层次式面向对象并行环境中并行支撑层的实现*)

The Implementation of Parallel Supporting Layer in the Hierarchical Object Oriented Parallel Environment

刘建中 马晓星 蔡 敏 李英军 吕 建

(计算机软件新技术国家重点实验室 南京大学计算机软件所 南京210093)

Abstract Hierarchical Object Oriented Parallel Environment(HOOPE) provides an easy-to-use, highly efficient and portable parallel developing environment for domain-specific applications. One of the key issues in the construction of this parallel developing environment is the implementation of the parallel-supporting layer, which makes the domain applications in the upper layers independent of the specific computer architecture in the lower layer. We implement this layer in the form of a parallel array class library with which users can write parallel programs in a way similar to writing serial programs. This paper describes the design and implementation of this array class library as well as the testing and performance analysis. Our development experiences show that the parallel-supporting layer is highly efficient and supports the developing of HOOPE well.

Keywords SPMD, MPI, Virtual Shared Grid(VSG), Ghost Boundary

一、引言

在石油勘探、气象和航空航天领域,大规模并行计算已经得到了广泛的应用。传统的并行应用开发方法直接使用底层的并行开发环境,开发者需要考虑到异构的硬件体系结构和软件平台等细节,并了解底层的消息通信机制,才能够进行实际的开发工作,往往使得程序代码不够清晰,调试起来也相当困难,因而开发效率较低。我们以油气勘探领域三维叠前深度偏移应用为前景,设计并开发了层次式面向对象并行环境HOOPE[1~3],以期为领域并行应用开发提供一个良好的并行开发环境,HOOPE 系统的目标是:(1)简化并行应用的开发过程,缩短开发周期;(2)提高并行应用软件的质量和运行功效;(3)便于应用程序在不同的并行体系结构之间移植;(4)提供一个具有良好的开放性和可扩充性的并行框架。图1是 HOOPE 系统的基本结构。

从 HOOPE 层次式体系结构图中可以看到,并行支撑层对底层的并行通信环境与高层的应用层、功能组件层和抽象数据类型层进行分离,HOOPE 体系结构采用逐层代理机制。并行支撑层直接使用消息传递接口在处理器节点之间进行数据传递(通信管理);对数据进行分割并将它们分配到各个处理器节点上,在这些节点上进行真正的数据运算(虚拟节点管理)。也就是说,并行支撑层实现这些底层的并行细节。而抽象数据层则直接使用并行支撑层,无需使用底层的MPI[4]。功能构件层使用抽象数据层中的抽象数据类型完成特定的计算任务;应用层则通过组装若干个功能构件体实现某个领域中的应用。这两层也可以直接使用并行支撑层。一般来说,开发者仅用到上面的四层,即涉及领域的应用层、功能组件层和抽象数据类型层,以及并行支撑层,而无需直接使用 MPI。

由上面的分析可见,在 HOOPE 系统中,并行支撑层非常关键。整个系统的实现的好坏很大程度上决定于并行支撑层的实现。首先,简化并行程序开发过程很重要的一个方面就是,让开发者在编写并行程序时不必过多地涉及底层的并行细节,而在 HOOPE 中,并行支撑层提供了并行数组类库,用户可以用类似于编写串行程序的方式编写并行程序,这样就大大地提高了编写并行程序的效率。第二,高层的并行组件使用并行数组类库直接编写,因而这些构件的质量和运行效率很大程度上决定于并行支撑层的质量和运行效率。第三,并行支撑层分离底层的并行细节和高层并行应用的实现,只要对不同体系结构的机器实现相应的并行支撑层,基本上就可以不修改高层的并行组件和应用程序,在不同体系结构的机器间进行移植。第四,由于并行支撑层保证高层组件和应用的平台无关性,使得HOOPE并行框架可以不断地增加新的高层应用组

