INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

H4.SMR/854-13

College on Computational Physics

15 May - 9 June 1995

Overview and Tutorial on PVM, Xab, and HeNCE

F. Massaioli

Università "la Sapienza"
Rome, Italy

Overview and Tutorial on PVM, Xab, and HeNCE

Al Geist
Mathematical Sciences Section
Oak Ridge National Laboratory

Los Alamos Summer Computational Workshop

June 22, 1992
LANL NM

Major Features of PVM

- **Easy to install** – can be done by any user.
- **Easy to configure** your own personal parallel computer
  Just list machines in a host file.
- Your configuration can **overlap** with other user's PVM's.
- **Easy to write programs** for PVM
  Uses standard message–passing interface.
- C and **Fortran** applications are supported.
- **Multiple applications** can run simultaneously on one PVM.
- Source of entire package is **under 400 Kbytes.**

Heterogeneity

- PVM supports heterogeneity at three levels

- **Application**

  Subtasks can exploit the architecture best suited to their solution.

- **Machine**

  Computers with different data formats are supported as well as different serial and parallel architectures.

- **Network**

  Different network types can make up a Parallel Virtual Machine. For example, FDDI, Token Ring, Ethernet

Installing PVM3 on a Machine

Follow directions in the header of pvm3_shar to cat and unpack shar file.

Shar file will unpack into a directory called pvm3

Copy the file **pvm3/Cshrc.stub** into your .cshrc file after your path is defined.

This stub automatically sets the correct value of **$ARCH**

And sets the correct path to **pvm** and **pvmd3** for this machine.

If PVM has already been installed on the machine
Then put the path to pvm and pvmd3 in bottom of stub

Compiling PVM3

For each architecture to be used with PVM

```
% cd pvm3
% make
```

Creates:

pvmd3
pvm
libpvm3.a
libfpvm3.a

And installs them in:

pvm3/lib/ARCH/

Copy these four files to **pvm3/lib/ARCH/** on other machines of this same architecture.

Starting PVM3

Three methods of starting PVM are

console **directly** **background**

console % pvm
pvm>

help	**version**	**id**
conf	**add**	**delete**
ps	**spawn**	**kill**
pstat	**mstat**	**sig**
reset	**quit**	**halt**

directly %pvmd3 hostfile

background %pvmd3 hostfile &

Running Applications

Compiling C:

% cc –o task file.c libpvm3.a

add libgpvm3.a if groups used

Compiling Fortran:

% f77 –o task file.f libfpvm3.a libpvm3.a

Default location PVM looks for user executables is:

pvm3/bin/ARCH/

PVM tasks can be started from any UNIX prompt on any computer in the virtual machine or console.

PVM 3 Task ID

All PVM3 tasks are identified by an integer task id

- **Including the pvm console and pvmds**

  uniform identification and communication system

- **Dynamic Process Groups – possible overlapping**

  requires a unique task identifier separate from group

- **Multiprocessor integration**

  encoded in the task id is the host address, the CPU number (on multiprocessors) , and the process ID

- **Scalability**

  The task ID is supplied by the local pvmd and requires no global communication to generate a unique identifier

PVM Error Handling

In C routines error status is returned by the function
In Fortran error status is returned by subroutine argument INFO

Return value >= 0 means OK

Return value < 0 means error

Stderr and stdout from all spawned tasks is redirected to /tmp/pvml.nnnnn on the host where PVM was started.

spmd.c
spmd.f

93/04/13
12:00:19

```
C-----------------------------
c  SPMD Fortran example using PVM 3.0
C-----------------------------
      program spmd
      include '../include/fpvm3.h'
      PARAMETER( NPROC=4 )

      integer mytid, me, i
      integer tids(0:NPROC)

c     ---------
c     Enroll in pvm
c     ---------
      call pvmfmytid( mytid )

c     ---------
```

```
c     Find out if I am parent or child - spawned processes have pare
nts
c     ------------------------------------------------------------
      call pvmfparent(tids(0))
      if( tids(0) .lt. 0 )  then
        tids(0) = mytid
        me = 0
c       ----------------------
c       start up copies of myself
c       ----------------------
        call pvmfspawn('spmd',PVMDEFAULT,'*',NPROC-1,tids(
1),info)
c       -------------------------
c       multicast tids array to children
c       -------------------------
        call pvmfinitsend( PVMDEFAULT, info )
```



