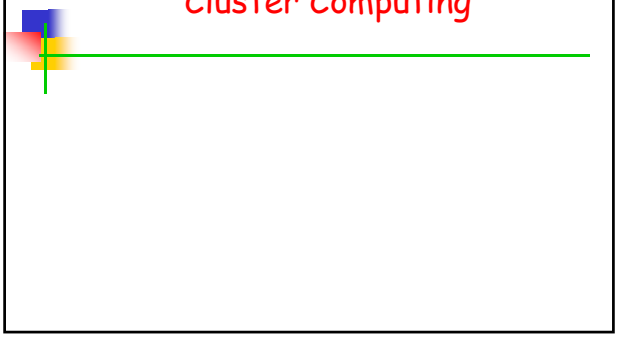


PVM = Parallel Virtual Machine

- Software package
  - Standard daemon: pvmd
  - Application program interface
- Network of heterogeneous machines
  - Workstations
  - Supercomputers
  - Unix, NT, VMS, OS/2

The *pvmd3* process

- Manages each host of vm
  - Message router
  - Create, destroy, ... local processes
  - Fault detection
  - Authentication
  - Collects output
- Inter-host point of contact
  - One pvmd3 on each host
- Can be started during boot

The program named *pvm*

- Interactive control console
  - Configuration of PVM
  - Status checking
- Can start pvmd

The library *libpvm*

- Linked with application programs
- Functions
  - Compose a message
  - Send
  - Receive
- System calls to pvmd
- libpvm3.a, libfpvm3.a, libgpvm3.a

libpvm3.a

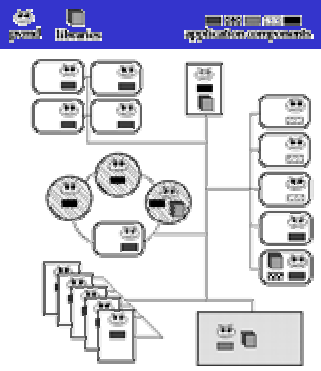
- Initiate and terminate processes
- Pack, send, and receive messages
- Synchronize via barriers
- Query and change configuration of the pvm
- Talk to local pvmd3
- Data format conversion (XDR)

libfpvm3.a

- Additionally required for Fortran codes

libgpvm3.a

- Dynamic groups of processes



First PVM Example: Hello World!

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

main() {
  int cc, tid;
  char buf[100];

  printf("i'm t%x\n", pvm_mytid());
  cc = pvm_spawn("hello_other",
                 0, 0, "", 1, &tid);

  if (cc == 1) {
     cc = pvm_recv(-1, -1);
     pvm_bufinfo(cc, 0, 0, &tid);
     pvm_upkstr(buf);
     printf("from t%x: %s\n",
            tid, buf);
  } else
     printf(
        "can't start hello_other\n");

  pvm_exit();
  exit(0);
}
```

```
/* hello_other */
#include "pvm3.h"

main() {
   int ptid;
   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);
}
```


Another Example

- Master-Worker (Master-Slave) Paradigm

Master (in Master-Worker)

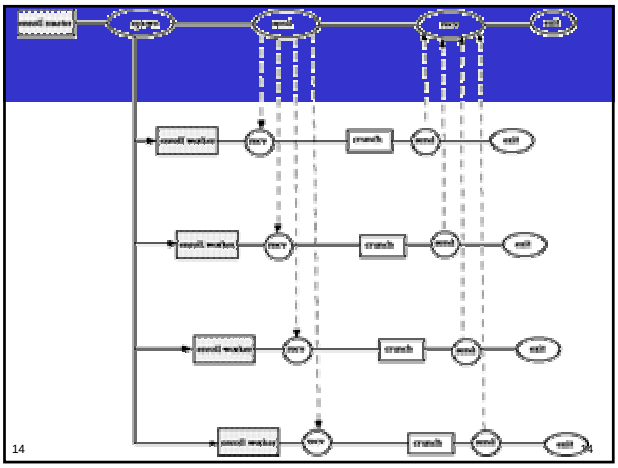
```
#include "pvm3.h"
#include <stdio.h>
#define SIZE 1000
#define NPROCS 5
main(){
  int mytid, task_ids[NPROCS];
  int a[SIZE], results[NPROCS];
  int sum = 0, i, msgtype;
  int num_data = SIZE/NPROCS;

  /* enroll in PVM */
  mytid = pvm_mytid();
  for (i = 0; i < SIZE; i++)
    a[i] = i % 25;

  /* spawn worker tasks */
  pvm_spawn("worker", (char **)0,
    PvmTaskDefault, "", NPROCS,
    task_ids);
```

```
  /* send data to worker tasks */
  for (i = 0; i < NPROCS; i++) {
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&num_data, 1, 1);
    pvm_pkint(&a[num_data*i],
          num_data, 1);
    pvm_send(task_ids[i], 4);
  }

  /* wait and gather results */
  msgtype = 7;
  for (i = 0; i < NPROCS; i++) {
    pvm_recv(task_ids[i],
          msgtype);
    pvm_upkint(&results[i], 1,1);
    sum += results[i];
  }
  printf("The sum is %d \n",sum);
  pvm_exit();
}
```

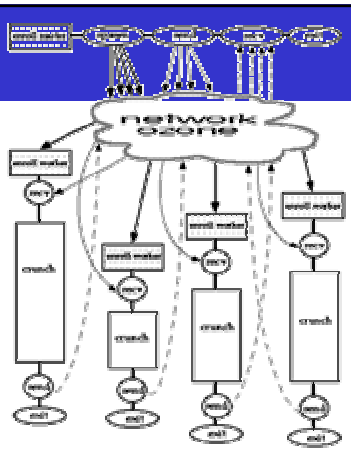
Worker (in Master-Worker)

