网络并行虚拟平台 PVM 3*

孙家昶　（中国科学院计算中心）

摘要：网络机群系统是并行系统与应用的一个重要发展方向。它具有高性能价格比、灵活性强以及可扩展性等优点，适合我国国情。异构网络并行虚拟平台 *PVM(Parallel Virtural Machine)* 是国际上近年推出的软件系统，已被众多计算机公司所接受为公共标准化的并行软件平台。一批高水平的串行软件正在逐步移植到 *PVM* 平台上。本文主要取材于 *PVM 3.0* 的使用说明，同时也参考了有关的材料，较详细介绍 *PVM3* 的功能以及使用。

一、引论

1.什么是 PVM?

* PVM 是 Parallel Virtual machine(并行虚拟计算机)的简称。这是一个软件系统，它的主要功能是把网络上各种同构或异构的计算机都利用起来，给用户提供一个统一的和灵活的并发计算资源。简言之，PVM 是负责管理由不同计算机构成的网络系统的软件。

* 网络上连接的计算机可以是标量工作站，也可以是共享或局部存储多处理机、向量机、甚至是超级计算机。网络连接的方式也可以是多样的。例如以太网(Ethernet)及光纤网(FDDI)等。

* PVM 对用户配置池中每台机器提供软件支撑，给用户提供一个统一的、通用的、功能齐全的并发应用计算环境。

* 用户程序可用 C 或 Fortran 编写，用户通过调用 PVM 库程序完成诸如进程初始化、信息传递与接收以及同步等功能。

* 用户可有选择地控制特定应用模块的执行方式，在以下三个级别上进行操作：

(*)透明级：应用程序中的每个源程序要由 PVM 自动分配到一个合适的网络结点机上运行；

(*)体系相关级：用户可以指定每一个原任务应该在哪一种特定的体系结构的计算机上运行；

(*)结点相关级：用户可以指定一个原任务要在哪一个结点机上运行。

2.为什么要研制 PVM?

据 PVM 研制者之一，著名的数学软件 LINPACK 作者 J.Dongarra 所述，他们研制与推出 PVM 软件系统主要有以下几个动机：

* 并行计算机市场变化过快，以至并行软件的研制与开发跟不上并行机更新淘汰的周期(目前国际上一个具体并行机机种的更新周期只有 3–5 年)；

* 为提高性能价格比，提高现有计算机资源的利用率；

* 为开拓并行计算用户范围。对大多数涉及并行计算的部门而言，他们只是需要时才使用并行环境，此时如能提使给他们适当的并行环境，他们没有必要非要购买并行机，特别是为不具备并行计算机的教学科研部门提供为讲授并行编制课程所需的并行环境。

3.PVM 研制简史

PVM 最早是由 Emory 大学的 V.Sunderam 与 Oak Ridge 国家实验室的 R.Mancheck 开始研制(1989 年夏)，现尚在继续开发中，1993 年 4 月推出 3.1 版，1994 年 5 月推出 3.3 版。参与新版本研制开发的还有 Oak Ridge 国家实验室的 A.Geist、J.Dongarra，Carnegie Mellon 大学的 A.Begnelin 及田纳西大学 W.Jiang 等。

PVM 研究项目属于美国高科技基础研究，由美国自然科学基金会、能源部及田纳西州全力资助，同时也得到美国各大计算机公司的支持。

4.PVM 的可移植性与可伸缩性(Scalability)

PVM 通过 UNIX 系统，已可转植到下列机器上运行：

80386/486 with BSDI	Alliant FX/8
DEC Alpha/OSF-1	BBN TC2000
DEC Microvax	Convex
DEC station	Cray YMP and C90
DG Aviion	IBM 3090
HP 9000/300	Intel Paragon

* 国家“863”计划资助项目

HP 9000 / 700	Intel iPSC / 2
IBM RS / 6000	Intel iPSC / 860
IBM / RT	Kendall Square KSR-1
NeXT	Sequent Symmetry
Silicon Graphics IRIS	Stardent Titan
Sun 3	Thinking Mach.CM-2,5
Sun 4, Sparc	