*)本文受国家自然科学基金项目(69873021)和国家杰出青年基金项目(61525204)资助。

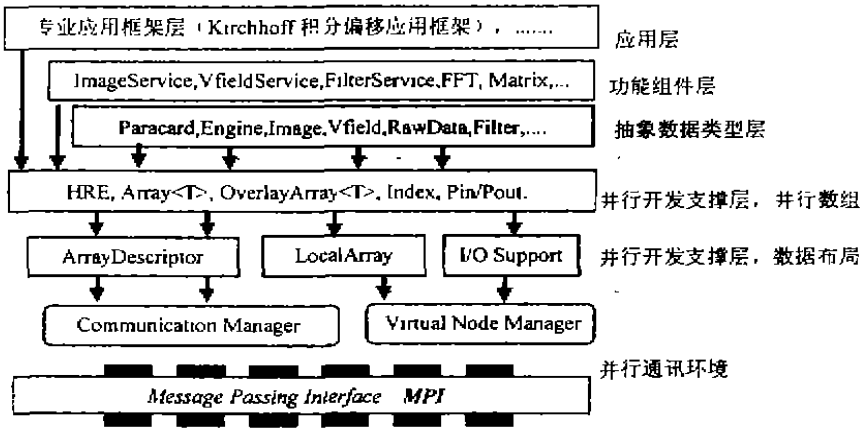

图1 HOOPE 应用框架层次式体系结构

件，也就保证了 HOOPE 并行环境的开放性和可扩充性。下面我们就对并行支撑层的设计和实现进行详细的讨论。

二、HOOPE 并行数组类库的设计

并行支撑层以并行数组类库形式实现，这是考虑到大规模科学计算中，尤其是油气勘探领域中，所涉及的主要程序设计模型是单程序多数据流模型（Single Program Multiple Data Stream，SPMD）和规格化网格的问题，因而数组是一种最通用的数据结构，可以在数组的基础之上构造其他复杂的数据结构。SPMD 模式是指在并行计算系统的各个节点上运行相同的程序代码，但作用于不同的数据这样一种并行处理方式，这种并行计算模型适用于很多科学工程计算问题，例如，油气勘探领域的旅行走时表和偏移成像计算都采用 SPMD 计算模型[5]。围绕着并行数组类库，我们还提供了其他一些并行计算中经常使用到的功能，例如并行输入输出等功能。这样既保证了并行计算领域所需要的一些基本功能都得到了实现，使得并行数组类库精简易用，可以达到较高的运行效率，又可以让用户比较容易地扩充数组类库，增加所需的功能。

2.1 设计目标

HOOPE 并行数组类库的设计目标是：首先，要给应用开发者提供一个简单易用的并行开发环境，屏蔽底层的并行细节，用户使用经过抽象的并行数组，可以用串行的方式开发并行应用程序，应用程序无需进行任何修改即可在并行机上运行。第二，高层应用领域的程序直接使用并行数组类库开发，不涉及到底层的并行细节，因而可以移植到不同体系结构的并行机上，保证了应用程序具有良好的可移植性。第三，应保证 HOOPE 并行数组类库具有较高的运行效率。第四，用户可以根据实际需要对 HOOPE 数组类库进行一些扩充，保证其可扩充性。

2.2 系统结构

运用 Envelope-Letter 设计模式，我们设计了并行数组的结构，如图2所示，用户仅能看到 SPMD 意义下全局并行数组对象，它为用户提供了全局数组分布及操作等功能。数组描述对象负责全局概念与局部概念之间的映射，包括全局坐标与局部坐标之间的转换，数组操作所需通信策略的制定等。局部数组负责管理局部数据的存储和局部数组操作。一个数组描述对象可能与多个具有相同分布的并行数组相关；一个局部数组也可能与多个并行数组相关，这些并行数组同享一块数据存储，是同一个数组的多个视图（视图也是合法的数组对象）。

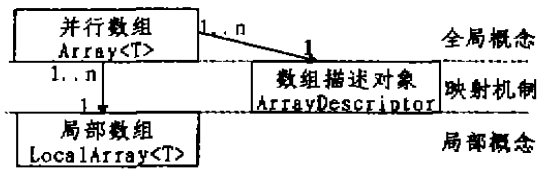

图2 并行数组结构

并行数组各部分运行时的协作原理如图3所示。图3中各个动作应以 SPMD 的语义理解，b 与其它节点通信应解释为各节点协作执行通信计划。通信结果将下步执行局部数组操作所需的非本节点数据预先取到一个缓冲区里，这样 c 执行局部数组操作时就无需进行零散通信了。这不仅简化了程序设计，更重要的是有效地提高了通信效率——在通信总量不变的情况下，一次通信要远好于多次零散通信。由于涉及的数据有时很大，为节约缓冲区空间，b 和 c 可以交叉执行。

