NIST PowerParallel Primer

Karin A. Remington

Mathematical and Computational Sciences Division, NIST ITL

August 28, 1998

Note: This document was generated automatically from the WWW pages located at URL address http://math.nist.gov/mcsd/Staff/KRemington/Primer/primer.html.

Disclaimer

The mention of specific products, trademarks, or brand names in this Primer is for purposes of identification only. Such mention is not to be interpreted in any way as an endorsement or certification of such products or brands by the National Institute of Standards and Technology. All trademarks mentioned herein belong to their respective owners.

Contents

1 Overview of the NIST SP2

The IBM Scalable POWERparallel System 2 (SP2) is a distributed memory multiprocessor consisting of a collection of RS6000 workstation nodes (running the AIX operating system) together with a Scalable POWERparallel Switch providing communication between the nodes.

1.1 SP2 hardware information

- 30 "SP2" Nodes, with IP addresses grand[1-16].nist.gov and grand[18-31].nist.gov

  Each of these nodes is the rough equivalent of an RS/6000 Model 390.

  - 66.7 MHz clock
  - 64 KB cache line
  - 512 MB of local memory

  NOTE: it has been observed that two of these nodes (grand24 and grand25) have a slightly faster clock than the remaining nodes.

- 6 "Power2 SuperChip" thin nodes, with the IP addresses grand[32-37].nist.gov
  - 120 MHz clock
  - L1 cache: 128KB data/32KB instr
  - 512 MB of local memory
- 1 "SP2-Wide" Node, with the IP address grand17.nist.gov This node is the rough equivalent of an RS/6000 Model 590.
  - 66.7 MHz clock
  - 64 KB cache line
  - 128 MB of local memory
- Approximately 2.5 GB of local disk space in /tmp on each of the nodes.
- A 98B RS/6000 rack system serving as a control workstation, danube.nist.gov.
- Summary table of SP Processor Nodes for comparison¡!figure Summary table of SP Processor Nodes for comparison¿

- SP Switch
  - Two-level cross bar switch
  - TB3 switch adapter with peak hardware performance of 150 MB/s.

For recent changes to the local configuration, see What's New with the SP2 system (http://danube.nist.gov/new.html) at the NIST SP2 website.

1.2 Interactive vs. batch access

The current NIST SP2 configuration allows users to interactively access the front-end machine `danube.nist.gov` and a small test pool of four nodes (grand17-grand20). The remaining nodes are available for batch jobs.

> This interactive pool is intended to be used only when interactivity is *absolutely required*, for example, in program debugging and run-time performance monitoring.

To execute large parallel jobs using the SP2 nodes, users must submit their jobs through a batch system, which partitions the machine into node pools and controls access to these pools so that jobs are single-threaded **(i.e. when you run on a set of SP2 nodes, you are in a "dedicated" environment)**.

1.3 Sources of information and documentation

There is quite a bit of electronic information you can browse through to get acquainted with the SP2 and PVM or MPI.

Locally, you will find IBM-supplied SP2 docs (in postscript) on danube.nist.gov in the directory /doc. For more information on the differences between PVM and MPI, see the section on Choosing a Message Passing Library (p.11).

Online man-pages are another valuable source of information, in particular for quick information about message passing routine syntax. Man-pages are available for both MPI and PVM, and are accessed by typing: `man routine`. Note, although information is provided for Fortran as well as C, *you must request the information using the C routine name.*

Another local source of information is a documentation system called Info-Explorer, located on danube. It has a X-windows user-interface, and so you must remember to set your `$DISPLAY` environment variable to your local console and add danube.nist.gov to your xhost list. Use the command "setenv

DISPLAY your_console.nist.gov:0.0" on danube to set the DISPLAY variable, and the command "xhost +danube.nist.gov" on your console to allow danube to open a window on your console. InfoExplorer can then be invoked with the command "info [-l libraryname]", where *libraryname* can be used to specify the user manual of interest. Currently, the following are available:

```
info -l xlf    (Fortran manuals)
info -l pe     (Programming Environment manuals)
info -l wb     (Workbench manuals)
info -l LoadL  (LoadLeveler manuals)
info -l pvme   (pvm3 manual)
```

If you start up InfoExplorer without a library specified, you can link to one later in the *"Task & Topic Index"* window (not through the *"Welcome to the InfoExplorer..."* window, which also appears) by pulling down the menu labelled "info" and choosing "Library" and "Select".

The SP User Group Home Page (http://spud-web.tc.cornell.edu/HyperNews/get/SPUserGroup.html) at Cornell has links to current discussion groups, educational sites, and a host of other information. To quote their introduction: "This WWW information server is designed for IBM SP users, systems administrators, and other people interested in parallel programming to share information. It is a place to ask questions, seek answers, and contribute knowledge to the other people interested in the IBM SP system."

There is a well-maintained WWW home page for PVM, PVM on Netlib (http://www.epm.ornl.gov/pvm/pvm_home.html) and some very useful SP training materials on the WWW site at the Cornell Theory Center (http://www.tc.cornell.edu/-Edu/Education.and.Training.Materials/Education.html) Note that the CTC was one of the first IBM SP sites and has a long-term affiliation with IBM, which contributes to their excellent support system.

See also the Frequently Asked Questions (FAQ) list for our local site (http://danube.nist.gov/aix_faq.html) .

2 Distributed-memory programming

2.1 The "art" of distributed-memory programming

If you have not already discovered this, you will probably soon realize that there are significant differences in programming a distributed-memory (DM) machine compared to a conventional machine. In fact, some might say there is a real "art" to DM programming, and a way of thinking that just is not required elsewhere.

The primary reason DM machines are more difficult to use is the fact that, not only is the data in memory distributed, but, in general, *the programmer is responsible for ensuring that data is in the right spot at the right time*, typically by using a message passing library to send and receive data across a network to and from processing nodes in the machine. (A notable exception, of course, is virtual shared-memory machines, such as the KSR, which have operating systems designed to manage distributed data without explicit user control.) This responsibility on the shoulders of the user is far from trivial, particularly considering the fact that data movement across a network doesn't always behave predictably. When data messages are delayed due to backlog on the network, for example, program synchronization becomes an issue, and a given program may not behave deterministically - a characteristic that many programmers have always taken for granted and counted on as an indisputable fact.

Where does the "art" come in? Primarily in finding the right way to view an application so that a data distribution which maximizes efficiency comes to the fore. It's likely that with enough effort, virtually any distribution of data across a machine can be made to work. However, if the goal is to have a program that actually runs *faster* on multiple nodes, rather than slower, the most obvious data distribution might not achieve the goal. "Art" might occasionally translate to patience, while the application developer experiments with a range of distributions until the most efficient is found.

2.2 SPMD versus MPMD programming

The SP2 is equipped to run in both Single Program Multiple Data (SPMD) and Multiple Program Multiple Data (MPMD) modes. The former requires that each node run an identical (single) program, though the data being processed on each node may differ. The latter allows the user to run different programs on each of the nodes. Probably the most familiar example of MPMD programming is the manager-worker framework, where the user composes two program: a manager program running on one processor to perform I/O or other program set-up functions and then coordinate the efforts of the other processors, and a worker program running on the remaining processors to perform the basic parallel tasks in the application. Often, SPMD programs are designed to mimic this manager-worker model, since there is always a portion of work in an application which it makes sense to perform on only one processor. This mimicking is accomplished by having one of the processors in the pool act alternately as a manager and a worker. *NOTE: One important advantage to adopting the SPMD mimic over the original MPMD manager-worker model is that the resulting code can be run on only one processor, thus greatly facilitating debugging.* We include a conversion to SPMD example (p.10) to illustrate how a typical MPMD manager-worker pair can be converted into a SPMD program.

2.3 Choosing a message passing library: PVM(e) or MPL?

If you have programmed in PVM in the past, and perhaps even have developed your application for workstation clusters or other machines using PVM, you are most likely interested in putting the PVM portability to use by porting your code without modification to the SP2. Unfortunately, *there can be a performance issue regarding your choice of message-passing library.* Standard PVM release 3.3.11 is installed on the SP2, and supports use of the SP2 as both cluster of workstations with ethernet interconnections (RS6K architecture) as a more tightly coupled system using the SP Switch (SP2MPI architecture). To use the switch while maintaining a good portion of PVM portability, IBM has provided their own proprietary version of PVM, called PVMe ("e" for "enhanced").

> **Note:** PVMe *is* able to access the switch, but there are some catches: it is compatible with a previous version of PVM (currently, 3.2.6), and it restricts the user to one PVM process per node - **if you try to spawn more processes than you have nodes, PVMe will fail.**

If you currently use any new features of PVM3.3 (global reduce operations, for example), then you will have to back-track to 3.2.6 compatibility to run on the switch, and, if you are accustomed to being able to overlap PVM processes on a node, you will have some work to do to adjust to the PVMe restriction. On the bright side, IBM seems committed to continuing to upgrade PVMe in the footsteps of PVM, so eventually those nice PVM3.3 features should be added to PVMe.

An alternative is to give up the PVM portability edge and use IBM's own message passing library, MPL. For obvious reasons, IBM has designed this library to use the switch, and it comes complete with global communication routines and other features not found in PVMe. Further, many of IBM's parallel programming tools for tasks such as program visualization and performance monitoring are available only to users of MPL. The structure of the message passing calls is very similar to those of Intel's NX and Thinking Machines' CMMD - so porting is not necessarily a difficult task. If you have familiarity with these other vendors syntax for message passing, using MPL should be pretty straightforward. If your only message passing experience is with PVM, then the sample programs included later in the tutorial may be useful, allowing you to directly compare a PVM(e) program with a corresponding MPL program which performs the same task. Also, later sections Message Passing with PVM(e) (p.12) and Message Passing with MPL (p.18) provide some examples of how to send and receive messages in each library.

3 Message passing with PVM(e)

PVMe is the IBM proprietary version of the widely used PVM message passing library from Oak Ridge National Laboratory. Its compatibility with the public domain PVM package generally lags one release behind. (For example, the current release of PVMe is compatible with PVM 3.2.6)

We will assume that the reader has a basic understanding of the concept of message passing communication of data on distributed memory parallel architectures. PVM(e) is described here primarily by example, but for the interested reader, extensive PVM documentation is available from the PVM authors at PVM on Netlib (http://www.netlib.org/pvm3/index.html) and from the online documentation available on danube (p.8).

3.1 Setting up the PVM environment

Before executing a PVM(e) message passing program, a "Virtual Machine" (VM) must be initiated by the user. This is done by invoking what is known as the PVM(e) daemon, a process which sets up and maintains the information needed for PVM(e) processes to communicate with one another. In general, the user can "customize" the VM by specifying which hosts it should include, the working directory for PVM processes on each host, the path to search for executables on each host, and so on. PVMe behaves slightly differently than PVM, since nodes are controlled through a Resource Manager. Rather than specifying particular nodes for the virtual machine, the user requests a certain number of nodes, and the Resource Manager reserves these nodes for that one particular user. This is to allow the user dedicated access to the High Performance Switch for the duration of their PVMe job.

One difference between PVM and PVMe on the SP2 is how information is supplied to the daemon. For PVM, a command line argument is used to pass the name of a "hostfile". The PVM hostfile is a formatted text file which specifies this information with a simple syntax understood by PVM. An example is found in hostfile (program listing p.157) .

For PVMe, a command line flag to the daemon, "-n ⟨number⟩", specifies the number of nodes to request from the Resource Manager. To override the default settings for the working directory and executable search path, environment variables can be used. See the PVMe description in /doc/pvme/pvme.ps and the section Interactive tutorial examples (p.32) for more information.

3.2 Checking the PVM environment

Both PVM and PVMe provide a utility for checking the status of the VM. This utility, called the PVM console, is started by entering the command "pvm." A variety of query commands can then be entered at the console prompt to determine, for example, which hosts belong to the VM, which processes are running under PVM, etc. See the documentation PVM on Netlib (http://-www.epm.ornl.gov/pvm/pvm_home.html) for detailed information.

3.3 Getting started with PVM programming

With some exceptions (noted in section on differences between PVM and PVMe (p.16)), PVM and PVMe are source code compatible. However, it is *vital* that your program is linked with the appropriate PVM or PVMe message passing libraries; these are NOT interchangeable. A code compiled and linked for PVM must be relinked to run under PVMe. See the section on Compiling message passing programs (p.24) for sample makefiles with the proper library paths for linking PVM/PVMe programs. With this important difference noted, we will use PVM to refer to both PVM and PVMe throughout the remainder of this section,

In coding a message passing program in PVM, calls to communication routines from PVM are included in the same way one makes subroutine calls in Fortran, or function calls in C. A program which requires PVM communication must "enroll" itself in the PVM virtual machine before these calls are made. The call to enroll a program in PVM is:

`call pvmfmytid(nodeid)` or, in C: `nodeid = pvm_mytid();`

3.4 Sending a PVM message

Messages are prepared for sending by packing a buffer with the data items to be included in the message, with one "pack" call for each data item. This allows the user to easily include data of varying types within the same message. A "initialization" call is made just prior to packing to initialize the message buffer. Once packed, the buffer is sent to a specified destination with a "send" call. The initialization call serves primarily to set whether the data is to be converted to the portable XDR format prior to sending (first argument to pvminitsend is PVMDATADEFAULT, or 0), or sent in its raw form (first argument to pvminitsend is PVMDATARAW, or 1). A third possible value for the first argument to the initialization call is PVMDATAINPLACE, which eliminates the copying of the message into the buffer during packing and therefore improves efficiency. Packing in this context copies a pointer to the data and its size into the buffer (note: the data within each pack must occupy contiguous memory locations). *When PVMDATAINPLACE is used, care must be taken not to modify the data before the message is actually received.* For details about the trade-offs between these initialization schemes, the reader is referred to the PVM or PVMe reference guides. The examples provided in this primer use the PVMDATADEFAULT option, which give the highest degree of portability and protection from unintentional data overwrites.

