Dual-Object：面向对象的并行程序设计*

袁 伟 孙永强

（上海交通大学计算机科学与工程系 上海 200030）

摘要 面向对象的并行程序设计提供了类似于共享内存模型对通讯和计算的抽象能力，从而非常适合于大型并行软件系统的开发．但是基于远程对象调用的分布式对象的实现效率一直是面向对象方法在分布式/并行程序设计中得到广泛应用的障碍．本文介绍了并行机MANNA上所采用的面向对象的并行程序设计模型——Dual-Object模型．该模型通过引入从语义角度出发给出的数据一致特性的描述，在一定程度上解决了实现效率低下的问题．其次，文章通过程序设计实例详细地讨论了基于Dual-Object模型的扩展C++并行程序设计，并给出了部分实际测试结果．

关键词 并行程序设计，面向对象方法，Dual-Object模型，远程对象调用．

中图法分类号 TP311.11

近年来有关分布式系统的研究表明，基于对象的并行程序设计方法非常适合于构造大型并行系统．一方面，面向对象的方法提供了类似于共享内存模型对通讯和计算的抽象能力；另一方面，对象隐藏了并行系统中各部分状态和行为的细节，从而增强了并行系统的模块化和可维护性．其次，对象的继承机制也简化了并行软件系统的复用性．

远程过程调用RPC(remote procedure call)[1]和网络API编程是两种常见的分布式计算的程序设计方法．这两种方法的主要缺点是程序员需过多考虑底层的通讯细节而非所需解决的并行问题本身．[2]远程对象调用ROI(remote object invocations)[3,4]是面向对象的并行程序设计中的一种常见的分布式对象的实现方法．它提供一种相对于共享内存模型更为自然的描述对象间相互作用的机制，同时也可在任何基于共享内存模型或消息传递模型的并行计算平台上获得实现．相对于RPC或API方法，面向对象的方法具有更好的可编程性、可移植性和代码的复用性．唯一的问题是ROI在分布式内存的并行体系结构中的实现效率．频繁地对远程对象进行访问将极大地增加系统的通讯开销．虽然对象的复制可在一定程度上降低系统的开销，但是当对象的状态需要经常性变化时，维持分布式对象一致性的开销将在很大程度上降低并行计算系统的潜在性能加速比．值得庆幸的是并行应用系统并非在任何时刻均要求维持分布式对象的一致性，所以面向应用的分布式对象的部分数据一致性维持方法将可提高系统的效率．上海交通大学与德国GMD FIRST合作项目中，MANNA并行计算机上的Dual-Object并行程序设计模型[4]正是这样一个面向应用的解决方法．Dual-Object将分布式对象划分为复制和非复制两个部分，从语义角度出发来分析和处理分布式对象，通过引入维持部分数据一致性的实现策略的说明，较好地解决了基于ROI的分布式对象的实现效率问题．

本文首先介绍了Dual-Object模型和基于该模型的扩展C++语言．然后，通过并行程序设计实例详细讨论了基于该模型的并行程序设计方法，并给出了MANNA并行机上的部分测试结果．

1 Dual-Object模型

定义 1. 一个Dual-Object对象由逻辑上紧密相关而物理上又分离的两个部分组成：原型(Prototype)和类体(Likeness)(如图1所示)．其中，

prototype ::=⟨public, private⟩,

likeness ::=⟨extract(public),ref_prototype⟩

ref_prototype是表示该类体所属原型的全局唯一标识．显然，原型表示了一个对象的内部结构，likeness则描述了从该对象原型的public部分所提取出来的全局共享的状态．

---

* 本文研究得到国家自然科学基金资助．作者袁伟，1972年生，博士，主要研究领域为并行程序设计及其优化实现技术．孙永强，1931年生，教授，博士导师，主要研究领域为计算机理论，软件．

本文通讯联系人：袁伟，上海 200030，上海交通大学计算机科学与工程系

本文1996-12-09收到原稿，1997-02-17收到修改稿

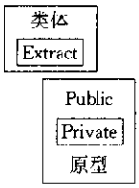

图1 Dual-Object

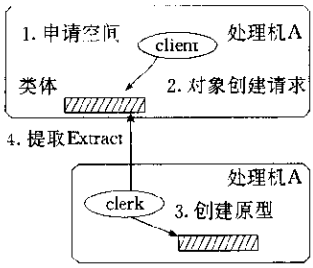

图2 Dual-Object对象

从应用的角度来看，一个 Dual-Object 实例是对象的本地实例和至少一个全局共享的远程实例的组合，原型作为本地实例存放在实现该对象的处理结点上，类体作为一种全局共享状态的数据载体存放在各个使用该 Dual-Object对象的进程各自所在的处理结点上.

