第 6 卷第 7 期
1995 年 7 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 7
Jul. 1995

分布式程序设计语言 DC 及其在松散耦合分布式环境中的实现*

田籁声　黄莲淑　夏　滨

（吉林大学计算机科学系，长春 130023）

摘要　DC 是一种支持分布式程序设计的编程语言，它是由 C 语言向上兼容扩充得到的. 本文介绍 DC 及其在工作站网络环境下运行时支持系统的设计与实现，最后给出测试结果.

关键词　分布式程序设计，松散耦合，工作站网络.

人们普遍认为分布式处理将是今后计算机的主要处理模式. 分布式处理是一门综合性的科学技术. 程序设计语言是它的一个重要研究课题. 没有好的、实用的分布式程序设计语言，任何分布式系统都难以推广应用. 现代分布式系统多数是用高速局部网连接若干小型、微型计算机或工作站组成的松散耦合系统. 在这种系统中，由于机器之间没有共享的主存储器，所以一般的并发程序设计语言（如并发 Pascal，Modula 等）无法应用，必须使用分布式程序设计语言.

国外对分布式程序设计语言的研究开始于 70 年代中期. 目前已有多种分布式语言发表. 例如 CSP，Ada，SR，Argus 等，其中有的已经商品化，如 Ada. 不过，这些语言多数处于发展阶段，离成熟和标准化相距甚远，并且它们的应用普及程度很低. 造成这种局面的原因是多方面的. 我们认为其中有两个重要的原因：第一，语言是全新的，包罗万象，用户不愿从头学起. 第二，语言设计时不考虑实现；语言实现时不提供通用接口. 语言运行时支撑（run－time support）系统要靠用户自己根据具体分布式环境研制，而一般用户不具备这种能力.

基于上述状况，我们着手研制一种较为简单，但很容易推广应用的分布式程序设计语言 DC. DC 是以广泛使用的、优秀的编程语言 C 为基础，向上兼容扩充得到的. 由于 DC 是 C 语言的超集，增加的语言成份不多，所以熟悉 C 语言的用户很容易接受. 另外，语言实现时兼顾到运行时支撑系统，提供通用的软件接口，所以易于推广到各种分布式环境中.

1　DC 语言简介

DC 是由 C 语言向上兼容扩充得到的. 并发机制的设计主要参考 AT&T BELL 实验室

* 本文 1993－11－13 收到，1994－02－24 定稿

作者**田籁声**，女，1938 年生，教授，主要研究领域为计算机网络与分布式系统. **黄莲淑**，女，1968 年生，助教，主要研究领域为计算机网络与分布式系统. **夏滨**，女，1968 年生，助教，主要研究领域为计算机网络与分布式系统.

本文通讯联系人：田籁声，长春 130023，吉林大学计算机科学系

© 中国科学院软件研究所 http://www.jos.org.cn

Gehani 和 Roome 的"Concurrent C"[1,2]. 扩充部分包括 13 条新语句,提供多种并发机制. 如进程说明和创建,进程同步和交互,进程终止和异常结束,优先级说明和等待多个事件等.

1.1 扩充的会合(rendezvous)机制(事务处理)

DC 采用了 Concurrent C 中的扩充的会合机制(称事务处理). 这实际是一种同步信息传送模型. 由程序员来定义通过同步信息传送进行交互的进程. 同步信息传送原语把进程同步与信息传送结合在一起. 两个进程按如下方式进行交互:首先实现同步,而后传送信息,最后继续进行各自的单独活动. 这种同步称作会合. 在一个简单会合中,信息传送是单向的. Concurrent C 中扩充的会合机制能进行双向信息传送. 从顾客观点看,扩充会合几乎同函数调用一样.

