第6卷第11期
1995年11月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 11
Nov. 1995

一种基于 Message Passing 的并行程序设计环境*

温钰洪　沈美明　王鼎兴　郑纬民

(清华大学计算机系,北京 100084)

摘要　并行处理技术的发展使得高性能的并行计算机系统不断地推出,而方便、灵活的并行程序设计环境在并行计算机系统的推广应用中起着重要的作用.如何编写有效的并行程序代码以及如何将现有的串行应用程序并行化已成为并行处理的重要研究课题.

关键词　消息传递,处理机分配,结点程序加载,并行程序模型.

随着计算机并行处理技术的发展,特别是并行计算机系统体系结构的研究,并行计算机系统越来越多地得到人们的重视.在科学计算、信息处理等研究领域,并行计算已经取得了很好的加速比.

在并行计算的实际问题中,有一大部分问题可以将其所处理的数据进行划分,采用同一处理过程进行计算,即所谓的 SPMD 模型问题.经研究表明,SPMD 在天气预报数值计算、石油勘探等实际应用中占有相当的比例,对这类问题采用并行计算机系统来求解,以缩短其问题求解时间,具有很大的意义.

并行计算机系统在计算机体系结构的研究中取得了很大的发展.但对于实际使用中,如何进行并行程序设计,如何将串行程序并行化则成为了并行计算机系统推广应用的巨大障碍[1].并行计算机系统以其丰富的计算资源,提高了计算能力,但同样也为用户如何操作、使用并行计算机系统带来了困难,因而研究设计方便、灵活的并行程序设计环境,使用户能有效地、合理地使用并行计算机系统的资源,充分发挥其巨大的计算能力,成为了并行处理的研究重点.

另一方面,随着计算机网络技术的飞速发展,计算机网络数据传输的速率越来越高,从而使得利用网络工作站进行并行计算成为了可能,并且已经取得了重大的进展.在网络工作站机群系统中,具有并行计算结点增减方便,以及投资小、风险小等特点,通过高速的计算机网络环境,可以很容易地构造大量计算结点的并行处理系统,同时,因为工作站具有独立工作能力强的特点,在机群系统不进行并行计算时,也能单独地使用,因而这样的并行计算系

* 本文 1994－03－21 收到,1994－08－09 定稿

作者**温钰洪**,1967 年生,博士生,主要研究领域为并行计算机体系结构,并行程序设计环境.**沈美明**,女,1938 年生,教授,主要研究领域为并行与分布处理,并行程序开发环境.**王鼎兴**,1937 年生,教授,博士导师,主要研究领域为并行与分布计算机系统,计算机组织与系统结构.**郑纬民**,1946 年生,教授,主要研究领域为并行计算,并行编译系统.

本文通讯联系人:温钰洪,北京 100084,清华大学计算机系

统投资风险非常小.

无论是何种类型的并行计算系统,都不是仅仅有了硬件支持环境就够了,并行计算的推广应用都需要有大量的并行程序环境以及并行计算工具,以利于进行并行程序设计,同时将现行成熟的串行应用程序改写为并行应用程序.

我们所设计的并行程序设计环境则是基于以上几种考虑而研究设计的.它具有能适应不同的并行计算机系统体系结构,不同的计算结点以及程序设计简单、方便等特点.并行程序设计环境是根据 Express 系统而重新改造、开发设计的,在利用现有 Express 系统的一些基本概念及库函数特性的基础上,我们自行研究设计了并行程序设计环境的核心部分,即运行系统核心(Runtime System Kernel)[2,3],以使其对于并行计算应用程序能有更好的并行计算加速比.我们已经在 SPARC 以太网工作站环境和 Transputer 并行加速器环境上分别实现了这样的并行程序设计环境.文中所给出的实验结果则是在 Transputer 并行加速器上应用并行程序设计环境所得到的并行计算结果.

1 并行程序设计环境系统结构

并行程序设计环境采用层次化的体系结构(如图 1 所示),在底层硬件资源及操作系统之上,构造一个并行程序设计环境运行核心,由它负责实现对并行计算机系统硬件资源的管理、并行计算任务的划分、处理机的分配及计算时各处理机之间的通讯管理.通过运行核心,完成并行程序在并行计算机上的实现,并做到使底层并行计算机体系结构对用户的透明.系统运行核心的实现中,引入面向对象(OO)的机制,以使并行程序设计环境管理更加有效[4,5].

