支持多种并行计算模型的面向对象框架研究*

吕 建[1,2] 陆 陪[1,2] 于大川[1,2] David L. Shang[3]

[1](南京大学计算机软件新技术国家重点实验室 南京 210093)

[2](南京大学计算机软件研究所 南京 210093)

[3](Motorola公司软件系统研究实验室 美国)

摘要 为了支持并行程序设计,几乎所有的程序设计语言均通过提供并行与同步通信机制来支持某一高级并行计算模型,如Ada语言的任务与会合机制以及Java语言的线程和同步方法.显然,这样的程序设计语言仅能支持一种高级并行计算模型.尽管单模型的途径对某些应用来说简单而有效,但由于现实世界中的问题往往较为繁杂而难以完全用单一模型来解决.文章采用面向对象的语言机制和框架技术来解决此问题.通过分析现有各种语言中高级并行计算模型的共性,提出了若干新的面向对象语言机制.以此为基础,提出了并行面向对象框架的概念,并讨论用其表达和使用高级并行计算模型的方法.结果是,可在某种意义下将高级并行计算模型从语言中分离出来,而单一语言可由多个并行框架加以支撑,从而使得在单一面向对象语言中支持多种高级并行计算模型成为可能.

关键词 并行模型,并行框架,嵌套类,抽象函数类,隐式代码,auto机制.

中图法分类号 TP311

从解决多任务操作系统中的并发问题开始,直到MPP系统的出现,并行程序设计一直是计算机领域的研究热点[1,2].由于分布式系统、并行系统和实时控制系统中程序设计的需要,如何在程序设计语言中支持并行程序设计是程序设计语言中的重要问题之一.支持并行程序设计的语言,如Ada83[3],Concurrent Pascal,Actor[4]以及较新的Java[5]等,基本上都是在传统程序设计语言的基础上,加入一些并行机制以及同步和通信机制,来支持特定的并行计算模型.例如Ada83的具有会合机制的Task模型、Concurrent Pascal的基于管程的Process模型、CSP的同步通信模型[6]和ABCL/1的基于多种消息传递模式的Actor模型[7].

对这些程序设计语言进行研究不难发现,对于某一种程序设计语言,它往往只能支持一种特定的并行计算模型,并且支持相应模型的并行与同步通信机制均是通过关键字的形式紧密地嵌入到语言中去的.但是,客观世界中的并行问题复杂多样,某一种特定的并行模型虽然能够有效地解决某一类问题,但往往并不能适用于所有的场合.例如,基于共享变量的管程模型能较好地解决多任务操作系统中的资源共享问题,但要实现远程过程调用就显得很困难;而消息传递类的模型则恰恰相反.因而,客观世界的具体问题往往需要一种程序设计语言来支持多种并行计算模型.

在这种情况下,可以使用一些低层程序设计技术来模拟实现所需要的但该语言并不支持的并行模型.然而,这种模拟实现是不自然的,常常使得系统结构较差,难以理解、维护和重用.

本文提出了一种基于面向对象框架的方法来解决上述问题.在对并行模型进行研究的基础上,我们提出了一些新的面向对象语言机制,使用这些机制并结合与并行相关的低层系统原语,可将用关键字表达的并行模型

---

* 本文研究得到国家攀登计划项目基金、国家杰出青年科学基金和与美国Motorola合作项目基金资助.作者吕建,1960年生,博士,教授,博士生导师,主要研究领域为软件自动化,面向对象语言与环境,并行程序形式化方法.陆陪,1974年生,硕士生,主要研究领域为面向对象语言与环境,并行程序形式化方法.于大川,1977年生,硕士生,主要研究领域为面向对象语言与环境,并行程序形式化方法.David L. Shang,1958年生,博士,主要研究领域为面向对象语言与环境.

本文通讯联系人:吕建,南京210093,南京大学计算机软件新技术国家重点实验室

本文1998-01-22收到原稿,1998-03-30收到修改稿

© 中国科学院软件研究所 http://www.jos.org.cn

从程序设计语言中分离出来,而采用并行面向对象框架来加以描述.不仅如此,所提供的语言机制可使得并行面向对象框架的使用和程序设计语言中关键字的使用同样方便和直观.由于同一种语言可支持多种不同的并行框架,从而使得用单一程序设计语言来支持多种并行模型成为可能.本文首先对两类典型的并行计算模型进行分析,然后提出几种有利于表达和使用各种并行模型的面向对象语言机制,接着引入了并行框架机制用于表达并行模型,最后是一个详细的例子,通过给出简化了的 Java,Ada83 和所对应模型(分别代表了共享量和消息传递两类模型)的并行框架,来说明用同一种程序设计语言来支持不同的并行模型是可能的.