1.2 进程和事务处理

DC 的并发构件是进程. 进程是进程定义的一个实例. 一个进程定义包括两部分:一个类型(规格说明)和一个体(实现). 进程类型定义的一般形式为:

Process spec 进程类型名(参数说明表)

{事务处理说明}

进程体的形式为:

Process body 进程类型名(参数名表)

{复合语句}

每一个进程都有自己的控制流,与其他进程并行. 进程可以提供一组入口供其他进程调用,以此实现进程间的同步和信息传送. 这些入口就是事务处理. 一个进程类型定义必须对该进程的每一个事务处理给出说明. 事务处理说明很象一个函数说明,只是前面加关键字 trans,并且参数类型是显式规定的:

trans 返回类型 事务处理名(参数说明表)

扩充的会合模型具有一个顾客进程(该进程启动一次交互——事务处理调用)和一个服务员进程(该进程等待交互). 每次事务处理调用,由顾客进程传送变元类型和返回值类型.

2 研制环境和总体方案

本文工作环境是一个 Sun 工作站网络,由 24 台 Sun4 工作站经以太网互连而成. 操作系统均为 SunOS 4.1.1.

总体方案由图 1 示意. 系统主要包括两个子系统:预编译器 PP 和运行时支撑系统 RTS. 图中用户节点机(基地机)是指网中用户预编译和启动执行 DC 程序的那台机器. 网中其他节点机统称远程机.

用户的 DC 源程序经 PP 预处理后,生成若干可以独立编译并行执行的 C 语言子模块,以及一组链表构成的与运行时支撑子系统 RTS 之间的软件接口. 运行时支撑系统根据接口中提供的信息,选择网中可以利用的空闲机,将 C 语言子模块分派到相应的网络节点机上. 之后,RTS 控制网中 C 模块并行编译和执行,相互合作完成一个共同的任务.

3 预编译子系统

DC 的实现,采取预编译加函数库的方法. 这种方法虽然冗余代码多,效率较低,但简单

© 中国科学院软件研究所 http://www.jos.org.cn

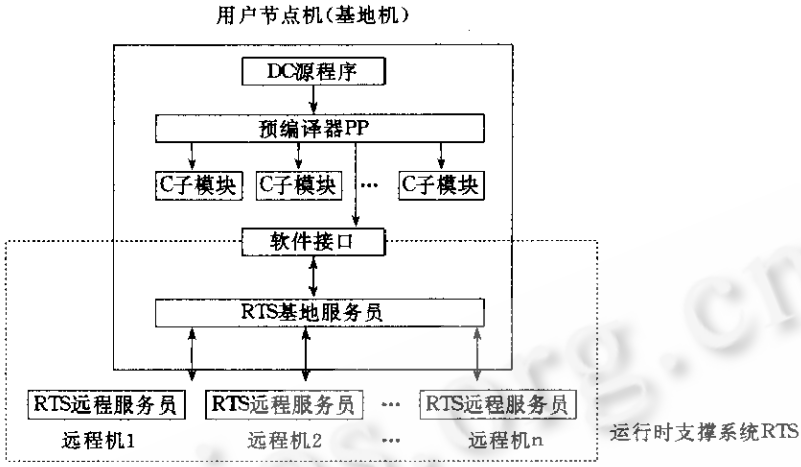

图1 总体示意图

易行并且可移植性好.

预编译子系统 PP 主要完成两项任务.

3.1 将 DC 源程序转换成可以独立编译和并行执行的 C 语言子模块

转换过程包括词法分析,语法分析,类型一致性检查,变量处理和目标结构生成等几个步骤. 实现中利用了 UNIX 提供的两个有效的工具 Lex 和 Yacc.

对变量的处理主要包括:

(1)对全局变量的处理

DC 语言中的全局变量(也称外部变量),与 C 语言相同,用 extern 说明. 这些变量外部于所有函数,可为多个函数共享. 当将 DC 源程序分解成多个可并行的子模块时,为确保生成的每个子模块都能独立编译和正确执行,需将源程序的全局变量作为每个子模块的全局变量,并统一管理在机间互斥共享.

(2)对 main()中变量的处理

一个 DC 程序开始执行时,只有一个进程. 这个进程称为初始进程. 它是运行时支撑系统的组成部分. 初始进程调用名为 main()的函数. 此后,该函数必须建立所有其他进程. 也就是说,建立进程的运算符 create 只能出现在 main()中.