PVM3 系统目前已可以在 100 台结点机上运行(每个并行机亦按一个结点机计),其效率与可伸缩性主要受网络传输限制。随着网络协议的改进,FDDI 光纤网以及其它高速网的出现,PVM 系统今后几年内可望能具有更强的可伸缩性,可扩至上千台结点机。

二、PVM3 系统的结构与特性

1.PVM 系统的组成

PVM 系统主要由核心程序(pvmd)、库函数(Libpvm)以及应用程序三部分组成。它们各自的功能如下:

(1)Pvmd 核心程序(daemon program)。

(*)驻留在并行虚拟机的每个结点机上运行,形成用户空间进程;

(*)提供结点机之间的通信,维持数据包的顺序性以免丢失;

(*)鉴别任务;

(*)在结点机上执行进程;

(*)提供错误的诊断;

(*)比应用程序具有更强的健壮性,采用有限自动机方式避免死锁;

(*)每个结点机上的核心程序完全对称,避免主从式引起的系统瓶颈问题;

(*)实时地通过广播功能将结点的当前运行信息传送到其它结点。

(2)Libpvm 程序库

这是 PVM 的接口程序库(记为 Libpvm3.a),包含用户可调用的标准化子程序,如消息传递、产生进程、任务分类以及虚拟机变化等。每一个应用程序都必须连接 Libpvm,其主要功能是实现低层 PVM 子调用"syscall"功能,如进程管理(进程的适应、注册与同步)数据传送(初始化、发送与接收)、共享存贮机制(提供进程可共享数据段的建立、赋值及删除)以及上锁、开锁操作。

(3)应用元程序

这是用户用 PVM 消息传递调用写成的。是 PVM 中可执行的"任务"。

2.PVM3 版的主要特性

由于 PVM 前两个版本存在很多本质上的缺陷,特别是连接多处理器端口有困难,伸缩性差,不具备容错能力等,PVM3 在很多方面作了重新设计和改进。主要有以下八个方面:

(1)更新用户界面名 为防止 PVM 子程序与计算机用户的库函数重名,PVM3 的用户子程序凡属 C 语言的,均以 PVM__开头,Fortran 的均以 PVMf 开头。

(2)整型任务标识符 进入 PVM 的所有进程都以一个整型标识符标记为 tid。tid 是 PVM 中标识进程最主要的与最有效的方法。因为,tids 整型量在整个虚拟器中是唯一的,它们是由局部核心程序 PVMd 提供的,而不是由用户选择的。PVM3.0 中包含有某些子程序会返回 tid 值,使用户应用时能够识别系统中的其它进程,例如 pvm__mytid、pvm__gettid、pvm__parenl 等。

(3)进程控制 PVM 能够把一个用户进程转变成为一个 PVM 任务,或再变为一个正常的进程。例如可动态增加或减少结点机,启动或中断 PVM 任务;向其它的 PVM 任务发送信号;找出有关虚拟机配置的信息以及有效 PVM 任务的信息。

(4)容错措施 如果某个结点机有问题,PVM 会自动诊断发现,并把该结点从虚拟机目录中删掉,应用程序可询问结点的状态,若有需要可由应用程序增加一个替代的结点机。PVM 不试图自动恢复由于结点机出问题而被删掉的任务。结点机的容错问题依然是应用开发者的责任,PVM3 已提供建立容错编程系统的平台。

(5)动态进程群 在 PVM3.0 的顶层已实现动态进程群,一个进程可以同时属于多个群,这些群可在计算的任何时刻动态改变,逻辑上与诸如广播及障碍等任务群有关的函数采用用户显式定义群名字当作变量,进入或退出一个有名群的各个任务则由系统提供子程序,诸任务均可询问其它群成员的信息。

(6)信号传输 PVM 提供二种方法向别的 PVM 任务传输信号。一种方法是向另一个任务传送一个 UNIX 信号。第二种方法,是用传送消息通知一组任

务，消息的内容涉及用户指定应用程序可检验的标记，包括一个任务的退出，删除或增加一个节点，某节点出错等。

(7)通信　PVM 提供各任务之间用以包装及发送信息的子程序，能够向单一任务异步发送，也能按任务单子多向传输。接收的信息可以是源程序，也可以是带有块或非块接收子程序的信息标记。接收时“野卡”(wildcard)专门挑出，其内容不予接收。可调用子程序以返回接收信息的情况。信息缓冲区是动态分配的，可发送或接收信息的最大规模只受所指定结点机可用内存的大小的限制。信息可以存储或转运。