2.3 主要功能

HOOPE 数组类库主要包括以下一些功能：

·数组的创建和分布；

·数组的向量索引与标量索引；

·数组的运算操作，包括基本的加减乘除运算以及赋值运算；

·数组的逐元(elementwise)运算，即将对数组对象的操作映射为对各个元素的操作，例如对数组 A 和函数 sin，A.elementwise(sin)是求数组 A 的所有元素相应的正弦值；

·数组的聚合(aggregate)操作。求数组所有元素的和、所有元素的最大值和最小值；

·数组的并行输入/输出操作。提供数组的文件并行输入/输出支持。例如数组对象可以从一个 Pfin 对象(对应一个输入文件)读入数据，通过 saveData 方法将数据保存到一个 Pfout 对象(对应于一个输出文件)中。

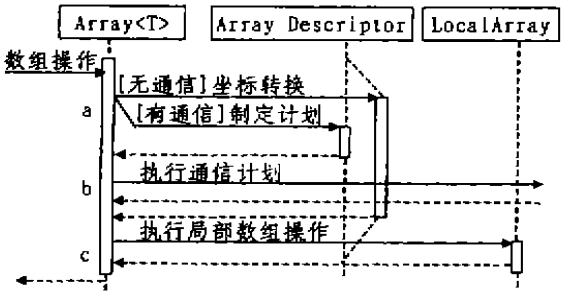

图3 并行数组动态时序图

2.4 主要技术

(1)虚拟共存网格(VSG) 高性能并行计算机体系结构发展的趋势是从共享内存设计转向分布式内存设计，这就需要通过消息传递机制显式地进行并行控制，使得程序设计环境变得非常复杂。主要问题是缺少很好的共享存储设计模型[4]。虚拟共享内存(Virtual Shared Memory)模型试图无需对程序的算法结构和处理器以外的数据进行显式的假定的情况下即可支持并行，但这往往导致对处理器以外的数据大量的存取，使得性能不高。我们将虚拟共享内存模型限定于规则的网格结构之上，有关数组的计算基于全局的下标，通信模式在运行时刻确定(为提高效率可能会使用一些调度策略作预先缓存)，所需的消息由并行数组类库自动生成[7]。用户使用数组语法可以向编译器和运行系统提供有关数据结构(数组结构)和算法(数组操作)足够的信息，这些信息使得数组类库可以将通信次数和通信数据量最小化，从而可以达到与显式的消息传递几乎相同的效率。在这一点上，虚拟共享网格相对虚拟共享内存具有很大的优势。

并行应用框架中的并行数组类涉及两种基本通信模型：

·虚拟共享网格更新(VSG-Update)。在实现虚拟共享网格时，采用所有者计算规则，对于二元运算指定左操作数为所有者，这种简单的规则可以处理并行环境下计算所需的通信。例如 A＝B＋C，其中 A，B，C 三者的数据分割各不相同，执行时首先构造一个与 B 分割一致的临时数组 T，然后再由拥有 B 数据的各个节点并行执行加操作。若处理器 P_i 上有 B 的部分数据 B_i，而与 B_i 相应的 C 的部分数据 C_i 可能不在 P_i 上，此时，通过消息传递把 C_i 从其他处理器(可能有多个)上发送过来，由 P_i 执行 $T_i=C_i+B_i$，然后执行 A＝T，同样由拥有 A 数据的各处理器执行赋值运算(见图4)。

```
HOOPE 用户定义：
Array(int)A,B,C; A=B+C
HOOPE 内部实现：
1. 创建临时数组 P1.T11,T12,P2.T2
   T=B+C
   P1.T11=B11+C1,接收 P2数组 C21
      T12=B12+C21
   P2:T2=B2+C22,传送 C21到 P1
   P3:空闲
2. 收集结果
   A=T
   P1.传送 T1到 P3
   P2.传送 T2到 P3
   P3:从 P1接收 T1
      从 P2接收 T2
   A=T
```