Sample of Fortran message sending with PVM:

```
c  Initialize the buffer bufid:
      call pvmfinitsend(0, bufid)
c  Pack the real value rstart:
```

```
      call pvmfpack(REAL8, rstart, 1, 1, rc)
c  Pack the real 10-element vector data:
      call pvmfpack(REAL8, data, 10, 1, rc)
c  Pack the integer*4 value chunksize:
      call pvmfpack(INTEGER4, chunksize, 1, 1, rc)
c  Send the message to the node given in tids(1) with a 'tag' of 100:
      call pvmfsend(tids(1), 100, rc)
```

Sample of C message sending with PVM:

```
/* Initialize the buffer bufid (to use default XDR encoding): */
      bufid = pvm_initsend(PvmDataDefault)
/* Pack the real value rstart: */
      rc = pvm_pkdouble(&rstart, 1, 1)
/* Pack the real 10-element vector data: */
      rc = pvm_pkdouble(data, 10, 1)
/* Pack the integer*4 value chunksize: */
      rc = pvm_pkint(&chunksize, 1, 1)
/* Send the message to the node given in tids[0] with 'tag' 100:*/
      rc = pvm_send(tids[0], 100)
```

3.5 Receiving a PVM message

Messages can be received selectively based on a combination of the source (which node is sending the message) and tag (a qualifier specified by the user in the corresponding "send" call). A "wildcard" value (-1) can be specified for one or both of these receive call arguments to allow a message to be received regardless of the source or tag. When a message fitting the specified qualifications arrives at the node, it is received into a buffer. To retrieve the message contents, the buffer must be unpacked, *in the same order in which it was packed at the sending end.*

Sample of Fortran message receiving with PVM:

```
c  Receive a message from node hostid with tag 100 into a buffer:
      call pvmfrecv(hostid, 100, bufid)
c  Unpack the real value rstart:
      call pvmfunpack(REAL8, rstart, 1, 1, rc)
c  Unpack the real 10-element vector data:
      call pvmfunpack(REAL8, data, 10, 1, rc)
c  Unpack the integer*4 value chunksize:
      call pvmfunpack(INTEGER4, chunksize, 1, 1, rc)
```

Sample of C message receiving with PVM:

```
/* Receive a message from node hostid with tag 100 into a buffer: */
      pvm_recv(hostid, 100)
/* Unpack the real value rstart: */
      rc = pvm_upkdouble(&rstart, 1, 1)
/* Unpack the real 10-element vector data: */
      rc = pvm_upkdouble(data, 10, 1)
/* Unpack the integer*4 value chunksize: */
      rc = pvm_upkint(&chunksize, 1, 1)
```

3.6 Broadcasting a message in PVM

Work in progress...

3.7 Differences between PVM and PVMe

Work in progress...

- Number of processes per node limited in PVMe
- Only one daemon process for PVMe. For PVM, in contrast, there is a main daemon on the parent node, and a sub-daemon on each node in the VM.
- Availability of pvm_catchout, and routing of node stdout
- The PVMe version of pvm_config() has a different (longer) argument list than public domain PVM. Read PVMe man pages for correct call; This extra parameter will cause an error if compiled with the public domain PVM.
- Generation of trace information

3.8 PVM/PVMe example in Fortran

On danube.nist.gov, the directory /usr/local/doc/Examples/pvm+pvme/Fortran contains the following simple Fortran example for PVM(e):

```
Filename        Style   Description
--------        -----   -----------
manager.f       MPMD    A basic "hello world" type program
worker.f
```

This directory includes the following files:

- README.pvm (program listing p.113)
- README.pvme (program listing p.116)
- make.pvm (program listing p.119)
- make.pvme (program listing p.121)
- manager.f (program listing p.123)
- manager.input.pvm (program listing p.126)
- manager.input.pvme (program listing p.127)
- user.cleanup (program listing p.128)
- user.setup (program listing p.129)
- worker.f (program listing p.130)

To run the sample program in batch under ** ORNL PVM **, follow the instructions given in README.pvm (program listing p.113)

To run the sample program in batch under ** IBM PVMe **, follow the instructions given in README.pvme (program listing p.116)

3.9 PVM/PVMe example in C

On danube.nist.gov, the directory /usr/local/doc/Examples/pvm+pvme/C contains the following simple C example for PVM(e):

```
Filename        Style   Description
--------        -----   -----------
manager.c       MPMD    A basic "hello world" type program
worker.c
```

This directory includes the following files:

- README.pvm (program listing p.96)
- README.pvme (program listing p.99)
- make.pvm (program listing p.102)
- make.pvme (program listing p.104)
- manager.c (program listing p.106)
- user.cleanup (program listing p.109)
- user.setup (program listing p.110)
- worker.c (program listing p.111)

To run the sample program in batch under ** ORNL PVM **, follow the instructions given in README.pvm (program listing p.96)

To run the sample program in batch under ** IBM PVMe **, follow the instructions given in README.pvme (program listing p.99)

4 Message passing with MPL

MPL is the IBM proprietary message passing library. It doesn't feature the same portability that PVM boasts, however, a number of tools have been developed specifically for the SP2 which require the use of MPL message passing. (Information on these tools will soon be added to this primer.)

We will assume that the reader has a basic understanding of the concept of message passing communication of data on distributed-memory parallel architectures. MPL is described here primarily by example, but for the interested reader, MPL documentation (p.8) is available online.

The function/subroutine calls covered in this tutorial are:

```
In C:                In Fortran:
----------------------------------------
mpc_environ()        call MP_ENVIRON()
mpc_task_query()     call MP_TASK_QUERY()
mpc_send()           call MP_SEND()
```

```
mpc_wait()          call MP_WAIT()
mpc_recv()          call MP_RECV()
mpc_brecv()         call MP_BRECV()
mpc_bcast()         call MP_BCAST()
```

4.1 Getting started

As with PVM, in MPL one includes calls to communication routines in the same way one makes subroutine calls in Fortran, or function calls in C. Corresponding to the pvm_mytid() call in PVM, MPL provides the MP_ENVIRON() call to determine the number of processors (set at runtime) and the task id of the processor running a particular instance of the program. The syntax of the MP_ENVIRON call is:

```
call MP_ENVIRON(nprocs,nodeid)
```

Or, in C:

```
rc=mpc_environ(&nprocs, &nodeid);
```

MPL provides the MP_TASK_QUERY call to obtain information about system variables and constants (beyond what's returned by the MP_ENVIRON call). Depending on the last argument to the MP_TASK_QUERY call (the "query-type"), various information can be retrieved (see the man-page for complete information). The most common use of this call is to obtain the system values for "dontcare" and "allgrp", which are the "wildcard" value, used for indiscriminately receiving messages, and the default processor group containing all nodes, often used in broadcasting information across the machine.

Sample of a Fortran MP_TASK_QUERY call:

```
      integer nbuf(4), dontcare, allgrp
c
c     Learn value of allgrp and dontcare, for later use with broadcast:
c
      call MP_TASK_QUERY(nbuf, 4, 3)
      dontcare = nbuf(1)
      allgrp = nbuf(4)
```

Sample of a C mpc_task_query() call:

```
   int nbuf[4], allgrp, dontcare;
/*
/  Learn the value of allgrp and dontcare, for use with broadcast.
*/
   rc = mpc_task_query(nbuf, 4, 3);
   dontcare = nbuf[0];
   allgrp = nbuf[3];
```

4.2 Sending an MPL message

Messages are sent in MPL with a call to MP_SEND (or mpc_send) for each piece of data to be sent (contrasting with the pack and send model in PVM). The call to the send routine returns immediately, without waiting for the receive to actually take place at the destination node. Because of this "asynchronous" behavior, it is important to pair a send call with a call to MP_WAIT (or mpc_wait) if execution should not proceed until the destination processor has actually received the data. For example, if the data is going to be modified at the source after a send, a wait call before this modification will ensure that the original data is transferred, rather than the modified data.

Sample of Fortran message sending with MPL:

```
c
c       Send msgleng consecutive bytes of data, starting with
c       location nodemsg, to node 0 (host) with a tag of 'msgid'
c
        call MP_SEND(nodemsg, msgleng, 0, 100, msgid)
c
c       Before proceeding, wait for nodemsg to be received at the host:
c
        call MP_WAIT(msgid, nbytes)
c
c       Now, nodemsg can be safely modified...
c
```

Sample of C message sending with MPL:

```
/*
```

```
/     Send msgleng consecutive bytes of data, starting with
/     location nodemsg, to node 0 (host) with a tag of 'msgid'
*/
      rc = mpc_send(nodemsg, msgleng, 0, 100, msgid);
/*
/     Before proceeding, wait for nodemsg to be received at the host:
*/
      rc = mpc_wait(&msgid, &nbytes)
      if (rc) {
         print("Error in mpc_wait(). Return value: %i\nExiting.\n", rc);
         exit(-1);
      }
/*
/     Now, nodemsg can be safely modified...
*/
```

4.3 Receiving an MPL message

Messages can be received selectively based on a combination of the source (which node is sending the message) and tag (a qualifier specified by the user in the corresponding "send" call). A "wildcard" value can be specified for one or both of these receive call arguments to allow a message to be received regardless of the source or tag. When a message fitting the specified qualifications arrives at the node, it is copied into the location specified in the receive call. Unless the specific "blocking" receive call is made (MP_BRECV), the program will not wait for the message to actually arrive before it continues processing. Therefore, if you need to use the data being received, you must either use the MP_BRECV call, or post a MP_WAIT call which will halt execution until the message is received. It is the non-blocking strategy which allows for the possibility of using cycles during network delays to perform useful work. However, it also can give the false impression that data has been received when it has not, thus creating difficult to locate bugs!

Sample of Fortran message receiving with MPL:

```
c
c  Receive a message msgleng bytes long from node i with tag 100 into the
c  consecutive memory locations beginning at 'nodemsg'
c  NOTE: This is a 'blocking' receive.
c
```

```
          call MP_BRECV(nodemsg, msgleng, i, 100, msgid)
```

Sample of C message receiving with MPL:

```
/*
/  Receive a message msgleng bytes long from node i with tag 100 into the
/  consecutive memory locations beginning at 'nodemsg'
/  NOTE: This is a 'blocking' receive.
*/
      rc = mpc_brecv(nodemsg, msgleng, &i, &type, &msgid);
```

4.4 Broadcasting a message in MPL

Broadcasts are an easy way to distribute data from one node to all other nodes (for example, sending program parameters from a host program to the node programs). The MP_BCAST command performs the broadcast, taking the location of the data item to be sent as the first argument, the length of the message as the second argument, and the source node and participants as the final two arguments. Typically, all nodes are included in the participant list, and there is a special value used to indicate this. The broadcast can be tailored to use only subsets of the nodes, however. See the online POE documentation for more details.

Sample of a Fortran broadcast with MPL:

```
c
c     Broadcast the 8 byte double rstart from node 0 to all others:
c
      call MP_BCAST(rstart,8,0,allgrp)
c
c     Broadcast the 4 byte integer chunksize from node 0 to all others:
c
      call MP_BCAST(chunksize,4,0,allgrp)
```

Sample of a C broadcast with MPL:

```
/* Broadcast the 8 byte double rstart from node 0 to all others:      */
   rc = mpc_bcast(&rstart,sizeof(double),nodeid,allgrp);
/* Broadcast the 4 byte integer chunksize from node 0 to all others: */
   rc = mpc_bcast(&chunksize,sizeof(int),nodeid,allgrp);
```

4.5 MPL example in Fortran

On danube.nist.gov, the directory /usr/local/doc/Examples/mpl/Fortran contains the following simple Fortran example for MPL:

```
Filename          Style   Description
--------          -----   -----------
mplmanager.f      MPMD    A basic "hello world" type program
mplworker.f
```

This directory includes the following files:

- README.mpl (program listing p.85)
- make.mpl (program listing p.88)
- mplmanager.f (program listing p.90)
- mplworker.f (program listing p.92)
- user.cleanup (program listing p.94)
- user.setup (program listing p.95)

To run the sample program in batch under ** IBM MPL **, follow the instructions given in README.mpl (program listing p.85)

4.6 MPL example in C

On danube.nist.gov, the directory /usr/local/doc/Examples/mpl/C contains the following simple C example for MPL:

```
Filename          Style   Description
--------          -----   -----------
mplmanager.c      MPMD    A basic "hello world" type program
mplworker.c
```

This directory includes the following files:

- README.mpl (program listing p.73)
- make.mpl (program listing p.76)

- mplmanager.c (program listing p.78)
- mplworker.c (program listing p.81)
- user.cleanup (program listing p.83)
- user.setup (program listing p.84)

To run the sample program in batch under ** IBM MPL **, follow the instructions given in README.mpl (program listing p.73)

5 Compiling message passing programs

The Unix utility "make", or a variant of it, is by far the easiest way to control the compilation and linking of your programs. The user is responsible for describing dependencies between programs in a specially formatted file call a "makefile". In this file, the user can also set which compilers to use, which libraries to include for proper linking, how to "clean-up" directories, and so on. To get started with make, copy a sample makefile from the Examples directory (two are reproduced here) and modify it to suit your needs.

Note that in the two makefiles shown here, different compilers are specified (see the lines beginning with `FC =` and `CC =`). MPL programs require that the mpxlf or mpcc compiler be used, while PVMe programs require only the standard xlf or xlc compiler.

- Sample MPL program makefile (program listing p.48)
- Sample PVM program makefile (program listing p.64)
- Sample PVMe program makefile (program listing p.66)

6 Using the LoadLeveler batch control system

To access nodes of the NIST SP2, outside of the three-node interactive pool, users must submit their jobs to the batch system, IBM LoadLeveler. NIST is currently running Version of 1.3 of LoadLeveler. The commands used to interact with the batch system given below.

6.1 Batch commands

llsubmit *command.file* Submit a job to the batch system. The argument to this command is the name of a file containing special LoadLeveler commands which describe the user's program and machine requirements.

llq [-l]

Query a job's status in the batch queue.

llstatus Query the status of the node pools.

llcancel *batchid* Kill a job that appears to be hung (on the queue or while processing) or is otherwise not performing as expected. The argument to this command is the four digit job identifier for the process to be cancelled, and can be extracted from the output of the "llq" command.

llclasses Display a current list of the available batch classes on the SP2.

classes Local version of ¡it¿llclasses¡/it¿ giving a short listing of available batch classes.

For more details on each of these commands, see the local LoadLeveler documentation (http://danube.nist.gov/ll.html)

6.2 Batch job preparation utilities

Since both PVM(e) and MPI typically require the user to play an interactive role in the set-up of the parallel environment, a job submitted through Loadleveler *must mimic the steps a user would take in an interactive environment.* This is accomplished via a shell script, which is invoked when the LoadLeveler job becomes active on the queue. This shell script can be user-defined, but for routine usage, the user may benefit from using prewritten generic scripts. These scripts make use of information provided in the LoadLeveler command file to invoke the appropriate environment control tool (either POE (MPI) or PVM), set the correct number of processors, copy files to/from the local node address space, and start the user's MPMD or SPMD program(s) with the correct arguments and/or input redirection. The scripts do require that certain program information be correctly specified in the LoadLeveler command file, so a user interface has been provided which automates the generation of this command file. This utility is invoked by typing `extoll` at the Unix prompt in the directory where your executables are stored. (Note that this utility superceeds "X-llcreate" and "llcreate".) The program prompts the user for specific information about the parallel program, and then creates an appropriate command file which includes the appropriate generic script file to properly execute the user program on the nodes. For more information, see the Extoll documentation (p.26).

7 Batch job utilities: Extoll

7.1 Introduction to Extoll

Extoll provides a convenient "point-and-click" style interface for generating a LoadLeveler command file matched to a user's specific parallel program. (Note that Extoll superceeds the utilities X-llcreate/llcreate, providing more robust and streamlined services.)

Note: when using `Extoll`, your DISPLAY environment variable must be set to your working console. To accomplish this, type "setenv DISPLAY console.internet.address:0.0" at the unix prompt on danube, and "xhost +danube.nist.gov" at the unix prompt on your console.