预编译时,每当 PP 扫描到一个 create,都要生成一个并行子模块. 这样,main()中所有的变量说明和第一个 create 语句之前的可执行语句,应为所有的并行子模块共享. 所以预处理时,将 main()中变量说明及第一个 create 语句之前的所有可执行语句,复制到每个子模块的相应位置上.

(3)进程体中内部变量的处理

由于在预编译过程中,进程体(body 语句)被转换为函数形式,所以进程体中内部变量说明语句不变,直接作为对应函数的局部变量说明.

3.2 建立 PP 与 RTS 间的软件接口

预编译器 PP,除完成常规的 DC 到 C 的转换外,还负责建立它与运行时支撑子系统 RTS 之间的通用软件接口.

接口由 5 个表组成:进程类型名表,事务处理说明表,事务处理调用表,事务处理实参表

© 中国科学院软件研究所 http://www.jos.org.cn

和事务处理结果表.其中前2个表是由PP填写的静态表.后3个表在用户程序执行过程中由RTS动态填写.

该接口有以下特点:(1)结构简单、清晰;(2)提供的信息是完备的;(3)具有通用性——对RTS的实现环境没有限制,可以是微机/工作站网络,也可以是单处理机系统或多处理机系统.

为了提高系统效率,将软件接口放在基地机的共享存储器区段中.SunOS的共享存储器区段,允许两个或两个以上进程共享,但不提供互斥机制.实现中,我们利用P.V原语和信号灯保证进程间互斥访问.

在分布式环境中实现时,运行时支撑系统将各并行子模块派往网中各节点机时,也要将该接口复制到相应的节点机上.

4 运行时支撑子系统

运行时支撑子系统建立在通用网络操作系统SunOS之上.提供以下功能:

①按一定原则,将C语言子模块分配到网中空闲节点机上.

②控制多个并行的C语言子模块在网中不同机器上分别编译和执行,管理各子模块间的交互以实现正确的同步和信息传送.

③管理共享资源,避免死锁.

④处理异常事件(机器瘫痪,通信失效等).

运行时支撑子系统由图1下半部分示意.它主要由两类服务员组成:基地服务员和远程服务员.用户节点机上的服务员,称为基地服务员.它包括子模块分配进程,本地服务进程和若干代理进程.远程机上的服务员,称为远程服务员.它包括远程守护进程,远程执行进程,I/O管理进程,通信管理进程以及远程执行监督(watchdog)进程.基地服务员由用户命令启动.远程服务员始终在网中各节点机的后台运行.

DC语言用户调用PP预编译自己的源程序后,再键入命令RTS,便启动了运行时支撑子系统.此后,用户程序就在运行时支撑子系统的控制下在网中分布并行运行.其执行细节对用户透明.用户感觉不到网络的存在,仿佛程序是在本地单机执行,只是速度提高了.

图2示意一个用户程序执行过程中,运行时支撑系统各部分工作情况.

(1)用户键入RTS命令后,运行时支撑系统的基地服务员立即启动.它首先激活子模块分配进程.子模块分配进程查看接口中进程类型名表,当发现已经有子模块登记时,便通知本地服务进程修改网络文件系统(NFS)状态,以便远程执行的子模块能够访问基地节点上的相应文件.

(2)子模块分配进程按一定的调度策略,为子模块分配节点机,并且为每个获得空闲节点机(远程机)的子模块启动一个代理进程.代理进程的主要作用是:与指定的远程机建立连接;在屏幕上开一个tty子窗口与远程机对应;将进程类型名表和事务处理说明表复制到远程机上.当一个子模块在远程机上执行时,其代理进程就是它在基地机上的"影子".用户与远程执行子模块的交互均通过该代理进程进行.

(3)基地机代理进程向相应的远程机发送远程执行请求和子模块.当信口中有远程执行请求时,远程服务员的远程守护进程被唤醒.