图4 VSG 虚拟共享网格的 Owner 计算规则示例

·重叠更新(Overlap Update)。使用上述 VSG 更新时，每次二元数组运算都要进行通信。在不对齐(即各节点上的数据分割方式不一致)的数组间进行数组运算时，可能不得不采用这种通信模式。但如果参与运算的数组是对齐的(这种情况经常出现)，这种通信模式显然不合适。我们采用的技术是：使数组分割的数据块在其边界上与其相邻的数据块重叠一部分，那些与相邻数据块重叠的边界称为虚拟边界(Ghost Boundary)。它们实际上是相邻节点上(部分)数据的只读拷贝。由于许多运算主要与相邻元素有关，而这些相邻元素可以直接从虚拟边界中读取，因此就可以避免很多通信。只有在进行赋值操作以后，才根据需要更新各个虚拟边界，以使它们与实际的数据保持一致(见图5)。采用虚拟边界策略，在许多对齐数组操作情况下，一条数组语句才进行一次通信，而不是在每次二元数组运算中都进行通信，从而大大提高了并行数组运行的效率。

基于 VSG 的并行数组分布在各个处理器上，数组内部需要保存数据分割及其与各处理器的对应信息，并处理虚拟边界的刷新以及 VSG 更新所需的不规则数据传递。通过综合使用上述两种通信模型(如果所需数据在虚拟边界中，则使用 Overlap 模型，否则使用

VSG 模型)，并行数组类库运行系统可以自动高效地进行所需的通信。

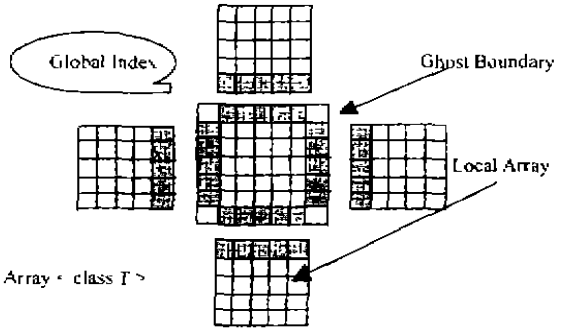

图5 并行数组 VSG 重叠更新

(2)内存管理 并行数组实现中一个很重要的问题是临时数组对象管理。在表达式 A=B+C 中，需要建立一个临时对象 Temp 来保存 B+C 的结果，然后再将这个临时对象赋值给 A，A=Temp。这里我们可以对赋值操作进行优化：将 Temp 的原始数据指针拷贝给 A，而不真正逐一拷贝原始数据本身。此时临时对象管理需要删除临时对象 Temp(不删除其原始数据，而要删除 A 的原始数据)。对临时对象生命周期的管理非常重要，如果没有恰当的管理，临时对象就会累积起来，浪费大量的内存资源。

临时对象管理使得执行数组语句时可以保留尽量少的临时对象。临时对象的生命周期由并行数组操作函数主动控制。在进行数组对象运算时，可察看是否有临时对象参与，若有，则可采取更高效的处理。例如，在计算 A=B+C+D 时，C+D 产生一个临时对象 Temp，而在做 B+Temp 时就不再生成新的临时对象而将 Temp 当作累加器重复使用。接下来执行赋值时，发现 Temp 是临时对象，于是拷贝其原始数据指针，并删除(空的)Temp 对象。这样，整个表达运算结束时没有任何临时对象累积下来。通常，一个数组表达式的计算过程中最多只生成一个或两个临时对象，而在(分离的)表达式之间不会有临时对象的积累。这一点对于较大规模的数组计算来说是非常重要的。

三、测试及性能分析

在 Sun HPC3000分布式共享内存并行计算机上，我们使用 Sun 的 MPI 并行库实现了 HOOPE 并行数组类库，并对它进行了测试。测试条件为 Sun HPC3000，4 CPUs，主频248MHz，内存512M，磁盘容量18G，网络传输速率为1000 Mbits/sec。

