国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第12卷第4期　1990年12月　Vol.12 No.4

机器人力反馈依从控制器多机系统分布式程序设计的实现

肖人庆
（自动控制系）

摘　要　本文介绍在机器人力反馈依从控制器多机系统中采用程序设计语言C与分布式库函数相结合实现分布式程序设计的方法，论述了分布式库函数形式，支持分布式库函数设计的多机操作系统功能及其在设计过程中应予以重视的某些问题，并给出了分布式程序设计的一个简例。

关键词　程序设计，分布式计算机，机器人/分布式程序设计，分布式库函数，分布式并发进程，原语

分类号　TP312，TP242

由于机器人技术的不断发展，目前具有力传感器的机器人已经出现。这种机器人除完成位置控制外还要控制力或力矩，即实现力／位置混合控制。具有这种功能的控制器称力反馈依从控制器。设计力反馈依从控制器的关键技术之一是研制一个多机并行处理系统，以完成一个采样周期内的大量计算。我们研制的力反馈依从控制器多机系统由4个32位微处理器及一个I/O处理器组成，其中4个处理器完成用户作业的多任务并行处理。

多机并行处理系统属于分布式计算机范畴。在分布式计算机系统上用户如何进行分布式程序设计是研究设计分布式计算机系统必须解决的一个问题。

1　分布式程序设计实现途径

分布式计算机系统出现后，在70年代后期提出了分布式程序设计。一个分布式程序含有若干个可以独立运行的程序模块，当分布式程序运行时，这些模块动态地分配到若干个计算机上并行执行。分布式程序运行时有两个明显的特征：

（1）分布式并发性

分布式并发性不同于单处理机中的多任务并发，由于分布式系统具有多个计算机，并发不仅表现在宏观上，还表现在微观上。单机系统的并发仅仅是宏观上的，微观上是各任务分时串行运行。

1990年2月12日收稿

(2) 分布式并发模块间的交互作用

由于分布式计算机系统能够彼此协同配合共同完成一项任务，分布式程序的模块之间必须相互传递信息以便协同彼此的动作。

顺序程序设计语言，如Fortran，pascal，c 等不具有描述并发事件的成份，也缺少描述并发任务间通信与同步的手段，不能满足设计分布式程序的需要。为了研究分布式程序设计语言，往往采用两种不同的途径：

其一，扩充已有的顺序程序设计语言，增加并发性与交互性描述成份。

其二，专门设计用于分布式程序设计的新语言。

由于分布式计算机系统的差异，在上述途径指导下开发的一些语言，如并行 pascal、并发 c 、MOD、DP、ADa、SR等未能得到普遍的接受。

由于机器人力反馈依从控制器的控制对象主要是机械手这类特定对象，控制器的多机系统可以设计成专用系统，可采用扩充 C 语言方案。如果直接增加并发性及交互性描述成份，则需要修改 C 编译器，困难大。我们的做法是间接扩充 C 语言功能，以适应分布式程序设计的需要。具体方法是通过 C 语言与分布式库函数相配合来实现分布式程序设计。这种方法无需修改 C 编译器，但需要设计一个多机操作系统内核来支持分布式库函数运行。

2 分布式库函数

C 源程序由若干个函数组成，其中有且仅有一个主函数 main( )。分布式库函数是能产生分布式并发进程、进行进程通信与同步以及完成有关计算任务的一类函数，它由若干个可以独立运行的程序模块组成。分布式库函数功能由分布式计算机系统中的若干台计算机并行执行来完成，故称分布式库函数。

设计分布式库函数时，函数内含有产生分布式并发进程的成份，有完成进程通信与同步操作的成份。这些，C 语言没有提供。为了支持分布式库函数设计，我们自行设计了多机操作系统内核，提供了若干条原语，供库函数调用。分布式库函数可以是下列形式：

```
        函数名( )
        {
      变量或结构说明;
模块1 { fork (参数表);
      { send (参数表);
      { C 语句序列……;
      { join (参数表);
模块2 { receive (参数表);
      { C 语句序列……;
      { send (参数表);
      { join (参数表);
```

其中：fork产生分布式进程族。

join实现分布式进程族内同步。

send 及 receive 实现分布式并发进程间的信息发送与接收，它们都是多机操作系统的原语。

为使函数中模块1、2能在两个计算机上并行运行，将模块1、2分别作为两个进程的正文段。当fork执行时，进程进入有关队列，由处理机管理自动地将它们分配到不同CPU上去运行。