Dual-Object 模型通过引入管理进程(Clerk)来协调对象的生成、调用和分布式对象的数据一致性. 对应于每个对象的原型，系统自动生成一个 clerk 管理进程来处理该对象的操作：①当系统需要生成一个新的 Dual-Object对象时，对象的实例化请求发送给能处理该类对象的 clerk 进程，然后，由该 clerk 完成原型的内存分配和初始化，并将原型的全局标识以及生成类体返回给请求生成操作的对象(如图 2 所示). 显然，当对象的生成请求来自于同一地址空间的进程时，Dual-Object 对象的生成采用 LOI(local object invocation)方式进行. ②当需要删除某 Dual-Object 对象时，该对象的拥有者向 clerk 发送一个释构请求，由 clerk 删除该对象的原型实例并同时删除该对象的类体实例. 显然，原型仅可由对象的拥有者进行释构操作，通过其他复制类体所进行的非对象拥有者的释构请求将会被 clerk 所拒绝.

定义 2. 为维持原型和类体间的数据一致性，系统必须在修改原型的 public 部分状态后重新构造新的类体. 我们称这一类的数据一致性问题为对象的垂直一致性.

在 Dual-Object 模型中，远程对象对原型的访问大多通过远程驻留的类体来进行. 类体与原型间的垂直一致性由两者间参数和结果传递过程同步操作来解决.

定义 3. 由于一个原型可能拥有多个类体，所以系统需维持多个类体间的数据一致性. 我们称这一类的数据一致性问题为对象的水平一致性.

对象的水平一致性问题的困难程度远高于垂直同步. 作为水平同步的一个特例是原型的 public 部分为空，即原型仅由私有部分(Private)构成而类体为空. 在这一情况下，水平同步问题很自然地消除了. 如何建立类体间的水平一致性是模型得到有效实现的关键问题. Dual-Object 模型解决该问题的出发点是：通过程序员根据具体应用程序的语义作出的实现策略描述来完成分布式对象的数据一致性管理，从而在面向对象的并行程序设计的抽象层次与实现效率上获得了一个折衷.

数据一致性处理可分为以下两种方式：

(1) 垂直一致性：由 clerk 完成对原型的提取操作(Extract)，并更新相关的所有类体实例；

(2) 水平一致性：当对象的某一方法更新了类体中的变元后，由 Dual-Object 运行系统执行一个全局的合一处理(Unification)维持各个复制类体以及原型的数据一致性(如图 3 所示).

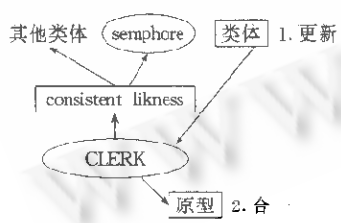

图3 水平一致性

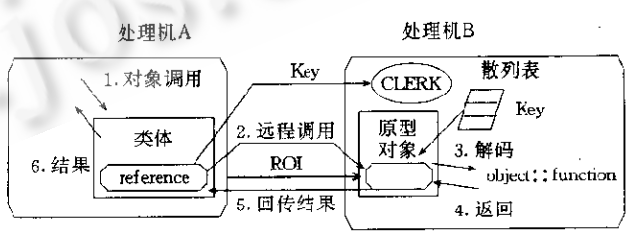

图4 Dual-Object的ROI调用

类体上的操作直接通过标准的本地对象调用(LOI)进行，而作用于对象私有部分的操作(即原型上的操作)通过 ROI 方式在当前原型所在的结点上进行(如图 4 所示). 由于 clerk 和原型对象具有相对的独立性，所以 clerk 和原型并非要求驻留在同一处理结点上. 原型对象可以在系统运行过程中自由迁移于并行计算机的不同结点，而由 clerk 来协调类体和原型间的函数调用和数据一致性. 显然，对于同一地址空间的原型操作，可通过 LOI 方式直接作用于原型或通过由该原型推导出的类体以 ROI 方式访问. 唯一的区别在于，后一方式可增强原型存储方式的透明性，以便于原型对象在并行处理结点网络上的迁移.

定义 4. 原型的 ROI 调用方式可划分为 3 类：

(1) 不同结点：被调用原型和调用进程分别处于不同的处理机；

(2) 不同地址空间:被调用原型和调用进程处于同一处理机的不同地址空间;

(3) 同一地址空间:被调用原型和调用进程处于同一处理机的同一地址空间.

