一个基于消息传递接口和面向对象的图像信号处理并行向量库*

姜春茂[1]，吴翔虎[2]，段　莹[3]，曲明成[2]，刘　敏[2]，李志聪[1]
(1.哈尔滨师范大学 计算机科学与信息工程学院，哈尔滨 150025；2.哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001；3. 哈尔滨商业学校，哈尔滨 150001)

摘　要： 传统并行软件系统的设计和实现存在着开发效率低、质量难以保证和可移植性差等问题。针对这些问题，采用开发标准并行库的方法加以解决。借鉴高性能嵌入式计算软件计划(high performance embedded computing software initiative，HPEC_SI)的解决方法，基于消息传递接口(message passing interface，MPI)的消息传递机制，对图像/信号处理中的一些典型并行算法以类组件的方式进行封装，设计和实现了具有面向对象特征的、用于图像/信号处理的并行向量库，提供给应用软件开发人员一个良好的开发环境。通过测试和实验证明，该库可以高效地实现相应的向量矩阵并行算法，并具有简单易用、可复用性和可移植性强、效率高的特点。

关键词： 面向对象；消息传递机制；层次型结构；并行算法
中图分类号： TP311　　**文献标志码：** A　　**文章编号：** 1001-3695(2012)12-4560-04
doi:10.3969/j.issn.1001-3695.2012.12.041

Design and implementation of image/signal processing parallel library based on MPI and object-orientation

JIANG Chun-mao[1]，WU Xiang-hu[2]，DUAN Ying[3]，QU Ming-cheng[2]，LIU Min[2]，LI Zhi-cong[1]
(1. *School of Computer Science Technology & Information Engineering*，*Harbin Normal University*，*Harbin* 150025，*China*；2. *School of Computer Science & Technology*，*Harbin Institute of Technology*，*Harbin* 150001，*China*；3. *Harbin Commercial School*，*Harbin* 150001，*China*)

Abstract： There are many problems in the design and realization of traditional parallel software systems，such as the low development efficiency，the difficulty to ensure the quality and the poor portability. This paper developed the standard parallel library to resolve these problems. This paper used the idea of HPEC_SI(high performance embedded computing software initiative) for reference，based on the message passing mechanism of MPI(message passing interface)，enveloped some typical parallel algorithms in image/signal processing domain as classes，designed and finished an object-oriented，image/signal processing standard parallel library to provide application software programmers an excellent development environment. It proves that the matrix-vector parallel algorithms can be implemented effectively by the library through experiments，and it's easy to use，portable and reusable greatly and effective.

Key words： object-oriented；message passing mechanism；hierarchical framework；parallel algorithm

传统的并行软件系统的设计和实现存在着开发效率低、质量难以保证、可移植性差等问题。一个新的系统以及一个系统在不同运行平台上的新版本，往往要从头重新设计开发，系统代码难以复用[1,2]，专业软件开发人员需要掌握较多的并行编程知识，开发周期过长，产品的质量也很难得到保证，可维护性和可移植性差。

VSIPL 和 MPI 是当前技术比较成熟的工业标准[3]，MPI 在矩阵运算的并行算法方面得到了一些研究[4]。VSIPL 是基于对象的，仍不是面向对象的 API，它不支持 C ++ 面向对象的编程方法。因此，在具体应用中使用 VSIPL 会使代码显得复杂难懂，且不符合人们习惯的思维方式。另外，它没有对并行的支持，只能应用在一些单处理器的系统中，因此对性能有了一定的限制，而目前多核处理器及其系统软件已经应用得非常广泛[5]。在两者基础上进行的并行应用软件的开发虽然保证了一定的性能，但是由于软件开发人员需要考虑并行的细节而带来了开发效率过低；另外，可移植性也不能达到令人满意的效果。

本文借鉴两者的思想和优点并加以结合，取长补短，考虑开发一个并行库，在用户和并行机之间架设一座桥梁：一方面使用户可以把精力集中到所研究的专业领域上，无须考虑这些计算在并行机上是如何实现的；另一方面使用户程序不受系统结构的影响，开发出可移植可扩展的并行程序。