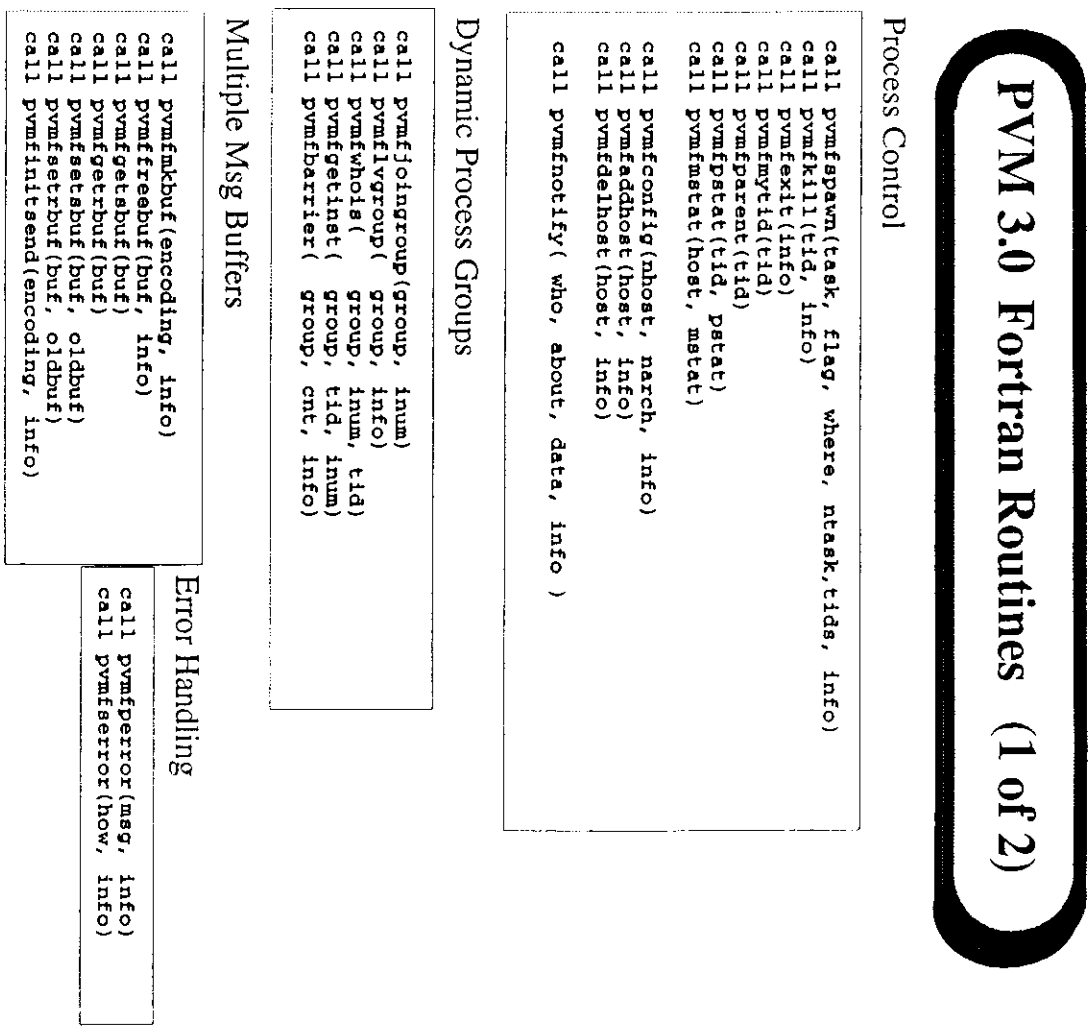
```
        call pvmfpack( INTEGER4, tids, NPROC, 1, info )
        call pvmfmcast( NPROC-1, tids(1), 0, info )
      else
c       -------------------------------
c       receive the tids array and set me
c       -------------------------------
        call pvmfrecv( tids(0), 0, info )
        call pvmfunpack( 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-----------------------------------------------------
      print*,'me =',me, '  mytid =',mytid
      call dowork( me, tids, NPROC )

c     ------------------------
c     program finished exit pvm
c     ------------------------
      call pvmfexit(info)
      stop
      end

      subroutine dowork( me, tids, nproc )
      include '../include/fpvm3.h'
c-------------------------------------------
c Simple subroutine to pass a token around a ring
c-------------------------------------------
```



```
      integer me, nproc
      integer tids( 0:nproc)

      integer token, dest, count, stride, msgtag

      count  = 1
      stride = 1
      msgtag = 4

      if( me .eq. 0 ) then
        token = tids(0)
        call pvmfinitsend( PVMDEFAULT, info )
        call pvmfpack( INTEGER4, token, count, stride, info )
        call pvmfsend( tids(me+1), msgtag, info )
        call pvmfrecv( tids(nproc-1), msgtag, info )
        print*, 'token ring done'
```

```
else
   call pvmfrecv( tids(me-1), msgtag, info )
   call pvmfunpack( INTEGER4, token, count, stride, info )
   call pvmfinitsend( PVMDEFAULT, 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
```