显然,最后一种方式的实现效率最高,而第一种方式的实现效率最低.

定义 5. Call-by-likeness 通过类体访问 Dual-Object 状态的方式可划分为两个阶段:

(1) 直接访问当前类体中的公共部分(Public);

(2) 若本地类体无法完成该调用,则通过向管理该类体所对应原型的 clerk 发送 ROI 请求来完成(如图 4 所示).

我们称这一通过类体的参数传递方式为 call-by likeness.

通过类体对 Dual-Object 访问与直接访问原型来对 Dual-Object 进行访问的区别在于,原型采用一般的参数传递方式(即 call-by-reference)而类体采用了 call-by-likeness. 当类体中不存在所需数据时,call-by-likeness 退化为 call-by-reference. 所以,类体的 call-by-likeness 实际上提供了避免远程操作的可能性.

clerk 的作用非常类似于 CORBA(common object request broker architecture)[2]系统中 broker 的作用,协调类体/client 和原型/server 间的协同工作. 不同之处在于类体可提供 CORBA 系统 IDL STUB 所不具有的智能代理机制(call by likeness),从而减少了不必要的重复远程通讯.

2 支持 Dual-Object 的 C++语言

MANNA 并行机通过基于 Dual-Object 模型扩展的 C++语言,提供对基于远程对象调用的面向对象的程序设计方法的支持. 其基本出发点是通过由程序员在源程序中以注释方式来显式描述 Dual-Object 类中每个方法的调用方式和参数传递语义. 扩展部分的语法采用注释方式是为了使并行程序无需修改便可直接由普通 C++编译器得到非分布式并行处理环境中的实现.

Dual-Object 类通过关键词 global 进行说明,类的 protect 说明逻辑上可被等价看作是 public 进行处理. 一个典型的 Dual-Object 类的说明如下所示:

```
class Neuron /*! ::neuron ! */: ...{
    private:
            double      threshold;     // a private member
    protect:
            double      weight;        // data , subject to
    public:
            int         x,y;           // unification/extraction
      //a public method
          double        value();   /*! global ! */
          void          fire();    /*! trigger ! */
       /*! copy ! */ neuron * replicate(); // result annotation
      /*! move ! */ neuron * migrate();
       void          connect(/*! copy ! */ Neuron &);
        // parameter annotation
}/*! global ! */
```

对于 Dual Object 类中每个成员函数可以采用以下几个关键词来说明函数调用时所需进行的数据一致操作,而具体的实现则由 Dual-Object 编译系统自动完成(诸如生成类体和原型的 C++类的定义以及相关 clerk 的类定义).

(1) in:在该函数调用前,系统进行合一操作(参见第 2 节)维持数据的水平一致性;但是在调用后,无需提取操作以维持 Dual-Object 的垂直一致性;

(2) out:在该函数调用前,系统无需执行合一操作;但是调用后必须进行提取操作;

(3) inout:in 和 out 的综合(缺省设置);

(4) global:系统无需进行合一或提取操作以维持处理一致性;

(5) const:标记该函数仅读取该对象的状态,它隐式说明系统需在该函数调用前进行合一操作;

(6) local:标明该函数仅涉及本地类体中 public 变元,从而不必采用 ROI 调用;

(7) trigger:标明调用者在将参数传递给被调用者后,两者同时并行运行. 它也隐式说明系统需在该函数调用前进行合一操作.

(8) restricted:标明该函数仅限于通过原型进行调用;

(9) idempoten:标明该函数不影响所涉及任何对象的状态.

C++通常提供两种方式的参数传递:call by value 和 call-by-reference,但是基于对象的分布式并行环境一般都

提供参数传递过程中有关对象复制和迁移的参数传递方式.例如,call-by move 将参数对象迁移至函数调用者所在处理结点;call-by-copy 则通过将参数对象复制并维持数据一致性来进行参数传递.[5]扩展的C++语言吸收了下列参数传递语义,但是将其中的复制对象的数据一致性交由程序员负责.

(1) in:等价于传统的 call-by-value;

(2) out:等价于传统的 call-by-result,函数返回结果存放于静态分配的空间而返回指向该地址空间的指针;

(3) inout:既是参数又是返回结果(call-by-value-result);

(4) copy:在被调用对象的地址空间中申请一段自由内存,用于复制参数对象;

(5) move:类似于 copy,但是对象调用者地址空间的原参数对象被删除.