1　并行库的总体设计

本文采用层次式的结构模式。层次式结构的主要结构关系如下：系统中的每个层次均承担相对独立的任务，每一层次均可由不同需要的用户使用，上面层次可以使用下面层次的对

收稿日期： 2012-05-07；**修回日期：** 2012-06-14　　**基金项目：** 黑龙江省自然科学基金资助项目(F201139)

作者简介： 姜春茂(1972-)，男，辽宁庄河人，副教授，博士，主要研究方向为多核操作系统、移动 P2P、云计算等(hsdrose@126.com)；吴翔虎(1968-)，男，教授，博士，CCF 高级会员，主要研究方向为嵌入式计算、空间计算、操作系统、网格计算；段莹，讲师，硕士；曲明成(1980-)，男，博士，主要研究方向为网格计算、嵌入式计算、企业智能计算；刘敏，女，硕士；李志聪(1972-)，男，副教授.

象来完成其特定的功能,而下面层次的对象主要用做上面层次的支撑环境[3,6,7]。将并行向量库划分为四个层次,即应用组件层、抽象数据层、数据划分层和并行支撑层,如图 1 所示。

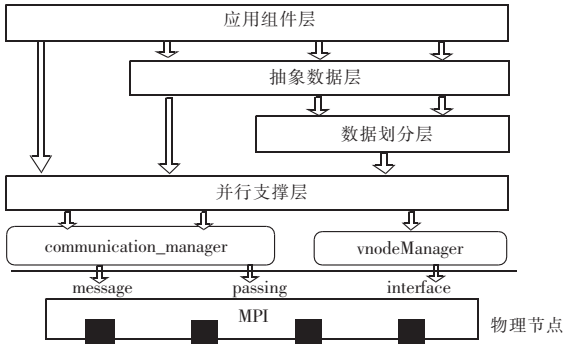

图 1　并行向量库层次型体系结构

并行向量库利用层次式结构,可使开发人员直接使用应用组件层和抽象数据层,像开发顺序程序一样来构造并行应用程序而不必涉及具体的并行实现细节。开发人员可以利用数据划分层对抽象数据层的类对象的划分进行描述,也可直接使用并行支撑层进行应用程序的开发,或用其增加新的面向领域的数据类型及其相应操作,以扩充抽象数据层乃至应用组件层。这样就为并行应用程序开发人员提供了一个具有易使用性、易复用性和易扩展性的新型并行应用开发环境。

层次式体系结构的主要特点如下:

a)其分层结构有利于将并行应用开发的任务分解。并行向量库主要集中于软件系统的低级层次,而应用开发者则可直接使用上面层次的功能来构造并行应用软件,从而可简化并行程序开发过程,缩短并行应用软件开发周期。

b)由于并行应用软件通常基于应用组件层和抽象数据层,因此只要将并行支撑层稍作修改即可在顺序、分布和并行平台上运行而不必修改源代码,从而便于应用程序在不同的并行体系结构间的移植。

c)由于并行应用软件常用的关键组件已集成在并行向量库之中,因此可集中采用现有的各种测试、验证和优化技术来保证其质量和运行功效,从而提高并行应用软件的质量和功效。而下面层次的对象主要用做上面层次的支撑环境。例如应用组件层中的并行组件主要是利用抽象数据层中所提供的领域数据类型及其操作,根据领域的需要加以并行组合而成;抽象数据层中的高级领域数据类型及其相关操作的并行实现则基于并行支撑层中所提供的各种机制;而并行支撑层的实现则基于 MPI 环境。

2　并行库的详细设计

2.1　抽象数据层

抽象数据层包含了应用领域并行数据类型,它仅仅是一组带有并行数据划分信息及其操作符的被封装对象。它将其信息和操作传递给数据划分层和并行支撑层进行数据划分、数据通信和数据操作符实现。