PVM 3.0 Fortran Routines (2 of 2)

Mesasage Passing

```
call pvmfpack( what, xp, nitem, stride, info )

call pvmfsend( tid, msgtag, info)
call pvmfmcast( ntask, tids, msgtag, info )

call pvmfnrecv( tid, msgtag, buf)
call pvmfrecv(  tid, msgtag, buf)
call pvmfbufinfo(buf, bytes, msgtag, tid, info)

call pvmfunpack( what, xp, nitem, stride, info )
```

what options

STRING	REAL4
BYTE1	COMPLEX8
INTEGER2	REAL8
INTEGER4	COMPLEX16

spmd.c

spmd.c

93/06/18 13:36:55

```
int i;

/* enroll in pvm */
mytid = pvm_mytid();

/* find out if I am parent or child */
tids[0] = pvm_parent();
if( tids[0] < 0 )       /* then I am the parent */
  {
    tids[0] = mytid;
    me = 0;
    /* start up copies of myself */
    pvm_spawn("spmd", (char**)0, 0, "", NPROC-1, &ti
ds[1]);
```

```
    /* multicast tids array to children */
    pvm_initsend( PvmDataDefault );
    pvm_pkint(tids, NPROC, 1);
    pvm_mcast(&tids[1], NPROC-1, 0);
  }
  else   /* I am a child */
  {
    /* receive tids array */
    pvm_recv(tids[0], 0);
    pvm_upkint(tids, NPROC, 1);
    for( i=1; i<NPROC ; i++ )
      if( mytid == tids[i] ){ me = i; break; }
  }
```



```
/*-------------------------------------------------------
 * all NPROC tasks are equal now
 * and can address each other by tids[0] thru tids[NPROC-
1]
 * for each process me => process number [0-(NPROC-1)]
 *-------------------------------------------------------
 */
    printf("me = %d mytid = %d\n",me,mytid);
    dowork( me, tids, NPROC );

    /* program finished exit pvm */
    pvm_exit();
    exit(1);
}
```

```
/* Simple example passes a token around a ring */

dowork( me, tids, nproc )
    int me;
    int *tids;
    int nproc;
{
    int token;
    int dest;
    int count  = 1;
    int stride = 1;
    int msgtag = 4;
```



```
    if( me == 0 )
    {
      token = tids[0];
      pvm_initsend( PvmDataDefault );
      pvm_pkint( &token, count, stride );
      pvm_send( tids[me+1], msgtag );
      pvm_recv( tids[nproc-1], msgtag );
      printf("token ring done\n");
    }
    else
    {
      pvm_recv( tids[me-1], msgtag );
      pvm_upkint( &token, count, stride );
      pvm_initsend( PvmDataDefault );
```

```
    pvm_pkint( &token, count, stride );
    dest = (me == nproc-1)? tids[0] : tids[me+1] ;
    pvm_send( dest, msgtag );
  }
}
```

spmd.c

PVM 3.x C Routines (1 of 2)

Process Control

```
int   cc = pvm_spawn(char *a.out, char **argv, int flag,
                     char *where, int cnt,  int *tids)
int   cc = pvm_kill( int tid)
void     pvm_exit()
int  tid = pvm_mytid()
int  tid = pvm_parent()
int stat = pvm_pstat(int tid)
int stat = pvm_mstat(char *host)

int   cc = pvm_config(int *host, int *arch,
                      struct hostinfo **hostp)
int   cc = pvm_addhosts(char **hosts)
int   cc = pvm_delhosts(char **hosts)
```

Dynamic Process Groups

```
int cc   = pvm_bcast(     char *group, int msgtag)
int inum = pvm_joingroup( char *group)
int cc   = pvm_lvgroup(   char *group)
int tid  = pvm_gettid(    char *group, int inum)
int inum = pvm_getinst(   char *group, int tid)
int cc   = pvm_barrier(   char *group, int cnt)
int size = pvm_gsize(     char *group)
```