编译过程首先由预编译器 P++将源程序(文件后缀为.pre)转变为由 PEACE 操作系统的通讯机制支持的标准C++语言程序,然后利用标准C++预处理器和 i860-C 编译系统完成可执行代码的编译(如图 5 所示).

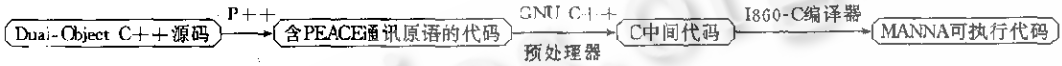

图5 编译流程

3 Dual-Object 程序设计

Dual-Object C++语言提供了一个类似于共享内存模型的并行程序设计模型.由于现有的大量并行算法均基于 PRAM 模型,所以程序员很容易从并行算法的抽象描述编写出基于 Dual-Object 的并行程序.其次,面向对象的方法隐藏了并行系统中各部分状态和行为的细节,从而并行程序具有良好的模块化和可维护性.

采用 Dual-Object 可以非常简单地实现分布式系统中的 client/server 模式和虚拟共享内存模式的并行程序设计.为了说明 Dual-Object C++语言的程序设计方法,我们在此给出了 MANNA 并行机 PVM3.0[6]的部分程序代码.由于 PVM 系统实现中需给每个处理结点赋予一个唯一的 PVM 虚拟机标号,所以我们采用了类似于虚拟共享内存的 Dual-Object 程序设计.PVMinit 类对象的原型驻留在 MANNA 机的物理 0 号结点中,而其他处理机结点上生成相应的类体.各处理结点的并行进程通过 PVMinit 类对象来实现对变元 NextPvmId 的访问,从而获得唯一标识的 PVM 虚拟机号.

```
Dual-Object 类的定义
    //PVMinit.pre
    extern int GetNewPvmID();
    extern void SetPvmFlag();
    class PVMinit/*! ::pvminit !*/{
        int id;
    public:
        PVMinit(){
          id=GetNewPvmID();
        }
        int getpvmid()/*! idempotent!
    */
        {return id;}
        void Barrier(char[16],int);
    }/*! global !*/;
```

```
    部分主程序:
    /*PVMinit 类 GetNewPvmID 方法的实现*/
    static int NextPvinId=0;
    int GetNewPvmID(){
        printf("now generating a new PVMID=%d
            \n",NexPvmId);
        return NextPvmId++;
    }
    void primer:action(/*PEACE C++程序的主模块
*/
    ......
      if(HOST==0){//get PVM initialized at node 0
        PVM=new PVMinit();
        My_PvmId=PVM→getpvmid();
        printf("LOI:Got ID T%x\n",My_PvmId);
      }
    else if(HOST>0){
      PVM=new PVMinit();
      My_PvmId=PVM→getpvmid();
      printf("call by Likeness:Got ID %d\n",
My_Pvmid);
    }
    ......
    }
```

混沌问题是一个常见的并行处理领域,Mandel 集计算是一个典型的例子.MANNA 并行机 Mandel 实现采用了 Farm 并行计算模式:计算任务由 0 号结点集中分配,通过 ROI 方式调用其他处理结点上 Mandel 对象的 mandel_line 方法来启动计算任务在 MANNA 并行机各处理结点的运行.该程序实例充分利用了面向对象方法中的对象间的自然

并行性,Dual-Object 对象 Mandel 的类定义如下:

```
class Mandel/*!  ::mandel!  */{
        public:
          Mandel();
          ~Mandel();
        /*!  copy !  */mandelbuf*mandel_line(/*!  in!  */double xs,
                                              /*!  in!  */double ys,
                                              /*!  in!  */double xe,
                                              /*!  in!  */int np,
                                              /*!  in!  */int mx);
    }/*!  global!  */
```

为了测试本文方法的效率,我们在 MANNA 并行机上实际运行了基于 Dual-Object 的 Mandel 程序(测试结果如表 1 所示). MANNA 机每个结点为基于共享内存的 I860 双 CPU 结构,每个结点间通过高速通讯网络互联,前端机采用 SUN 工作站,它用于系统加载和计算结果的图形显示.