它的设计采用面向应用领域的方法,定义该领域相关的高级数据类型,可以根据图像/信号处理中并行数据的具体特点设计出不同的抽象数据类型。本文定义的抽象数据类型主要有两类:一类属于不可并行类型,即标量(scalar),实际上就是通常使用的基本数据类型;另一类是可并行类型,即数组类(view),根据维数的不同分为不同的实现。抽象数据类型 view 类有两种实现,分别为向量(vector)和矩阵(matrix),它们都属于模板类,需要在模板参数列表中给出元素的数据类型。元素的数据类型可以是上面定义的不可并行类型的一种,用户在定义抽象数据类型(可并行类型)时,系统会根据数据类型的不同给出相应的类的实现。每个类的数据成员包括维数、域说明、数据映射信息、存储数据块和其他变量,用来表示抽象数据类型的相关属性。其中,向量和矩阵相应的维数被赋值为 1 和 2。

这里重点解释一下域说明(domain)。

Domain 是一个用于描述数据对象规模的类,代表了数据元素索引值的逻辑集合。一维的 domain 类对象可以描述一维的数据对象(vector)的范围,在构造它时需要对它的起始索引值、步长以及长度进行赋值。为了方便起见,如果构造函数中只提供了长度值,则起始索引值默认赋值为 0,步长默认赋值为 1。二维的 domain 类对象用来描述一个二维的数据对象(matrix)的范围,它实际上由两个一维的 domain 类对象组成;同理,三维的 domain 类对象则可由三个一维的 domain 类对象组成,用来描述一个三维的数据对象的范围。其中维数(dim)作为一个模板参数出现,用户在定义不同的维数参数时系统就会调用相匹配的 domain 模板类构造函数。在后面的数据划分和分配中,依然要根据 domain 类对象的具体描述来实现。它可以用来描述数据对象的全局域,也可以用来描述数据对象在某个处理器上的局部域。

域说明描述了数据对象各方向上的范围,数据映射信息则提供了数据对象各维上的划分信息。Map 类用来表示数据映射信息,它描述了数据对象在每一个方向上是如何分配的,从而把每个数据对象划分成子块;同时还规定了每个子块是如何分配给每个进程的。它是由数据划分层提供的类,本文将在后面作详细说明。

在知道某个进程标志的条件下,通过数据映射信息可以得到数据对象在某个进程中的局部域。可以根据这个域来分配一个局部的内存块来存储本地的局部数据,存储数据块描述了指向这个存储空间的指针。每个进程中只拥有本地的数据块,这样就实现了数据分布。但是每个进程只能对本地的数据进行直接访问和赋值,当需要访问其他进程中的数据时,需要通过并行支撑层提供的功能来实现。

另外还为抽象数据类型提供了访问和赋值操作。这里采用了统一地址空间编程的风格,用户可以根据需要对数据对象进行处理,无须考虑具体的实现细节。并行库可以调用并行支撑层的一些功能对数据对象的元素进行处理。对于一些其他的操作,则会根据类型的不同而有所不同。

2.2　数据划分层

数据划分层主要封装了数据对象的并行划分信息,它可以根据划分信息对数据进行分块,同时调用并行支撑层的一些功能,并提供一定的映射机制分配给相匹配的处理器,为抽象数据层提供并行数据对象[8]。

Map 类是数据划分层的主要类组件,Map 类对象保存了必要的信息用来进行地址的转换。

2.2.1　基本划分方式

对于数组型即可并行数据类型,既然它们的元素是按不同方向来组织的,就可以通过定义每一维上的划分方式进而划分

整个数据对象。因此,首先来介绍在某一方向上的三种基本划分方式:Block_dist(分块划分)、Cyclic_dist(循环划分)和Whole_dist(全局划分)。这三种基本划分方式是数据对象整体分块划分的基础。

Block_dist(分块划分)的划分方式是将这一方向上的数据统一切成具有连续元素的块,这种方式有利于保持数据的整体性。例如,定义一个Block_dist的类对象block_dist:

```
Block_dist block_dist(NUM_Sub);
```