Multiple Msg Buffers

```
int buf    = pvm_mkbuf(int encoding)
int cc     = pvm_freebuf(int buf)
int buf    = pvm_getsbuf()
int buf    = pvm_getrbuf()
int oldbuf = pvm_setsbuf(int buf)
int oldbuf = pvm_setrbuf(int buf)
int buf    = pvm_initsend(int encoding)
```

Error Handling

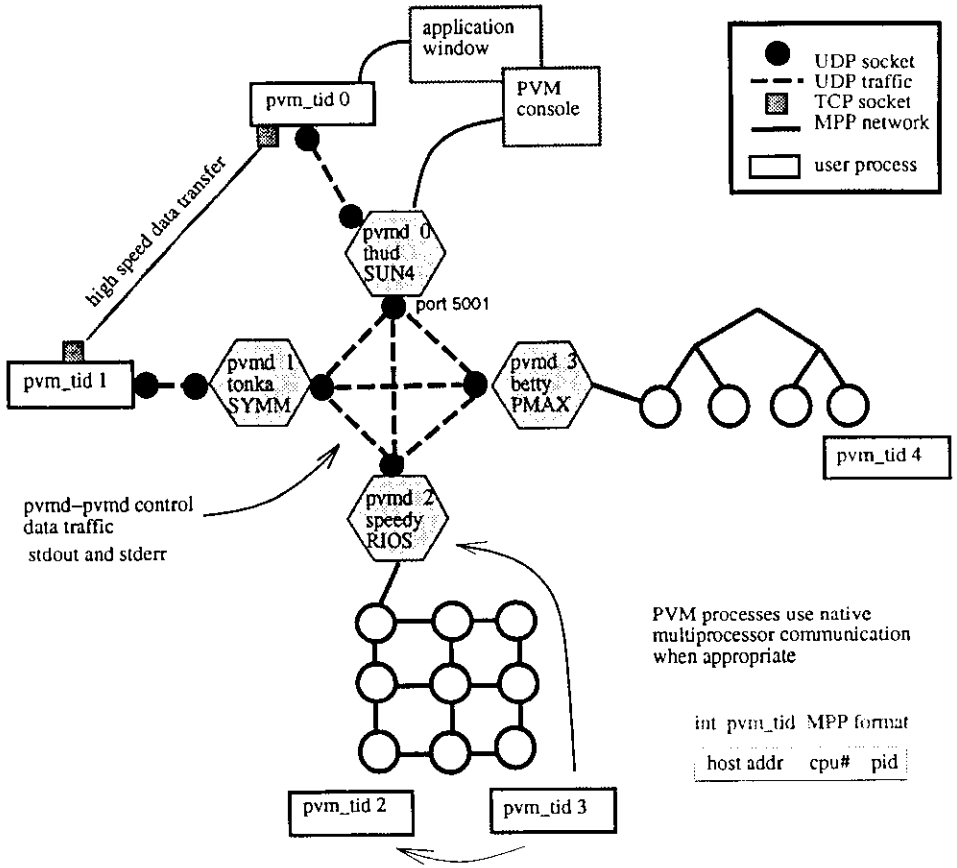
```
int info = pvm_perror(char *msg)
int info = pvm_serror( int how)
```

PVM 3.x C Routines (2 of 2)

Mesasage Passing

```
int cc = pvm_advise( route )

int cc = pvm_pkbyte(  char   *cp, int cnt, int std)
int cc = pvm_pkcplx(  float  *xp, int cnt, int std)
int cc = pvm_pkdcplx( double *dp, int cnt, int std)
int cc = pvm_pkdouble(double *dp, int cnt, int std)
int cc = pvm_pkfloat( float  *fp, int cnt, int std)
int cc = pvm_pkint(   int    *np, int cnt, int std)
int cc = pvm_pklong(  long   *np, int cnt, int std)
int cc = pvm_pkshort( short  *np, int cnt, int std)
int cc = pvm_pkstr(   char   *cp)

int cc = pvm_send(  int tid, int msgtag)
int cc = pvm_mcast( int tids, int ntask, int msgtag)

int buf = pvm_probe(int tid, int msgtag)
int buf = pvm_nrecv(int tid, int msgtag)
int buf = pvm_recv( int tid, int msgtag)
int cc  = pvm_bufinfo(int buf, int *len,
                      int *msgtag, int *tid)

int cc = pvm_upkbyte(  char   *cp, int cnt, int std)
int cc = pvm_upkcplx(  float  *xp, int cnt, int std)
int cc = pvm_upkdcplx( double *dp, int cnt, int std)
int cc = pvm_upkdouble(double *dp, int cnt, int std)
int cc = pvm_upkfloat( float  *fp, int cnt, int std)
int cc = pvm_upkint(   int    *np, int cnt, int std)
int cc = pvm_upklong(  long   *np, int cnt, int std)
int cc = pvm_upkshort( short  *np, int cnt, int std)
int cc = pvm_upkstr(   char   *cp)
```