When Extoll is invoked, the user is prompted for information about the communication library used (PVMe, MPL, or Oak Ridge PVM), the style of their program (SPMD or MPMD), arguments or input required, batch class and number of processors, and whether a custom setup or cleanup script is required. Finally, when the "Create *.cmd File" button is pressed, a command file is generated based on the input information.

7.2 Use of Extoll

To use `Extoll`, simply invoke it in the directory where your executables are stored (this is often the same as your source directory, or `~/pvm3/bin/RS6K`). check the appropriate boxes and fill in requested information in the entry panel. The script is relatively generous in interpreting your input; for example, the program is insensitive to the case of most responses. In fact, most input can be handled with a click of a button, eliminating the need to worry about typos in the input.

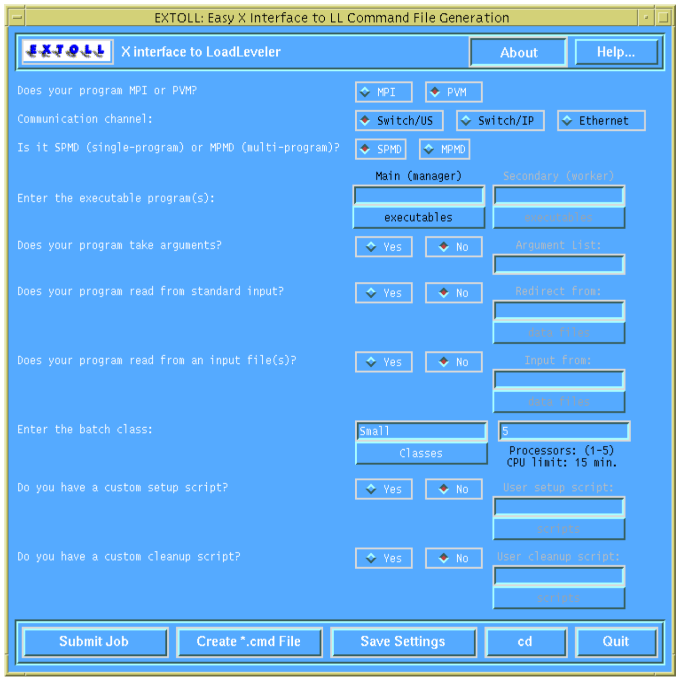

Figure 1: A sample view of the Extoll interface

Of course, with the use of this utility, you maintain the flexibility of editing the generated command file, which could be used, for example, to correct any mistyped program names, etc. You may wish to experiment with various combinations of responses to the queries, studying resulting command file, to help your understanding of how information is relayed to the generic script. If you run into difficulty because your particular program doesn't fit any of the standard scenarios handled by `Extoll` and the appropriate generic script file, please send mail to *kremington@nist.gov* and we will try to update the scripts to accommodate your needs.

A short (8-page) user's guide is available, see ¡A HREF="http://math.nist.gov/acmd/Staff/KRemington/Primer/extoll-doc.ps"¿ Extoll: Easy X-interface to LoadLeveler for Parallel Jobs on the NIST IBM SP2¡/A¿, or the file /usr/local/apps/EXTOLL/doc/extoll-doc.ps on danube.

7.3 Generic Batch Scripts for Message Passing Programs

The Extoll utilitly relies on the use of generic batch scripts, one for each of the available message passing libraries. These scripts are intended to be system supported and maintained, so that all users benefit from locally developed improvements in batch script handling. Actual support of these scripts may vary. Please contact the author at kremington@nist.gov for current information.

7.4 Sample command file

```
#!/bin/ksh
# -----------------------------------------------------------------
# LoadLeveler Command Script:  main.cmd
# Automatically generated at:  Fri Aug 28 14:20:21 EDT 1998
# EXTOLL V. 2.0
# -----------------------------------------------------------------
#
# @ job_name = main
# @ initialdir = /danube/nist/karin/EXTOLL/bin
# @ output = $(initialdir)/$(Cluster).out
# @ error = $(initialdir)/$(Cluster).err
# @ notification = Error
# @ notify_user = karin@danube.nist.gov
# @ class = Small
# @ job_type = parallel
# @ min_processors = 4
# @ max_processors = 16
# @ environment = CLUSTER=$(Cluster); LL_JOB=TRUE
# @ restart = no
# @ checkpoint = no
# @ requirements = (Adapter == "hps_user")
# @ queue
#
# -----------------------------------------------------------------
#    Set up and start an SP2MPI PVM job which uses the SP2
#    switch in User Space mode (high performance).
#
#    NOTES:  > Executables must be compiled according to the
#              PVM documentation in README.mp.
#            > Processes are limited to one per node, and each
#              pvm_spawn generates an "extra" process.
#
```

```
#    (Please see PVM documentation in README.mp for more info.)
# ------------------------------------------------------------------
#    This stub is to set PVM_ROOT and PVM_ARCH in your .cshrc file.
#    See the EXTOLL documentation for proper .cshrc setup.
#
cp $EXTOLL_ROOT/bin/.pvm-switch.stub $HOME/.llpvmstub
# ------------------------------------------------------------------
#
#    Create a PVM hostfile with the Executable path set to /tmp/karin
#             ( using switch addresses hssw*.nist.gov)
#
cat /tmp/$LOADL_STEP_OWNER.$LOADL_STEP_ID.hostfile |
    sed -e 's/grand/hssw/' >> $CLUSTER.hostfile
# ------------------------------------------------------------------
export PVM_ROOT=$HOME/pvm3
export PVM_ARCH=SP2MPI
export PVM_PROCESSOR_LIST=$LOADL_PROCESSOR_LIST
export STARTHOST=`hostname | sed -e 's/grand/hssw/'`
export LOADL_PROCESSOR_LIST=`echo $LOADL_PROCESSOR_LIST |
                             sed -e 's/grand/hssw/g'`
export MP_PROCS=4
export MP_RMPOOL=1
export ARGUMENTS=""
export STANDARD_IN=""
export FILES_IN="my.data"
# ------------------------------------------------------------------
#    Copy executables and 'cd' to working directory:
# ------------------------------------------------------------------
for host in $PVM_PROCESSOR_LIST
do
  rsh $host "ksh -c '[ -d /tmp/karin ] || mkdir /tmp/karin'"
  rsh $host "cd /danube/nist/karin/EXTOLL/bin; cp -f main worker /tmp/karin"
done
cd /tmp/karin
# ------------------------------------------------------------------
#    Now, invoke the daemon and run the [setup] program [cleanup]:
# ------------------------------------------------------------------
$PVM_ROOT/lib/$PVM_ARCH/pvmd3 -n$STARTHOST $CLUSTER.hostfile &
sleep 15
/danube/nist/karin/EXTOLL/bin/user.setup
main $ARGUMENTS < $STANDARD_IN
/danube/nist/karin/EXTOLL/bin/user.cleanup
# ------------------------------------------------------------------
#    Delete executables and other files from working directory:
```

```
# ---------------------------------------------------------------
cd /danube/nist/karin/EXTOLL/bin
for host in $PVM_PROCESSOR_LIST
do
  rsh $host 'rm -r /tmp/karin"'
done
# ---------------------------------------------------------------
#    Finally, halt the daemon and copy back the PVM log file:
# ---------------------------------------------------------------
$PVM_ROOT/lib/$PVM_ARCH/pvm <<EOJ
halt
EOJ
export myuid
myuid=`id -u`
cp /tmp/pvml.$myuid $CLUSTER.pvmlog
echo Batch job completed.
# ---------------------------------------------------------------
```

8 Interactive access to the NIST SP2

8.1 Introduction

The current SP configuration allows users to interactively access the front-end machine (danube) and a small test pool of four nodes (grand17-grand20). This interactive pool should be used to facilitate debugging, or to study performance with interactive analysis tools. To execute large parallel jobs using the SP2 nodes, users must submit their jobs through a batch system, which partitions the machine into node pools and controls access to these pools so that jobs are single-threaded.

Using the interactive nodes is much like using a workstation cluster. You begin by logging onto one of the interactive nodes (grand17, grand18, grand19 or grand20) and then setting up the parallel environment you wish to use. ORNL PVM, IBM's PVMe and IBM's POE(MPL) are all available on the interactive nodes. ORNL PVM may be of particular interest for use with various third-party performance tracing tools, such as XPVM, which are not compatible with PVMe.

8.2 Managing message passing programming sessions

An interesting problem is posed by the existence of two separate and incompatible libaries both sharing the name PVM (ORNL PVM, and IBM PVMe). Without proper care, it becomes quite difficult to manage directories, makefiles, etc. so that PVM and PVMe programs can coexist, and you as a user can switch between one environment and the other without substantial effort. Care is also required to ensure that the interactive environment does not interfere with any batch processing you may have on queue.

These problems can be addressed on the NIST SP2 with two strategies:

- First, the process of setting up your interactive environment can be automated to ensure that all necessary environment variables are set, and that the appropriate PVM daemon and console commands, consistent with the desired environment, are enlisted. This automation is accomplished by appending several lines to your .cshrc file which have been designed to enable switching from one environment to the other during a single interactive session by typing simple commands: "startpvm", "startpvme", or "startmpl".

- Second, batch environments can be designed in a self-contained way, so that they are impervious to anything going on in the user's interactive session (or in their .cshrc file). This is ensured with the use of the generic batch scripts provided to work in conjuction with the batch job preparation utilities X-llcreate and llcreate. If you require the use of a completely customized batch script, and it currently relies on your .cshrc file, you may wish to modify it so that it is self contained, that is, so that all of the variables required by PVM(e) are explicitly set in the batch script, rather than being inherited from the parent environment. The generic scripts can be used as a guide in making any necessary modifications.

8.3 Debugging PVM(e) programs

PVM(e) provides simple facility for spawning processes under a standard debugger. If the pvmfspawn call provides "PVMDEBUG" as the second argument (or "PvmTaskDebug" for spawns in C programs), an xterm window running the spawned process under a debugger will be generated for each spawned process.

This is not by any means a "graceful" debugging environment, but can be used to get a "quick-and-dirty" view of the processes. To invoke the debugger (for PVM or PVMe programs), first compile your programs with the -g compiler option, and set up your PVM(e) environment by issuing "startpvm" or "startpvme" following by starting the daemon. Then, instead of simply executing your main programs, type

debug ⟨main_program⟩

Debug has been aliased to a system supported script which check that your DISPLAY variable is set and that the appropriate debugger scripts are available to PVM(e). The debug script is based on the serial dbx debugger; for help on available dbx commands, type "man dbx" on danube.

8.4 Tutorial examples

The directory /usr/local/doc/interact contains two short tutorials on interactive use of the PVM, PVMe, and MPL environments, and demonstrates how to switch from one environment to another. The tutorial directory includes the following:

- README.FIRST (program listing p.40)
- mpl
  - README.debug (program listing p.43)
  - README.mpl (program listing p.46)
  - make.mpl (program listing p.48)
  - mplmanager.f (program listing p.50)
  - mplworker.f (program listing p.52)
- pvm+pvme
  - README.debug (program listing p.54)
  - README.pvm (program listing p.57)
  - README.pvme (program listing p.60)
  - make.pvm (program listing p.64)
  - make.pvme (program listing p.66)

- manager.f (program listing p.68)
- worker.f (program listing p.71)

9 Some tips on debugging parallel programs

1. Make sure the serial code is bug-free!
    - To this end, it is a good practice to write programs (whenever possible) in a style which can be run on one processor. We include a conversion to SPMD example (p.10) to illustrate how a typical MPMD program pair can be converted into a SPMD program.
    - This step should eliminate any basic indexing errors, and ensure the serial program logic is correct.
    - In Fortran, use `IMPLICIT NONE`, if you don't already. This helps catch mistyped variable names by flagging them as undeclared.
2. Debug your PVM code on your own workstation, using multiple tasks, but a single host.
    - This step will make a coarse pass at eliminating message passing logic errors.
    - Examine code carefully, looking for:
        - Unmatched `send`'s and `recv`'s
        - Message tags that don't match
        - Non-unique message tags; that is, messages which aren't properly distinguished. (This is a problem especially when receiving with wildcard (`-1`) parameters.)
        - Messages that are unpacked upon receipt in a different order than they were packed before sending. The ordering must be identical.
    - Use `pvm_catchout()` to route node stderr/stdout to your console window. (Note: This is not available with the current release of PVMe on the SP2. Node output from PVMe is automaticaly routed to stdout of the main process.)

- Use the `PVMDEBUG` (Fortran) or `PvmTaskDebug` (C) parameter in the `spawn` call. This generates an xterm running "dbx" for each spawned process. (Make sure code is compiled with `-g`.)

  Sample Fortran call:

```
 call pvmfspawn('simplenode', PVMDEBUG, '*',
*                             (nprocs-1), tids, info)
```

  Sample C call:

```
rc = pvm_spawn("simplenode", (char**)0, PvmTaskDebug, "",
                                       (nprocs-1), tid);
```

3. If possible, debug your PVM program on multiple interactive workstations.

   - See the section on interactive access to the SP2 (p.31) for information on how to use debugging on the 4-node interactive pool.
   - If possible, test to see whether the program behaves differently on the interactive nodes of the SP2 than it does on a cluster of non-SP2 workstations.
   - If the program runs on a cluster, but fails regularly or sporadically on the SP2, then the bug is most likely to be highly sensitive to communication timing, and therefore *difficult to track down.* You may wish to seek the assistance of the scientific consulting staff (975-2968).

Finally, *when all else fails*...

4. Try debugging through the batch queue on the SP2 itself. (This feature will be available with an upcoming version of the X-Loadlgen utility.)

   - Make sure that your program(s) are compiled with the `-g` compiler option to `xlf` or `xlc`.
   - See the `llcreate` documentation (p.**??**) for help in generating a loadleveler command file which will enable debugging. Copy the `debugger` and `debugger2` scripts to your working directory.
   - Submit as usual, *but only when the queues are empty and you are free to wait by your console for you program to run.*

10 Converting an MPMD program to SPMD

There are at least two important advantages to programming in a SPMD style:

- **Debugging** - The program can easily be run sequentially, thus facilitating an important layer of debugging.

- **Efficiency** - Since each node on the SP2 under PVMe can run only one process, (a notable restriction of PVMe which is described in Choosing a Message Passing Library (p.11)), the manager part of a manager-worker program requires its own dedicated node, therefore eliminating one from the pool of nodes available to do the distributed work. In the SPMD form, the manager process spawns worker, and then goes on to do its own share of the distributed workload - thus making more efficient use of the available node pool.

We will assume in this discussion that PVM(e) is being used as the message passing library, however, the same strategy applies equally well to MPL programs.

The key to writing a manager-worker type program in SPMD style is to include a branch in your program just after a `pvm_mytid()` and `pvm_parent()` call, based on whether the particular program instance was spawned (== has a parent). If the program detects that it has no parent, it takes the role of the manager process and does all the start-up functions, including spawning worker tasks and distributing work. It may set aside a portion of the parallel work for itself, so that it does not sit idle while the parallel calculations are carried out. When finished with its share of the calculations, it can then receive messages back from the "worker" processes, perhaps containing the results of the parallel computations.

