面向对象的并行程序设计环境框架

A Framework of Object-Oriented Parallel Program Designing Environment

李庆华 徐 权

(国家高性能计算中心 武汉430074)

(华中理工大学计算机科学与工程学院 武汉430074)

Abstract Wieldy and high-powered parallel program designing environment is the key of popularizing parallel processing technology. In this article, we will discuss a framework of object-oriented parallel program designing environment, which is based on PVM, together with some characters of OOP. The framework aims at find a likely way to combine parallel program designing technology with OOP.

Keywords PVM, OOP, COW, Parallel, Class, Object

一 引言

人们普遍认为并行处理技术将成为下一世纪计算机领域的主流技术。近年来，从不同的角度入手，已经研制出各种各样的并行程序设计环境，有的从程序设计语言的角度出发，在程序设计语言中引入了有利于并行性开发的语言元素，如 Linda、Emerald 等，前者是基于共享存储计算模型的，而后者是基于消息传递计算模型的[1]；还有的从程序辅助开发工具的角度出发，像 C、FORTRAN 77等，开发了并行操作原语库，著名的产品有 PVM、MPI 等，它们都是基于消息传递计算模型的[2,3]。虽然有如此多的开发环境，但它们都不同程度地存在这样那样的问题，有的不支持异构环境，有的只支持单一的并行计算模型，还有的容易出现负载失衡的情况。同时，它们都有一个共同的缺点是，用户界面不够友好，利用它们进行并行程序开发仍然是件费时、易出错的工作。

在众多的并行程序设计环境中，PVM 和 MPI 的普及性好。PVM 在并行程序设计方面具有如下一些特性：对异构环境的支持[2]；通信原语简单易用，并支持对多种数据类型的传输；支持组通信和动态组的概念；采用了动态虚拟机的思想，并对虚拟机有完善的管理措施。

而 MPI 的主要特点是：充分考虑了通信效率的问题，采用并行 I/O、主动消息等手段来提高通信效率[4]；支持组通信、点对点的通信、进程拓扑、进程组等[5,6]；对同步、异步、阻塞、非阻塞的通信原语以及广播通信等都有很好的支持[5]。

尽管 PVM 具有上述的一些优点，但它仍然存在一些致命的问题，如：

·仅支持 TCP/IP 协议连接的网络；

·消息传递的效率较低，不支持异步消息传递；

·仅支持消息传递的计算模型；

·程序开发仍然费时、易出错，软件的维护与复用也十分困难[1]；

·当有多个虚拟机运行在 COW 系统上时，结点会出现计算负载失衡的情况。

为了解决上述这些问题，我们考虑把面向对象程序设计(OOP)的技术和 PVM 原有的一些良好特性结合起来，期望获得更加易用、有效的并行程序设计环境。采用 OOP 的方法，程序员可以很方便地定义客观世界的模型，并获得抽象性很高的软件框架。这一点对于并行程序设计来说具有特殊重要的意义，因为现实世界是并行的，如果能用对象在计算机中逼真地描述这种并行，将会极大地推动并行程序设计环境技术的发展。

二 并行程序设计环境的框架

我们提出如图1所示的并行程序设计环境的框架，这个新框架是 OOP 和 PVM 相结合的产物，因此，我们称之为 OOPVM。从图中我们可以看出 OOPVM 具有分层的结构，这样可以使它具备更好的可扩展性和易维护性。

李庆华 教授，博士生导师，国家高性能计算中心(武汉)主任，主要研究领域为并行处理与高性能计算、图形图像处理以及智能化软件系统。徐 权 硕士，主要研究领域为并行处理与高性能计算。

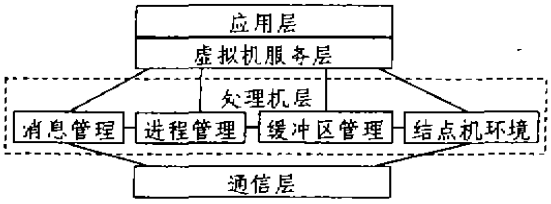

图1 OOPVM 的框架结构

1. 通信层

通信层的作用是负责最底层的通信接口的实现，并向上层提供高效的数据传输原语。这种数据传输主要是指一种可靠的面向字节流的传输[8]。通信层参考了网络 OSI 模型的结构，也具有分层的对象结构，每一层负责消息传递的不同方面，并使其整体构成上层对象的通信平台。其所有向上提供的数据传输服务具有统一的调用格式，而原语的实现可以有一种基于 OS 的通用方式，也可以由具体的工作站、服务器的制造商来提供。如果是后一种方式，传输原语可以获得更好的硬件支持，从而提高传输效率。这很有点像数据库中的 ODBC 的概念。这样做，一方面可以提高 PVM 在现有的异构环境中消息传递的效率，另一方面，如果把对各种协议的仿真实现在传输原语中，无疑可以扩展现有的异构环境，使 PVM 能在使用异种协议互联的 COW 系统上运行。