PVM 3.0 Internal Details



Popular Distributed Programming Schemes

Master / Slave

Master task starts all slave tasks and coordinates their work and I/O.

SPMD (hostless)

Single Program Multiple Data model
Same program executes on different pieces of the problem.

Functional

Several programs are written each performs a different function in the application.

Load Balance Methods

Static Load Balancing

Problem is divided up and tasks are assigned to processors only once. The number or size of tasks may be varied to account for different computational powers of machines.

Dynamic Load Balancing by Pool of Tasks

Typically used with master/slave scheme, the master keeps a queue of tasks and continues to give them to idle slaves until the queue is empty. Faster machines end up getting more tasks naturally (used by xep example in pvm3).

Dynamic Load Balancing by Coordination

Typically used in SPMD scheme, all the tasks stop and redistribute their work either at fixed times (f.e. every 10 iterations) or if some condition occurs (f.e. the load imbalance between tasks exceeds some limit).

PVM Workshop

Adam Beguelin
School of Computer Science
Pittsburgh Supercomputing Center
Carnegie Mellon University

Parallel computing concepts

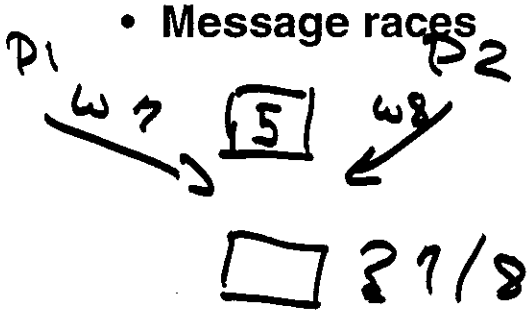
- Kinds of parallelism
- Nondeterminacy
- SPMD programming
- Load balancing
- Granularity
- Speedup and efficiency

Kinds of parallelism

- **Data parallelism**
  Same operation on different data
  – Incrementing all values of a vector in parallel
- **Functional parallelism**
  Different operations in parallel
  – Molecular dynamics and graphical rendering
- **Vector parallelism**
- **Pipelining**

Nondeterminacy

- **Parallel access to a shared resource**
  – Parallel writes
  – Parallel reads and writes
- **Message races**



SPMD programming

Single program, multiple data

- Running the same program on multiple processors at the same time.

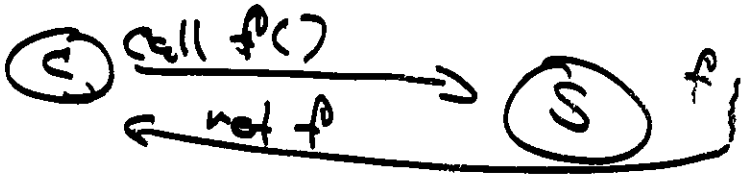
Programming Distributed Memory

Remote procedure call

- Procedure calls carried out remotely by server
- No explicit parallelism
- Familiar interface

Message passing

- Explicitly passing data via messages
- Explicit partitioning of data



Load balancing

Effort to keep all processors busy

- Processor speeds may vary
- Type of work may differ
- Amount of work may differ

Kinds of load balancing

Static load balancing

- A priori division of work
- Little overhead

Dynamic load balancing

- Adapts to current conditions
- Performance prediction not needed
- More runtime overhead

Load Balancing

- Trying to keep all processors busy
  - Different (possibly unknown) amounts of work
  - Different processor speeds
  - External load imposed on processors
- Static load balancing
  - Work divided before starting
  - Based on work, CPU speed
  - Need to predict processing time for work
  - Works well for some algorithms, machines
- Dynamic load balancing
  - Adapts to changing load
  - Adapts to varying amounts of work
  - More runtime overhead
  - Simple methods
    - Job queue manager
  - More complex methods
    - Preemptive - workers are interrupted, rescheduled
    - Process migration