它的含义是把这一方向上的数据按尽量公平的原则划分成NUM_Sub个具有连续元素的子块。假定相应维长为m,则在每个处理机上分配一块$\lceil m/\text{NUM_Sub}\rceil$个连续元素($\lceil\ \rceil$表示取上整),最后一块数据不足时,用空闲单元补足。例如有某一维维长为7,划分成4个子块,即NUM_Sub等于4,则每个子块的元素索引号顺次为(0,1)、(2,3)、(4,5)、(6)。第4个子块需要由一个空闲单元补足。

Cyclic_dist(循环划分)的划分方式是将这一维的元素循环分配给固定数目的子块,这种方式有利于改善负载平衡。例如,可以用下面的形式来定义一个Cyclic_dist的类对象cyclic_dist:

```
Cyclic_dist cyclic_dist(NUM_Sub);
```

它的含义是顺序给每个子块每次分配一个元素,每个处理机上元素个数相同,不足时用空闲单元补足。假定相应维长为m,则在每个处理机上分配一块$\lceil m/\text{NUM_Sub}\rceil$个元素,每个子块的元素索引号的间隔是NUM_Sub。例如有某一维维长为7,划分成4个子块,即NUM_Sub等于4,则每个子块的元素索引号顺次为(0,4)、(1,5)、(2,6)、(3)。第4个子块需要由一个空闲单元补足。

Whole_dist(全局划分)也称退化划分,即该方向上的数据都存放在一个子块中,即不作分割。

上面介绍的三种划分方式的有序排列就可以用来说明一个数据对象的划分方式,用Map(映射)类来表示。这里需要对数据对象的每一维指明它的划分方式。例如,可以这样定义一个用来说明向量(vector)的数据划分方式的Map类:

```
template〈Block_dist〉Map
map(Block_dist(NUM_Sub));
```

则可以在定义一个向量时,把map作为构造函数的参数,那么这个向量将被分成NUM_Sub个具有连续元素的子块。

同理,可以这样定义一个说明矩阵(matrix)的数据划分方式的Map类:

```
template〈Block_dist, Cyclic_dist〉
Map map(Block_dist(NUM_Sub1),Cyclic_dist(NUM_Sub2));
```

用这个map作为参数的矩阵将有NUM_Sub1 × NUM_Sub2个子块。例如,定义一个具体的7×7矩阵,并指定NUM_Sub1和NUM_Sub2都为4,则这个矩阵被划分为16个子块,如图2所示。

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)

(a)划分前

(0,0)	(0,1)	(0,2)	(0,3)	(0,4) (0,5) (0,6)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4) (4,5) (4,6)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4) (1,5) (1,6)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4) (5,5) (5,6)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4) (2,5) (2,6)
(6,0)	(6,1)	(6,2)	(6,3)	(6,4) (6,5) (6,6)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4) (3,5) (3,6)

(b)划分后

图2 矩阵划分

2.2.2 数据划分接口

作为数据划分层的主要类组件,Map类提供了一些公共接口用来实现一些具体的功能。这些功能是基于并行支撑层来实现的,同时为抽象数据层和应用组件层提供了有效的支持。通过对数据划分层功能的分析,Map类主要提供了下面的功能:

a)全局地址与局部地址的相互转换。具体的并行算法在实现过程中经常要用到全局地址与局部地址的相互转换。对于一个元素的处理,最后都归结到对某个进程中的数据块的访问。而在访问局部数据的某个元素时,也常常需要知道它的全局地址。