2. 处理机层

处理机层是整个 OOPVM 的核心，它包含了 PVM 的 pvmd，可以看作 pvmd 的类化版本。但在原有的 pvmd 的基础上，处理机层增加了一些对进程对象、消息、缓冲区的管理功能。这一层采用了嵌套的对象结构，处理机层的整体是一个系统对象，我们称为处理机对象，其它的各种服务都封装成子对象嵌在其中。处理机对象 Processor 的结构大致如下：

```
class Processor{
    unsigned long Handle;
    string Name;
    class Message_Manage; //消息管理
    class Course_Manage; //进程管理
    class Buffer_Manage; //缓冲区管理
    class Pvm_Session; //结点机环境管理
} //End of Processor
```

下面对其中几个主要的子对象做简略的介绍：

(1)消息管理 消息管理类 Message_Manage 也是一个嵌套的对象类，主要包括基本消息类 Message_Object、消息队列类 Message_Queue_Object、消息调度类 Message_Schedule_Object。基本消息类是所有消息类的父类，它是一个抽象类，定义了消息的初始化、终止及与消息队列、缓冲区对象的交互等。通过继承基本消息类，用户可以定义特殊的消息对象，发送自定义的数据类型。消息队列类和消息调度类是负责提供具体的消息服务的类，包括消息的维护与路由等，与 pvmd 提供的消息功能相同。

```
Class Message_Manage{
  Class Message_Object{ //基本消息类
    Unsigned Long message_id,
    Constructor create();
    Procedure Sendto_MessageQueue();
    ……;
  Class Message_Queue_Object{
    Unsigned Long Message_Queue_id,
    Struct Message_Queue{
      Class Buffer_Object *Buffer Pointer; //待发消息指针
      Struct Message_Queue *Pro;
      Struct Message_Queue *Next;
      ……}
      Struct Message_Queue *Head;
      ……;
    class Message_Schedule_Object{
      Procedure Message_Right(); //为消息赋予权值
      ……}
} //End of Message_Manage
```

(2)进程管理 其主要任务是进程的创建、终止、挂起、唤醒等，这些都与原来的 PVM 相同。同时，进程管理还应支持进程对象的迁移，当一个进程对象处于活动状态时，它所封装的各种属性就记录了对象的当前状态。当该对象被迁移到其它结点机上时，这些属性对于重新启动进程对象有至关重要的意义。为了支持这种迁移，进程管理必须维护一张表，表上记录了对象的名称和目的结点，其目的是针对被迁移的对象进行消息转发。

```
class Course_Manage{ //进程管理
    …… //进程创建、终止、挂起、唤醒、迁移等
    private struct Course_Move_Grid{
      string old_CourseObject_Name;
      unsigned long old_CourseObject_Handle;
      unsigned long Current_Processor_Handle;
      string Current_Processor_Name;
      ……} //End Struct Course_Move_Grid
} //End Class Course_Manage
```

(3)缓冲区管理 OOPVM 的缓冲区管理与 PVM 大致相同，但 PVM 中的缓冲区管理存在问题。当多个进程或线程交替运行时，由于不同的进程/线程使用不同的发送或接收缓冲区，但进程打包的数据缺省都装入活动缓冲区，且任一时刻，活动缓冲区只有一个，因此，在这种情况下很容易造成发送和接收的张冠李戴的错误。为了解决这个问题，我们决定把活动缓冲区的概念缩小到进程/线程的级别，即每个进程/线程都拥有自己的活动缓冲区，这个缓冲区是该进程/线程用 oopvm_mkbuffer()创建的多个缓冲区中的一个。缓冲区管理将维护一张表，表上记录了进程/线程对象与活动缓冲区的对应关系，以后，进程/线程发送接收数据的操作都将针对自己的活动缓冲区进行。

```
Class Buffer_Manage{
  Class Buffer_Object{
    Unsigned Long Buffer_id;
    String Buffer_Object_Name;
    Boolean Active;
    ……}
```

```
Class Active_Buffer_Grid{
  Unsigned Long item_id,
  String Course_Object_Name;
  String Buffer_Object_Name;
  ……}
Function buffer_mk(int),
……}
```