If the program detects that it was spawned from another process, then it bypasses all the start-up functions, and goes right to work on it's share of the global problem, then possibly sends to the "manager" process results of its calculations.

Following is an example of how a Fortran MPMD style program was transformed into a SPMD program:

The MPMD style program in
manager.f (program listing p.123)

and

worker.f (program listing p.130)

is converted to the SPMD program

spmd.f (program listing p.132)

See the directory /usr/local/doc/Examples/spmd for the complete spmd example, including a makefile and README file with instructions for building and executing the program.

11 Batch debugging

Work in progress...

12 XPVM program monitoring on the SP2

12.1 Program visualization with XPVM

The tool XPVM, developed at Oak Ridge National Laboratory, allows for viewing trace information generated by PVM programs. (Note that PVMe programs do not generate trace information which can be used with XPVM.) XPVM can be used on the interactive nodes, or in the batch environment.

12.2 Requirements

To use XPVM, your program must spawn processes with the "PvmTaskTrace" option. Further, if your program has *any* input requirements, or if you want to monitor a job running in batch under the system pvm.script file, you will need to include a subroutine call in your program to enable it to attach to a pre-established XPVM session at run-time. (This is due to limitations of XPVM, which cannot spawn programs with input requirements and which makes assumptions on the executable directory where your program resides.) Programs with no input requirements may be spawned from within an XPVM session on the interactive nodes, and do not require this extra subroutine call. To cover the most general case, we will assume that the program to be monitored cannot be spawned from with XPVM.

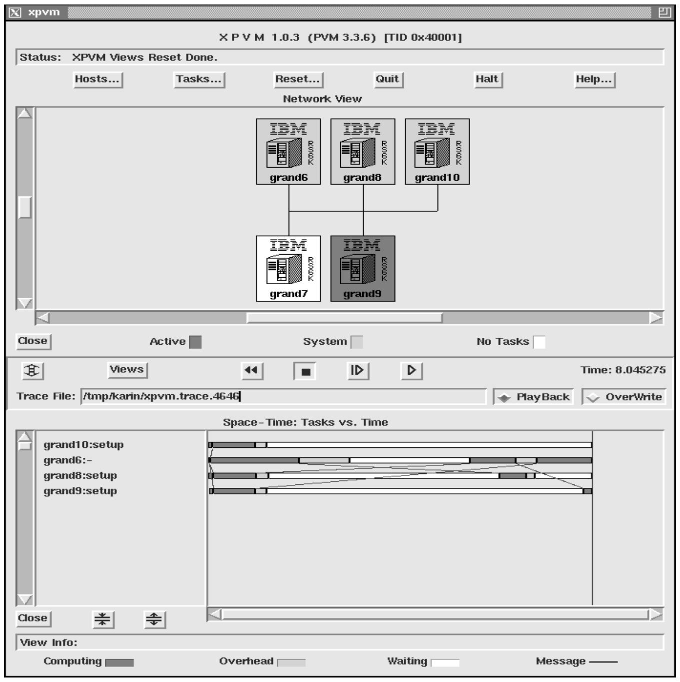

Figure 2: A sample view of the XPVM console window

12.3 Tutorial example: Matrix transposition

The tutorial included in the directory /usr/local/doc/Examples/transp demonstrates the use of XPVM to monitor a SPMD program running in the batch environment. The SPMD program setup.c generates a 3-D matrix distributed across the nodes, and calls a Fortran routine (in transp.f) to perform a matrix transposition. Since both C and Fortran are used, this example also serves to demonstrate how to link multi-language programs. A simple timer is also included, and can be used to get an estimate of elasped wallclock time.

- README.pvm (program listing p.137)
- make.pvm (program listing p.141)
- realtime.c (program listing p.143)
- setup.c (program listing p.145)

- transp.f (program listing p.149)
- user.cleanup (program listing p.154)
- xpvmconnect.c (program listing p.155)

Note the following:

- Your DISPLAY variable must be set on the node running the batch job. The easiest way to do this is to include your DISPLAY setting in your .cshrc file.
- The "grand" node running your batch process must have permission to display on your console. Since it can be difficult to anticipate which specific node will run a batch job (particularly in the case of TEST classes), you may want to add all "grand" nodes to your xhost list on your workstation console.
- Use extreme care when submitting a job to the batch system with the XPVM monitoring option. Interaction is required to properly terminate the XPVM session. If you are not at your console when the job is run, the XPVM will remain open for the maximum time permitted for the batch class which it is running under.

12.4 XPVM on the interactive node pool

To run this tutorial, copy the source and README files listed above to your own area and build the executable program (setup) just as you would for the batch tutorial. Note the arguments required for the exectuable program. (See README.pvm for details.) To use XPVM on the interactive node pool, first follow the intructions for setting up an interactive PVM session, since it is probably easiest to start XPVM *after* you've successfully started PVM. To proceed with XPVM:

- Open *two shell sessions (windows)* running on one of the interactive nodes.
- Check to ensure your DISPLAY is set and that the interactive grand node has permission to open on your console.

- Then, in one of the interactive shell sessions (windows), execute /usr/local/bin/xpvm. You need not specify a hostfile, since the job which is being traced will add hosts to XPVM as processes are spawned. After XPVM successfully starts, and you've selected a tracefile (or accepted the default tracefile), switch to the second window.

  It may seem like a more natural approach to start XPVM as a background process, but pecularities in the XPVM environment (and the fact that it generates its own shell) typically cause this approach to fail.

- In the second interactive shell session (window), execute the program by typing its name and arguments. The included call to xpvm_connect() will cause its trace information to be displayed in the XPVM display area. When the program finishes, you may switch to "playback" mode, and replay the trace file and examine various "views" of the collected information.

- When finished, you may choose to "quit" XPVM, which leaves the PVMvirtual machine intact, or "halt" XPVM, which "halts" the PVM daemon as XPVM is exited. You should choose "quit" if you intend to continue using PVM after leaving XPVM, and "halt" if you are finished with your PVM session.

A Interactive introductory README file

Program Listing: README.FIRST

```
  ***   Welcome to interactive PVM(e) and MPL on the NIST SP2!   ***

   This directory contains two tutorial subdirectories: one for
PVM and  PVMe (combined because the same source file can be used),
and a second for MPL.   For each message passing library, there is
a detailed README file containing step-by-step tutorial instructions
for building and running the sample program.

  ***   Before you begin !! ***

   Before you get started, you should append the lines attached to
the end of this README.FIRST file to your .cshrc file.  (If your default
shell is ksh, please contact karin@cam.nist.gov for the appropriate
commands to add to your startup file.)

   You may select either ORNL PVM (pvm), IBM's PVMe (pvme), or IBM's
MPL (mpl)  as your default message passing environment by editing the
.cshrc line indicated as the default setting.  By choosing a default,
you are in no way committing yourself to a particular library, and
you can easily switch between the two versions during any login session
with a single command (startpvm, startpvme, or startmpl).

   To run the tutorials, you should make your own copy of the sample
directory. Here's an easy way to accomplish this (assuming you don't
already have a directory called ~/interact):

   prompt> /usr/bin/cp -r /usr/local/doc/interact .

   After you've modified your .cshrc file and created your own sample
directory, you may begin your interactive session by logging in to
grand17, grand18, grand19, or grand20.

   The README files give directions for proceeding with the tutorials.

   Good luck!


#-----------------------------------------------------------------
```

```
#  Add these lines to your .cshrc file to facilitate switching
#  between PVM and PVMe.
#-------------------------------------------------------
#
# Setup aliases for switching between pvm and pvme:
#
alias startpvm "exec $SHELL -i pvm"
alias startpvme "exec $SHELL -i pvme"
alias startmpl "exec $SHELL -i mpl"
alias set_pvm_pool /usr/local/apps/LLCREATE/bin/set_pvm_pool
alias debug /usr/local/apps/LLCREATE/bin/debug
setenv PVM_ARCH RS6K
setenv PVM_ROOT /usr/local/apps/pvm3
#
# Choose whether this session will be PVM or PVMe...
#
# Set your desired default here:
#
set PVM_LIB=pvm
#
# This will change the default if an argument is passed to csh:
#
if ( $1 != "" ) then
  set PVM_LIB=$1
endif

if ( ${?PVM_LIB} )  then
   if ( ${PVM_LIB} == "pvm" ) then
      unset PVMEPATH
      unset PVMDPATH
      setenv PVM_ROOT /usr/local/apps/pvm3
      alias pvm $PVM_ROOT/lib/$PVM_ARCH/pvm
      alias pvmd3 $PVM_ROOT/lib/$PVM_ARCH/pvmd3
   else if ( ${PVM_LIB} == "pvme") then
      unsetenv PVM_ROOT
      setenv PVMEPATH $HOME/pvme/bin/RS6K
      setenv PVMDPATH /usr/lpp/pvm3
      alias pvm /usr/lpp/pvm3/pvm
      alias pvmd3 /usr/lpp/pvm3/pvmd3e -n 4
      alias pvmd3e /usr/lpp/pvm3/pvmd3e -n 4
   else if ( ${PVM_LIB} == "mpl") then
      unsetenv PVM_ROOT
      unsetenv PVMEPATH
      unsetenv PVMDPATH
```

```
      setenv MP_PROCS 4
      setenv MP_RMPOOL 3
      alias poe /usr/bin/poe
   endif
endif
#-------------------------------------------------------
```

End: README.FIRST

B MPL interactive example files

Program Listing: README.debug

```
* IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL *
--------------------------------------------------------------------------
|  Instructions for POE debugging:
|  To be used AFTER first running the interactive example as is (w/o debugging).
--------------------------------------------------------------------------

1) Make sure you are in the IBM POE(MPL) environment, type:

   prompt> startmpl

2) In your copy of the sample directory, type:

   prompt> make -f make.mpl

   The makefile will direct the compilation and linking of the
   programs.

3) Set the environment variable describing the parallel programming
   model (spmd or mpmd).  For this tutorial, the proper setting is mpmd:

   prompt> setenv MP_PGMMODEL mpmd

3) In the sample directory, start the pdbx debugger (which, in turn, starts
   the Parallel Operating Environment (POE).  For this tutorial, we take
   advantage of the command preprocessing option of the pdbx command
   to read (and then execute) some debugging commands from a file.

   prompt> pdbx -c debug.commands

   The commands in the file debug.commands load the mplmanager executable
   on node 0, and create a group of "workers", loading the mplworker
   executable on each of the "worker" nodes.
   (These commands could also be entered by hand after pdbx is started.)

   The response should resemble the following:


>  grand17>  pdbx -c debug.commands
>  Version 2.0                 Dec 16 1994 13:51:24
```

```
>
> WARNING: 0031-402  Using css0 as euidevice for User Space job
> WARNING: 0031-403  Forcing dedicated adapter for User Space job
> WARNING: 0031-403  Forcing dedicated adapter for User Space job
> WARNING: 0031-403  Forcing dedicated adapter for User Space job
> WARNING: 0031-403  Forcing dedicated adapter for User Space job
>
> pdbx(all) 0029-2040 3 tasks added to group workers
>
> pdbx(all)
> pdbx(all) reading symbolic information ...
> reading symbolic information ...
> reading symbolic information ...
> reading symbolic information ...
> [1] stopped in mplmanager at line 27 in file "mplmanager.f"
>    27         call MP_ENVIRON(nprocs,nodeid)
> [1] stopped in mplworker at line 21 in file "mplworker.f"
>    21         call MP_ENVIRON(nprocs,nodeid)
> [1] stopped in mplworker at line 21 in file "mplworker.f"
>    21         call MP_ENVIRON(nprocs,nodeid)
> [1] stopped in mplworker at line 21 in file "mplworker.f"
>    21         call MP_ENVIRON(nprocs,nodeid)
> 0031-504  Partition loaded ...
>
> pdbx(all)
```

4) Breakpoints for each node (or group of nodes) can be set using various pdbx commands. See the Parallel Environment Operation and Use reference manual (in /doc/pedocs/peopsuse.ps) or the pdbx manpage for details.

   The command "cont" ("continue") can be use to start the loaded programs running, or continue execution after breakpoints have been reached.

5) Enter the command "quit" to quit the pdbx debugger.

```
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------
```

End: README.debug

Program Listing: README.mpl

```
* IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL *
--------------------------------------------------------------------------
|  This directory contains a simple Fortran example for MPL
--------------------------------------------------------------------------
|
|  Filename           Style   Description
|  --------           -----   -----------
|  mplmanager.f       MPMD    A basic "hello world" type program
|  mplworker.f
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.mpl
make.mpl
mplmanager.f
mplworker.f

To interactively run the sample program under ** IBM MPL ** :

0) If this is your first use of the interactive nodes, make sure you
   have read and followed the instructions in ../README.FIRST !!
   You will be asked to add lines to your .cshrc file, copy the
   sample directory to your own area, and rlogin to one of the
   interactive nodes.

1) Switch to the IVM MPL environment, type:

   prompt> startmpl

     (If mpl is your default environment, set in your .cshrc file,
      and you haven't switched to another environment, then you can skip
      the startmpl step.  See ../README.FIRST for more information.)

   In your copy of the sample directory, type:

   prompt> make -f make.mpl
```

```
   The makefile will direct the compilation and linking of the
   programs.  The executables are created in the current directory
   in this example.

2) Set the environment variable describing the parallel programming
   model (spmd or mpmd).  For this tutorial, the proper setting is mpmd:

   prompt> setenv MP_PGMMODEL mpmd

3) In the sample directory, start the POE (Parallel Operating Environment)
   on the nodes of the interactive pool by typing:

   prompt> poe

   You'll be prompted to enter the names of executables, one for each
   node of the machine.

   When you enter the last executable name, the program automatically
   begins running.  To run the program again, you repeat this step,
   starting with the command "poe".

6) Examine the results file generated by the mplmanager program:

   prompt> cat mplmanager.results

8) Clean up...  type:

   prompt> make -f make.mpl new

   This will remove the executable, object, output and error files,
   to restore the directory to its original state.

--------------------------------------------------------------------
|  Again,
|  please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------
```

End: README.mpl

Program Listing: make.mpl

```
#--------------------------------------------------------------------------
# This makefile is used to compile MPL examples on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#--------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.

FC = mpxlf
FFLAGS= -g
CC = mpcc
CFLAGS = -g

# EXEDIR sets the directory where executables are put.
# Some people put all PVM executables in a bin directory, but
# here they are just kept in the current working directory.

EXEDIR=.

# LFLAGS may give paths to include for proper linking.
# (none required for MPL programs)

# FLIBS and CLIBS may give particular libraries required.
# (none required for these programs)

# INCL gives required include paths.
# (none required in this case)

#--------------------------------------------------------------------------
# This part is typically user-defined, giving the program names, etc.
#--------------------------------------------------------------------------

all: mplmanager mplworker


mplmanager: mplmanager.o
 $(FC) $(LFLAGS) $(FFLAGS) mplmanager.o $(FLIBS) \
 -o $(EXEDIR)/mplmanager

mplworker: mplworker.o
```

```
 $(FC) $(LFLAGS) $(FFLAGS) mplworker.o $(FLIBS) \
 -o $(EXEDIR)/mplworker

.f.o:
 $(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
 $(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
 rm -f core mpl *.o mpl*.*.out mpl*.*.err

new :
 rm -f core *.o mplmanager mplworker \
 mpl*.results mpl*.*.out mpl*.*.err *.cmd *.cmd.tmp
```