b)进程号与子块号之间的相互转换。当函数参数列表中需要指定具体的进程号时,若不指定则默认为当前进程。这个过程实际上满足一定的映射关系。

c)根据子块号获得相应的域值。获得子块的局部域,实际上就是获得某个子块的大小。

d)根据全局地址得到它所在的子块序号和进程号。对于一个元素的处理,也就是对某个子块的局部数据的访问,需要获得它所在的子块序号和进程号。

数据划分层的Map类提供的具体的成员函数接口定义是针对上面的功能来实现的,如表1所示。

表1 数据划分层功能接口列表

函数名	功能	参数	返回值
global_from_local_index	全局地址与局部地址的相互转换	子块号和数据元素的局部地址	数据元素的全局地址
local_from_global_index		子块号和数据元素的全局地址	数据元素的局部地址
subblock_from_processor	进程号与子块号的转换	具体的进程号	进程的本地子块序列
processor_from_subblock		具体的子块号	子块所在的进程号
subblock_global_domain	获得子块号相应的域值	子块号	子块的全局域
subblock_local_domain			子块的局部域
subblock_from_global_index	获得子块序号和进程号	元素的全局地址	元素所在的子块序号
subblock_from_global_index			元素所在的进程号

这些功能的实现是在并行支撑层的支持下完成的,同时它们为抽象数据层和应用组件层提供了实现的基础。

2.3 并行支撑层

并行支撑层基于MPI的消息传递机制[3,9],实现了一组透明的进程管理、节点通信机制,从而为数据划分层、抽象数据层和应用组件层提供抽象级别较高的并行环境。

通信器(communicator)类是并行支撑层的主要构件之一。并行库在库的初始化过程中就会建立一个communicator对象,为系统建立了一个消息传递的平台。Communicator类是基于MPI实现的,为并行库提供了消息传递机制,它的具体功能有进程管理、点对点通信、组通信等。

2.3.1 MPI的并行支持特征

作为并行库,尽管要求它应该具有较高的执行效率,但首先要具备良好的通用性、可移植性和健壮性,可以不加修改地在各种同构/异构网络上正确运行[7]。对并行向量库的消息传递机制需求描述如下:

一个安全的通信空间,确保库可以发送和接收点对点消息而不会打扰系统中的其他点对点消息。组操作把一个进程组(来自于同一个通信子)作为一个参与集,而没有参与的进程继续执行不受影响。进程的抽象命名是基于虚拟拓扑结构的,或者至少是基于组序列号名称的,因此可以避免硬件依赖,并使得应用程序代码更直观。

MPI 中完善的消息传递机制为保证并行库的上述性能提供了条件。MPI 的目标是开发一个广泛应用于编写消息传递程序的标准,给用户提供一个高可用性的应用编程接口,兼顾异构环境和可移植性,保证通信的有效性和数据传输的可靠性。当前流行的各类 SMP、DSM、MPP 和微机机群等并行机均支持 MPI,它提供了一个理想的消息传递并行机模型,如图 3 所示。

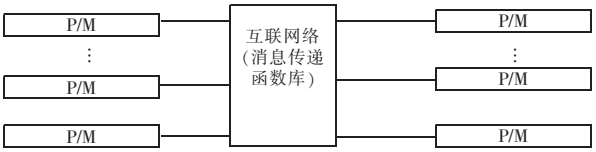

图 3　理想的消息传递并行机模型

图中 P 表示进程,M 表示局部存储空间,多个 P/M 模块通过互联网络相互连接。在该结构中,各个进程之间可以直接通信,但是各个进程只能直接访问自身的局部存储空间(本地访问),对其他进程的局部存储空间的访问(远端访问)只能通过消息传递来实现。它实际上属于典型的分布式存储并行机。

MPI 具备一些特征可以满足上面提出的并行向量库的需求:

进程组,即所有参加通信的进程的集合。它定义了一个在点对点通信中的相关组的命名序列。

通信子包括两部分,即进程组和通信上下文。

消息标志,在消息传递机制中,点对点通信是大量出现的。

针对并行库上层结构对并行支撑层的功能需求,基于 MPI 的消息传递机制,为 communicator 类设计了如表 2 所示的成员函数接口。

表 2　并行支撑层功能接口列表

分类	函数名	实现功能	参数	返回值
进程管理	get_rank	获取当前进程的序列号	无	当前进程号
	get_size	本程序中所用到的进程数	无	进程数
点对点通信	buf_send	以缓存通信模式发送数据	发送消息的起始地址,数据个数,数据类型,目标进程号,消息标志,通信子	操作成功则返回 0,否则返回错误状态
	send	标准通信模式发送数据	接收消息的起始地址,数据个数,数据类型,目标进程号,消息标志,通信子,返回状态	
	recv	接收数据	通信对象,返回状态	
	wait	等待某个通信请求完成	消息起始地址,数据个数,数据类型,根进程号,通信子	
组通信	barrier	进程同步功能	无	
	broadcast	消息的广播操作	消息的发送和接收地址,数据个数,数据类型,操作类型,通信子	
	allreduce	全归约操作	消息的发送和接收地址,数据个数,数据类型,操作类型,通信子	

2.4　应用组件层

本文将重点阐述并行库应用接口的设计,即应用组件层的实现。它的功能是给用户提供一个易用的接口,完成某种特定的并行处理功能。这里针对图像/信号处理领域中应用软件的需求[3],选择较为优化的并行算法进行封装,并以类组件的形式提供给用户。

2.4.1　矩阵转置

矩阵转置的功能由 MatTrans(矩阵转置)类来完成,用户在创建一个 MatTrans 类对象时,把原矩阵和结果矩阵作为该对象构造函数的参数,这样在该类对象完成它的 operate 操作时,就会把并行计算的结果存放在结果矩阵中。

2.4.2　矩阵—向量乘法

矩阵—向量乘法也是在信号处理过程中经常用到的基本运算,应用组件层的 MatVec 类用来完成这个功能。这里把参加运算的矩阵和向量以及结果向量作为构造函数的参数,则在该 MatVec 类对象进行 operate 操作时,并行运算的结果就被存放在结果向量中。

2.4.3　矩阵乘法

矩阵乘法是信号处理中使用频率比较高的运算。本文用 MatMat 类来实现。同样地,两个参加运算的矩阵以及结果矩阵作为构造函数的参数,结果矩阵在该类对象完成 operate 操作时被赋予并行计算的结果。

一个 $m \times n$ 阶矩阵 $A = [a_{ij}]$ 乘以一个 $n \times k$ 的矩阵 $B = [b_{ij}]$ 就可以得到一个 $m \times k$ 的矩阵 $C = [c_{ij}]$,它的元素 c_{ij} 为 A 的第 i 行向量与 B 的第 j 列向量的内积。矩阵相乘的关键是相乘的两个元素的下标要满足一定的要求(即对准)。

2.4.4　矩阵 LU 分解

矩阵的 LU 分解是一种常用的基本矩阵运算,它是求解线性方程组的基础,尤其在解多个同系数阵的线性方程组时特别有用。应用组件层的 MatLU 类用来完成矩阵 LU 分解的并行计算。MatLU 类对象的构造函数有三个参数,一个是参加运算的矩阵,另外两个是结果矩阵。MatLU 类对象的 operate 操作完成矩阵 LU 分解的并行处理。

2.4.5　矩阵求逆

矩阵求逆(matrix inversion)是一种常用的矩阵运算。同样地,把原矩阵和结果矩阵作为相关组件类 Mat Inver 类构造函数的参数,在类对象完成它的 operate 操作时,就会把并行计算的结果存放在结果矩阵中。

3　实验测试

表 3 列出了测试程序的基本执行信息。每个测试程序中设有输出函数,可以让各进程输出当前进程的本地数据和运算结果,从而获得并行算法的执行情况。输出结果中有测试程序中的数据对象名、进程标志、子块号、元素的局部地址、全局地址和元素值等信息。