(4)*结点机环境管理*　主要包括一个结点机环境对象，利用该对象的属性和方法，用户可以方便地获取环境参数、如虚拟机的构成情况、结点机负载情况、进程/线程运行情况等。该对象包含了 PVM 的 pvm_config()、pvm_mstat()、pvm_task()等函数的功能，利用对象提供的方法还可以完成虚拟机的动态配置，如添加、删除结点等。

```
Class Pvm_Session{
  Char **nodes;
  Struct Node_Type{……}; //结点机的型号、配置等
  Function Add_Node();
  Function Delete_Node();
  ……} //End of Pvm_Session
```

上面是处理机对象的几个主要模块。在 OOPVM 系统中，通信层和处理机层将合起来构成系统的核心。每个处理机对象将对应一台结点机，在结点机上启动 OOPVM 的过程就是初始化处理机对象的过程，启动完毕后，处理机对象就运行在后台，用户可以使用控制台或系统提供的对象库来访问处理机对象。另外，由于在面向对象的系统中，对象的访问是通过名字进行的，因此处理机对象必须对从本机启动的所有任务对象及其子对象进行名字的绑定，以使用户可以通过名字透明地访问虚拟机内的所有对象。

3. 虚拟机服务层

虚拟机服务层，顾名思义是为用户编写应用程序、构造虚拟机提供服务的，并为用户提供了一个进行 OOP 开发的编程界面。在这一层，系统提供了可供用户继承的各种抽象类，如进程类、线程类、管道类、邮箱类、原子数据类和支持各种数据类型传递的类。用户在开发并行应用程序时，可以通过继承上面这些抽象类生成用户自定义的数据对象。用户开发并行性的过程就是操纵这些数据对象的属性和方法的过程。下面简要介绍一下这些类：

(1)*进程/线程类*　这两种对象类是提供给用户的可以并行的基本单元。用户继承这些类生成进程/线程对象后，只要完成对象的执行体的代码，父类的构造函数将自动处理虚拟机的并行资源的分配。构造函数通常是与处理机对象交互，获取虚拟机当前的配置信息和负载情况，然后将对象复制到合适的处理机结点上执行，复制的主要目标是使虚拟机负载平衡。当然，用户也可以通过重载构造函数来自定义复制策略，但在重载的构造函数中要首先运行父类的构造函数，同样地，在重载的析构函数中也要首先运行父类的析构函数。

```
class Thread{
    constructor create();
    destroy();
    run(); virtual //用户定义的执行体
    ……}
```

(2)*管道和原子数据类*　这两种类是为在分布存储的 COW 系统上模拟共享存储的计算模型而提供的。用户使用管道类可以生成管道对象，通过管道对象，不在同一结点机上的进程可以互访对方定义的变量。用户使用原子数据类可以生成原子数据对象。原子数据对象相当于一个全局共享变量，该对象有点像 CONST 类型的数据，一经产生后就只能写入一次，且任一时刻只能有一个进程能读，当有多个进程来访时，多余的进程将进入等待队列。如果该对象被写入新值，将会产生一个新的对象挂在原对象的后面，并被赋予要写入的新值，以后的写操作都将针对新的对象进行，原对象在等待队列为空后会被自动清除。

原子数据类是一个嵌套类，它内部包括了一个存放实际数据的原子类，我们称为 ADO。原子数据类的实例与其它对象实例一样，通过响应消息来为其它对象提供服务。原子数据类的算法如下：

算法1　原子数据类的主算法

第一步：初始化数据对象，申请空间；

第二步：进入处理消息的循环：

1. 接受消息 ReceiveMessage()；
2. Case DATA_SET

IF 当前 ADO 的等待队列为空 THEN

把当前 ADO 赋予新值；

ELSE

创建新的 ADO 对象并赋予新值，我们称它为当前 ADO；

将新的 ADO 对象放在 ADO 链表的末尾；

END；

3. Case DATA_GET

检查当前 ADO 对象的等待队列，若发现消息主体的对象句柄则忽略此消息；

检查其它 ADO 对象的等待队列，若发现消息主体的对象句柄则从队列中取掉该句柄，若该 ADO 对象的等待队列为空，则同时删除此 ADO 对象；

将消息主体的对象句柄加入当前 ADO 对象的等待队列；

END；

4. Case DATA_FREE

若某个 ADO 对象的等待队列不为空，则返回出错信息；否则，释放当前 ADO 对象，并退出消息循环；

END;

5. Case NO_MESSAGE(没有消息)

按照FIFO的原则,检索所有的等待队列,并向等待的对象返回对应的ADO的值;

End;

第三步:释放原子数据对象空间;