End: make.mpl

Program Listing: mplmanager.f

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program mplmanager
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     This is a trivial "hello world" program for illustrating the
c     use of the SP2
c
c     This manager program spawns copies of the program 'worker'.
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      implicit none
      integer maxnworkers
c
c     The following integers are used in the MPL calls...
c
      integer i, j
      integer nprocs, nworkers
      integer nodeid, msgid, namelen, noderesult
      integer nbuf(4), dontcare, allgrp, hostnm_, rc
      character*7 my_hostname
c
c
c     Call mp_environ to return the # of nodes available (nprocs)...
c     and this node's id:
c
      call MP_ENVIRON(nprocs,nodeid)
c
c     Get manager hostname:
c
      rc = hostnm_(my_hostname)
c
c     Number of workers must be fixed in the case of MPL.
c     You can vary the number of processors with the environment
c     variable MP_PROCS
c
      nworkers = nprocs-1
c
c     Learn value of allgrp and dontcare, for use with broadcast.
c
      call MP_TASK_QUERY(nbuf, 4, 3)
```

```
      dontcare = nbuf(1)
      allgrp = nbuf(4)
c
c     Open the file for output
c
      open(21,file='mplmanager.results')
      write(6,100)
      write(21,100)
 100  format('------ MPL program: mplmanager ----- ',/)
c
c     Broadcast the manager hostname to all processors...
c
      call MP_BCAST(my_hostname,  7, 0, allgrp)
c
c     Send each processor an integer value to print out, modify and return.
c
      do 10 i = 1,nworkers
         call MP_BSEND(i, 4, i, 100, msgid)
 10   continue

c
c     Now, the manager receives and prints the results of the workers.
c
      do 20 i = 1, nworkers
        call MP_BRECV(noderesult, 4, i, 200, msgid)
        write(6,*) 'Result from task ', i, ': ',noderesult
        write(21,*) 'Result from task ', i, ': ',noderesult
 20   continue

c
c     Close the output file:
c

      close(21)

      stop
      end
```

End: mplmanager.f

Program Listing: mplworker.f

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program mplworker
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     This is the worker program corresponding to the manager program:
c
c     mplmanager.f
c
c     It receives an integer message from the manager, prints it,
c     modifies it by adding 100, and then sends the results back
c     to the manager.
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      implicit none
c
      integer nbuf(4), dontcare, allgrp
      integer nprocs, nodeid, msgid, i, hostnm_, rc
      character*7 my_hostname, manager_name
c
c     Call mp_environ to return this node's id:
c
      call MP_ENVIRON(nprocs,nodeid)
c
c     Learn value of allgrp and dontcare, for use with broadcast.
c
      call MP_TASK_QUERY(nbuf, 4, 3)
      dontcare = nbuf(1)
      allgrp = nbuf(4)
c
c     get node name:
c
      rc = hostnm_(my_hostname)
c
c     Participate in broadcast to get manager's hostname...
c
      call MP_BCAST(manager_name, 7, 0, allgrp)
c
c     Receive the start-up value from the manager...
c
      call MP_BRECV(i,4,0,100,msgid)
      write(6,100) nodeid, my_hostname, i, manager_name
 100  format('Stdout from node ',I9,' (',A,'):  Received ',I3,
```

```
     *                                       ' from host ', A ,'.')


c     Now, perform a trivial computation on the integer value:

      i = i + 100
c
c     Send the computed information back to the manager.
c
      call MP_SEND(i, 4, 0, 200, msgid)

      stop
      end
```

End: mplworker.f

C PVM/PVMe interactive example files

Program Listing: README.debug

```
* ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM *
--------------------------------------------------------------------------
|  Instructions for PVM(e) debugging:
|  To be used AFTER first running the interactive example as is (without debugging).
--------------------------------------------------------------------------

0) Edit the file manager.f so that the pvmfspawn command takes on the
   PVMDEBUG argument (i.e. replace PVMDEFAULT by PVMDEBUG).
   Without this change, the spawned tasks will not be debuggable.

1) Make sure you are in the ORNL PVM environment, type:

   prompt> startpvm

     (If pvm is your default environment, set in your .cshrc file,
      and you haven't switched to pvme, then you can skip the
      startpvm step.  See ../README.FIRST for more information.)

   In your copy of the sample directory, type:

   prompt> make -f make.pvm

   The makefile will direct the compilation and linking of the
   programs.  The executables are created in the current directory
   in this example, but this can be changed to the default executable
   directory, $HOME/pvm3/bin/RS6K, by editing the EXEDIR entry in
   the makefile.  It's highly recommended that you *first* run through
   the example "as is", and then experiment with modifications after
   you've confirmed that the basic example works for you.

2) Generate a hostfile for the interactive pool.  This can be easily
   done by running the command:

   prompt> set_pvm_pool

   In an interactive setting, this command will generate a hostfile
   (named "hostfile") which contains the 4 available interactive nodes,
   and uses the current working directory as the PVM working directory
   and executable path.  (The hostfile can later by modified to suit
```

```
   your needs.)

3) In the sample directory, start the PVM daemon on the nodes of the
   interactive pool by typing:

   prompt> pvmd3 hostfile &          <-- NOTE: the ampersand is important!
                                        If omitted, you must push the job
                                        to the background to continue.

4) Check the configuration through the pvm console:

   prompt> pvm       [RETURN]
   pvmd already running.
   pvm> conf         [RETURN]
   4 hosts, 1 data format
               HOST     DTID     ARCH   SPEED
   grand17.nist.gov    40000     RS6K    1000
   grand18.nist.gov    80000     RS6K    1000
   grand19.nist.gov    c0000     RS6K    1000
   grand20.nist.gov   100000     RS6K    1000
   pvm> quit         [RETURN]
   prompt>

  (Other responses, such as only one host being listed, are a sign that
   something has not gone right.  Please contact karin@cam.nist.gov.)

5) Run the program under the debugger by typing:

   prompt> debug manager

   You will be prompted to enter information about the location of your
   source files, and any global breakpoints you may want to set.
   For the provided example, enter a single period (meaning current directory)
   for the location of source files, and a null line for global breakpoints.

   Use the standard dbx commands (run, cont, list, stop at <linenum>, etc.)
   to examine the manager program.  When the pvmfspawn call is reached,
   for each spawned process, an X-window dbx session will be generated.
   Standard dbx commands may also be entered in these spawned dbx sessions.

6) To quit, enter quit in each spawned debug window, and quit in the manager
   debug window.

   Do NOT forget to HALT the pvm daemon (especially with PVMe sessions!)
```

```
   prompt> pvm    [RETURN]
   pvm> halt      [RETURN]
   prompt>

8) Clean up...  type:

   prompt> make -f make.pvm new

   This will remove the executable, object, output and error files,
   to restore the directory to its original state.


--------------------------------------------------------------------
|  Again,
|  please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------
```

End: README.debug

Program Listing: README.pvm

```
* ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM *
--------------------------------------------------------------------------
|  This directory contains a simple Fortran example for PVM and PVMe
--------------------------------------------------------------------------
|
|  Filename           Style   Description
|  --------           -----   -----------
|  manager.f          MPMD    A basic "hello world" type program
|  worker.f
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.pvm (this file) * ORNL PVM *
README.pvme
make.pvme
make.pvm
manager.f
worker.f

To interactively run the sample program under ** ORNL PVM ** :

0) If this is your first use of the interactive nodes, make sure you
   have read and followed the instructions in ../README.FIRST !!
   You will be asked to add lines to your .cshrc file, copy the
   sample directory to your own area,
      AND LOGIN TO ONE OF THE INTERACTIVE NODES (grand17-20)

1) After logging into an interactive node, switch to the ORNL PVM
   environment:

   prompt> startpvm

     (If pvm is your default environment, set in your .cshrc file,
      and you haven't switched to pvme, then you can skip the
      startpvm step.  See ../README.FIRST for more information.)
```

```
   In your copy of the sample directory, type:

   prompt> make -f make.pvm

   The makefile will direct the compilation and linking of the
   programs.  The executables are created in the current directory
   in this example, but this can be changed to the default executable
   directory, $HOME/pvm3/bin/RS6K, by editing the EXEDIR entry in
   the makefile.  It's highly recommended that you *first* run through
   the example "as is", and then experiment with modifications after
   you've confirmed that the basic example works for you.

2) Generate a hostfile for the interactive pool.  This can be easily
   done by running the command:

   prompt> set_pvm_pool

   In an interactive setting, this command will generate a hostfile
   (named "hostfile") which contains the 4 available interactive nodes,
   and uses the current working directory as the PVM working directory
   and executable path.  (The hostfile can later by modified to suit
   your needs.)

3) In the sample directory, start the PVM daemon on the nodes of the
   interactive pool by typing:

   prompt> pvmd3 hostfile &          <-- NOTE: the ampersand is important!
                                        If omitted, you must push the job
                                        to the background to continue.

4) Check the configuration through the pvm console:

   prompt> pvm       [RETURN]
   pvmd already running.
   pvm> conf         [RETURN]
   4 hosts, 1 data format
               HOST     DTID     ARCH   SPEED
   grand17.nist.gov    40000     RS6K    1000
   grand18.nist.gov    80000     RS6K    1000
   grand19.nist.gov    c0000     RS6K    1000
   grand20.nist.gov   100000     RS6K    1000
   pvm> quit         [RETURN]
   prompt>
```

```
  (Other responses, such as only one host being listed, are a sign that
   something has not gone right.  Please contact karin@cam.nist.gov.)

5) Run the program by typing:

   prompt> manager

   You will be prompted to enter the number of processes you wish
   to spawn.  (NOTE: although PVM does not limit the number of
   spawned processes, the dimension of the array of process ids
   does impose an effective limit. In this example, *the limit is 32*.)

   Run the program as many times as you like to verify the output....

6) Examine the results file generated by the manager program:

   prompt> cat manager.results

7) Upon completion, halt PVM:

   prompt> pvm    [RETURN]
   pvm> halt      [RETURN]
   prompt>

8) Clean up...  type:

   prompt> make -f make.pvm new

   This will remove the executable, object, output and error files,
   to restore the directory to its original state.


--------------------------------------------------------------------
|  Again,
|  please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------
```

End: README.pvm

Program Listing: README.pvme

```
* IBM PVMe * * IBM PVMe * * IBM PVMe * * IBM PVMe * * IBM PVMe * * IBM PVMe *
--------------------------------------------------------------------------
|  This directory contains a simple Fortran example for PVM and PVMe
--------------------------------------------------------------------------
|
|  Filename           Style   Description
|  --------           -----   -----------
|  manager.f          MPMD    A basic "hello world" type program
|  worker.f
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.pvm
README.pvme  (this file)  * IBM PVMe *
make.pvme
make.pvm
manager.f
worker.f

To interactively run the sample program under ** IBM's PVMe ** :

0) If this is your first use of the interactive nodes, make sure you
   have read and followed the instructions in ../README.FIRST !!
   You will be asked to add lines to your .cshrc file, copy the
   sample directory to your own area,
      AND LOGIN TO ONE OF THE INTERACTIVE NODES (grand17-20)

1) After logging into an interactive node, switch to the IBM PVMe
   environment:

   prompt> startpvme

     (If pvme is your default environment, set in your .cshrc file,
      and you haven't switched to pvm, then you can skip the
      startpvme step.  See ../README.FIRST for more information.)
```

```
   In your copy of the sample directory, type:

   prompt> make -f make.pvme

   The makefile will direct the compilation and linking of the
   programs.  The executables are created in the current directory
   in this example, but this can be changed to the default executable
   directory, $HOME/pvm3/bin/RS6K, by editing the EXEDIR entry in
   the makefile.  It's highly recommended that you *first* run through
   the example "as is", and then experiment with modifications after
   you've confirmed that the basic example works for you.

2) Unlike the ORNL PVM example, you need not create a hostfile.
   The Resource Manager handles the machine setup for you.
   At this point, you should should check the PVMEPATH environment
   variable and change it as necessary to point to the directory where
   your executables reside.   The default is $HOME/pvm3/bin/RS6K.
   For example, the command

   prompt> setenv PVMEPATH `pwd`

   will set the executable path to be the current directory
   (required for this example).

3) In the sample directory, start the PVM daemon on the nodes of the
   interactive pool by typing:

   prompt> pvmd3 &          <-- NOTE: the ampersand is important!
   - or -                       If omitted, you must push the job
   prompt> pvmd3e &             to the background to continue.

   ( pvmd3 and pvmd3e are aliased to the same command,
   so you may use whichever you prefer.)

   This (in effect) requests a single user "lock" on the switch.

   If the switch is available, you will see output from PVMD, and
   finally the ready message:

   PVMD: Ready for 4 hosts...

   If someone else has accessed the switch, you will be prompted to
   wait a while and retry.
```

```
   A similar will result if YOU have already accessed the switch.
   The error message

   PVMD: Error while allocating node , capacity has been exceeded.

   indicates that you already have accessed the switch, and must
   halt the daemon if you wish to restart it.


4) Check the configuration through the pvm console:

   prompt> pvm       [RETURN]
   pvmd already running.
   pvm> conf         [RETURN]
   4 hosts, 1 data format
                      HOST     DTID     ARCH   SPEED
                   grand17  fff0001     RS6K    1000
                   grand18  fff0002     RS6K    1000
                   grand19  fff0003     RS6K    1000
                   grand20  fff0004     RS6K    1000
   pvm> quit         [RETURN]
   prompt>

  (Other responses, such as only one host being listed, are a sign that
   something has not gone right.  Please contact karin@cam.nist.gov.)

5) Run the program by typing:

   manager

   You will be prompted to enter the number of processes you wish
   to spawn.

   *** IMPORTANT ***:  Since this is PVMe, there is a hard limit on
   the number of PVM processes, and this cannot exceed the number of
   processors in the virtual machine.  Therefore, you must enter a number
   ** less than or equal to 3 **; choosing a larger input value will
   result in failure of the program!

   Run the program as many times as you like to verify the output....

6) Examine the results file generated by the manager program:

   prompt> cat manager.results
```

```
7) Upon completion, halt PVMe:

**   This step is VERY IMPORTANT, since you must release the switch   **
**   before anyone else has access to it.  DO NOT FORGET THIS STEP!   **

   prompt> pvm    [RETURN]
   pvm> halt      [RETURN]


8) Clean up...  type:

   prompt> make -f make.pvme new

   This will remove the executable, object, output and error files,
   to restore the directory to its original state.


--------------------------------------------------------------------
|  Again,
|  please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------
```