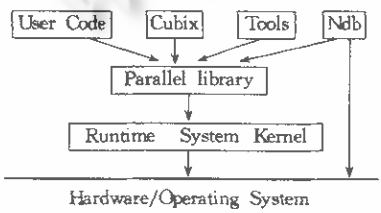

图1 并行程序设计环境体系结构

并行程序库则通过大量的函数,使用户能完成对处理机分配、消息通讯、程序加载及运行的控制,用户在程序中只需描述清楚所要解决的问题规模、通讯方式,然后调用库函数即可,程序设计简单、方便.

并行程序设计环境的最上层,为系统所提供的大量的实用工具及用户程序.通过使用工具,用户可以对并行程序的运行情况进行综合的管理,得出统计结果,以便于分析及提高程序的执行效率,使用户能对并行程序进一步改进.同时,系统还提供用户对并行程序执行时的调试工具,以利于对并行程序运行时的动态控制.

2 并行程序设计环境机制

在并行程序设计环境中,应用程序的并行执行主要分为申请分配计算结点,在计算结点上加载执行程序代码并启动运行,Message Passing 进行数据交换,回收计算结果等几个步骤.

(1) 处理机分配及程序加载

在运行并行程序之前,最基本的操作是分配应用程序所需的处理机并加载应用程序.依据应用程序所选择的不同的程序模型,并行程序设计环境中有多种不同的处理机分配机制,

在 Cubix 程序模型中，程序员不用关心处理机如何分配，由系统来完成. 在 Host－Node 模型中，则有以下几种方法(依据 Express 的方式)：

① 程序员不用关心处理机如何进行分配，只关心所分配的处理机数；

② Host 控制，应用程序需要在 Host 上运行一段代码，由它来控制处理机的分配以及在其上的数据加载和数据回收；

③ 全控制，应用程序需要完全控制分配使用和加载并行程序，其中包括使用哪一个物理处理机以及将其用作何用.

在结点分配中，应用程序分配到一组处理机，系统给出一统一的编号，作为此应用程序的处理机标识. 所有操作均在这一组处理机中，如果在应用程序不完全控制的情况下，处理机分配则考虑如下几个启发因素：系统中有空闲结点未被占用；各结点之间的距离总和最小.

以上策略分配的结点，可以保证结点间通讯时有最短的路径.

(2) 通讯机制

并行程序设计环境的底层系统是以异步的点到点的消息传递 Message Passing 系统作为基础的，这意味着一个结点可以随时将一个消息通过系统核心的 Buffer 及其路由选择，发向另一个结点.

在系统中，同步消息传递是将读(Read)函数挂起，直到其所要求的数据到达为止的办法来实现的(如图 2 所示). 在并行程序环境中，同时提供同步的和异步的两种通讯方式，这分别适用于不同类型的应用程序，实际应用中可选用合适的一种方式.

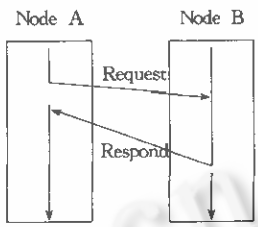

图2 同步通讯方式

在通讯系统中，是围绕 Message 来构造的. 其中每个消息指一个写(Write)函数所产生的结果，它包含一些没有结构的数据. 每个消息由一个读(Read)函数来消费，如果读函数的消息长度小于写函数所写出的消息长度，读函数取走少量所需的数据. 反之，则读函数返回指明没有足够的数据的报告.

在消息标识中，除了通常的数据 buffer、数据长度、结点地址之外，同时还采用消息类型(Type)来标识. 这个值在于使接收结点能更为清楚地明白要接收什么样的类型的数据，这对于多任务系统的程序环境而言，可以清楚地标明这一消息是给这一任务而非另一任务，而在实时系统中，程序员则可以利用这一 Type 值来将一些消息置为高优先级，而另一些置为低优先级.

当一个结点接收消息时，利用接收标识来选取所需要的消息. 在 Message Passing 中，每个所发送的消息可带有一个相应的 Type 类型值，接收结点则利用这一类型值与结点地址号来选取消息. 当有相同类型的消息同时到达时，则选择最先到达的消息；另外，接收结点也可利用 NOCARE 这一特殊值来更灵活地选取所有可能的消息.

① 消息、结点和类型

在 Message Passing 中，每个消息是一个没有结构定义的数据流，用户可以从任意结点，在程序执行的任意时刻发送消息，系统不以任何方式解释消息. 因而，用户可以任意地传送整数、浮点数、字符串、一结构数据等任意的数据. 在接收结点中，则在用户程序一级协议，

按所确定的消息数据结构来解包，获得所需要的数据(消息定义采用 Express 方式).