算法1终止。

这两个类都定义了一系列私有方法,用来控制多进程/线程通过网络透明地存取它们的实例。当用户使用这两种类时,通常是使用它们来说明一组变量,然后就可以像使用普通变量一样来使用这些对象实例。

(3)*邮箱类和各种消息类* 这些类为用户提供了多种消息传递的方式。邮箱类是一个全局类,主要用于异步消息的处理。各种消息类包括传递整型、实型、字符串等数据类型的类,用这些类可以定义各种待传的数据实例,然后调用统一的发送函数将它们发送出去。

4. 应用层

应用层就是用户的应用程序,它包含了用户继承或自定义的各种数据对象,这些对象与处理机对象相互作用,并行完成用户定义的任务。

三 框架分析

上面我们对OOPVM的框架作了描述,对于这个框架的结构、每一层所包含的类、类的用途都已有所了解,下面我们就来分析一下这个框架的特点:

1. 用户界面

由于整个框架的设计采用了对象的思想,因此,一方面,从程序设计的角度,用户可以充分发挥面向对象的程序设计语言(如C++)的优势,开发出扩展性、并行性俱优的应用程序;另一方面,从软件工程的角度,与传统的结构化程序相比,面向对象的应用程序非常易于维护和复用。我们可以把一些常用的并行算法做成对象库,当用户开发程序时,可以直接继承、使用这些对象,这样就可以大大减轻程序开发的复杂度。

2. 异步消息传递

原来的PVM提供了很多用于同步消息传递的函数,但对于异步消息传递的支持却不够好。在OOPVM中,系统提供了邮箱类用来支持异步消息传递。邮箱类的实例通常是一个全局对象,每个进程/线程对象都可以访问它。进程/线程通常是把异步消息发往邮箱对象,邮箱对象将负责这些消息的维护,要处理异步消息的进程/线程会定期地去检索邮箱,取回发给自己的消息,并对消息作出相应动作,从而完成一次异步消息的处理过程。

3. 进程/线程类

原来的PVM基本上是基于进程间并行的,进程的初始化、执行以及终止的代码都由用户完成,这使得用户的工作非常繁琐。引入进程类后,进程的初始化与终止都由系统完成,用户只要完成执行部分的代码,即只要重载进程类中定义的RUN()函数即可。同时,为了提高对处理机资源的利用率,OOPVM还提供了线程类,线程类与进程类有大致相同的结构和使用方法,但线程类通常是作为进程类的嵌套类来定义的。一个进程对象在运行时可以产生多个线程对象,并根据名称来访问这些线程对象,线程对象的分配在缺省情况下是由系统完成的,但用户可以通过重载类的构造函数来自定义对象的分配,当然,在线程类的构造函数中必须调用父类的构造函数。线程的引入充分提高了系统资源的利用率。

4. 对象迁移

原来的PVM在定义虚拟机时通常是一个任务对应一个虚拟机,这样当有多个任务运行在同一个COW系统上时,多个虚拟机重叠的部分负载就很重,从而使虚拟机出现计算调度失衡的情况。这是目前并行计算中常出现的问题,针对这个问题,人们提出了许多解决的办法,其中的一种办法就是在进程运行的过程中,将其动态地从一个结点"般"到另一个结点上去。对于进程来说,这种迁移比较麻烦,一方面要保持进程的现场状态,使得进程在迁移后能从原来的地方继续向下执行,另一方面要给进程重新分配任务标志TID,并能转发仍然使用旧标志发送的消息。引入OOP的机制后,对象的封装性为解决这个问题提供了便利。对于进程对象,RUN()函数中使用的变量都可以在对象中定义,这样就可以较容易地将进程的状态保存下来,并与进程封装在一起发送出去。另外,由于使用了对象名称来存取对象,这样,在对象发生迁移后,它仍然可以使用原来的名称,同时在对象迁出时要在迁出结点维护的迁移信息表中进行注册,并在迁入结点的对象信息中注册,重新绑定自己的名字,以利于消息的转发。

5. 对多种计算模型的支持

原来的PVM基本上只能支持消息传递的并行计算模型,形式单一。OOPVM通过引入几个附加类,从而扩展了PVM能够支持的计算模型。在前面我们介绍了两个类:管道类和原子数据类,使用这两种类,可以很容易地在COW系统这种分布存储的系统上模拟实现共享存储的并行计算模型。这样,用户在程序开发时通过使用不同的类的实例,可以将消息传递与共享存储这两种模型的优势充分结合起来,从而开发出并行性能较高的应用程序。