End: README.pvme

Program Listing: make.pvm

```
#------------------------------------------------------------------------
# This makefile is used to compile the simple PVM sample on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.
ARCH = RS6K
PVM_ROOT = /usr/local/apps/pvm3
PVMLIB = $(PVM_ROOT)/lib/$(ARCH)/libgpvm3.a $(PVM_ROOT)/lib/$(ARCH)/libpvm3.a
PVMFLIB = $(PVM_ROOT)/lib/$(ARCH)/libfpvm3.a

FC = f77
FFLAGS= -g
CC = cc
CFLAGS= -g

# EXEDIR sets the directory where executables are put.
# Typically, executables are kept in the PVM default binary directory
# $(HOME)/pvm3/bin/RS6K,  or in current working directory.
# If you choose not to move executables to the default location,
# you must let PVM know where the executable reside through the 'ep='
# extension in the hostfile.
# See example and PVM documentation for details.

EXEDIR=$(HOME)/pvm3/bin/RS6K
EXEDIR=.

# LFLAGS gives paths to include for proper linking:

# FLIBS and CLIBS may give particular libraries required.
# (in this case, the pvm3 library and the Fortran pvm3 library)

FLIBS=$(PVMFLIB) $(PVMLIB)
CLIBS=$(PVMLIB)

# INCL gives required include paths.

INCL=-I$(PVM_ROOT)/include
```

```
#-------------------------------------------------------------------------
# This part is typically user-defined, giving the program names, etc.
#-------------------------------------------------------------------------

all: include manager worker

#
# Special consideration is given for the fact that xlf compiles do not
# search the include path for include files. So, the proper include file
# is linked into the source directory.  The fortran files are also touched
# to guarantee they are compiled with the proper header.
#
include:
 touch *.f
 ln -s -f $(PVM_ROOT)/include/fpvm3.h .

manager: manager.o
 $(FC) $(LFLAGS) $(FFLAGS) -o $(EXEDIR)/manager manager.o $(FLIBS)

worker: worker.o
 $(FC) $(LFLAGS) $(FFLAGS) -o $(EXEDIR)/worker worker.o $(FLIBS)

hostlist: hostfile
 cp hostfile $(EXEDIR)

.f.o:
 $(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
 $(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
 rm -f core $(EXEDIR)/manager.results $(EXEDIR)/manager*.*.out \
 $(EXEDIR)/manager*.*.err $(EXEDIR)/hostfile

new :
 rm -f core *.o $(EXEDIR)/manager $(EXEDIR)/worker *.results *.*.out \
 *.*.err $(EXEDIR)/hostfile *.cmd.tmp fpvm3.h
```

End: make.pvm

Program Listing: make.pvme

```
#-------------------------------------------------------------------------
# This makefile is used to compile the simple PVMe sample on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#-------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.

FC = xlf
FFLAGS= -O
CC = xlc
CFLAGS= -O

# EXEDIR sets the directory where executables are put.
# Typically, executables are kept in the PVM default binary directory
# $(HOME)/pvm3/bin/RS6K,  or in current working directory.
# If you choose not to move executables to the default location,
# you must let PVM know where the executable reside through the 'ep='
# extension in the hostfile.
# See example and PVM documentation for details.

EXEDIR=$(HOME)/pvm3/bin/RS6K
EXEDIR=.

# LFLAGS gives paths to include for proper linking:

LFLAGS=-L/usr/lpp/ssp/css/libtb2 -lcss

# FLIBS and CLIBS may give particular libraries required.
# (in this case, the pvm3 library and the Fortran pvm3 library
# and proper include path for linking PVMe programs)

FLIBS=-lpvm3 -lfpvm3
CLIBS=-lpvm3
INCLUDE=-bI:/usr/lib/pvm3e.exp

# INCL gives required include paths.

# INCL=
```

```
#--------------------------------------------------------------------------
# This part is typically user-defined, giving the program names, etc.
#--------------------------------------------------------------------------

all: include manager worker

#
# Special consideration is given for the fact that xlf compiles do not
# search the include path for include files. So, the proper include file
# is linked into the source directory.  The fortran files are also touched
# to guarantee they are compiled with the proper header.
#

include:
 touch *.f
 ln -s -f /usr/lpp/pvm3/lib/fpvm3.h .

manager: manager.o
 $(FC) $(LFLAGS) $(FFLAGS) manager.o $(FLIBS) -o $(EXEDIR)/manager \
 $(INCLUDE)


worker: worker.o
 $(FC) $(LFLAGS) $(FFLAGS) worker.o $(FLIBS) -o $(EXEDIR)/worker \
 $(INCLUDE)


.f.o:
 $(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
 $(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
 rm -f core $(EXEDIR)/manager.results $(EXEDIR)/manager*.*.out \
 $(EXEDIR)/manager*.*.err $(EXEDIR)/pvmnodelist pvmnodelist

new :
 rm -f core *.o $(EXEDIR)/manager $(EXEDIR)/worker $(EXEDIR)/*.results \
 $(EXEDIR)/*.*.out $(EXEDIR)/*.*.err $(EXEDIR)/pvmnodelist pvmnodelist\
 $(EXEDIR)/*.cmd $(EXECDIR)/*.cmd.tmp fpvm3.h
```

End: make.pvme

Program Listing: manager.f

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program manager
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     This is a trivial "hello world" program for illustrating the
c     use of the SP2
c
c     This manager program spawns copies of the program 'worker'.
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      implicit none
c
c     Note: full path is given in INCL compilation flag in makefile:
c
      include 'fpvm3.h'
      integer maxnprocs
      parameter (maxnprocs = 32)
      integer tids(maxnprocs)
c
c     The tids array is used by PVM to hold the identifiers for the
c     spawned worker programs.  The values are used to direct messages to
c     the workers in the pvmfsend command.
c
c     The following integers are used in the PVM calls...
      integer info, i
      integer nprocs
      integer hostid, fromid
      integer rc, bufid, hostnm_
      character*7 my_hostname
c
c     The following integer is used to store the message returned by the workers...
      integer noderesult
c
c     Enroll the manager program in PVM and return an id for this process...
c     read input to determine number of processes to create...
c

      call pvmfmytid(hostid)
c
c     Get manager hostname:
```

```
c
      rc = hostnm_(my_hostname)
c
c     Read from standard input the number of processes to create...
c
      write(6,*)'Enter the number of PVM processes to create:'
      write(6,*)'NOTE: for PVMe, must be < number of nodes in pool!'
      read(5,*)nprocs

c
c     Open the file for output
c
      open(21,file='manager.results')
      write(6,100)
      write(21,100)
 100  format('------ PVM program: manager ----- ',/)

c
c     Spawn nprocs worker programs. Their task ids will be placed in the
c     tids array for later use in message passing.
c
      call pvmfspawn('worker', PVMDEFAULT, '*', nprocs, tids, info)
      if (info .ne. nprocs) then
        write(6,*) 'Error in spawning; Return code is ', info
        call pvmfexit(rc)
        stop
      endif
c
c     multi-cast (broadcast) the manager hostname to all processors...
c
      call pvmfinitsend(0,bufid)
      call pvmfpack(STRING, my_hostname, 7, 1, rc)
      call pvmfmcast(nprocs, tids, 100, rc)
c
c     Send each processor an integer value to print out, modify and return.
c
      do 10 i = 1,nprocs
       call pvmfinitsend(0, bufid)
       call pvmfpack(INTEGER4, i, 1, 1, rc)
       call pvmfsend(tids(i), 200, rc)
 10   continue

c
c     Now, the manager receives and prints the results of the workers.
```

```
c
      do 20 i = 1, nprocs
        call pvmfrecv(-1, 300, bufid)
        call pvmfunpack(INTEGER4, fromid, 1, 1, rc)
        call pvmfunpack(INTEGER4, noderesult, 1, 1, rc)
        write(6,200) fromid, noderesult
        write(21,200) fromid, noderesult
 20   continue

200   format('Result from task ',I10,': ',I4)

c
c     Close the output file:
c
      close(21)

c
c     Call pvmfexit to terminate the PVM program gracefully...
c
      call pvmfexit(rc)

      stop
      end
```

End: manager.f

Program Listing: worker.f

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program worker
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     This is the worker program corresponding to the manager program:
c
c     manager.f
c
c     It receives an integer message from the manager, prints it,
c     modifies it by adding 100, and then sends the results back
c     to the manager.
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      implicit none
c
c     Note: full path is given in INCL compilation flag in makefile:
c
      include 'fpvm3.h'
      integer nodeid, hostid, bufid, hostnm_
      character*7 my_hostname, manager_name
c
      integer rc, i
c
c     Enroll this program in PVM and return an id for this process...
c
      call pvmfmytid(nodeid)
c
c     Get worker hostname:
c
      rc = hostnm_(my_hostname)
c
c     Determine manager's id so that this worker can direct messages there...
c
      call pvmfparent(hostid)
c
c     Receive a broadcast containing the manager's hostname:
c
      call pvmfrecv(hostid, 100, bufid)
      call pvmfunpack(STRING, manager_name, 7, 1, rc)
c
c     Receive (and unpack) the start-up information from the manager...
c
```

```
      call pvmfrecv(hostid, 200, bufid)
      call pvmfunpack(INTEGER4, i, 1, 1, rc)

      write(6,100) nodeid, my_hostname, i, manager_name
 100  format('Stdout from node ',I9,' (',A,'):  Received ',I3,
     *                                  ' from host ', A ,'.')

c     Now, perform a trivial computation on the integer value:

      i = i+100

c
c     Send the computed information back to the manager.
c
      call pvmfinitsend(0, bufid)
      call pvmfpack(INTEGER4, nodeid, 1, 1, rc)
      call pvmfpack(INTEGER4, i, 1, 1, rc)
      call pvmfsend(hostid, 300, rc)
c
c     Call pvmfexit to terminate the PVM program gracefully...
c
      call pvmfexit(rc)

      stop
      end
```

End: worker.f

D MPL batch example files (C)

Program Listing: README.mpl

```
* IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL *
--------------------------------------------------------------------------
|  This directory contains a simple C example for MPL
--------------------------------------------------------------------------
|
|  Filename           Style   Description
|  --------           -----   -----------
|  mplmanager.c       MPMD    A basic "hello world" type program
|  mplworker.c
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.mpl
make.mpl
mplmanager.c
mplworker.c
user.cleanup
user.setup

To run the sample program in batch under ** IBM MPL ** ,
making use of the batch job preparation utility xllcreate:

Set up your environment for xllcreate and copy the MPL example
(if not already done):

   1) a) Add the following lines to your .cshrc file:
         setenv LLC_ROOT /usr/local/apps/LLCREATE
         alias llcreate ${LLC_ROOT}/bin/llcreate
         alias xllcreate ${LLC_ROOT}/bin/xllcreate
      b) Remove from your .cshrc any old references to XL_ROOT and (x)loadlgen
      c) Source your .cshrc file so these settings take effect
         (* Don't forget this step ! *)

   2) Copy the sample directory to your own area:
```

```
      /usr/bin/cp -r /usr/local/doc/Examples/mpl/C ~/<your_directory>

      This will create a subdirectory under your home directory
      called <your_directory>.

   3) Set your DISPLAY environment variable if not already set.

Build and submit the example:

   4) In the directory ~/<your_directory>, type: make -f make.mpl

   5) Call up the xllcreate program: xllcreate

   6) Experiment with the entry panel, using the Help button to
      get started.  You may use "Help" throughout xllcreate as needed.

      For the program to properly execute, you will need to enter
      the following information:

        The program is MPL, and MPMD.
        The main executable is "mplmanager", and the secondary "mplworker".
        The program does not take arguments, nor does it read
          from standard input.
        The program does not explicitly open any input files.
        The user.setup and user.cleanup scripts are optional.

       *-------------------------------------------------------------*
       * NOTE: A short postscript document describing X-llcreate     *
       *       is available. See $LLC_ROOT/doc/LLC-doc.ps.           *
       *-------------------------------------------------------------*

   7) Create the command file with "Create *.cmd File"

   8) Submit the job!

Experiment with your own program...

   9) Assuming all goes well with this example, please feel free to try
      out the utility on your own favorite example.  Simply call up
      xllcreate from the directory containing the executable(s).

                              * * * * * * *
          Comments and suggestions will be greatly appreciated.
```

```
Feel free to call ext. 5119 or send e-mail to karin@cam.nist.gov.
```

End: README.mpl

Program Listing: make.mpl

```
#--------------------------------------------------------------------------
# This makefile is used to compile MPL examples on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#--------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.

FC = mpxlf
FFLAGS= -O
CC = mpcc
CFLAGS = -O

# EXEDIR sets the directory where executables are put.
# Some people put all PVM executables in a bin directory, but
# here they are just kept in the current working directory.

EXEDIR=.

# LFLAGS may give paths to include for proper linking.
# (none required for MPL programs)

# FLIBS and CLIBS may give particular libraries required.
# (none required for these programs)

# INCL gives required include paths.
# (none required in this case)

#--------------------------------------------------------------------------
# This part is typically user-defined, giving the program names, etc.
#--------------------------------------------------------------------------

all: mplmanager mplworker


mplmanager: mplmanager.o
 $(CC) $(LFLAGS)  mplmanager.o $(FLIBS) \
 -o $(EXEDIR)/mplmanager

mplworker: mplworker.o
```

```
 $(CC) $(LFLAGS)  mplworker.o $(FLIBS) \
 -o $(EXEDIR)/mplworker

.f.o:
 $(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
 $(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
 rm -f core mpl *.o mpl*.*.out mpl*.*.err

new :
 rm -f core *.o mplmanager mplworker \
 mpl*.results mpl*.*.out mpl*.*.err *.cmd *.cmd.tmp
```

End: make.mpl

Program Listing: mplmanager.c

```
/*
/---------------------------------------------------------------------------
/  File: mplmanager.c
/---------------------------------------------------------------------------
/
/     This is a trivial "hello world" program for illustrating the
/     use of the SP2
/
/     This manager program spawns copies of the program 'worker'.
/
/---------------------------------------------------------------------------
*/
/* Contact: Karin A. Remington, karin@cam.nist.gov */

#include<stdlib.h>
#include<stdio.h>
#include<malloc.h>

main(argc, argv)
int argc; char *argv[];
{
      int i;
      int noderesult;
      int type;
      int msgid;
      int nprocs, nworkers;
      int nodeid;
      int nbuf[4], dontcare, allgrp;
      char *my_hostname;
      FILE *out_file;
/*
/     Call mp_environ to return the # of nodes available (nprocs)...
/     and this node's id:
*/
      mpc_environ(&nprocs, &nodeid);
/*
/     Get manager hostname:
*/
      my_hostname = (char *)malloc(8);
      gethostname(my_hostname,7);
/*
```

```
/     Number of workers must be fixed in the case of MPL.
/     You can vary the number of processors with the environment
/     variable MP_PROCS
*/
      nworkers = nprocs-1;
/*
/     Learn value of allgrp and dontcare, for use with broadcast.
*/
      mpc_task_query(nbuf, 4, 3);
      dontcare = nbuf[0];
      allgrp = nbuf[3];
/*
/     Open the file for output
*/
      out_file = fopen("mplmanager.results","w");
      printf("----- MPL Program mplmanager ----\n");
      fprintf(out_file,"----- MPL Program mplmanager ----\n");
/*
/     Broadcast the manager hostname to all processors...
*/
      mpc_bcast(my_hostname,  7,  nodeid, allgrp);
/*
/     Send each processor an integer value to print out, modify and return.
*/
      for (i=1;i<=nworkers;i++){
         mpc_bsend(&i, 4, i, 100, msgid);
      };
/*
/     Now, the manager receives and prints the results of the workers.
*/

      type = 200;
      for (i=1;i<=nworkers;i++){
        mpc_brecv(&noderesult, 4, &i, &type, &msgid);
        printf("Result from task %d: %d\n",i,noderesult);
        fprintf(out_file,"Result from task %d: %d\n",i,noderesult);
      };
/*
/     Close the output file:
*/
      fclose(out_file);

}
```

