Directions in Object-Based Concurrency, MIT Press, 1993.

19 Meseguer J. Solving the inheritance anomaly in concurrent object-oriented programming. In: Proceedings of ECOOP'93, Kaiserslautern, July 1993.

20 Andler S. Predicate path expressions. In: Proceedings of 6 POPL, ACM, 1979.

21 Ben-Ar M. Principles of concurrent and distributed programming. Prentice-Hall International, Hemel Hempstead, 1991.

22 Andrews G R, Schneider F B. Concepts and notations for concurrent programming. ACM Computing Surveys, 1983.

CCPP: THE DESIGN AND IMPLEMENTATION OF A CONCURRENT C++ LANGUAGE

WEN Dongchan WANG Dingxing ZHANG Ning

(*Department of Computer Science Tsinghua University Beijing* 100084)

Abstract COOL (concurrent object-oriented language) is a kind of highly efficient concurrent programming language. In this paper, a concurrent object model (concurrent C++, CCPP) and its language are presented. In this model, all objects are concurrent, and they communicate through asynchronous message sending. Concurrent objects are synchronized using a wait-when-necessary policy, and the synchronization constrains among concurrent threads within an object are described by path expressions. CCPP is an extension of C++, implemented by a precompiler. CCPP allows sequential/concurrent code reuse.

Key words Concurrent object-oriented language, concurrent programming, C++.

Class number TP312CC