表 1　MANNA 并行机上的性能(采用 Mandel 程序测试)　　单位:Megaflops

处理机数目							平均值
cpu=2×1	37.31	36.72	37.32	38.01	36.10	39.31	37.46
cpu=2×2	31.38	72.17	74.63	73.73	88.68	93.46	72.34
cpu=2×3	107.54	104.91	97.15	129.56	135.09	141.35	119.26
cpu=2×4	142.08	199.27	173.28	182.43	219.58	178.47	182.52
cpu=2×5	143.43	164.20	224.29	238.93	244.45	277.83	215.52
cpu=2×6	310.71	311.54	152.16	309.48	330.25	405.48	303.29
cpu=2×7	297.34	408.28	353.63	357.39	364.52	364.52	357.61

测试结果显示,对于不同计算区域,MANNA 并行机均表现出较强的并行处理能力.这表明 Dual-Object 模型确实提供了一个较好的程序设计模型,并且部分数据一致性实现策略有助于提高分布式对象的实现效率.该测试结果也表明随着负载的变化,并行计算机的利用率变化较大,这表明负载均衡问题仍然是获得最佳并行处理性能的关键问题.从另一个侧面而言,这表明 Dual-Object 模型达到了我们所期望的设计目标:程序员仅需关心并行问题的本身而不必过多关心有关分布式对象的实现细节,进而提高了并行程序开发的效率.

4　结束语

Dual-Object 是一个描述并行/分布式系统的面向对象的方法.它不是依赖于分布式对象的完全一致性,而是从程序语义角度出发,提供一个面向程序员的分布式对象的数据一致性的实现方案.Call-by-likeness 参数传递方式减少了不必要的远程数据通讯,进而提高了基于 ROI 方式的分布式对象的实现效率.

虽然 Dual-Object 是作为一个并行程序设计模型而提出,但是它的基本原理亦同样适用于网络环境的分布式计算.CORBA[2]是目前面向对象的一个分布式计算环境标准,该标准已获得了许多公司(例如 IBM 和 HP)的支持.CORBA 采用 broker 来处理系统中服务器与客户间的消息.Broker 结构使得服务器的选择和服务器的实现均透明于客户端,从而有利于系统的修改以及支持多个分布式服务器和多种通讯方式(同步或异步).Dual-Object 模型与 CORBA 系统具有许多相似之处,例如 Dual-Object 模型的 Clerk 与 CORBA 系统的 Broker 具有异曲同工之用.主要区别在于,Dual-Object 模型提供了相对于共享内存模型更为自然的描述对象间的作用,clerk 类和类体类均可由 P++系统自动生成,从而为并行/分布软件系统的设计和开发提供了一个有效的工具.CORBA 的主要设计目标是提供一个支持分布式面向对象应用系统的可移植性以及系统间相互操作的标准,但是程序员在设计过程中仍需关心许多有关 Broker 的细节问题.其次,Dual Object 的类体起到了智能代理(Smart Agent)的作用,call-by-likeness 提供了 CORBA 系统中 IDL stubs 所无法提供的优化实现的可能性.

实际测试数据显示负载均衡问题仍是提高并行系统处理效率的关键,本文进一步的研究方向就是研究如何基于 Dual-Object 模型来实现并行系统的负载调度.

参考文献

1　Nelson B J. Remote procedure call. Technical Report of CMU-CS-81-119. Carnegie-Mellon University, 1982

2 Randy Otto. Understanding CORBA (common object request broker architecture). Prentic Hall PTR, 1996
3 Black A. Distribution and abstract types in emeralds. IEEE Transactions on Software Engineering, 1987, 13(9): 65～76
4 Jorg Nolte. Language-level support for remote object invocations. Technical Report of German National Research Center for Computer Science. Berlin, Germany, 1995
5 Norton Charles D, Szymanski Boleslaw K. Object-oriented parallel computation for plasma simulation. Communication of the Association for Computing Machinery, 1995, 38(10): 88～100
6 Al Geist *et al*. PVM3 user's guide and reference manual. Oak Ridge National Laboratory. Oak Riceg, Tennesse 37821, May 1994

Dual-Object Approach to Object-Oriented Parallel Programming

YUAN Wei SUN Yong-qiang

(Department of Computer Science and Engineering Shanghai Jiaotong University Shanghai 200030)

Abstract Objects provide a uniform treatment of both communication and computation similar to a shared memory model. Thus, object paradigm is suitable for construction of large parallel software system. ROI (remote object invocations) provide a natural mechanism for remote interaction between objects. Unfortunately, the invocation performance blocks the widespread of object-oriented parallel programming paradigm. This paper introduces the Dual-Object model for MANNA machine, which attacking the invocation overhead problem at a semantic level via annotations. Some examples and experimental data are presented to show this dual object parallel programming with extended C++ language.

Key words Parallel programming, object-oriented, Dual-Object, remote object invocations.
Class number TP311.11