End: mplmanager.c

Program Listing: mplworker.c

```
/*--------------------------------------------------------------------------
/  File: mplworker.c
/--------------------------------------------------------------------------
/
/     This is the worker program corresponding to the manager program:
/
/     mplmanager.f
/
/     It receives an integer message from the manager, prints it,
/     modifies it by adding 100, and then sends the results back
/     to the manager.
/--------------------------------------------------------------------------
*/
/* Contact: Karin A. Remington, karin@cam.nist.gov */

#include<stdlib.h>
#include<stdio.h>
#include<malloc.h>

main(argc, argv)
int argc; char *argv[];
{
      int nbuf[4], dontcare, allgrp;
      int nprocs, nodeid;
      int type;
      int hostid;
      int msgid;
      int i;
      char *my_hostname;
      char *manager_name;
/*
/     Call mp_environ to return this node's id:
*/
      mpc_environ(&nprocs, &nodeid);
/*
/     Learn value of allgrp and dontcare, for use with broadcast.
*/
      mpc_task_query(nbuf, 4, 3);
      dontcare = nbuf[0];
      allgrp = nbuf[3];
/*
```

```
/     Get worker hostname:
*/
      my_hostname = (char *)malloc(8);
      gethostname(my_hostname,7);
/*
/     Participate in broadcast to get manager's hostname...
*/
      manager_name = (char *)malloc(8);
      mpc_bcast(manager_name,  7, 0, allgrp);
/*
/     Receive the start-up value from the manager...
*/
      hostid = 0;
      type = 100;
      mpc_brecv(&i, 4, &hostid, &type, &msgid);
      printf("Stdout from node %d (%s):  Received i = %d from host %s\n",
                nodeid, my_hostname, i, manager_name);
/*
/     Now, perform a trivial computation on the integer value:
*/
      i = i + 100;
/*
/     Send the computed information back to the manager:
*/
      mpc_bsend(&i, 4, 0, 200, msgid);

}
```

End: mplworker.c

Program Listing: user.cleanup

```
#!/bin/ksh

echo "${0##*/}: The user.cleanup script is running in directory:  "
pwd
if [ "$CMD" = "pvm.script" ]
then
   echo "${0##*/}: Copying back pvm log file to ./pvmlog ... "
   cp /tmp/pvml.$MYUID $INITIALDIR/pvmlog
fi

#for host in $hostlist
#do
#   echo "Here is where I would copy back output from ${host} ..."
#done

echo "${0##*/}: Finishing the user.cleanup script."
```

End: user.cleanup

Program Listing: user.setup

```
#!/bin/ksh

echo "${0##*/}: Doing some mock setup work to test the user.setup facility..."
echo "${0##*/}: The setup script has the following variables set:"
echo "${0##*/}: USER: $USER"
echo "${0##*/}: TMPDIR: $TMPDIR"
echo "${0##*/}: hostlist:\n$hostlist"
echo "${0##*/}: The script is being executed in the following directory:"
pwd
echo "${0##*/}: Finishing the user.setup script."
```

End: user.setup

E MPL batch example files (Fortran)

Program Listing: README.mpl

```
* IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL * * IBM MPL *
--------------------------------------------------------------------------
|  This directory contains a simple Fortran example for MPL
--------------------------------------------------------------------------
|
|  Filename           Style   Description
|  --------           -----   -----------
|  mplmanager.f       MPMD    A basic "hello world" type program
|  mplworker.f
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.mpl
make.mpl
mplmanager.f
mplworker.f
user.cleanup
user.setup

To run the sample program in batch under ** IBM MPL ** ,
making use of the batch job preparation utility xllcreate:

Set up your environment for xllcreate and copy the MPL example
(if not already done):

   1) a) Add the following lines to your .cshrc file:
         setenv LLC_ROOT /usr/local/apps/LLCREATE
         alias llcreate ${LLC_ROOT}/bin/llcreate
         alias xllcreate ${LLC_ROOT}/bin/xllcreate
      b) Remove from your .cshrc any old references to XL_ROOT and (x)loadlgen
      c) Source your .cshrc file so these settings take effect
         (* Don't forget this step ! *)

   2) Copy the sample directory to your own area:
```

```
      /usr/bin/cp -r /usr/local/doc/Examples/mpl/Fortran ~/<your_directory>

      This will create a subdirectory under your home directory
      called <your_directory>.

   3) Set your DISPLAY environment variable if not already set.

Build and submit the example:

   4) In the directory ~/<your_directory>, type: make -f make.mpl

   5) Call up the xllcreate program: xllcreate

   6) Experiment with the entry panel, using the Help button to
      get started.  You may use "Help" throughout xllcreate as needed.

      For the program to properly execute, you will need to enter
      the following information:

        The program is MPL, and MPMD.
        The main executable is "mplmanager", and the secondary "mplworker".
        The program does not take arguments, nor does it read
          from standard input.
        The program does not explicitly open any input files.
        The user.setup and user.cleanup scripts are optional, but
           user.cleanup will copy back an output file you won't
           otherwise see.

       *-------------------------------------------------------------*
       * NOTE: A short postscript document describing X-llcreate    *
       *       is available. See $LLC_ROOT/doc/LLC-doc.ps.          *
       *-------------------------------------------------------------*

   7) Create the command file with "Create *.cmd File"

   8) Submit the job!

Experiment with your own program...

   9) Assuming all goes well with this example, please feel free to try
      out the utility on your own favorite example.  Simply call up
      xllcreate from the directory containing the executable(s).
```

```
                        * * * * * * *
        Comments and suggestions will be greatly appreciated.
    Feel free to call ext. 5119 or send e-mail to karin@cam.nist.gov.
```

End: README.mpl

Program Listing: make.mpl

```
#--------------------------------------------------------------------------
# This makefile is used to compile MPL examples on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#--------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.

FC = mpxlf
FFLAGS= -g
CC = mpcc
CFLAGS = -O

# EXEDIR sets the directory where executables are put.
# Some people put all PVM executables in a bin directory, but
# here they are just kept in the current working directory.

EXEDIR=.

# LFLAGS may give paths to include for proper linking.
# (none required for MPL programs)

# FLIBS and CLIBS may give particular libraries required.
# (none required for these programs)

# INCL gives required include paths.
# (none required in this case)

#--------------------------------------------------------------------------
# This part is typically user-defined, giving the program names, etc.
#--------------------------------------------------------------------------

all: mplmanager mplworker


mplmanager: mplmanager.o
 $(FC) $(LFLAGS) $(FFLAGS) mplmanager.o $(FLIBS) \
 -o $(EXEDIR)/mplmanager

mplworker: mplworker.o
```

```
 $(FC) $(LFLAGS) $(FFLAGS) mplworker.o $(FLIBS) \
 -o $(EXEDIR)/mplworker

.f.o:
 $(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
 $(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
 rm -f core mpl *.o mpl*.*.out mpl*.*.err

new :
 rm -f core *.o mplmanager mplworker \
 mpl*.results mpl*.*.out mpl*.*.err *.cmd *.cmd.tmp
```

End: make.mpl

Program Listing: mplmanager.f

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program mplmanager
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     This is a trivial "hello world" program for illustrating the
c     use of the SP2
c
c     This manager program spawns copies of the program 'worker'.
c
c     Contact: Karin A. Remington, karin@cam.nist.gov
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      implicit none
      integer maxnworkers
c
c     The following integers are used in the MPL calls...
c
      integer i, j
      integer nprocs, nworkers
      integer nodeid, msgid, namelen, noderesult
      integer nbuf(4), dontcare, allgrp, hostnm_, rc
      character*7 my_hostname
c
c
c     Call mp_environ to return the # of nodes available (nprocs)...
c     and this node's id:
c
      call MP_ENVIRON(nprocs,nodeid)
c
c     Get manager hostname:
c
      rc = hostnm_(my_hostname)
c
c     Number of workers must be fixed in the case of MPL.
c     You can vary the number of processors with the environment
c     variable MP_PROCS
c
      nworkers = nprocs-1
c
c     Learn value of allgrp and dontcare, for use with broadcast.
c
```

```
      call MP_TASK_QUERY(nbuf, 4, 3)
      dontcare = nbuf(1)
      allgrp = nbuf(4)
c
c     Open the file for output
c
      open(21,file='mplmanager.results')
      write(6,100)
      write(21,100)
 100  format('------ MPL program: mplmanager ----- ',/)
c
c     Broadcast the manager hostname to all processors...
c
      call MP_BCAST(my_hostname,  7, 0, allgrp)
c
c     Send each processor an integer value to print out, modify and return.
c
      do 10 i = 1,nworkers
         call MP_BSEND(i, 4, i, 100, msgid)
 10   continue

c
c     Now, the manager receives and prints the results of the workers.
c
      do 20 i = 1, nworkers
        call MP_BRECV(noderesult, 4, i, 200, msgid)
        write(6,*) 'Result from task ', i, ': ',noderesult
        write(21,*) 'Result from task ', i, ': ',noderesult
 20   continue

c
c     Close the output file:
c

      close(21)

      stop
      end
```

End: mplmanager.f

Program Listing: mplworker.f

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program mplworker
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     This is the worker program corresponding to the manager program:
c
c     mplmanager.f
c
c     It receives an integer message from the manager, prints it,
c     modifies it by adding 100, and then sends the results back
c     to the manager.
c
c     Contact: Karin A. Remington, karin@cam.nist.gov
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      implicit none
c
      integer nbuf(4), dontcare, allgrp
      integer nprocs, nodeid, msgid, i, hostnm_, rc
      character*7 my_hostname, manager_name
c
c     Call mp_environ to return this node's id:
c
      call MP_ENVIRON(nprocs,nodeid)
c
c     Learn value of allgrp and dontcare, for use with broadcast.
c
      call MP_TASK_QUERY(nbuf, 4, 3)
      dontcare = nbuf(1)
      allgrp = nbuf(4)
c
c     Get worker hostname:
c
      rc = hostnm_(my_hostname)
c
c     Participate in broadcast to get manager's hostname...
c
      call MP_BCAST(manager_name, 7, 0, allgrp)
c
c     Receive the start-up value from the manager...
c
      call MP_BRECV(i,4,0,100,msgid)
```

```
      write(6,100) nodeid, my_hostname, i, manager_name
 100  format('Stdout from node ',I9,' (',A,'):  Received i = ',I3,
     *                                        ' from host ', A ,'.')


c     Now, perform a trivial computation on the integer value:

      i = i + 100
c
c     Send the computed information back to the manager.
c
      call MP_SEND(i, 4, 0, 200, msgid)

      stop
      end
```

End: mplworker.f

Program Listing: user.cleanup

```
#!/bin/ksh

echo "${0##*/}: The user.cleanup script is running in directory:  "
pwd

if [ "$CMD" = "pvm.script" ]
then
   echo "${0##*/}: Copying back pvm log file to ./pvmlog ... "
   cp /tmp/pvml.$MYUID $INITIALDIR/pvmlog
fi

#for host in $hostlist
#do
#   echo "Here is where I would copy back output from ${host} ..."
#done

echo "${0##*/}: Finishing the user.cleanup script."
```

End: user.cleanup

Program Listing: user.setup

```
#!/bin/ksh

echo "${0##*/}: Doing some mock setup work to test the user.setup facility..."
echo "${0##*/}: The setup script has the following variables set:"
echo "${0##*/}: USER: $USER"
echo "${0##*/}: TMPDIR: $TMPDIR"
echo "${0##*/}: hostlist:\n$hostlist"
echo "${0##*/}: The setup script is being executed in the following directory:"
pwd
echo "${0##*/}: Finishing the user.setup script."
```

End: user.setup

F PVM/PVMe batch example files (C)

Program Listing: README.pvm

```
* ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM *
--------------------------------------------------------------------------
|  This directory contains a simple C example for PVM and PVMe
--------------------------------------------------------------------------
|
|  Filename           Style   Description
|  --------           -----   -----------
|  manager.c          MPMD    A basic "hello world" type program
|  worker.c
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.pvm (this file) * ORNL PVM *
README.pvme
make.pvme
make.pvm
manager.c
user.cleanup
user.setup
worker.c

To run the sample program in batch under ** ORNL PVM ** ,
making use of the batch job preparation utility xllcreate:

Set up your environment for xllcreate and copy the PVM/PVMe example
(if not already done):

   1) a) Add the following lines to your .cshrc file:
         setenv LLC_ROOT /usr/local/apps/LLCREATE
         alias llcreate ${LLC_ROOT}/bin/llcreate
         alias xllcreate ${LLC_ROOT}/bin/xllcreate
      b) Remove from your .cshrc any old references to XL_ROOT and (x)loadlgen
      c) Source your .cshrc file so these settings take effect
         (* Don't forget this step ! *)
```

```
   2) Copy the sample directory to your own area:

      /usr/bin/cp -r /usr/local/doc/Examples/pvm+pvme/C ~/<your_directory>

      This will create a subdirectory under your home directory
      called <your_directory>.

   3) Set your DISPLAY environment variable if not already set.

Build and submit the example:

   4) In the directory ~/<your_directory>, type: make -f make.pvm

   5) Call up the xllcreate program: xllcreate

   6) Experiment with the entry panel, using the Help button to
      get started.  You may use "Help" throughout xllcreate as needed.

      For the program to properly execute, you will need to enter
      the following information:

        The program is PVM, and MPMD.
        The main executable is "manager", and the secondary "worker".
        The program takes a single argument, giving the number of
          processes to spawn.
        The program does not explicitly open any input files.
        The user.setup and user.cleanup scripts are optional, but
           user.cleanup will copy back an output file you won't
           otherwise see, as well as the pvm log file.

       *-------------------------------------------------------------*
       * NOTE: A short postscript document describing X-llcreate    *
       *       is available. See $LLC_ROOT/doc/LLC-doc.ps.          *
       *-------------------------------------------------------------*

   7) Create the command file with "Create *.cmd File"

   8) Submit the job!

Experiment with your own program...

   9) Assuming all goes well with this example, please feel free to try
      out the utility on your own favorite example.  Simply call up
```

```
xllcreate from the directory containing the executable(s).


                        * * * * * * *
         Comments and suggestions will be greatly appreciated.
    Feel free to call ext. 5119 or send e-mail to karin@cam.nist.gov.
```