Granularity

Number of operations between communication

- **Appropriate granularity depends on**
  - **Computation vs Communication rates**

Coarse Granularity

- **Networks of workstations**

Fine Granularity

- **Shared memory machines**
- **Vector supercomputers**

Speedup

Sequential vs parallel execution times

- Speedup = $T_{sequential} / T_{parallel}$
- Use tuned sequential version

Superlinear speedup

- $T_{sequential} / T_{parallel}$ > # processors
- Due to:
  - Cache
  - Memory
  - Heterogeniety

Efficiency

Speedup / # processors

- Measures effective use of processors
- Reveals scalability of the application

PVM: What is it?

Parallel Virtual Machine

- System for Heterogeneous Parallel Programming
- Process control
- Communication

Design Goals

Applications

- Straightforward, well understood paradigms
- Graphical development tools
- Debugging and profiling facilities

System

- Support for multiple architectures
  - Multiprocessors and networks
- Efficient distributed algorithms, protocols, failure resilience
- Ease of installation, operating and admin interface

Programming Model

- Application consists of tasks
- Task is the unit of concurrency
  - Named by task id or group
- Tasks communicate via messages
  - Send, receive, and multicast
- Tasks can also synchronize via barriers
- Rich collection of status and housekeeping calls

Major components of PVM

The PVM Library

- FORTRAN
- C/C++

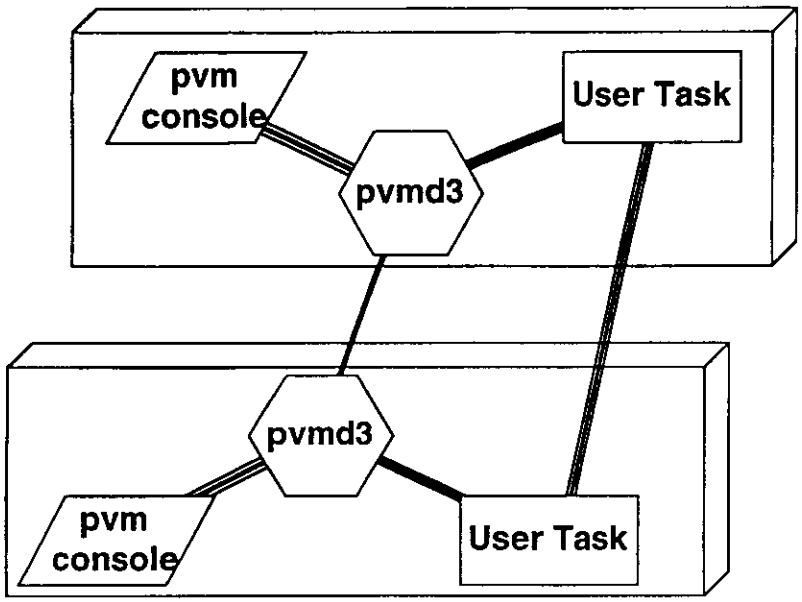
The daemon

- A Unix process
- pvmd3 user process on each host

The console

- PVM task
- User interface to the system

PVM Task Architecture



Starting PVM

- **Use the console**

```
% pvm
pvm> conf
1 host, 1 data format
                    HOST     DTID     ARCH   SPEED
   DAO.NECTAR.CS.CMU.EDU    40000 sun4_mach    1000
```

- **Add machines**

```
pvm> add duck concord
2 successful
                    HOST     DTID
                    duck    c0000
                 concord   100000
```

% rsh duck date
Oct 9 ...

Starting PVM with a Hostfile

Start the pvmd3 directly

```
% pvmd3 hf &
```

Host file contains:

- Host names and options

  * dx=/usr/local/bin/pvmd

  dao lo=adamb ep=bin

  scfea lo=beguelin wd=input/data

PVM Process Control

- Spawn processes anywhere
  - Processes named by task id
- Kill process started in PVM
- Status of processes

Process Control

- The task id
- Spawning
- Killing
- Notify

The Task ID

- Unique integer id for a task
- Obtained by pvm_mytid() pvmfmytid
- Concatenation of host and task numbers
  - Slightly different on MPP machines

cc = pvm_mytid()

pvmfmytid(&cc);

The pvm_spawn call

rc = pvm_spawn(char* task, char **argv, int flag, char* where, int ntask, int* tids)