© 中国科学院软件研究所 http://www.jos.org.cn

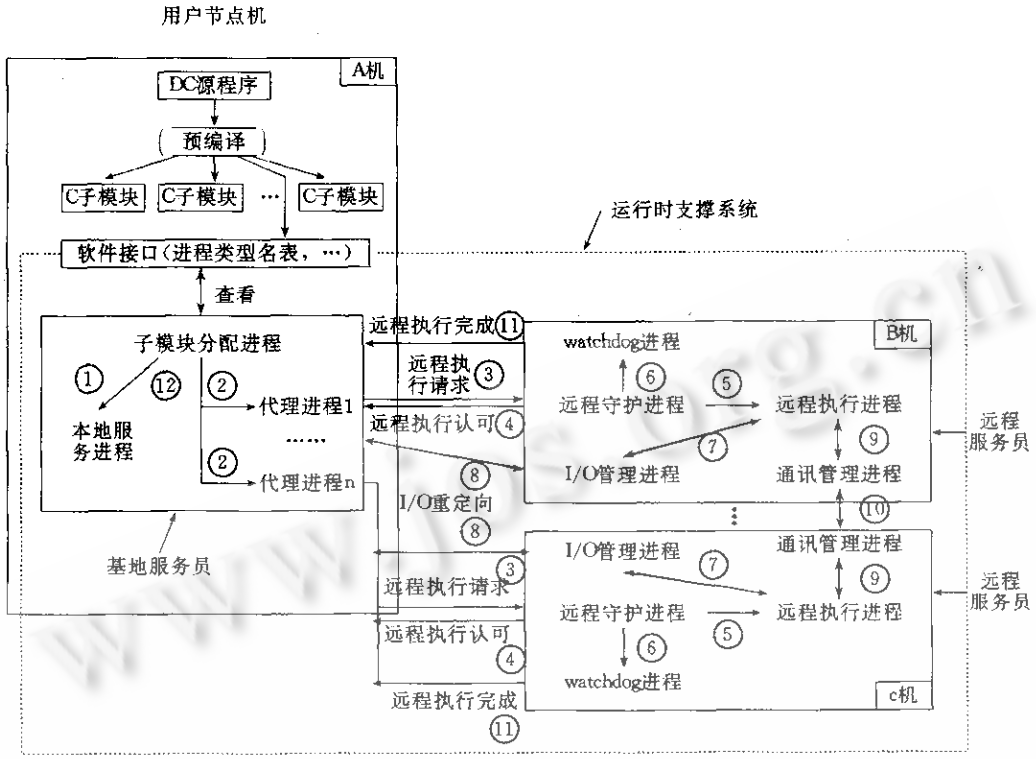

图2 DC程序执行过程示意图

(4)若没有异常情况，远程守护进程向基地机的代理进程回送一个远程执行认可，表示远程机可以为用户服务.

(5)远程守护进程建立一个远程执行进程，它负责启动 C 语言子模块的编译和执行.

(6)远程守护进程启动远程执行监督(watchdog)进程，用以监视子模块的执行情况.

(7)在子模块执行过程中，若有输入输出操作，则通知 I/O 管理进程.

(8)远程机的 I/O 管理进程与基地机的代理进程联系，将远程执行子模块的 I/O 定向到基地机该子模块的 tty 子窗口上.

(9)在子模块执行过程中，若有事务处理调用则启动通信管理进程.

(10)通信管理进程向相应的目的机(事务处理的服务员进程) 发出事务处理调用.

(11)watchdog 进程监视子模块的远程执行状态，并且每隔一定时间(例如 10 秒)，向基地机相应的 tty 子窗口输出一个圆点，以表示本子模块正在远程执行. 当子模块执行完毕，watchdog 进程向基地机的代理进程发回一个远程执行完成的信件.

(12)当所有的子模块都执行完或程序满足终止条件时，基地机的子模块分配进程便通知本地服务进程恢复 NFS 状态. 到此，本次基地服务员任务完成，自动撤消.