四 性能评价

从前述的框架描述与分析中我们可以看出,

OOPVM 由于引入了 OOP 的思想，并加进了多层次的、功能丰富的类，其系统维护的开销将明显比 PVM 要大。因此，对于小规模的并行应用程序而言，使用 OOPVM 并没有优势，相反还会降低程序运行的效率。但当问题的规模增大时，使用 OOPVM 可以充分挖掘 COW 系统的资源，并能采用多种手段调度对象在不同的结点机上运行，从而提高了解决问题的效率和速度，同时，支持多种计算模型的对象机制的引入，也扩大了 OOPVM 能够解决问题的范围。

在"数据库并行查询核心系统"的研制过程中，我们利用二元及多元连接查询对 OOPVM 平台的性能进行了测试。在多个结点上进行二元及多元连接并行查询前，对分布在多个结点上的数据重新进行 Hash 划分的通信时间是整个查询时间的重要组成部分，为了缩短这个重划分时间，我们使用了 OOPVM 来实现 Hash 划分。下面是二元和三元带条件连接查询中使用 OOPVM 前后的数据重划分时间比较。

test1、test2和 test3分别是含有10万、20万和30万个元组的数据表，unique1和 onepercent 是元组属性。

(1)二元连接

查询语句：select * from test1，test2 where test1. nnique1 = test2. nnique1 and test1. onepercent=0；

查询结果：1 rows selected

结点数	使用 OOPVM 前	使用 OOPVM 后
1	42.770s	44.520s
2	23.186s	24.220s
4	11.446s	9.182s
8	5.202s	3.550s

(2)三元连接

查询语句：select * from test1，test2 where test1. uniquel = test2. uniquel and test2. uiniquel = test3. uniqnel and test1. onepercent=0；

查询结果：1 rows selected

结点数	使用 OOPVM 前	使用 OOPVM 后
1	86.820s	87.471s
2	48.172s	48.220s
4	21.982s	15.103s
8	12.540s	7.222s

从上面的重划分时间比较中我们可以看出，随着划分规模的扩大以及结点数的增多，使用 OOPVM 平台可以有效地提高划分效率。

在仿真模拟方面，OOPVM 也具有较大的优势，使用 OOPVM 的程序员会很自然地采用面向对象的方法来建立问题的模型，这种模型可以把现实世界的组织结构在计算机上再现，并且直观、易懂。现实世界中各种事物间的联系与影响可以通过对象间的消息传递与共享来实现，OOPVM 为程序员在 COW 系统上实现面向对象的模型提供了良好的平台，因此对于模拟现实世界中的并行成分较大的问题，OOPVM 有其独特的优势。

结束语 PVM 是目前应用较为广泛的一种并行程序开发工具，其核心 PVMD 具有稳定的性能、并能支持广播、组通讯、动态组、多虚拟机重叠等功能，因此获得众多程序员的青睐。OOPVM 的引入对原来的 PVM 作了根本的改变，克服了它原有的一些缺点，但同时也保留了它的很多优点。从长远的角度看，OOPVM 也不是解决问题的最终办法，并行应用程序执行的效率和代价最终还要操作系统的支持，如果操作系统能对可并行的资源进行有效的维护与管理，无疑将大大降低并行应用程序设计的难度，并增强并行程序执行的效率。因此可以考虑将 OOPVM 嵌入操作系统的核心中，将 OOPVM 原有的一些对并行资源的管理功能，如对象的动态迁移、虚拟机状态采集等，转移给操作系统来完成，这样可以极大地精简 OOPVM 的系统结构，对于降低 OOPVM 系统的运行成本具有重要的意义。

参 考 文 献

1 Goscinski A M. Finding，expressing and managing parallelism in programs executed on clusters of workstations. Computer Conmunications，*1999*，*22*：*998～1016*

2 PVM 3 User's Guide and Reference Manual. Jan. 1994

3 MPI：A Message-Passing Interface Standard. May 1994

4 Hempel R. The Status of the MPI Message-Passing Standard and Its Relation to PVM，Parallel Virtual Machine-EuroPVM'96，Springer，Berlin，1996

5 MPIF，Message Passing Interface Forum，MPI：A Message Passing Interface Standard ·［Computer Science Technical Report CS-94-230］. University of Tennessee，April 1994

6 Snir M，et al. MPI：The Complete Reference. MIT Press，Cambridge，MA，1996

7 Geist G A. Advanced Programming in PVM. Parallel Virtual Machine-EuroPVM'96，Springer，Berlin，1996

8 Jia W. Communicating object group and protocols for distributed systems. The Journal of Systems and Software，1999，45：113～126