在 Message Passing 系统中，每个消息都有其所属的结点号来进行标识. 并行程序设计环境中，每个结点都有相应的逻辑结点号，而 Host 则对应于一特殊的结点号. 并行程序设计环境的 Host 是并行计算机系统中的一个特殊结点，其上可提供诸如图形、磁盘操作以及加载用户程序等系统服务，同时，也是所有结点程序所要进行 I/O 服务的处理机. 在并行计算机系统中，它被视为一连结在并行机网络中的结点，因而 Host 与结点间的 Message Passing 也采用相同的方式进行.

消息结构中的最后一个限制是 Type 类型参数，其作用在于更清楚地标识 Message. 在系统中 0—16383 的 Type 类型值作为应用程序的消息类型值，16383 以上则作为系统消息类型值.

在用户应用程序中，也可不加类型值限制. 但应用程序中，如果合理地使用 Type 类型值来定义区分消息，则能很好地调试、运行并监控应用程序的运行. 另外，在并行程序设计环境的 Cubix 程序模型中，由于用户不写 Node 程序，而 Host 程序由系统提供的 General I/O Server 来完成，在这一程序模型中，消息传递都由系统来统一完成，因而就可以去掉 Message Type 值的设定.

② Blocking 通讯

在并行程序设计环境中，最基本的消息通讯函数为 exread 和 exwrite，这些函数可以任意在 Host 和 Node 上使用，消息的传递由系统调用(采用 Express 函数形式)：

```
status = exwrite( *buffer, length, &dest, &type);
```

来完成，其中 buffer 为指向一个 length 长度的消息的指针，length 可以为任意值，长度为 0 的消息通常可用于使各个处理机间的执行达到同步.

在接收消息时，使用系统调用：

```
status = exread( *buffer, length, &src, &type);
```

来完成. 系统在选择消息时，通过 src 和 type 两个限制参量来确定，如果 src 为 NOCARE，则可读取从任意结点发送来的消息. 系统调用 exread 采用 Blocking 机制，若结点所需的某一消息未收到，则执行程序被挂起.

在 exwrite，exread 的基础上，并行程序设计环境中同时还提供了实现结点间广播通讯的 exbroadcast. 将发送与接收数据合为一体的 exchange. 在所选定的结点间收集数据，并进行一些合并操作，如求平均、最大、最小值等的 excombine. 将数据并入一个数据 buffer 中的 excone，以及在各处理机间设置同步点的 exsync 等系统通讯函数，所有这些都构成了并行程序设计环境的 Blocking 通讯机制.

③ Non—Blocking 通讯

对于诸如实时控制处理等实际应用，需要很快、很灵活地输入数据，并进行 Pipeling 操作，它们处理数据集中的一部分，同时等待另一部分数据的到达. 这些应用若采用 Blocking 通讯机制无法实现，因而需要采用 Non—Blocking 技术.

在实时系统中，通常产生一个服务进程来接收并处理消息，同时不打断另一部分应用程序处理其它来源的数据.

并行库函数 exhandle 就提供了当一旦接收到消息，即刻就处理的机制，其系统调用为：

```
type = 123;
src = NOCARE;
status = exhandle(*func, &src, &type);
```

这样,当结点一旦接收到 type 123 的消息,则由用户定义的函数 func 来对其进行处理.

在这种处理中,用户程序可以以最小的延时来处理接收到的消息,一旦当消息到达,则中断正在执行的程序,接收数据并传送到用户程序中.

这一函数的使用构成了并行程序环境中多任务程序设计的基础,也即构成了从消息类型空间到应用程序中一系列处理过程之间的映射关系.这个映射使得应用程序不用关心其它结点的计算,而自由地计算,直到有消息传送到达.这同样也构成了并行程序设计环境中的异步程序设计模型.

另一种异步通讯处理是 Non-Blocking 读和写:

```
status = exrecv(*buffer, length, &src, &type, &state);
status = exsend(*buffer, length, &dest, &type, &state);
```

对于 exrecv 函数而言,与 exread 相比,多了一个参数 state,调用此函数之后,该函数立即返回,不象 exread 那样挂起.若没有读到数据,则 state 为-1,src 和 type 的值不变;若读到了数据,则 state 置为所读到的消息长度,src 和 type 的值相应改变为接收到的消息中的值.对于 exsend 也同样,应用程序调用之后立即返回,而不管消息是否已经发送出去,因而当应用程序调用 exsend 之后,不能立即改变 buffer 中的值,除非 state 值已不为-1.