(8)多处理器集成　PVM 最初是为连接网络上的诸单机开发的。诸如 Intel iPSC／860 等分布存储多计算机通常只有一个主机或是一个结点处理器可与网络实际连接。所有其它处理器只能通过该计算机的内部通讯途径进行通讯。为了适应这种情况，PVM3.0 放松了对于 UNIX 环境以及 TCP(传输控制协议)的依赖。现在用 PVM3 写的程序可在 SUN 工作站机群网络上运行，也可在 Intel Paragon 一组结点机上运行，又可以在全世界分布的多处理器计算机组成的异构网络上运行，不需要补写任何顾客特殊的信息传递指令。在一个多处理器内部，PVM 设计为采用该机内部的通讯调用，同一个多处理器二个节点之间信息直接传输 Intel iPSC／860 已经集成到 PVM 中，因而 Intel 的 NZ 信息传递子程序已用于内节点通讯。Cray，Convex，SGI，Intel 以及 IBM 等公司还在提供他们各自多处理机。一旦推出就与 PVM3.0 的可兼容。更多的多处理机将在以后的版本 PVM3.x 时增加上。

三、PVM3 系统的运行过程与程序结构

1. 我们以图 1 为例说明 PVM3 的运行过程。假设网络上有四台机器：三台工作站(Alpha,RS6k,SUN4)及一台具四个结点的并行机 iPSC／860。现有九个任务，其中任务 4 为控制，任务 9 为图形显示，其它均为计算任务。

由此可见，PVM 的主要运行过程如下：

(1)在每个结点机上装入 PVM 核心程序；

(2)用户将应用问题划分为相对独立的多任务，并确定哪一类任务(或哪个任务)需要在网络上运行的计算机种类。如上图中任务 4 控制块在 RS6k 上进行，三项与图形显示有关的任务 1，2，3 在 Alpha 工作站上运行，其计算结果通过高速网连接到 SUN4 工作站，组成任务 9 图形显示，另有四个计算任务 5～8 分配到并行机 iPSC／860 上运行。

(3)各个任务模块之间的通信与同步通过调用 PVM 用户库函数中提供的相应函数实现。

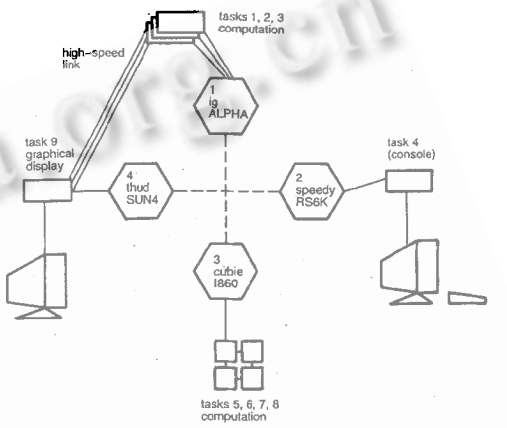

图　1

图例 ◇pvmd 过程 □pvm 任务 △(socket)接口

(4)用户需要自行完成各任务模块在所运行结点机中的编译(将 C 或 Fortran 源程序变成可运行代码)。

(5)PVM 系统根据用户提供的任务清单自动在各个结点机上创立相应的进程，管理这些进程的通信与同步。

(6)在用户程序完成或者出错情况下返回运行结果状态。

总之，PVM 系统负责管理网络上异构结点机上用户进程的初始化、通信、同步、处理器的分配以及不同结点机之间数据格式转换等工作。

2.PVM 系统程序结构与目录组成

总目录名为～／pvm3。所有的库函数与可执行程序均装入～／pvm3／1:b／ARCH／。控制命令 PVM 须进入你的 shell path。

附：PVM3 目录中的子目录结构与功能说明。

README	PVM系统装入与启动说明
MAKEFILE	建立可执行情况的批文件
[DOC]	系统的文本手册
[SRC]	系统的C语言源程序与头文件，约348k

内含若干[ARCM]子目录与38个源程序。

[CONSOLE] 控制命令目录，给用户一个shell管理虚拟机及相应任务，内含若干[ARCH]子目录与四个文件，约35k

[INCLVDE] 包括C与Fortran所需调用的二个头文件：PVM3.H及FPVM3.H，约11k

[LIB] 包括启动PVM3以及DEBUG调试程序等七个文件，约6.6k (装入后所含的[ARCH]库函数与可执行文件未计入)