用户用C语言设计分布式程序，程序包括两部分：一个主函数，若干个分布式库函数。主函数调用分布式库函数实现分布式并发处理。多机操作系统内核在管态下运行，用户程序可在目态下运行，也可以在管态下运行。为保证多机系统的安全，用户程序宜于在目态下运行，但管／目态转变频繁时需一定的开销。分布式程序仅在多机操作系统内核支持下运行，无需UNIX操作系统介入。

3　多机操作系统内核设计的一些考虑

本文论述的分布式程序设计方法中，分布式库函数起着关键的作用。设计分布式库函数需多机操作系统给予有力支持。确定多机操作系统功能时，应着重考虑力反馈依从控制特点和支持分布式库函数设计的需要，不追求通用多机操作系统的功能及特征，以免系统庞大给设计、调试、维护所带来的困难。设计过程中力求高效率与精炼，以适应于实时控制的需要。我们设计的多机操作系统内核主要包括：分布式并发进程的产生与会合，分布式并发进程的通信与同步，进程管理与控制，处理机管理，共享存贮器管理以及中断处理等部分。

本文仅就与分布式库函数设计紧密相关的几个问题谈谈设计过程中的一些考虑。

3.1　分布式并发进程的产生与会合

要进行分布式程序设计必须有控制产生分布式并发进程的手段。为此，在多机操作系统内核中设置两条原语：fork（参数表），join（参数表）。原语 fork 创建一族分布式并发进程，含一个父进程及若干子进程。一族内的进程在若干个CPU上并行执行。从增加系统的并发性考虑，一族内进程多则系统并发性强，但由于程序的计算量是确定的，分布式并发进程越多，进程粒度越小，多机操作系统开销增大，系统并行处理效率往往下降。因此，确定分布式并发进程个数时，应综合考虑系统中 CPU 个数，系统并发性，操作系统开销及并行处理效率等多种因素，权衡确定。由于分布式库函数可能调用其它库函数，允许 fork 嵌套使用。此时，出现一种进程，对于外层 fork，它是子进程，对内程 fork，它是父进程，即具有父/子双重身份的进程。资源分配时，若对其特殊处理，如进程控制块及信箱可以不再分配，双重身份的进程撤消则无需释放进程控制块与信箱。当分布式程序执行时，这种具有父/子身份的进程会频繁产生，这样特殊处理后将明显减少操作系统开销。

原语 join实现进程族交会，它可视作进程族内同步，称族同步。可用族同步计数器来实现族同步，执行 join 时，计数器减1．如结果非零，执行 join 的进程若为父进程则阻塞；若为子进程则撤销；如结果为零，对于子进程则唤醒父进程。

3.2 分布式并发进程通信

实现力反馈依从控制过程中，分布式并发进程间需要相互协同与配合。创建分布式并发进程是为了实现多机并行处理和缩短程序执行一遍所需时间。某些分布式并发进程计算的初始值要由另外的进程动态地给出，计算的结果需要传送，即分布式并发进程间需要通信。

在分布式计算机系统中，进程通信可按下列方式之一进行：

基于共享存贮器的进程通信；

基于信件传递的进程通信。

前者适用于紧耦合多机系统，后者主要用于没有共享存贮器的分布式计算机系统。

我们在多机操作系统内核中设置了进程通信原语来支持进程通信，它们是：send（参数表），receive（参数表）。

原语send完成信息发送，原语receive 完成信息接收。发送与接收按异步方式进行。由于分布式并发进程在不同的CPU上，它们的正文段与数据段处在不同CPU的局部存贮器中。虽然我们设计的多机系统中有共享存贮器，分布式并发进程也不宜直接通过共享变量方式进行通信，否则总线冲突过于激烈，引起并行处理效率急骤下降。在设计多机通信原语时有几点要特别注意：

第一，*C* 源程序中的变量是编译时静态分配存贮空间的，由于分布式并发进程分布的任意性，某些进程的变量地址在一些CPU上是合理的。而在另一些 CPU 上却是非法地址空间。为了解决这一问题，在进行分布式并发进程的处理机分配时，须对指针变量的值进行相应调整，以确保进程通信的顺利进行。