表 3　测试程序的执行信息

测试组件	执行功能	程序名	原参数	结果参数	进程数	是否正确
MatTrans	矩阵转置	MatTrans1	8×8 矩阵	8×8 矩阵	4,3	是
		MatTrans2	64×64 矩阵	64×64 矩阵	12,9	是
MatVec	矩阵—向量乘法	MatVec1	6×12 矩阵和 12×1 列向量	6×1 列向量	4,3	是
		MatVec2	24×48 矩阵和 48×1 向量	24×1 向量	12,9	是
MatMat	矩阵乘法	MatMat1	两个 6×6 矩阵	6×6 矩阵	3,2	是
		MatMat2	两个 18×18 矩阵	18×18 矩阵	9,6	是
MatLU	矩阵 LU 分解	MatLU1	6×6 矩阵	两个 6×6 矩阵	3,2	是
		MatLU2	16×16 矩阵	两个 16×16 矩阵	9,6	是
Mat Inver	矩阵求逆	Mat Inver1	9×9 矩阵	9×9 矩阵	3,6	是
		Mat Inver2	16×16 矩阵	16×16 矩阵	4,8	是

输出结果显示,每个组件在使用不同的进程数运行时,组件可以根据进程数实现相应的并行处理算法,获得相同的运算结果。由此可以证明并行库应用组件层的类组件能够满足并行算法实现的正确性,并具有一定的可移植性。　(下转第 4570 页)

选择 70% 的样本(163 个)作为训练样本,利用 K-均值算法对焦比进行聚类,之后利用 C4.5 算法进行规则抽取,得到若干规则;将剩余 30% 的样本(70 个)作为测试样本,按照得到的规则进行分类。重复 100 次实验,训练样本、测试样本的规则覆盖平均准确率分别为 93.1%、85.3%。

2.3 结果分析

从决策树结果中可以看出,山西焦使用比例为 14% 是样本分类的第一个分割点,说明山西焦对于焦比的影响是最重要的。现场在进行燃料配比时,应充分考虑该因素。

通过抽取得到的规则全面地包含了各种燃料与焦比的影响关系,可以帮助现场工程师更深入地认识 COREX 燃料配比对生产的影响规律。在炉况、燃料供应发生变化时,可以根据规则调整燃料配比。

3 结束语

本文提出了基于决策树规则抽取的 COREX 燃料配比模型,利用决策树 C4.5 规则抽取方法对宝钢 COREX-3000-2$^{\#}$炉的实际生产数据进行学习,给出了燃料配比与焦比之间定量关系的知识规则,为现场生产燃料配比决策提供了有效的理论依据,在研究燃料配比、节约生产成本、提高生产效率等方面具有重要的现实意义。

参考文献:

[1] 吴铿,张二华,万鹏,等. 关于 COREX 流程熔融气化炉风口前理论燃烧温度的思考[J]. 煤炭学报,2010,35(10):1730-1734.

[2] 李海峰,王臣,邹宗树,等. COREX 喷煤模型及应用分析[J]. 过程工程学报,2009,9(S1):349-353.

[3] 李维国. COREX-3000 生产现状和存在问题的分析[J]. 宝钢技术,2008(6):11-18.

[4] 王子宏,唐志宏,游坚,等. 2 号高炉用焦炭热性能及其配煤分析[J]. 柳钢科技,2009(4):5-9.

[5] 魏继中. 高炉生产配煤配矿决策支持系统设计与实现[D]. 武汉:武汉科技大学, 2009.

[6] 吴胜利,韩宏亮,徐少兵,等. 高炉优化配料数学模型的研究[J]. 钢铁,2007,42(9):19-23.

[7] 吴敏,丁雷,曹卫华. 基于混合粒子群算法的烧结配料优化[J]. 信息与控制,2008,37(2):242-246.

[8] 吕学伟,白晨光,邱贵宝,等. 基于遗传算法的烧结配料综合优化研究[J]. 钢铁,2007,42(4):12-15.

[9] ZHANG Rui-jun, LU Jie, ZHANG Guang-quan. A knowledge-based multi-role decision support system for ore blending cost optimization of blast furnaces[J]. European Journal of Operational Research, 2011,215(1):194-203.