[LIBFPVM] Fortran调用所需的头文件与库函数。内含若干[ARCH]子目录与42个文件，23k

[PVMGS] PVM成组服务器管理目录，内含若干[ARCH]子目录与七个有关文件。28k

[EXAMPLES] 内含四个Fortran与C应用实例，共11个文件，(包括一个Fortran调用PVM自检程序) 35k

[GEXAMPLES] 成组PVM命令实例。共有九个文件，包括一个MAKEFILE文件及八个C语言文件，未列入Fortran文文。10k

[XEP] 应用x-window显示分形的C语言实例，内含若干[ARCH]子目录及九个文件，共42k。

[BIN] [ARCH]可执行程序

四、用户界面

PVM中的子程序用C语言编写。Fortran用户可以通过PVM3所提供的用户界面调用这些子程序。主要有下列六个方面的界面：

1.进程控制程序

进入PVM: pvmfmytid
退出PVM: pvmfexit
派生进程: pvmfspawn
删除任务: pvmfkill

2.信息表示程序

相关进程信息: pvmfparent
进程状态信息: pvmfstat
结点状态信息: pvmfmstat
系统配置信息: pvmfconfig
任务信息: pvmftasks
动态配置信息: pvmfaddhost
pvmfdelhost
信号传输信息: pvmfsending

3.出错信息程序

出错状态打印: pvmfperror
出错打印控制: pvmfserror

4.消息传递程序之一:缓冲区管理程序

缓冲区初始化 pvmfinitsend
创造新缓冲区 pvmfmkbuf
择放缓冲区 pvmffreebuf
返回发送缓冲区 pvmfgetsbuf
返回接收缓冲区 pvmfgetrbuf
指定发送缓冲区 pvmfsetsbuf
指定接收缓冲区 pvmfsetrbuf

5.消息传递程序之二:数据打包与拆包

数据打包 pvmfpack
数据拆包 pvmfunpack

这二个程序的功能分别是把给定的数据类型数组打包进入有效的发送缓冲区，以及把有效的消息缓冲区的内容拆包成为所指定的数据类型数组，拆包时的变量形参含义必须与打包时相同。

6.消息传递程序之三:发送与接收

结点与结点间的立即发送:
pvmfsend
全局广播发送: pvmfmcast
成组接收: pvmfrecv
非成组接收: pvmfnrecv
局部广播发送: pvmfbcast
障碍同步: pvmfbarrier

五、PVM编程实例(Fortran语言部分)

所附的两个例子说明如何在PVM环境下编写Fortran程序

例1:Master / Slave模式

这是常用的一种并行编程平台，由Master程序派生出一批Slave进程，并调度这些计算进程，各Slave进程之间允许有通信。

Master 程序的主框图如下:

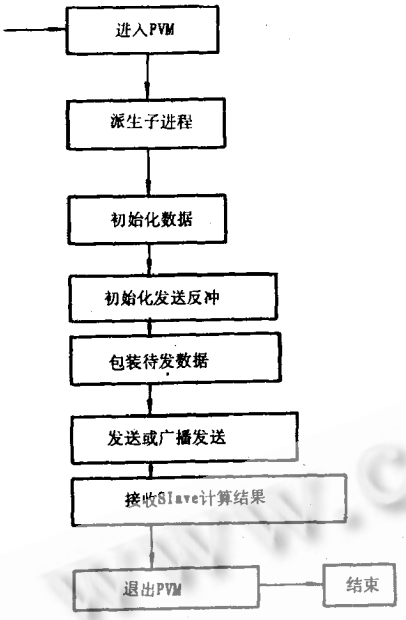

图　2

Slave 程序的主框图如图 3

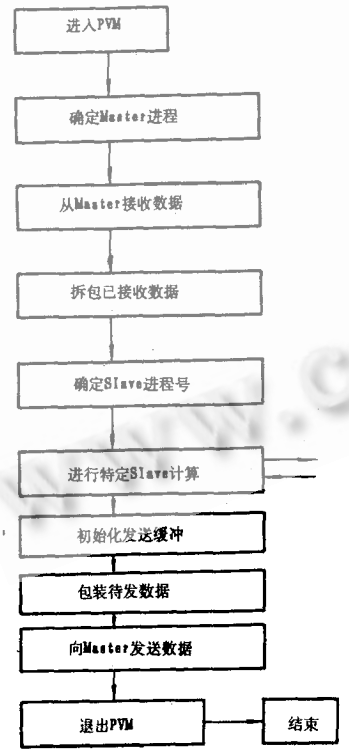

图　3

例 2: SPMD 模式(Single-program multiple data)

在这种模式下,所有进程都是地位对称的,没有一个 Master 进程在指挥。当处理器很多时可以防止由 Master 引成的瓶颈。这时只需一份程序。这类程序有时称为"无主机程序",必须从 Unix 提示符下启动。

SPMD 程序的主框图如图 4

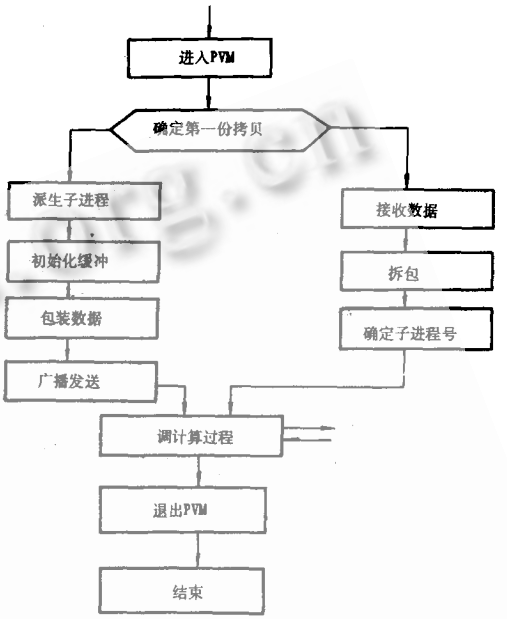

图　4

程序 1

```
                program master1
c INCLUDE FORTRAN HEADER FILE
      include 'fpvm3.h'

      integer i,info,nproc,numt,msgtype,who,mytid,tids(0:32)
      double precision result(32),data(100)
      character * 12 nodename,arch

c     Enroll this program in PVM
      call pvmfmytid(mytid)

c     Initiate nproc instamces of slave1 program
      print *,'How many slave programs(1-32)?'
      read *,nproc
      nodename = 'slave1'
      call pvmfspawn(nodename,PVMDEFAULT,' * ',nproc,tids,numt)

c--------Begin user program----------
      n = 100
      call initiate-data(data,n)

c     Broadcast data to all node programs
```

```
      call pvmfinitsend(0,info)
      call pvmfpack(INTEGER4,nproc,l,l,info)
      call pvmfpack(INTEGER4,tids,nproc,l,info)
      call pvmfpack(INTEGER4,n,l,l,info)
      call pvmfpack(REAL8,data,nl,info)
      msgtype=1
      call pvmfmcast(nproc,tids,msgtype,info)

    wait for results from nodes
      msgtype=2
      do 30 i=1,nproc
        call pvmfrecv(-1,msgtype,info)
        call pvmfunpack(INTEGER4,who,l,l,info)
        call pvmfunpack(REAL8,result(who),l,l,info)
 30   continue
c-------End user program--------
c     program finished leave PVM before exiting
      call pvmfexit( )
      stop
      end
      program slavel
c INCLUDE FORTRAN PVM HEADER FILE
      include'fpvm3.h'

      integer info,mytid,mtid,msgtype,me,tids(0:32)
      double precision result,data(100)
      double precision work

c     Enroll this program in PVM
      call pvmfmytid(mytid)
c     Get the master's task id
      call pvmfparent(mtid)

c-------Begin user program--------

c     Receive data from master
      msgtype=1
      call pvmfrecv(mtid,msgtype,info)
      call pvmfunpack(INTEGER4,nproc,l,l,info)
      call pvmfunpack(INTEGER4,tids,nproc,l,info)
      call pvmfunpack(INTEGER4,n,l,l,info)
      call pvmfunpack(REAL8,data,n,l,info)

c     Determine which slave I am(0--nproc-1)
      do 5 i=0,nproc
        if(tids(i).eq.mytid)me=i
 5    continue

c     Do calculations with data
      result=work(me,n,data,tids,nproc)

c     Send result to master
      call pvmfinitsend(PVMDEFAULT,info)
      call pvmfpack(INTEGER4,me,l,l,info)
      call pvmfpack(REAL8,result,l,l,info)
      msgtype=2
      call pvmfsend(mtid,msgtype,info)

c-------End user program--------

c     program finished.Leave PVM before exiting
      call pvmfexit( )
      stop
      end
```