首先我们对HOOPE 并行数组与C 语言编写的串行程序进行了基本功能对比测试，包括算术运算、逐元

操作	数组相加	数组相乘	数组与数相减	数组与数相除	数组求和	求数组最大值	逐元操作	索引操作
串行 C	176.31	176.16	81.44	116.24	86.63	105.67	52.22	76.61
1CPU	180.32	179.92	106.45	151.27	102.58	108.58	87.70	97.27
2CPU	89.52	89.54	53.11	74.15	51.75	54.39	43.97	49.50
3CPU	60.32	60.41	35.84	48.69	33.21	36.38	30.33	33.01
4CPU	44.83	45.43	26.83	35.91	25.36	28.31	22.53	25.20

图6 基本运算测试结果

运算、数组的聚合操作、索引操作等，图6给出了部分结果。网格大小为12000×1200，数据类型为双精度，为测试精确，对每一种操作都重复循环100次，表格中数据的单位为秒。由表中数据我们可以看到，HOOPE 并行数组的加速比很高，随着处理器数目的增加，部分基本操作甚至出现了非线性加速比。原因主要是，一方面因为 HOOPE 并行数组采用了虚拟共享网格等技术，另一方面随着处理器数目的增加，处理器 cache 的作用越来越大，从而导致非线性加速比的出现。因为 HOOPE 数组需要一部分代码额外处理节点之间的通信，所以一般情况下一个 CPU 时，HOOPE 数组比串行的 C 语言编写的数组操作要稍慢。

另外我们以一个科学计算中经常用到的两维 Laplace 方程

$$d^2V/dx^2+d^2V/dy^2=\beta$$

为例，对它用 Jacobi 迭代法进行求解，对 HOOPE 进行了综合测试，网格大小为300×300，单精度，迭代次数为100次，根据加速比计算公式：

$$S=T_1/T_n$$

我们计算了相应的加速比，结果如图7所示。

从以上测试我们可以看到 HOOPE 并行数组类库具有较高的并行效率，加速比也很高，在效率这一方面已经达到了设计目标。

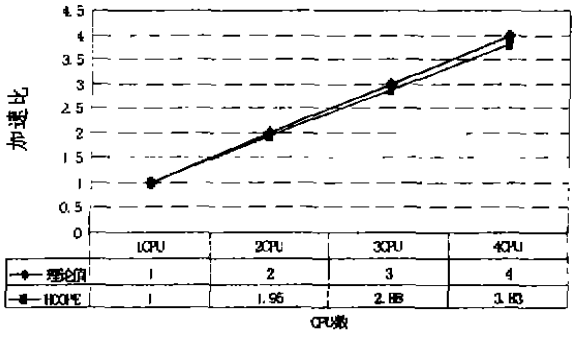

图7 求解 Laplace 方程的加速比比较

结论 上面介绍了 HOOPE 并行支撑层的主要技

(下转第24页)

做一简单分析。假一数据库大小为 M 块，读集合和写集合大小为 B 块，则两个事务发生数据冲突的最大可能性为[5]：

$$1-\left(\frac{M-2B+1}{M-B+1}\right)^{B}$$

给 B，M 赋几个样值，我们得到下表（如图3）：

B	M	冲突几率 P(C)
5	100	0.0576
10	500	0.0025
100	1,000	0.1130

图3 冲突几率表

以上的数据说明，乐观方法在大多数情况下是行之有效的。就算法本身而言，控制器的效率主要由冲突检测算法和刷新无效块算法的时间复杂度来决定。冲突检测的时间与事务数和更新链上的块数有关，假定事务数为 N，事务更新链上的块数最多为 L，则冲突检测的时间复杂度为 $O(N*L^2)$，刷新无效块的时间取决于无效块的块数，时间复杂度为 O(L)。

控制器的调用也影响着系统的效率。冲突检测模块的执行时间较长，但只在每个事务提交前调用一次；刷新模块则调用频繁一些，每当事务做查询操作都需调用一次，但其时间消耗不多。因此，整体上看，控制器提高了系统的效率。

结束语 本文在分析了索引并发控制难点的基础上，提出了一种实现策略，并在 DM2 上实现。本算法具有普适性，不仅适用于 B^n 树，而且也适用于其它索引结构如 B^+ 树等。