1 并行模型分析

传统程序设计语言所支持的并行模型大致有两类:共享量类和消息传递类.

共享量就是指多个并行执行单元利用读写共享变量的方法来实现同步和通信.此类模型主要有:Concurrent Pascal 的基于管程的 Process 模型、Smalltalk 的基于 PV 操作的 Process 模型[8]和 Java 的基于管程的多线程模型.

消息传递就是指多个并行执行单元利用各种各样的消息传递手段来实现同步和通信.此类模型主要有:Ada83 的具有会合机制的 Task 模型、CSP 的同步通信模型和 ABCL/1 的使用多种通信模式的活动 Actor 模型.

对上述模型进行分析不难发现:程序设计语言主要通过提供两组关键字来支持某一特定的并行计算模型:其一是与并行执行单元有关的关键字,如 *Task*, *Process* 和 *Thread*;其二是与通信和同步有关的关键字,如 *Entry*, *Synchronized* 等.并且第 2 类关键字通常在第 1 类关键字所刻画的单元中使用.因此,要在单一语言中支持多种并行计算模型,就必须将其从程序设计语言中分离出来.而能否分离又取决于可否用面向对象的概念来解释与表达上述两组关键字及其相互间的关系.基本思路如下:

(1) 可以将 Thread,Process,Task 等作为特殊的类,每一个并行执行单元就是一个对象,这些对象的内容是可执行的代码.与常规对象所不同的是,在创建这些对象时,系统往往需要做一系列底层处理工作,例如分配控制块资源、注册到处理器上以供调度等.在对象的运行过程中,其工作方式类似于函数.在其消亡时,也要做一系列底层处理工作.为了刻画 Thread,Process 和 Task 这样特殊的类,我们引入了函数类的概念.

(2) Java 的 *synchronized* 关键字,Ada83 的 *entry*,*accept*,*call* 关建字等的主要作用是包含了一些与同步和通信有关的代码,我们称之为"隐式代码".例如,Java 程序中的某一个对象执行一个带 *synchronized* 关键字的函数时,必须先自动调用一段代码,这段代码检查该对象是否还有其他 *synchronized* 函数正在被别的线程所调用,由此来决定本线程是进入等待队列还是继续执行下去;Ada83 程序中执行一个 entry 时,调用 accept 语句的任务和调用 call 语句的任务都必须各自先调用一段同步通信的代码.如何灵活地刻画这段代码并使之方便直观地使用是问题的关键所在.为此,我们引入了 auto 机制.

(3) 通常,两组关键字之间呈嵌套关系,例如,Ada83 的程序是由多个 Task 构成的,而每个 Task 中就嵌有多个 entry.ABCL/1 的程序是由多个 Actor 构成的,每个 Actor 又有多个 script.为了反映这样的关系,我们引入了嵌套类结构.

(4) 为了能采用类似于关键字的方法使用由嵌套类刻画的并行计算模型,我们引入了嵌套继承机制,并采用前缀式的子类使用方法.

2 语言机制

基于上述对程序设计语言中并行计算模型的分析,我们在常规面向对象语言基础上提出了几种对描述和使用并行模型行之有效的语言机制.

(1) 将函数看作类,称之为函数类(function class).函数调用看作为函数类的实例化.常规类的实例化一般是指根据类描述给对象分配空间,然后调用类的构造函数来进行初始化工作.实例化的结果是生成一个新对象,这个对象一直保留在存储区域中,直到被显式地删除.而函数类对象的生成就是按照常规函数调用规程或某一函数,一旦调用结束,该函数类对象立刻消亡,并不保留在存储区域中.

© 中国科学院软件研究所 http://www.jos.org.cn

将函数看作为类后，我们发现，对类进行实例化生成对象的语义完全可以不局限于常规的面向对象语言函数调用的规定.因而程序员可以完全按照自己的需要，改写函数类的对象创建方法 *create* 和对象消亡处理方法 *terminate*，以此来定义不同的对象创建和消亡规程.这样，前文所述的 task，process，thread 等，都可以用类来描述了.

(2) 类可以多层次嵌套.即在一个类的内部可以定义另一个类.如果在类 A 中定义了类 B，我们称类 B 为类 A 的内嵌类，而类 A 为类 B 的外包类.如果一个类 C 不是任何一个类的内嵌类，则称其为自由类.