程序 2

```
            program spmd
c INCLUDE FORTRAN PVM HEADER FILE
      include'fpvm3.h'

      PARAMETER(NPROC=4)
      integer mytid,me,numt,i
      integer tids(O:NPROC)
c       ENROLL IN PVM
      call pvmfmytid(mytid)
c       FIND OUT IF I AM PARENT OR CHILD
      call pvmfparent(tids(0))
      if(tids(0).lt.0)then
        tids(0)=mytid
        me=0
c       START UP COPIES OF MYSELF
       call pvmfspawn('wpmd',PVMDEFQULT,'*',NPROC-1,tids(1),numt)
c       SEND TIDS ARRAY TO CHILDREN
       call pvmfinitsend(0,info)
       call pvmfpack(INTEGER4,tids,NPROC,1,info)
      call pvmfmcast(NPROC-1,tids(1),0,info)
      else
c       RECEIVE THE TIDS ARRAY AND SET ME
       call pvmfrecv(tids(0),0,info)
       call pvmfunpack(INTEGER4,tids,NPROC,1,info)
       do 30 i=1,NPROC-1
        if(mytid.eq.tids(i)) me=i
 30    continue
      endif
c-----------------------------------------------------------
c     all NPROC tasks are equal now
c     and can address each other by tids(0) thru tids(NPROC-1)
c     for each process me=> process number [0-(NPROC-1)]
c-----------------------------------------------------------
      call dowork(me,tids,NPROC)
c       PROGRAM FINISHED EXIT PVM
      call pvmfexit()
```

```
      stop
      end

      subroutine dowork(me,tids,nproc)
c-----------------------------------------------------------
c  Simple subroutine to pass a token around a ring
c-----------------------------------------------------------
      integer me,nproc,tids(0:nproc)
      integer token,dest,count,stride,msgtag
      count = 1
      stride = 1
      msgtag = 4

      it(me.eq.0)then
        token = tids(0)
        call pvmfinitsend(0,info)
        call pvmfpack(INTEGER4,token,count,stride,info)
        call pvmfsend(tids(me+1),msgtag,info)
        call pvmfrecv(tids(nproc-1),msgtag,info)
      else
        call pvmfrecv(tids(me-1),msgtag,info)
        call pvmfunpack(INTEGER4,token,count,stride,info)
        call pvmfinitsend(0,info)
        call pvmfpack(INTEGER4,token,count,stride,info)
        dest = tids(me+1)
        if(me.eq.nproc-1)dest = tids(0)
        call pvmfsend(dest,msgtag,info)
      endif
      return
      end
```

六、应用程序的编制与调试

编写应用程序可将 PVM 看作是一个通用的、灵活的并行计算资源，它支持消息传递计算模式。这个资源以三种不同访问方式：

(*)透明级模式 诸任务自动在最合适的机器上执行(一般是负载最轻的计算机)

(*)机型级模式 用户可以指定哪些特别的任务在哪种机型执行

(*)低级模式 规定执行任务的具体结点机

PVM 系统下的应用程序可有任意的控制与相关结构。换句话说，在执行一个并发程序的任何时刻，所存在的诸进程之间可有任意的关系。此外，任何进程可以与别的进程通信或同步。这是 MIMD 并行计算最通用的形式，而在实际中大多数并发应用程序是结构化的。SPMD 与 Master／Slave 是两种典型结构。前者所有进程是恒等的，后者一批 Slave 计算进程完成一个或多个 Master 进程所交付的工作。