5　性能测试及结论

为了检验本项工作的正确性并初步测试系统性能，我们用 DC 语言编写了若干程序在网中运行. 这里给出其中的两个例子，一个是 5 位哲学家就餐问题，另一个是求积分运算.

© 中国科学院软件研究所 http://www.jos.org.cn

5位哲学家就餐问题是个非常典型的例子,已被许多文献引用过.尽管它具有明显的简单性,但却能具体说明在并行程序设计中遇到的许多问题(共享资源,死锁等).所以,它常被用作检验并发程序设计机制合适程度的基准.由于5位哲学家中的每一位和5把叉子中的每一把都作为一个进程实现,所以用DC编写的5位哲学家就餐的源程序经预编译后,在基地机上形成10个C语言子模块.其中有5个叉模块,5个哲学家模块.哲学家由并行进程代表.叉子是哲学家的共享资源.运行时支撑系统根据接口信息,将这些子模块分派到网中10台机器上,然后控制它们分别编译执行.执行过程中,分布在10台机器上的子模块间要进行非常频繁的交互(事务处理调用).每位哲学家吃面时,必须先拿起右叉子,然后拿起左叉子.吃完后,要先放下左叉子后放右叉子.拿起叉子和放下叉子都对应哲学家模块所在机器对叉子模块所在机器的一次事务处理调用.这样,每位哲学家每吃一次面就有4次交互.本例运行结果非常好.可以让5位哲学家吃吃想想任意多次(试验时,最多试到100000次)不死锁,也不发生不确定的延迟(活锁).直到他们全部离开餐桌到阴间去,程序正常结束.

另一个例子是求积分运算($\int_{0.001}^{100} \exp(-x)$),想以此初步检测一下系统的性能.我们分别在单机和多台(4台)机器上循环计算该积分100次,400次,1000次.其结果如下:

循环次数	单机运行时间(秒)	多机(4台)运行时间(秒)	加速比
100	440	219	2.00
400	1762	705	2.49
1000	4406	1626	2.70

从结果看出,目前该系统运行时开销较大,只有在大计算量时才能获得较高的加速比.

以上检测结果说明,DC语言虽然简单粗糙,不甚规范,但比较实用.运行时支撑系统性能有待改进,但切实可行.DC语言及其运行时支撑系统是在Sun工作站网络中实现的,但由于设计了一个既简单清晰又比较完备通用的接口,所以很容易移植到其它分布式环境中.

参考文献

1 Gehani N H, Roome W D. Concurrent C. Software Practice & Experience, 1986,16:821—844.

2 Cmelik R F, Gehani N H, Roome W D. Experience with multiple processor versions of concurrent C. IEEE Transactions on Software Engineering, 1989,15(3):335—344.