由于函数被看作类，则类中的成员函数也可以看作嵌套的成员类.因此，所有具有成员函数的类都可以看作具有嵌套层次的类.嵌套类能够从包含关系的角度反映出整个计算模型的体系结构.采用嵌套类结构，可以比较方便地刻画并行计算模型中关键字之间的相互关系.

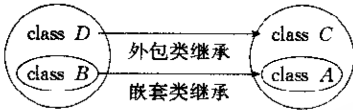

图1　具有嵌套结构的类的继承示意图

具有嵌套层次结构的类的继承同时牵涉到嵌套类和外包类的双重继承，它使得一个嵌套结构的性质可由相应的嵌套结构一致地继承.嵌套类继承规则如下：如果类 A 是类 C 的嵌套类，且类 C 继承类 D，类 A 继承类 B，则 B 必为一个自由类，或类 D 的嵌套类.如图 1 所示.

(3) auto 机制.如果一个类的构造函数和析构函数被说明成为 auto 的，则在其直接子类的构造函数执行之前，一定要先执行该类的构造函数；在其直接子类的析构函数执行之后，一定要执行该类的析构函数.当有多个类组成一条继承关系链时，由此类推，递归执行所有的 auto 函数.

在引入 auto 机制的基础之上，我们来讨论函数类的继承问题.首先，我们定义"抽象函数类"(abstract function class)，抽象函数类指的是不能进行实例化(即不能直接调用)的函数类，它只能用来被其他函数类继承.抽象函数类中没有类体，只有构造函数 *enter*( )和析构函数 *exit*( )，而且它们必须被声明为auto的.这样，根据 auto 的语义，一个函数 F_1(抽象的或非抽象的)继承一个抽象函数 F_2，指的是在函数 F_1 被调用之前，必须先调用 F_2 的构造函数 *enter*( )；在函数 F_1 被调用之后，必须调用 F_2 的析构函数 *exit*( ).

(4) 关键字类化机制.即将与传统并行语言中的与并行机制相关的关键字通过类的形式来描述.这种类可被称为关键字类.

Task，*process*，*thread* 等类的描述并行执行单元的关键字，可以通过前文所述的改变类的对象创建和消亡规程的方法来将它们封装成类.

与同步机制相关的关键字，如 *synchronized* 等，则可以将它们封装为抽象函数类.例如，我们将 Java 语言中 *synchronized* 关键字类化为抽象函数类 *Synchronized_function*，其隐式代码封装到该类的构造函数和析构函数(并将它们说明为 auto 的).而后，我们将所有需要互斥执行的函数说明为抽象函数类 *Synchronized_function* 的子类.这样，每当我们调用它们时，就会自动执行 *Synchronized_function* 中定义的隐式代码，达到互斥执行的目的(具体细节请详见第 4 节的例子).

关键字类化机制的实质就是，将以关键字形式紧密嵌入传统并行语言中的并行机制，以灵活的、可分离的类的形式加以封装.

事实上，我们可以将传统关键字的使用和关键字类的继承使用在语法形式上等同起来，使得仍然可以用类似关键字的方式来使用我们将关键字封装成的类.例如，"*task t*"在传统的语言中用关键字 *task* 表明 t 是一个可并行的任务.而在我们的关键字类化的思想下，它是"t 是 *task* 的一个子类"或"t 继承 *task*"的一种语法缩写，等同于"t is *task*" 或"t inherit *task*".

3　并行框架

运用上述的语言机制，我们就可以比较方便地描述并使用各种并行模型了.这种描述和使用是通过具体的并行框架来实现的.框架技术是当前面向对象技术的研究热点[9,10]，对于"框架"这一概念，目前尚没有统一的明确认识.本文所指的并行框架，就是为表达某种并行模型而联系在一起的一组类，这些类中含有所有反映该模型的并行单位构成和同步通信的机制.

© 中国科学院软件研究所　http://www.jos.org.cn

并行框架的描述显然要涉及到一些与系统底层紧密相关的操作,所以,其宏观结构可以用具有前面所给出的嵌套类等机制的面向对象语言来描述,但其中核心的隐式代码等操作必须利用系统提供的底层的原子操作来实现.关键字类化是描述并行框架的核心之所在,将原来紧密嵌入语言中的并行机制"抽取"出来,用类的形式加以封装.一种并行模型的全部相关并行机制都应该包含在其并行框架中,这样一种模型才能完全地体现在一个框架中.