```
#include "pvm3.h"
#include <stdio.h>
main(){
  int mytid;
  int i, sum, *a;
  int num_data, master;

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

  /* receive to be summed */
  pvm_recv(-1, -1);
  pvm_upkint(&num_data, 1, 1);
  a = (int *) malloc(num_data
        *sizeof(int));
  pvm_upkint(a, num_data, 1);
  sum = 0;
  for(i = 0; i < num_data; i++)
    sum += a[i];

  /* send sum back to master */
  master = pvm_parent();
  pvm_initsend(PvmDataRaw);
  pvm_pkint(&sum, 1, 1);
  pvm_send(master, 7);
  pvm_exit();
}
```

13 13



14



15 15

pvm_mytid

- Enrolls the calling process into PVM and generates a unique task identifier if this process is not already enrolled in PVM. If the calling process is already enrolled in PVM, this routine simply returns the process's tid.
- tid = pvm_mytid ();

16 16

pvm_spawn

- Starts new PVM processes. The programmer can specify the machine architecture and machine name where processes are to be spawned.
- numt = pvm_spawn ("worker",0,PvmTaskDefault,"",1,&tids[i]);
- numt = pvm_spawn ("worker",0,PvmTaskArch,"RS6K",1,&tid[i]);

17 17

pvm_exit

- Tells the local pvmd that this process is leaving PVM. This routine should be called by all PVM processes before they exit.

18 18

pvm_addhosts

- Add hosts to the virtual machine. The names should have the same syntax as lines of a pvmd hostfile.
- pvm_addhosts (hostarray,4,infoarray);

pvm_delhost

- Deletes hosts from the virtual machine.
- pvm_delhosts (hostarray,4);

pvm_pkdatatype

- Pack the specified data type into the active send buffer. Should match a corresponding unpack routine in the receive process. Structure data types must be packed by their individual data elements.

pvm_upk<datatype>

- Unpack the specified data type into the active receive buffer. Should match a corresponding pack routine in the sending process. Structure data types must be unpacked by their individual data elements.

pvm_send

- Immediately sends the data in the message buffer to the specified destination task. This is a blocking, send operation. Returns 0 if successful, < 0 otherwise.
- pvm_send (tids[1],MSGTAG);

pvm_psend

- Both packs and sends message with a single call. Syntax requires specification of a valid data type.

pvm_mcast

- Multicasts a message stored in the active send buffer to tasks specified in the tids[]. The message is not sent to the caller even if listed in the array of tids.
- pvm_mcast (tids,ntask,msgtag);

25 25

pvm_recv

- Blocks the receiving process until a message with the specified tag has arrived from the specified tid. The message is then placed in a new active receive buffer, which also clears the current receive buffer.
- pvm_recv (tid,msgtag);

26 26

pvm_nrecv

- Same as pvm_recv, except a non-blocking receive operation is performed. If the specified message has arrived, this routine returns the buffer id of the new receive buffer. If the message has not arrived, it returns 0. If an error occurs, then an integer < 0 is returned.
- pvm_nrecv (tid,msgtag);

27 27

PVM Collective Communication

28 28

pvm_barrier

- Blocks the calling process until all processes in a group have called pvm_barrier().
- pvm_barrier ("worker",5);

29 29

pvm_bcast

- Asynchronously broadcasts the data in the active send buffer to a group of processes. The broadcast message is not sent back to the sender.
- pvm_bcast ("worker",msgtag);

30 30

pvm_gather

- A specified member receives messages from each member of the group and gathers these messages into a single array. All group members must call pvm_gather().
- pvm_gather(&getmatrix,&myrow,10,PVM_INT, msgtag,"workers",root);

31 31

pvm_scatter

- Performs a scatter of data from the specified root to each of the members of the group, including itself. All group members must call pvm_scatter(). Each receives a portion of the data array from the root in their local result array.
- pvm_scatter (&getmyrow,&matrix,10,PVM_INT, msgtag,"workers",root);

32 32

pvm_reduce

- Performs a reduce operation over members of the group. All group members call it with their local data, and the result of the reduction operation appears on the root. Users can define their own reduction functions or the predefined PVM reductions
- pvm_reduce(PvmMax,&myvals,10,PVM_INT,msgtag ,"workers",root);

33 33

Prepare to Execute a PVM session

PVM expects executables to be located in ~/pvm3/bin/$PVM_ARCH
% ln -s $PVM_ROOT/lib ~/pvm3/lib

% cc -o myprog myprog.c -I$PVM_ROOT/include
-L$PVM_ROOT/lib/$PVM_ARCH -lpvm3

34 34

Create your PVM hostfile

- PVM hostfile defines your parallel virtual machine. It contains the names of all desired machines, one per line.

35 35

Create Your $HOME/.rhosts file

- Example .rhosts file

mamma.cs.wright.edu user02
fr2s02.mhpcc.edu user02
beech.tc.cornell.edu jdoe
machine.mit.edu user02

36 36

Start pvmd3

- % pvmd3 hostfile &
- This starts up daemons on all other machines (remote) specified in your hostfile.
- PVM console can be started after pvmd3 by typing "pvm".

37 37

Execute your application

% myprog

38 38

Quitting PVM

- Application components must include call of pvm_exit().
- Halting the master pvmd3 will automatically kill all other pvmd3s and all processes enrolled in this PVM.
- In pvm console: "halt"
- Running in the background: enter console mode by typing "pvm" and halt.

39 39