- task is executable name
- flag is PvmTaskDefault, PvmTaskHost, PvmTaskArch, PvmTaskDebug, and PvmTaskTrace
- where is either host or architecture
- ntask is number of tasks to start
- tids is an array of the created task ids or an error code
- rc is the number of tasks successfully started

PVM Message passing

- Send and receive messages
  - Message order is preserved between pairs of tasks
- Multicast to a list of tids
- Blocking and non-blocking receive
- Heterogeneous data formats supported
  - Data types are specified when building a message

Sending a message

bufid = pvm_initsend(int encoding)

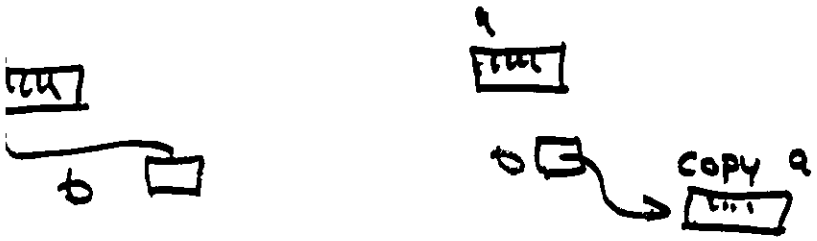
- Creates a new buffer, bufid

pvm_pkint(int* np, int cnt, int std)

- Packs cnt integers into the message buffer

rc = pvm_send(int tid, int msgtag)

- Sends the current message to task tid



Receiving a message

bufid = pvm_recv(int tid, int msgtag)

- Blocks until a message from task tid arrives with tag msgtag
- Message is placed in newly created buffer bufid

pvm_upkint(int* np, int cnt, int std)

- Unpacks integers from the message into np

An Example

Hello World

- Program hello1.c, the main program:

```
#include <stdio.h>
#include "pvm3.h"

main()
{
    int tid;        /* tid of child */
    char buf[100];

    printf("I'm t%x\n", pvm_mytid());

    pvm_spawn("hello2", (char**)0, 0, "", 1, &tid);
    cc = pvm_recv(-1, -1);
    pvm_bufinfo(cc, (int*)0, (int*)0, &tid);
    pvm_upkstr(buf);
    printf("Message from t%x: %s\n", tid, buf);
    pvm_exit();
    exit(0);
}
```

- Program hello2.c, the slave program:

```
#include "pvm3.h"

main()
{
    int ptid;       /* tid of parent */
    char buf[100];

    ptid = pvm_parent();
    strcpy(buf, "hello, world from ");
    gethostname(buf + strlen(buf), 64);
    pvm_initsend(PvmDataDefault);
    pvm_pkstr(buf);
    pvm_send(ptid, 1);
    pvm_exit();
    exit(0);
}
```

Buffer Encodings

PvmDataDefault

- Converts the data to XDR format

PvmDataRaw

- Performs no data conversion

PvmDataInPlace

- Does not copy data until send

Multicast Messages

pvm_mcast(int *tids, int ntask, int msgtag)

- Sends the current msg to all tids
- Can be received with pvm_recv or pvm_nrecv
- Order not preserved with normal sends

Buffer Manipulation

bufid = pvm_initsend(int encoding)

- Resets the current send buffer

pvm_bufinfo(int bufid, int *bytes, int *mtag, int * tid)

- Gets information about the message buffer

bufid = pvm_mkbuf(int encoding)

Creates a new buffer

- info = pvm_freebuf(int bufid)
- Frees the buffer

Message Routing

pvm_setopt(PvmRoute, PvmDontRoute | PvmAllowDirect | PvmRouteDirect)

- PvmDontRoute
  - Don't request or grant direct connections
- PvmAllowDirect (Default)
  - Don't request but allow direct connections
- PvmRouteDirect
  - Request and allow direct connections