[10] De MATOS U F, De CASTRO J A, Da SILVA A J. Analysis of the injection of different coal blending using a total blast furnace model [R]. 2005.

[11] 毛国君,段立娟,王实,等. 数据挖掘原理与算法[M]. 2 版. 北京: 清华大学出版社, 2007.

[12] QUINLAN J R. Induction of decision trees[J]. Machine Learning, 1986,1(1):81-106.

[13] QUINLAN J R. C4.5: programs for machine learning[M]. San Francisco: Morgan Kaufmann, 1993.

[14] 孙宏斌,王康,张伯明,等. 采用线性决策树的暂态稳定规则提取[J]. 中国电机工程学报,2011,31(34):61-67.

[15] MacQUEEN J. Some methods for classification and analysis of multivariate observations[C]//Proc of the 5th Berkeley Symposium on Mathematical Statistics and Probability. Berkeley: University of California Press,1967:281-297.

(上接第 4563 页)

4 结束语

本课题针对并行应用软件开发中存在的效率低、质量难以保证、可移植性差等问题,借鉴 HPEC_SI 计划的解决方法,基于 MPI 的消息传递机制,设计和实现了具有面向对象特征的、用于图像/信号处理的并行向量库[10]。

本文在对 HPEC_SI 计划解决方法的分析、总结和模型化的基础上,提出了并行向量库结构的四个层次:应用组件层、抽象数据层、数据划分层和并行支撑层。其中,抽象数据层、数据划分层和并行支撑层共同完成并行基础平台的构建;应用组件层在并行基础平台提供了应用接口的功能。这种层次型结构有利于并行应用开发的责任分解,提高了应用软件的可移植性和质量[11]。

本文对应用组件层中已经实现的类组件进行了测试。针对相同的类组件,通过设置不同的进程数来运行测试程序,都得到了正确的测试结果。证明类组件提供的并行处理的正确性是能够保证的,且具有一定的可移植性,通过比对可以看出,笔者开发的库具有简单易用、可复用性和可移植性强、效率高的特点。

参考文献:

[1] 陈国良,孙广中,徐云,等. 并行计算的一体化研究现状与发展趋势[J]. 科学通报,2009,54(8):1-43.

[2] 解庆春,张云泉,王可,等. SIMD 技术与向量数学库研究[J]. 计算机科学,2011,38(7):298-301.

[3] LU Jian, LI Ying-jun, MA Xiao-xing, *et al*. A hierarchical framework for parallel seismic applications[J]. Communications of the ACM, 2000,43(10):55-59.

[4] 周灿. 基于 MPI 的矩阵运算并行算法研究[D]. 重庆:重庆大学, 2010.

[5] 杨川,杨斌. 基于 TBB 的傅里叶变换多核并行化实现[J]. 计算机工程,2010,36(16):288-290.

[6] YU Hua-shan, HU Chang-jun, HUANG Qi-jun, *et al*. A time-slicing optimization framework of computation partitioning for data-parallel languages[J]. Journal Software, 2001,12(10):1434-1446.

[7] CAMPBELL D P, RICHARDS M A, MACKENZIE K M, *et al*. Facilitating middleware for polymorphous computing architectures[R]. Georgia: Georgia Institute of Technology, 2002.

[8] HOFFMANN H, KEPNER J V, BOND R A. S3P: automatic optimized mapping of signal processing applications to parallel architectures. [D]. [S.l.]: MIT Lincoln Laboratory. 2001.

[9] 杨川,杨斌. 基于 TBB 的傅里叶变换多核并行化实现[J]. 计算机工程,2010,36(16):288-290.

[10] CAMPBELL D. The high performance embedded computing software initiative: C++ and parallelism extensions to the vector, signal, and image processing library standard[C]//Proc of Users Group Conference. 2005:278-283.

[11] 曾小荟,刘清,罗文浪,等. 基于 FC-TR 网络的 MPI 通信性能优化研究[J]. 计算机应用研究,2011,28(4):1505-1508.