这两种异步通讯方式在并行程序设计环境中的引入,对于并行程序执行时速度的提高,是有很大好处的.

(3) I/O 机制

在我们的并行程序设计环境中,每个处理机被系统视为有计算能力的设备,结点上可进行计算,但不具有 Disk,没有 I/O 能力,处理机都通过 Host 机来完成 I/O 服务.

在 Host-Node 程序模型中,应用程序的执行由 Host 程序来完成控制,因而结点程序的 I/O 服务也通过上述系统中的多种通讯机制来完成.

并行程序设计环境的 Cubix 是一个具有 I/O 服务功能和操作系统服务器功能的子系统,即 General I/O Server,它可以使结点应用程序都能访问 Host 机上的操作系统资源.在 Cubix 程序模型中,应用程序不用写 Host 部分,由 Server 来完成,应用程序只用写结点上运行的部分程序.通过并行库函数的调用,应用程序还可以访问 Host 机的系统资源.同时,其程序设计方式还保持了与串行程序相似的程序设计方法(Cubix 并行程序设计模型采用与 Express 相同的方法).

在 Cubix 编程模型的 I/O 服务中,最底层为通过标准函数 Read, Write, Open, Rewind 等来完成.而上一层次则通过以下 3 种不同的模式来构成一个完整的分布 I/O 服务系统.

同步模式:各个处理机同时发出调用请求,并且各处理机获得相同的数据.这种方式的 I/O 服务有利于应用程序获取全局数据等共同的数据;

多模式:各个处理机同时发出调用请求,但各个处理机获得不同的数据.这种方式的 I/O服务有利于应用程序并行地处理同一数据源里的不同数据;

异步模式:各个处理机可随时地各自提出 I/O 申请,并独立地由 Server 来完成.这种方

式不易控制,且具有不可重复性,但对于一些实际应用也是很有帮助的.

在Cubix模型程序设计中,可以根据实际应用的需要,任意地选择一种方式的I/O服务.另外,同一应用程序也可随意地在几种I/O方式之间进行切换.通常对于Multi模式,适合于处理有格式定义的数据文件,而异步I/O模式,则适合于处理无格式定义的数据文件.

3 体系结构

我们所研究设计的并行程序设计环境则是基于上述并行计算机系统的使用所提出的.在我们的并行程序设计环境中,主要针对并行处理SPMD问题.在并行计算机中,各个处理机同时处理实际问题的不同数据,以获得实际问题求解的加速比.

在并行程序设计环境中,为保证并行计算机系统对于并行程序设计的硬件透明性,应提供并行计算机逻辑拓扑结构到实际物理结构的良好映射,通过这一映射,将程序设计的逻辑结构空间转换到并行计算机系统的物理空间.因而,在我们所设计的并行程序设计环境中,提供了多种并行计算机拓扑结构的逻辑描述方式,程序员在进行并行程序设计时,可根据具体应用程序的需要,任意地将并行计算机划分为一维、二维、Torus、Hypercubes等多种结构形式,而不必关心实际并行计算系统是如何进行连结的.所有的这些逻辑结构划分,只需在程序加载执行之前,通过调用相应的并行库函数即可,非常灵活方便.并行程序中所描述的逻辑结构在具体并行计算机系统上的实现,则由并行程序环境来完成.

并行程序设计环境除了在程序设计中提供逻辑拓扑结构的描述之外,同时提供了对于不同结点数的并行计算机系统的支持.在实际应用中,程序员可以随意对不同结点数的并行计算机系统进行逻辑结构划分,并行程序设计环境可根据相应的结点数生成相应的逻辑结构,因而具有系统一级的可扩展性.

在我们具体实现的并行计算机系统中,每个结点为Transputer T800处理机,32个处理机以Mesh网组成一个并行加速系统,系统体系结构如图3所示.

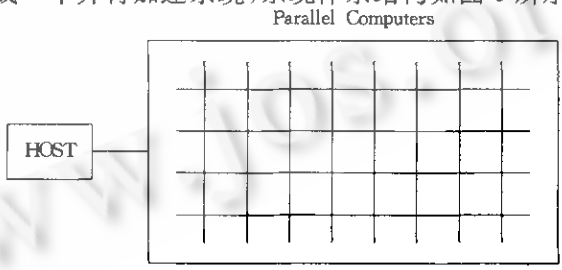

图3 并行加速系统体系结构

在并行计算机系统中,实际上是一个Multi-computers系统,各个处理机有各自的Memory,处理机之间通过Message Passing技术来完成处理机之间的通讯,底层通讯机制采用Worm-hole寻径技术,达到快速的消息传递.