附录:求积分运算的源程序清单

```
Process Spec summ()
{
  Trans void put(double sf);
};
Process Spec qromo1(int pi2,double a,double b,int
f);
Process Spec qromo2(int pi1,double a1,double b1,int
f);
Process Body qromo1(pi2,a,b,f)
{
  double sdd1=0.00,sf1;
  int i3;
  for(i3=0;i3<250;i3++) {
    sf1=qromo(pi2,a,b);
    sdd1=sf1+sdd1;
  }
  Sendt f.put(sdd1);
}
Process Body qromo2(pi1,a1,b1,f)
{
  double ss,ss2;
  int i1;
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
    for(i1=0;i1<250;i1++) {
      ss=qromo(pi1,a1,b1);
      ss2=ss2+ss;
    }
    Sendt f. put(ss2);
  }
  Process Body summ()
  {
    double sum=0.00;
    double sss;
    int i2;
    for(;;) {
      Accept put(sss)
      {
        sum=sum+sss;
        i2++;
      }
      if(i2==4)
        break;
    }
    printf("sum :%f\n", sum);
  }
  main()
  {
    int f1;
    int ppp,ppp1;
    double yy0,yy1;
    double funf(double);
    double fung(double);
    double qromo(int,double,double);
    ppp=0;
    ppp1=1;
    f1=Create summ() Processor jida7; // jida7 is a
machine name //
    yy0=0.001;
    yy1=100.00;
    Create qromo1(ppp1,yy0,yy1,f1);
    Create qromo2(ppp1,yy0,yy1,f1);
    Create qromo1(ppp1,yy0,yy1,f1);
    Create qromo2(ppp1,yy0,yy1,f1);
  }
  double funf(double x)
  {
    double g;
    g=log(x);
    return(g);
  }
  double fung(double x)
  {
    double po;
    po=exp(-x);
    return(po);
  }
  double qromo(int piu,double a,double b)
  {
    double s[15],h[15],ha[5],sa[5];
    void polint(double *,double *,int,double,double
*,double *);
    void midinf(int,double,double,double *,int);
    int j,t,u;
    double ss,dss;
    h[0]=1;
    for (j=0;j<14;j++) {
      midinf(piu,a,b,&s[j],j);
      if (j>=4) {
        t=j;
        for (u=0;u<5;u++) {
          ha[u]=h[t-4];
          sa[u]=s[t-4];
          t++;
        }
        polint(ha,sa,5,0.0E0,&ss,&dss);
        if (fabs(dss)<(1.0E-6)*fabs(ss))
          return(ss);
}
      s[j+1]=s[j];
      h[j+1]=h[j]/9.0E0;
      }
      return(ss);
    }
    void polint(double *xa,double *ya,int n,double x,
double *y,double *dy)
    {
      double c[10],d[10];
      int ns,i,m;
      double dif,dift,he,hp,den,w;
      ns=0;
      dif=fabs(x-x[0]);
      for (i=0;i<n;i++) {
        dift=fabs(x-xa[i]);
        if (dift<dif) {
          ns=1;
          dif=dift;
        }
        c[i]=ya[i];
        d[i]=ya[i];
      }
      *y=ya[ns];
      ns=ns-1;
      for (m=1;m<=n-1;m++) {
        for (i=0;i<n-m;i++) {
          ho=xa[i]-x;
          hp=xa[i+m]-x;
          w=c[i+1]-d[i];
          den=ho-hp;
          den=w/den;
          d[i]=hp*den;
          c[i]=ho*den;
        }
        if (2*ns<(n-m)) {
          *dv=c[ns+1];
        }
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
        ns=ns-1;
      }
      *y=*y+*dy;
      }
    }
    double funk(int poo,double yt)
    {
      double yr,funkk,y2,y3;
      yr=1.0E0/yt;
      if (poo==0) {
        y2=funf(yr);
        y3=yt*yt;
        funkk=y2/y3;
        return(funkk);
      }
      else {
          y2=fung(yr);
          y3=yt*yt;
          funkk=y2/y3;
          return(funkk);
      }
    }
    void midinf(int po,double aa,double bb,double *s,
int n)
    {
    double x,del,ddel,sum,ko,a,b;
    int tmm,j;
    static int it;
    b=1.0E0/aa;
    a=1.0E0/bb;
    if (n==0) {
      x=0.5E0*(a+b);
      *s=(b-a)*funk(po,x);
      it=1;
    }
    else {
        tnm=it;
        del=(b-a)/3.0E0*tnm);
        ddel=del+del;
        x=a+0.5E0*del;
        sum=0.0E0;
        for (j=0;j<it;j++) {
            ko=funk(po,x);
            sum=sum+ko;
            x=x+ddel;
            *s=(*s+(b-a)*sum/tnm)/3.0E0;
            it=3*it;
        }
    }
```

IMPLEMENTATION OF DC IN A LOOSELY—COUPLED DISTRIBUTED ENVIRONMENT

Tian Laisheng Huang Lianshu Xia Bin

(*Department of Computer Science, Jilin University, Changchun* 130023)

Abstract DC is a programming language which supports distributed programming. It is an upward—compatible parallel extension of C. This paper presents design and implementation of DC and its run—time support system in a workstation network. It also provides the test results.

Key words Distributed programming, loosely—coupled, workstation network.

© 中国科学院软件研究所 http://www.jos.org.cn