Exercise

Write a dot product program

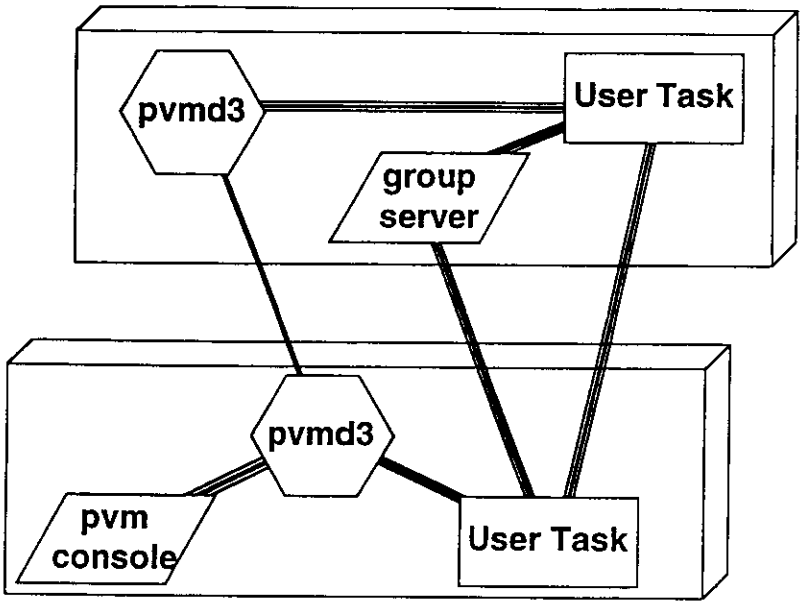
- Use the SPMD style
- Generate the vector values in parallel
- Multicast the solution to all tasks
- Calculate speedup for the tasks

Process groups

Groups provide a global name space

- Join and leave groups
- Broadcast to a group
- Barrier synchronization on a group
- Integer group id

Group Task Architecture



File I/O

Where does stdout/stderr go?

- /tmp/pvml.<uid> on master
- pvm_catchout()
- pvm_setopt(PvmOutputTid, tid)

The working directory of spawned tasks

- wd=working-dir
- The expected directory structure
  - Location of executables

Preventive Maintenance

Using strongly typed messages

- Specific message tags

Avoid wildcards when possible

- pvm_recv(pvm_parent, InputTag)

  NOT

- pvm_recv(-1, -1)

Tracing

- Tracing from the console
- Spawning with the trace flag
- Setting/getting the trace mask

Tracing from the console

PVM calls can be traced from the console

Set the calls to be traced

```
pvm> trace + spawn
```

Spawn the program with trace flag

```
pvm> spawn -@ xep
```

Events will be displayed on the console tty

XPVM

- X based PVM console
- Interface for machine configuration
- Shows output of tracing
- Interface for controlling tasks

Installing PVM

- FTP and untar the source
- Set the environment var PVM_ROOT
  setenv PVM_ROOT ~/pvm3
  (Put this in your .cshrc file)
- Change directories to PVM_ROOT
- Type "make"

Failure Notification

- pvm_notify(int what, msgtag, int cnt, int *tids)

Send message when

- Task exits
- Hosts are added
- Hosts are deleted

Failed Messages

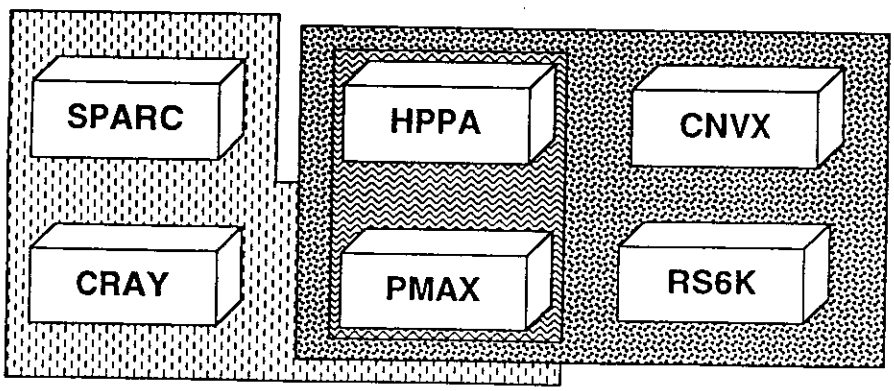
- pvm_trecv(int tid, tag, struct timeval *tmout)
  - tmout->tv_sec and tmout->tv_usec
- pvmftrecv(tid, msgtag, sec, usec, bufid)

PVM receive call with a time out

Returns negative if no message was recvd before timeout

Overlapping Virtual Machines

- Virtual machines are on a per user basis
- Multiple virtual machines per actual machine



PVM availability

- Send mail to netlib@ornl.gov
  Subject "send index from pvm3"
- FTP netlib2.cs.utk.edu
  Directory pvm3
  – Source, User's guide, etc.
  XPVM Directory pvm3/xpvm
- Bug reports or questions
  comp.parallel.pvm
  pvm@msr.epm.ornl.gov