在并行计算机程序环境中,各个处理机被视为计算设备,并行程序环境负责完成处理机在应用程序中的分配、程序加载及运行控制,结点仅仅具有计算能力,Host与Node之间也同样采用Message Passing的机制.

4　并行程序设计环境在天气预报中的应用

在天气预报数值计算问题中，二阶线性扩散方程的迭代算法是一个计算量较大，并可根据区域划分，在不同结点上执行相同的程序代码，处理不同的数据的 SPMD 问题. 我们将这一算法在 Transputer T800 并行加速系统中，利用并行程序设计环境进行求解，得到了如下的几组数据.

对于 50×50×10 的数据矩阵，在进行 100 次迭代时，得到加速比曲线(如图 4)和效率曲线(如图 5)的结果如下：

结点数	执行时间(秒)	加 速 比	效率(%)
1	622.27		
2	311.41	1.998	99.9
4	159.15	3.910	97.8
8	85.46	7.281	91.0
12	63.48	9.803	81.7
16	50.61	12.295	76.8
32	35.48	17.479	54.6

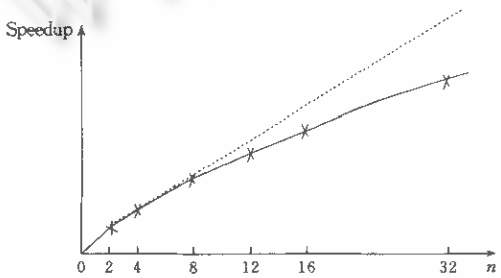

图4　加速比曲线

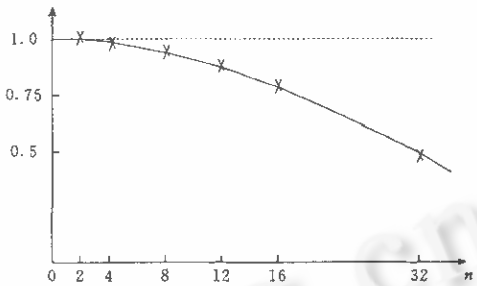

图5　效率曲线

从以上数据可以看出，随着结点数的增加，效率有所下降. 这主要是由于随着结点数的增加，通讯量也增加，而各结点的计算量相应减少，造成结点间通讯所用时间的比例增加，从而影响了计算的加速比及其效率. 如果将上述题按 1000 次进行迭代，得到：

16 个结点的执行时间：422.28 秒

32 个结点的执行时间：249.36 秒

32 个结点对 16 个结点的加速比 S = 1.71

而在 100 次迭代时的加速比 S=1.40. 因此，随着计算量的加大，并行计算的加速比和效率都会相应增加.

5　结　语

并行程序设计环境通过提供多层次的系统调用和并行库函数，使程序员能够方便、灵活地通过程序设计操作和使用并行计算机系统. 并行程序设计环境中，有丰富的 Message Passing 通讯机制，多种 I/O 服务，以及多种程序设计模型，对于并行计算机系统的推广应用必将起到积极的作用.

参考文献

1 Boari Maurelio, Ciampolini Anna, Corradi Antonio. Programming environment for transputer—based architecture. IEEE Transactions on Computers, 1991.

2 Parasoft. Express: porting express 3.2. Parasoft Co., Paradena, CA, 1992.

3 Parasoft. Express: users guide. Parasoft Co., Paradena, CA, 1988.

4 Henri E Bal *et al*. Programming languages for distributed computing system. ACM Computing Surveys, Sept. 1989, **21(3)**: 261—322.

5 Ciampolini A, Corradi A, Leonardi L. Parallel object models in the design of an environment for massively parallel architectures. In: First Workshop of the ESPRIT Parallel Computing Action, Southampton, UK, July 1990.

A PARALLEL PROGRAMMING ENVIRONMENT BASED ON MESSAGE PASSING

Wen Yuhong Shen Meiming Wang Dingxing Zhen Weimin

(*Department of Computer Science, Tsinghua University, Beijing* 100084)

Abstract With the developing of parallel processing technology, more and more high—performance parallel computer systems have been developed. The convenient and flexible parallel programming environment plays an important role in the spread of parallel computing. How to write efficient parallel codes and how to convert the existing sequential applications into parallel codes have become a very important issue in parallel processing.

Key words Message passing, processors allocation, node application loading, programming model.