1.一般的性能考虑

(1)任务精度。精度可用一个进程所接收的字节数与进程完成的浮点运算次数之比来度量。通过测算 PVM 配置中机器的计算速度以及连接机器的有效网络宽度，用户可以得他的应用程序任务粒度，一个粗略的下界估计。粒度愈大，加速比愈高，然而通常也降低有效的并行性。

(2)消息传送的字节数目。所接收的字节数目可以由许多小的消息合成，也可以是一些大的消息。采用大的消息可以减少总的消息起动时间，但不一定减少总的通信时间。这是因为小的消息通信可以与计算重迭，重迭的能力与发送消息的最佳数目与应用程序有关。

(3)适合于功能并行还是数据并行。功能并行是指不同的机型完成不同的任务。例如，向量机求解适合于向量化的部分问题，多处理机求解另一部分适合并行的问题，图形工作站用于实时生成数据的可视化。每个机器完成不同的功能(数据有可能相同)。

在数据并行模式中，数据分块且分布到 PVM 配置的所有机器中，对于每组数据完成相近的运算，不同进程之间交换信息，直至问题解决。数据并行对于分布存储多处理器是很普遍的，因为这可以伸缩到成百个处理器。许多线性代数、PDE 以及矩阵算法都是利用数据并行模式开发的。

当然，在 PVM 中可以结合这二种模式以发挥每个机器的能力。

2.网络因素

如果用户要在网络上运行应用程序，需要考虑网络对于程序开发的影响。一个用户的并行程序将与其它用户分享同一个网络。多用户、多任务的网络环境对于程序的通信和计算都产生复杂的影响。

首先要考虑到配置中每台机器不同计算能力的影响。这可能是由虚拟机中计算速度不同的异构机型而引起的。同一种工作站的不同型号机器的计算能力可以差别两个数量极。对于超级计算机，这种差别会更大。即使是用户指定的是同构机型，每个机器的有效性能也可有极大差别，这是因为所在配置机器的子集上运行着他

本人或者是其它用户的多任务。如果用户把他的问题分成若干相同的部分在每个机器上运行(这是并行化的一种通常方式),那么多任务的因素对他的程序执行有不利的影响。这时的应用程序运行速度将降到最慢机器上任务的执行速度。如果任务之间有联系,那么即使最快机器也要慢下来等待最慢任务发回的数据。

其次,要考虑到跨网络的长消息延滞的影响。这可能是使用远距离网络时机器间的距离长短而引起的,也可能是因为局部网上你自己的程序与其它用户程序争用引起的。倘若应用程序设计得每个任务只向相邻的任务发送消息,则可以假定不存在这种争用现象。在一台分布存储多处理器上假定没有这类争用现象,所有的传送是平行进行,然而在以太网上,如果这种传送陆续进行,将导致改变消息到达邻近任务的延滞,其它网络如token 环,FDDI 以及 HiPPI 都有引起改变延滞的功能。用户需要决定是否要在算法中指定延滞的误差。

第三,要考虑到其它用户共享资源时,计算性能与有效的网络带宽都要动态变化。一个应用程序运行一次得到了很好的加速比,几分钟后再次运行得到的加速比可能极差。在某一次运行中,应用程序能够有正常的同步机制,抛掉一些需要等待数据的任务。在最坏的情况下,应用程序运行时出现同步化错误,这只有当动态机器负载有波动时才会减慢运行速度,因为这种条件很难再生,这类错误很难找到。

3.负载平衡

在一个多用户网络环境下我们发现:负载平衡是提高性能最重要的措施。并行计算中有很多平衡负载的方法。在网络计算最通用的有以下三种:

(1)静态平衡。这是最简单的方法。只需把问题分割,将诸任务一次性分配到各处理器。作业开始前可以作数据分块,或者应用程序的头几步作数据分块。考虑到各种机器的不同计算能力,分配到一个稳定机器上的任务个数或任务规模可以有变化。因为所有这些任务一开始就是有效的。它们之间可以通信。对于一个负载不多的网络,静态负载平衡是很有效的。