并行框架的使用主要是通过前面所阐述的嵌套类继承来实现.通过继承框架中的类,用户类可以具有并行框架中的类与并行相关的性质(例如,继承一个线程类就可以得到一个可并行执行的线程),以及使用框架中的同步和通信机制(如隐式代码).一个简单的多线程任务类的框架继承使用如图2所示.

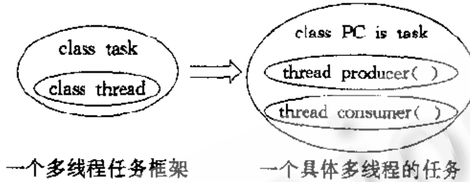

图2　通过继承使用并行框架的示意图

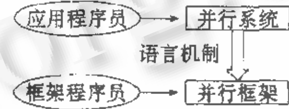

图3　并行框架设计与使用示意图

通过引入并行框架,我们成功地将语言和并行模型分离开来.一种面向对象语言,只要具有前文所讨论的语言机制,它本身无需任何并行机制,只要结合系统低层操作开发出反映若干不同并行模型的并行框架,便可以通过使用这些框架来进行各种不同领域的并行程序设计,从而达到以同一种语言支持不同并行模型的目的.

由于很多并行模型相当复杂,因而,并行框架本身的开发就不是一件简单的工作,所以,并行框架的程序设计完全可以从传统的程序设计中分离出来,我们称之为"框架设计"或"模型设计".而应用的并行程序设计可以从繁重的同步互斥问题的琐碎细节中解脱出来,直接利用框架程序员提供的高层并行框架来开发可靠性好、结构清晰、易于维护的并行系统,如图3所示.多个并行框架可以构成一个框架库,由语言的支撑开发环境来对它进行维护.

4　一个简单的例子

本节通过一个简单的例子来说明上述思想.这个例子基于前述的语言机制,通过一个并行框架来支持简化了的Java的多线程模型.为方便阅读,框架的宏观结构用类C++的伪代码来书写,一些具体的底层操作则用文字来描述.

简化了的Java语言多线程模型只提供线程类、类方法的*Synchronized*关键字说明和不带任何参数的*wait*( )和*notify*( )方法(只维护一个等待队列),其含义都与Java相同.其实这也就是一个最简单的Monitor模型[11].假设系统底层提供了PV操作,我们用PV操作来实现关于*Synchronized*关键字的全部隐式代码.

我们的并行框架全部地体现于一个Multi_thread类中,其中内嵌了支持并行执行单元的Thread类和支持同步机制的Synchronized_class类和.具体描述如下:

```
class Multi_thread{
    class Thread{
        terminate( ){ // 改变对象消亡规程
            ... // terminating, releasing occupied resource
            ... // Notify Multi_thread::self to kill the thread
        }
        meta create( encloser: Multi_thread){ // 改变对象创建规程
            ... // Create a thread according to code in self. Run()
            ... // Pushing the thread object into the stack
            ... // Pushing arguments into the stack
            ... // Get the entry point of selfclass
```

© 中国科学院软件研究所　http://www.jos.org.cn

```
            ... // Set the teminating address
            ... // Notify encloser to register/schedule the thread
        }
        virtual void Run( );// 由子类重载
    } // End of class Thread
    class Synchronized_class{
        Semaphone mutex, condsem, urgent; // Semaphone 是由系统提供的原子类型
        int wcount, urgentcount, condcount ; // 3个计数器
        wait( ){ // 供子类使用的函数
                condcount=condcount+1;
                if (urgentcount>0) V(urgent); else V (mutex);
                P(condsem);
                condcount=condcount+1;
        }
        notify( ){ // 供子类使用的函数
                 urgentcount=urgentcount+1;
                 if (condcount>0) {V(condsem); P(urgent)}
                 urgentcount=urgentcount-1;
        }
        abstract_function_class Synchronized{// 将 Synchronized 关键字类化为一个抽象函数类
            auto enter( ){P(mutex)} // 构造函数,被说明为 auto 的
            auto exit( ){ // 析构函数,被说明为 auto 的
                if (urgentcount>0) V(urgent); else V(mutex)
            }
        } // End of class Synchronized
    } // End of class Synchronized_class
} // End of class Multi_thread
```

在这里,我们通过改写对象创建和消亡方法 *terminate*( )和 *create*( ),实现了 Thread 类;通过用 PV 操作实现隐式代码,实现了 Synchronized 类以及其外包的 Synchronized_class 类;并将它们组合在 Multi_thread 类中.使用这个框架时,直接继承 Multi_Thread 类来生成用户的多线程应用,其中,重载 Thread 类中的 *Run*( )函数生成用户所需的线程类(作为 Java 提供的模型的简化,只要一个线程类的对象一经生成,即自行启动执行);继承 Synchronized_class 及其内嵌类 Synchronized 来生成需要进行同步控制的对象的类.