第二，在操作系统的总开销中，通信开销占比较大的比例。原语fork 嵌套时产生双重身份的父/子进程。如果处理机分配不当，例如将双重身份的子与父进程分配到不同的计算机上，则子与父也要通信。事实上，这种通信完全可以避免，在处理机分配时，只须将双重身份的父与子进程分配在同一个CPU上即可。

第三，在分布式计算机系统中，当出现下列情况：产生分布式并发进程，均衡系统负载，系统中CPU故障时，往往要进行进程迁移，即进程的正文段与数据段从一个CPU移至另一个CPU. 进程迁移由机间通信完成。为减少产生分布式并发进程时的进程迁移通信开销，在多机系统开工前可以静态地将用户程序分别向各个CPU加载，从而减少了进程迁移的通信开销。

3.3 进程管理与控制

多机系统中，任一时刻存在的进程数及进程状态都在改变着。经过一段时间间隔，有一些新进程产生，也有一些进程由于肩负的使命已经完成，从系统中消亡，系统中的总进程数大体上保持不变。这对任何计算机系统都是十分重要的。为此，多机操作系统的内核中安排了下列原语：

create（参数表）—创建原语

destroy（参数表）—撤消原语

block（参数表）—阻塞原语

wakeup（参数表）—唤醒原语

由于力反馈依从控制的顺序性，在实施分布式进程调度时采用 FIFO 原则。在分布式进程通信与同步过程中，有的进程会因某事件尚未发生而从运行状态变成阻塞状态，一旦等待的事件发生，它们又从阻塞状态变成就绪状态

4 分布式程序简例

```
main ( )
{
extern struct s1
        {
        int a,b,c,d,e,f,v1,v2,v3,v4,v5,v6,v7,v8;
        }       var1,*p1;
int     i;
p1=&var1;
(*p1).a=1;(*p1).b=2;(*p1).c=3;(*p1).d=4;
(*p1).e=5;(*p1).f=6;
sys();
p1->v1=(*p1).a+(*p1).b+(*p1).c+(*p1).v2+(*p1).v3+(*p1).v4;
p1->v1=p1->v1+p1->v5+p1->v6+p1->v7-p1->v8;
.....
}
struct s2
{
int     d1,e1,f1,u1,u2,u3,u4;
int     a1,b1,a2,b2,a3,b3,c3,a4,b4,c4,d4;
}       var2,*p2=&var2;
struct  s1
{
int     a,b,c,d,e,f,v1,v2,v3,v4,v5,v6,v7,v8;
}       var1,*p1;
struct  s3
{
int     a11,b11,a22,b22,a33,b33,c33,a44,b44,c44,d44;
int     v11,v22,v33,v44;
}       var3,*p3=&var3;
sys()
        {
int     i;
goto    e1;
```

```
e2: receive(1,2);
        for(i=1;i<200;i++)
        { p2->d1++; p2->e1++; p2->f1++; }
        p2->a1=80;p2->b1=4;p2->a2=150;p2->b2=200;
模块1   p2->a3=1000;p2->b3=500;p2->c3=20;p2->a4=100;
        p2->b4=50;p2->c4=10;p2->d4=6;
    sys1();
    send(1,2,&((*p2),d1),28);
    join(1,2);
e1: fork(1,1,&((*p1),v2),e2,&((*p2),d1));
    send(1,1,&((*p1).d),12);
模块2       for(i=1;i<150;i++)
            { ((*p1),a)++; ((*p1).b)++; ((*p1).c)++; }
    join(1,1);
    }
    sys1()
            {
    int     i;
    goto    m1;
m2: receive(2,2);
            for(i=1;i<130;i++)
            { p3->a22++; p3->b22++; }
            (*p3).v22=((*p3).a22)*6+((*p3).b22)*5;
    send(2,2,&((*p3).v22),4);
    join(2,2);
m3: receive(2,3);
            for(i=1;i<230;i++)
            { p3->a33++; p3->b33++; p3->c33++; }
            (*p3).v33=p3->a33*500-p3->b33*200+p3->c33*4;
    send(2,3,&((*p3).v33),4);
    join(2,3);
m4: receive(2,4);
            for(i=1;i<95;i++)
            { p3->a44++; p3->b44++; p3->c44++; p3->d44++; }
            p3->v44=p3->a44*56+p3->b44*72-p3->c44*20+p3->d44;
    send(2,4,&((*p3).v44),4);
    join(2,4);
m1: fork(2,3,&((*p2).u2),m2,&((*p3).a22),m3,&((*p3).a33),m4,&((*p3).a44));
```

```
send(2,1,&((*p2).a2),8,&((*p2).a3),12,&((*p2).a4),16);
        for(i=1;i<240;i++)
        {  ((*p2).a1)++;  ((*p2).b1)++;  }
        p2->u1=(p2->a1)*25+(p2->b1)*500;
join(2,1);
        }
```