(2)动态平衡。当计算负载出现变化时,动态负载平衡方式是必需的。最普遍的方式称为任务池的平台(Pool of Tasks paradiagm)。这种方式典型地在 Master / Slave 程序中实现。master 程序创建且掌握这个任务池,把诸任务承包给闻置的 Slave 程序去完成。任务池通常由排队实现。如果诸任务的规模改变,则将最大的任务排在队伍的前头。用这种方法,所有 Slave 处理器都保持忙碌,只要任务池内还有任务存在。因为这时每个任务均可在任何时刻启动或停止,这些方式对诸 Slave 程序之间不安排通信。Slave 只与 Master 及文件通信的应用程序较适用。

(3)负载定时再分配。预先确定时间表,届时重新检查与分配负载。例如在求解非线性 PDE 时,每个线性化步可作静态负载分配,在两个线性求解步之间诸任务检查问题已有哪些变化,重新分配网格点。这种基本的负载再分配方法还有几种不同变化。有的将过重的负载分配给相邻任务而不用同步化,有的是等待某个过程信号,表明它的负载平衡是否已超过给定的误差。超过就进行再分配,不是定时。

4.调试方法

一般来说,调试并行程序比调试串行程序要困难得多。不仅是因为有多个处理器同时运行,而且它们之间的交互也会引起出错。比如,某进程可能接收到一个错误数据,以至后面被除数为零。死锁是又一个例子。当某个编制出错引起所有处理器都在等待消息时就会出现死锁。所有的 PVM 标准程序如在执行中侦查出了错误,返回时给出相应的出错信息。

PVM 提供两个程序 pvm__perror 与 pvm__serror 以允许自动侦查并打印 PVM 其它程序调用时的出错信息。PVM 任务也可用在启动时用人工方式插入一些标准的串行调试语句,如 dbn。

派生 PVM 任务时可在调试情况下启动。在调用 pvmfspawn 时,令 flag=PVMDEBVG, PVM 将执行 shell script pvm3 / lib / debugger。启动主机的 x-窗口,这样 PVM 启动后,派生的任务就在所在的 x-window 调试状态下。处于调试状态下的这个任务可以在虚拟机任一个结点机上执行,只要在 pvnifspawn 中指定相应的 flag 与 where。这时所得的诊断打印语句不出现在用户屏幕上,而写在启动 PVM 结点机上形成 / tmp / pvm1.<u 文件。

经验表明,调试 PVM 程序可分为三个步骤:

(1)可能的话,先在单任务上运行程序,加入串行的调试状态。目的在于发现与并行化无关的(语法)式逻辑

错误，一旦错误排除可进入第二步。

(2)在单个机器上用 2～4 个进程运行程序，把单一的工作站名字列入 hostfile 文件中。PVM 将在此台工作站上多任务地产生进程，目的在于检查通信的语法与逻辑。例如，发送时消息用 tag 5，接收者却等待 tag 为 4 的消息。采用不唯一的消息 tag 是一个常犯的错误。假定总是采用相同的消息标记 tag，一个进程接收到某些初始数据来自三个分别的消息，但无法确定哪个消息中含有哪些数据。PVM 返回的信息涉及到所要的来源与 tag，使用户确信能唯一核对消息的内容。不唯一的 tag 出错通常是很难调试出来的，因为它对精细的同步化效应极敏感，而且不能通过反复运行再生。如果这个错误不能通过 PVM 出错指令或者从快速打印语句中确定，那么用户可采用完整的调试控制手段，使之某个或所有任务都在调试状态下启动，可加入转折点、变量追朔、单步化以及对每个进程追踪，甚至追踪到该进程与其它 PVM 任务之间的消息传递，不论后者是否在 dbx 状态下运行。

(3)用几台机器运行与第 2 步相同的 2～4 个进程，目的是检查因网络延滞产生的同步错误。这类错误常常在这个阶段被发现，它们对于消息到达的次序算法极敏感，而由逻辑出错引起的程序死锁，对网络的延滞也很敏感。在这个阶段可再次采用完整的调试控制，但可能不是很有用，因为此时调试的位置可能已经被移动或者被遮盖了。

参考文献：

[1]O.A.McBryan, An Overview of Message Passing Environments, "Parallel Computing", 20(1994), April, 417–444.

[2]J.Dongarra, etc. The PVM Concurrent Computing system, 同*[1],531–545.*

[3]a1 Geist,etc. PVM 3 User's Guide and Reffe renco Manua1,ORNL / TM–12187,Moy,1994.