下面就是一个包含产生者线程和消费者线程共享一个队列的多线程的并行应用例子.

```
Multi_thread Producer_and_Consumer{// Producer_and_Consumer 是 Multi_thread 的子类
    SharedQueue sq; // 先使用后定义 SharedQueue,关于 SharedQueue 的定义见下面.
    Thread producer(speed: integer){
            Run( ){ ... ; sq.Push(random_integer); ... // 以给定速度 speed 产生随机数放入共享队
            列 sq. }
            }
    Thread consumer(speed: integer){
            Run( ){ ... ; sq.Pop( ); ... // 以给定速度 speed 从共享队列 sq 中取出数并处理. }
    }
    Synchronized_class SharedQueue{// SharedQueue 类是 Synchronized_class 类的子类.
        Element * Head=NULL, Tail=NULL;// 共享队列的头指针和尾指针.
        Synchronized Push(int aValue) {// Push 函数是抽象函数类 Synchronized 的一个子类.
            Element * tmpNode;
            tmpNode=new Elment; tmpNode.value=aValue;
            if(Head==NULL) Head=tmpNode; else Tail.next=tmpNode;
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
                tmpNode.next=NULL; Tail=tmpNode;
                notify( );
            }
            Synchronized int Pop( ){ // Push 函数是抽象函数类 Synchronized 的一个子类.
                int TmpValue;
                if (Tail==NULL) wait( );
                TmpValue=Head.value;
                Head=Head.next;
                // 此处省去释放 Head 指针所指向空间的操作,假设系统支持自动垃圾收集.
                return TmpValue;
            }
        } // end of SharedQueue.
} // end of Producer_and_Consumer.
Producer_and_Consumer pc_test;
pc_test.producer(5); // 生成第1个 Producer 线程并运行,以速度5产生对象到共享队列中去
pc_test.consumer(10); // 生成第2个 Consumer 线程并运行,以速度10从共享队列中取出
pc_test.producer(8); // 生成第2个 Producer 线程并运行,以速度8产生对象到共享队列中去
```

上面这一个例子从形式上来看,Thread,Synchronized_class 和 Synchronized 像是作用于 producer/consumer,ShardQueue 和 Push/Pop 函数的传统的关键字,但实际上分别是 producer/consumer、类 SharedQueue 和函数类 Push/Pop 的父类.这样,通过继承,producer/consumer 是可并行执行的线程;它们共享的 ShardQueue 的对象 sq 中隐含了与其读写同步控制相关的信号量和计数器;同时,由于被 producer/consumer 线程调用的函数 Push/Pop 继承了抽象函数类 Synchronized,在开始调用它们之前,就自动调用 Synchronized 类中的 auto 构造函数 *enter*( );在调用它们结束之后,就自动调用 Synchronized 类中的 auto 析构函数 *exit*( );并且,Push/Pop 函数中可以直接使用 Synchronized_class 类中定义的 *wait*( )和 *notify*( )对 SharedQueue 中的变量进行读写,来达到同步的效果.这一系列内部实现对用户来说都是透明的,用户仍然以一种类似于传统的关键字的方式来使用并行框架中提供的并行机制和同步机制.

5　结束语

本文的主要工作在于:引入了若干面向对象语言机制,并采用并行框架技术将原来紧密嵌入某一种语言中的并行计算模型从语言中分离出来,从而使得用同一种语言来支持多种并行模型成为可能.将本文的思路拓广一下,不难发现,将表达计算模型的机制从语言中抽取出来以框架的形式来描述这一想法,不仅适用于并行计算领域,同样可以用于分布式计算和实时计算领域.

框架技术的研究是目前面向对象技术研究的热点.但从反映高层计算模型这一角度对框架进行的探讨还为数不多.由于计算机应用的领域越来越广,所需表达的高层计算模型也越来越丰富,探讨各种不同框架的描述和使用这些框架的面向对象语言机制是很有必要的.这样,才能大大地增强一种面向对象语言在某一领域中解决各种不同问题的能力.

上述研究成果已体现在基于 Transframe 语言的软件开发环境 MagicFrame 设计之中.MagicFrame 是我们和美国 Motorola 公司合作研究的项目,其主要设计目标之一就是以框架的手段来支持领域相关程序设计.