程序包含A、B、C三部分，A是主函数，B与C是分布式库函数sys( )及sys1( )。主函数调用分布式库函数sys( )。B中fork产生两个分布式并发进程，模块1对应子进程，模块2对应父进程。父进程用send给子进程发送初值，子进程用receive接收它。sys1( )；是调用分布式库函数sys1，计算结果由send发送给父进程。join为族内同步操作。分布式库函数sys1( )能产生四个分布式并发进程。

5 结 束 语

本文介绍的分布式程序设计方法具有容易实现、用户易于设计分布式程序的优点。专用的分布式程序设计语言如SR，由于它具有说明及控制并发成份，具有描述交互作用的手段，在实现分布式处理的总开销中一部分由编译静态完成，其余由运行支持核心软件完成。相比之下，本文介绍的方法运行效率可能会低一点。

参 考 文 献

[1] 孙钟秀.分布式计算机系统.国防工业出版社，1987

[2] 金蓝，周笛.分布式计算机系统的程序设计.小型微型计算机系统，1985,(1)

[3] 程玉石，杨继远，程代杰.基于MPSCU的并行语言PC.微型计算机,1988(2)

[4] 汤子瀛，杨成忠.计算机操作系统.西北电讯工程学院,1987

The Implementation of the Distributed Programming for the Multiprocessor System in a Robot Force -Feedback Compliance Controller

Xiao Renqing

(Departmen of Automatic Control)

Abstract

This paper introduces a method of the realization of the distributed programming by combining programming language C with distributed library function in the robot force-feedback compliance controlling multiprocessor system. The form of the distributed library function, the multiprocessor

国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第12卷第4期 1990年12月 Vol.12 No.4

一种用于机器人力反馈依从控制的计算机体系结构

邹逢兴

（自动控制系）

摘 要 本文提出了一种用于机器人力反馈依从（Compliance）控制的计算机体系结构——M1MD型的多微机并行处理系统，并从硬件系统结构和系统软件两方面予以了说明。该系统是为完成一项实际科研任务而设计的，它能有效地实现机器人力/位置混合控制中多任务的并行处理，它的实现将使伺服控制周期缩短到5ms以下，能满足各种机器人控制任务的需要。

关键词 机器人计算机系统，并行处理/力反馈控制，计算机体系结构，多微机系统

分类号 TP302，TP242

力反馈依从控制技术和力控制器是研究和开发智能机器人的关键性基础技术之一。由于机器人力反馈依从控制是以操作空间法为基础的力/位置混合控制，实现它比实现以关节级的PID算法为基础的纯位置控制要复杂得多，计算量要大得多，需对控制计算机的计算速度和存储容量等性能提出更高的要求。为此，近几年来许多国家的专家们都在结合力／位置混合控制器的研究，对机器人控制的计算机体系结构进行探索和研究。

实践证明，采用目前的单处理机系统来实现机器人力反馈依从控制是肯定不行的，即使采用16位以至32位的超级小型机也无法满足它的计算需要；必须采用多机并行处理体系结构，实现机器人力反馈依从控制中多重任务的并行处理和控制，才能满足要求。

从目前所见，多机并行处理系统用于机器人控制方面，主要有三种结构型式，即流水线式结构，并行处理机结构和多处理机结构。但综合起来看，还是以采用多指令流多数据流(MIMD)的多处理机结构更合适、更普遍；而在机器人控制的多处理机结构中，

1989年10月7日收稿

operating system function supporting the design of the distributed library function, and some problems which have to be emphasized in the design process are described. A simple distributed program example is also given.

Key words programming, distributed computer, robot/distributed library function, distributed concurrent processes, primitives