参 考 文 献

1 冯玉才. 数据库系统基础(第二版). 华中理工大学出版社，1993
2 DM2概要设计. 华中理工大学计算机学院数据库与多媒体技术研究所，1998.6
3 Silberschatz A, Korth H F. Database System Concepts (Third Edition). McGraw-Hill Press
4 Yu P S, Dias D M. Performance Analysis of Concurrency Control Using Locking with Deferred Blocking. IEEE Trans. Software Eng., 1993, 19(10): 982～996
5 Bhargava B. Concurrency Control in Database System. IEEE Trans. Knowledge and Data Eng., 1999, 11(1): 3～16
6 Kung H T, Rbinson J T. On Optimistic Methods for Concurrency Control. Trans. Database System, 1981, 6(2): 213～226
7 Davidson S B. Optimism and Consistency in Partitioned Distributed Database Systems. Trans. Database System, 1984, 17(3): 456～481
8 Gray J N. The Transaction Concept: Virtues and Limitations. In: Proc. VLDB Conf., Cannes, France Sept. 1981
9 Gray J N, Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, Calif., 1993

（上接第28页）

术、系统结构和主要功能，并对它进行了基本的测试和性能分析，我们认为 HOOPE 并行支撑层已经达到了设计目标，它具有以下一些特点：

首先，HOOPE 并行支撑层提供的并行数组类库具有易用高效的特点，开发者可以用类似于 FORTRAN90的语法和编写串行程序的方式编写并行程序。

第二，并行数组类库的设计使用了虚拟共享网格和虚拟边界等技术，使得 HOOPE 并行数组类库具有较高的运行效率，这样就保证了上层用 HOOPE 数组类库直接进行开发的应用程序具有较高的运行效率。

第三，由于上层的应用组件和应用程序直接使用 HOOPE 并行数组类库进行开发，基本上不涉及底层的并行细节，只需根据不同的体系结构实现相应的并行支撑层，就可以将这些组件和应用程序移植到不同体系结构的机器上。

第四，HOOPE 并行数组类库用 C++ 实现，采用了模板等面向对象技术，根据实际需要，开发者可以很容易地对它进行扩充。

HOOPE 并行支撑层还有一些方面的不足，例如，目前仅支持 SPMD 计算模式和规则的网格结构。在实现三维叠前深度偏移应用的过程中，我们看到，一些串行应用程序中的算法使用并行数组类库可以很容易地转化为并行应用程序，这种方法不同于用并行编译器对串行程序进行并行编译的做法，HOOPE 并行数组类库仅需标准的 C++ 编译器即可；也不同于完全用特定的并行语言编写应用程序的做法，而是用类库的方式提供了类似于串行语言的使用方式，因而使用起来更为灵活，更有利于将一些串行程序移植为并行应用程序。这种思路对于应用领域中利用已有的串行算法进行并行应用开发具有一定的指导意义。

参 考 文 献

1 李英军，吕建，王宏琳. 面向对象应用框架在油气勘探领域的应用研究. 软件学报，1999，10(4)
2 李英军，吕建. 一个科学计算领域的面向对象并行应用框架. 计算机工程与科学，1998，20(3)
3 李英军，吕建，于大川，马晓星. 一个层次式面向对象并行计算框架的设计. 电子学报，已录用
4 张林波，迟学斌，汪道柳，等. 网络并行计算与分布式编程环境. 科学出版社，1996
5 Li Yingjun, Lu Jian. Framework-Based Software Reuse for Seismic Processing Applications. In: Proc. of Technology of Object-Oriented Languages and Systems, TOOLS 31, IEEE Computer Society, Los Alamitos, California, 1999. 22～25
6 Gropp W, Lusk E, Skjellum A. Using MPI, Portable Parallel Programming with the Message-Passing Interface, The MIT Press, 1994
7 Lemke M, Schüller A, Solchenback K, Trottenberg U. Parallel processing on distributed memory multiprocessors. In: Proc. GI-20. Annual meeting 1990, Informatik Eachberichte Nr. 257, Springer, 1990