本文的工作只是一个尝试.在对并行框架的语言支持方面,只给出了函数作为类、嵌套类继承、关键字类化、auto 机制等语言机制.由于并行计算模型复杂多样,还需要有其他更强有力的机制来支持并行框架的描述和使用.下一步的工作设想是:(1) 从更广的范围,更深入地来分析各种不同的并行模型,从而给出更有效的语言机制;(2) 从实现的角度来具体设计实现一些典型的并行模型的框架;(3) 在已有工作[12~14]的基础上,讨论基于并行框架的软件规约与形式化方法.

参考文献

1　Snow C R. Concurrent Programming. Cambridge: Cambridge University Press, 1992

© 中国科学院软件研究所　http://www.jos.org.cn

2 Andrews Gregory R, Schneider Fred B. Concepts and notations for concurrent programming. Computing Surveys, 1983,15(1):3~43

3 Stratford-Collins M J. ADA: A Programmer's Conversion Course. New York: John Wiley and Sons, Inc., 1982

4 Beaudouin-Lafon Michel. Object-oriented Languages: Basic Principles and Programming Techniques. London: Chapman & Hall, 1994

5 Arnold Ken, Gosling James. The Java Programming Language. Reading, Massachusetts: Addison-Wesley Publishing Company, 1996

6 Hoare C A R. Communicating Sequential Process. Englewood Cliffs, NJ: Prentice Hall, 1985

7 Yonezawa A, Briot J-P, Shibayama E. Object-oriented concurrent programming in ABCL/1. In: Norman Meyrowitz ed. Proceedings of ACM SIGPLAN Conference'86 on Object-oriented Programming Systems, Languages and Applications (OOPSLA'86). New York: ACM Press, 1986. 252~268

8 Goldberg Adele, Robson David. Smalltalk-80: The Language and Its Implementation. Reading, Massachusetts: Addison-Wesley Publishing Company, 1983

9 Johnson Ralph E. Documenting frameworks using patterns. In: Andreas Paepcke ed. Proceedings of ACM SIGPLAN Conference'92 on Object-oriented Programming Systems, Languages and Applications (OOPLSA'92). New York: ACM Press, 1992. 63~76

10 Nierstrasz Oscar, Tsichritzis Dennis. Object-oriented Software Composition. Englewood Cliffs, NJ: Prentice Hall, 1995

11 Hoare C A R. Monitors: an operating system structuring conception. Communications of ACM, 1974,17(10):549~557

12 Lü Jian. Introducing data decomposition into VDM for tractable development of programs. ACM Sigplan Notices, 1995, 30(2):41~46

13 Lü Jian. Developing parallel object-oriented programs in the framework of VDM. In: McGregor J D ed. Annals of Software Engineering——Special Volume Object-oriented Software Engineering: Foundations and Techniques. Amsterdam: Baltzer Science Publishers, 1996. 199~211

14 Lü Jian. Sound and complete rules for data reification. Science in China (Series E), 1997,40(4):379~386

Researches on Object-oriented Frameworks Supporting Multiple Parallel Computing Models

LÜ Jian[1,2] LU Pei[1,2] YU Da-chuan[1,2] David L. Shang[3]

[1](*State Key Laboratory for Novel Software Technology Nanjing University Nanjing* 210093)

[2](*Institute of Computer Software Nanjing University Nanjing* 210093)

[3](*Software Systems Research Laboratory Motorola Inc. USA*)

Abstract In order to support parallel programming, almost all of the programming languages incorporate a high-level parallel computing model into the language by providing parallelism and synchronization mechanisms, such as Ada's task with rendezvous and Java's thread with synchronized method. Obviously, such a programming language can only support one high-level parallel computing model. Although the approach of one model is simple and fairly effective to some applications, unfortunately the problems in the real world are always too diverse to be solved by using a single parallel computing model. In this paper, a new object-oriented approach to this problem is proposed. After analyzing the high-level parallel models of the various languages, some novel object-oriented language mechanisms are presented. Based on them, the concept of object-oriented parallel framework is proposed and the method for expressing and using the high-level parallel computing models by parallel frameworks is discussed. As a result, the high-level computing models are separated from languages in some sense and more than one framework could be given within a single language. Therefore, using a single language to support the various parallel models turns to be feasible.

Key words Parallel model, parallel framework, nested class, abstract function class, implicit code, auto mechanism.

© 中国科学院软件研究所 http://www.jos.org.cn