End: README.pvm

Program Listing: README.pvme

```
* IBM PVMe * * IBM PVMe * * IBM PVMe * * IBM PVMe * * IBM PVMe * * IBM PVMe *
--------------------------------------------------------------------------
|  This directory contains a simple C example for PVM and PVMe
--------------------------------------------------------------------------
|
|  Filename           Style   Description
|  --------           -----   -----------
|  manager.c          MPMD    A basic "hello world" type program
|  worker.c
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.pvm
README.pvme  (this file) * IBM PVMe *
make.pvme
make.pvm
manager.c
user.cleanup
user.setup
worker.c

To run the sample program in batch under ** IBM PVMe ** ,
making use of the batch job preparation utility xllcreate:

Set up your environment for xllcreate and copy the PVM/PVMe example
(if not already done):

   1) a) Add the following lines to your .cshrc file:
         setenv LLC_ROOT /usr/local/apps/LLCREATE
         alias llcreate ${LLC_ROOT}/bin/llcreate
         alias xllcreate ${LLC_ROOT}/bin/xllcreate
      b) Remove from your .cshrc any old references to XL_ROOT and (x)loadlgen
      c) Source your .cshrc file so these settings take effect
         (* Don't forget this step ! *)

   2) Copy the sample directory to your own area:
```

```
      /usr/bin/cp -r /usr/local/doc/Examples/pvm+pvme/C ~/<your_directory>

      This will create a subdirectory under your home directory
      called <your_directory>.

   3) Set your DISPLAY environment variable if not already set.

Build and submit the example:

   4) In the directory ~/<your_directory>, type: make -f make.pvme

   5) Call up the xllcreate program: xllcreate

   6) Experiment with the entry panel, using the Help button to
      get started.  You may use "Help" throughout xllcreate as needed.

      For the program to properly execute, you will need to enter
      the following information:

        The program is PVMe, and MPMD.
        The main executable is "manager", and the secondary "worker".
        The program takes a single argument,  giving the number of
          processes to spawn.  THIS ARGUMENT MUST BE LESS THAN THE
          NUMBER OF NODES IN THE BATCH POOL!)
        The program does not explicitly open any input files.
        The user.setup and user.cleanup scripts are optional, but
           user.cleanup will copy back an output file you won't
           otherwise see.

       *-------------------------------------------------------------*
       * NOTE: A short postscript document describing X-llcreate    *
       *       is available. See $LLC_ROOT/doc/LLC-doc.ps.          *
       *-------------------------------------------------------------*

   7) Create the command file with "Create *.cmd File"

   8) Submit the job!

Experiment with your own program...

   9) Assuming all goes well with this example, please feel free to try
      out the utility on your own favorite example.  Simply call up
      xllcreate from the directory containing the executable(s).
```

```
                              * * * * * * *
          Comments and suggestions will be greatly appreciated.
     Feel free to call ext. 5119 or send e-mail to karin@cam.nist.gov.
```

End: README.pvme

Program Listing: make.pvm

```
#--------------------------------------------------------------------------
# This makefile is used to compile the simple PVM sample on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#--------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.
ARCH = RS6K
PVM_ROOT = /usr/local/apps/pvm3
PVMLIB = $(PVM_ROOT)/lib/$(ARCH)/libgpvm3.a $(PVM_ROOT)/lib/$(ARCH)/libpvm3.a
PVMFLIB = $(PVM_ROOT)/lib/$(ARCH)/libfpvm3.a

FC = f77
FFLAGS= -O
CC = cc
CFLAGS= -g

# EXEDIR sets the directory where executables are put.
# Typically, executables are kept in the PVM default binary directory
# $(HOME)/pvm3/bin/RS6K,  or in current working directory.
# If you choose not to move executables to the default location,
# you must let PVM know where the executable reside through the 'ep='
# extension in the hostfile.
# See example and PVM documentation for details.

EXEDIR=$(HOME)/pvm3/bin/RS6K
EXEDIR=.

# LFLAGS gives paths to include for proper linking:

# FLIBS and CLIBS may give particular libraries required.
# (in this case, the pvm3 library and the Fortran pvm3 library)

FLIBS=$(PVMFLIB) $(PVMLIB)
CLIBS=$(PVMLIB)

# INCL gives required include paths.

INCL=-I$(PVM_ROOT)/include
```

```
#-------------------------------------------------------------------------
# This part is typically user-defined, giving the program names, etc.
#-------------------------------------------------------------------------

all: manager worker

manager: manager.o
 $(CC) $(LFLAGS) $(CFLAGS) -o $(EXEDIR)/manager manager.o $(CLIBS)

worker: worker.o
 $(CC) $(LFLAGS) $(CFLAGS) -o $(EXEDIR)/worker worker.o $(CLIBS)

hostlist: hostfile
 cp hostfile $(EXEDIR)

.f.o:
 $(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
 $(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
 rm -f core $(EXEDIR)/manager.results $(EXEDIR)/manager*.*.out \
 $(EXEDIR)/manager*.*.err $(EXEDIR)/hostfile

new :
 rm -f core *.o $(EXEDIR)/manager $(EXEDIR)/worker *.results *.*.out \
 *.*.err $(EXEDIR)/hostfile manager.cmd *.cmd.tmp fpvm3.h pvmlog
```

End: make.pvm

Program Listing: make.pvme

```
#------------------------------------------------------------------------
# This makefile is used to compile the simple PVMe sample on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.

FC = xlf
FFLAGS= -O
CC = xlc
CFLAGS= -O

# EXEDIR sets the directory where executables are put.
# Typically, executables are kept in the PVM default binary directory
# $(HOME)/pvm3/bin/RS6K,  or in current working directory.
# If you choose not to move executables to the default location,
# you must let PVM know where the executable reside through the 'ep='
# extension in the hostfile.
# See example and PVM documentation for details.

EXEDIR=$(HOME)/pvm3/bin/RS6K
EXEDIR=.

# LFLAGS gives paths to include for proper linking:

LFLAGS=-L/usr/lpp/ssp/css/libtb2 -lcss

# FLIBS and CLIBS may give particular libraries required.
# (in this case, the pvm3 library and the Fortran pvm3 library
# and proper include path for linking PVMe programs)

FLIBS=-lpvm3 -lfpvm3 -bI:/usr/lib/pvm3e.exp
CLIBS=-lpvm3 -bI:/usr/lib/pvm3e.exp

# INCL gives required include paths.

# INCL=

#------------------------------------------------------------------------
```

```
# This part is typically user-defined, giving the program names, etc.
#--------------------------------------------------------------------------

all: manager worker

manager: manager.o
 $(CC) $(LFLAGS) $(CFLAGS) -o $(EXEDIR)/manager manager.o $(CLIBS)


worker: worker.o
 $(CC) $(LFLAGS) $(CFLAGS) -o $(EXEDIR)/worker worker.o $(CLIBS)


.f.o:
 $(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
 $(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
 rm -f core $(EXEDIR)/manager.results $(EXEDIR)/manager*.*.out \
 $(EXEDIR)/manager*.*.err $(EXEDIR)/pvmnodelist pvmnodelist

new :
 rm -f core *.o $(EXEDIR)/manager $(EXEDIR)/worker $(EXEDIR)/*.results \
 $(EXEDIR)/*.*.out $(EXEDIR)/*.*.err $(EXEDIR)/pvmnodelist pvmnodelist\
 $(EXEDIR)/*.cmd $(EXECDIR)/*.cmd.tmp fpvm3.h
```

End: make.pvme

Program Listing: manager.c

```
/*---------------------------------------------------------------------------
/  File: manager.c
/---------------------------------------------------------------------------
/
/     This is a trivial "hello world" program for illustrating the
/     use of the SP2
/
/     One argument is used, to tell the number of processes to create.
/
/  ** Warning:  For PVMe, the argument must be less that the number
/               of available processors.
/
/     This manager program spawns copies of the program 'worker'.
/
/---------------------------------------------------------------------------
*/

/* Contact: Karin A. Remington, karin@cam.nist.gov */

/*
/  Note: full path to pvm3.h is given in INCL compilation flag in makefile:
*/
#include "pvm3.h"

#include<stdlib.h>
#include<stdio.h>
#include<malloc.h>

main(argc, argv)
int argc; char *argv[];
{
      int *tid;
      int i;
      int noderesult;
      int type;
      int hostid, nodeid, msgid, rc;
      int nprocs, nworkers, narchs;
      struct hostinfo *hosts;
      FILE *out_file;
      char *my_hostname;
```

```
      nworkers = atoi(argv[1]);
/*
/     Enroll the manager program in PVM and return an id for this process...
/     and call pvmfconfig to return the # of nodes available (nprocs)...
*/
      hostid = pvm_mytid();
      pvm_config(&nprocs, &narchs, &hosts);
      if ( nworkers > nprocs - 1 ) {
         printf("Warning: If running under PVMe, too many workers have been requested.\n");
      }
/*
/     Allocate space for the array of task ids:
*/
      tid = (int *)malloc(nworkers*sizeof(int));
/*
/     Get manager hostname:
*/
      my_hostname = (char *)malloc(8);
      gethostname(my_hostname,7);
/*
/     Open the file for output
*/
      out_file = fopen("manager.results","w");
      printf("----- PVM Program manager ----\n");
      fprintf(out_file,"----- PVM Program manager ----\n");
/*
/     Spawn nworkers worker programs. Their task ids will be placed in the
/     tid array for later use in message passing.
*/
      rc = pvm_spawn("worker", (char**)0, PvmTaskDefault, "",
                                          nworkers, tid);

      if (rc != nworkers) {
         printf("Error in spawning;  Return code is %i\n",rc);
         pvm_exit();
         exit(1);
      }
/*
/     multi-cast (broadcast) the manager hostname to all processors...
*/
      pvm_initsend(0);
      pvm_pkstr(my_hostname);
      pvm_mcast(tid, nworkers, 100);
/*
```

```
/     Send each processor an integer value to print out, modify and return.
*/
      for (i=0;i<nworkers;i++){
         pvm_initsend(0);
         pvm_pkint(&i, 1, 1);
         pvm_send(tid[i], 200);
      };
/*
/     Now, the manager receives and prints the results of the workers.
*/
      for (i=0;i<nworkers;i++){
         pvm_recv(-1, 300);
         pvm_upkint(&nodeid, 1, 1);
         pvm_upkint(&noderesult, 1, 1);
         printf("Result from task %d: %d\n",nodeid,noderesult);
         fprintf(out_file,"Result from task %d: %d\n",nodeid,noderesult);
      };

/*
/     Close the output file:
*/
      fclose(out_file);

/*
/     Call pvmfexit to terminate the PVM program gracefully...
*/
      pvm_exit();

}
```

End: manager.c

Program Listing: user.cleanup

```
#!/bin/ksh

echo "${0##*/}: The user.cleanup script is running in directory:  "
pwd
echo "${0##*/}: Copying back results file from master PVM process ..."
cp $TMPDIR/manager.results $INITIALDIR/manager.results
if [ "$CMD" = "pvm.script" ]
then
   echo "${0##*/}: Copying back pvm log file to ./pvmlog ... "
   cp /tmp/pvml.$MYUID $INITIALDIR/pvmlog
fi

#for host in $hostlist
#do
#   echo "Here is where I would copy back output from ${host} ..."
#done
#echo "Finishing the user.cleanup script."
```

End: user.cleanup

Program Listing: user.setup

```
#!/bin/ksh

echo "${0##*/}: Doing some mock setup work to test the user.setup facility..."
echo "${0##*/}: The setup script has the following variables set:"
echo "${0##*/}: USER: $USER"
echo "${0##*/}: TMPDIR: $TMPDIR"
echo "${0##*/}: hostlist:\n$hostlist"
echo "${0##*/}: The script is being executed in the following directory:"
pwd
echo "${0##*/}: Finishing the user.setup script."
```

End: user.setup

Program Listing: worker.c

```
/*---------------------------------------------------------------------------
/  File: worker.c
/---------------------------------------------------------------------------
/
/     This is the worker program corresponding to the manager program:
/
/     manager.c
/
/     It receives an integer message from the manager, prints it,
/     modifies it by adding 100, and then sends the results back
/     to the manager.
/---------------------------------------------------------------------------
*/

/* Contact: Karin A. Remington, karin@cam.nist.gov */

/*
/  Note: full path to pvm3.h is given in INCL compilation flag in makefile:
*/
#include "pvm3.h"

#include<stdlib.h>
#include<stdio.h>
#include<malloc.h>

main(argc, argv)
int argc; char *argv[];
{
      int *tid;
      int i;
      int noderesult;
      int type;
      int nodeid, hostid, msgid;
      int nprocs, nworkers, narchs;
      struct hostinfo *hosts;
      FILE *out_file;
      char *my_hostname, *manager_name;
/*
/     Enroll the worker program in PVM and return an id for this process...
*/
      nodeid = pvm_mytid();
```

```
/*
/     Determine manager's id so that this worker can direct messages there...
*/
      hostid =  pvm_parent();
/*
/     Get worker hostname:
*/
      my_hostname = (char *)malloc(8);
      gethostname(my_hostname,7);
      manager_name = (char *)malloc(8);
/*
/     Receive a broadcast containing the manager's hostname:
*/
      pvm_recv(hostid, 100);
      pvm_upkstr(manager_name);
/*
/     Receive (and unpack) the start-up information from the manager...
*/
      pvm_recv(hostid, 200);
      pvm_upkint(&i, 1, 1);
      printf("Stdout from node %d (%s):  Received i = %d from host %s\n",
                nodeid, my_hostname, i, manager_name);
/*
/     Now, perform a trivial computation on the integer value:
*/
      i = i+100;
/*
/     Send the computed information back to the manager.
*/
      pvm_initsend(0);
      pvm_pkint(&nodeid, 1, 1);
      pvm_pkint(&i, 1, 1);
      pvm_send(hostid, 300);
/*
/     Call pvmfexit to terminate the PVM program gracefully...
*/
      pvm_exit();

}
```

End: worker.c

G PVM/PVMe batch example files (Fortran)

Program Listing: README.pvm

```
* ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM *
--------------------------------------------------------------------------
|  This directory contains a simple Fortran example for PVM and PVMe
--------------------------------------------------------------------------
|
|  Filename           Style   Description
|  --------           -----   -----------
|  manager.f          MPMD    A basic "hello world" type program
|  worker.f
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.pvm (this file) * ORNL PVM *
README.pvme
make.pvme
make.pvm
manager.f
manager.input.pvm
manager.input.pvme
user.cleanup
user.setup
worker.f

To run the sample program in batch under ** ORNL PVM ** ,
making use of the batch job preparation utility xllcreate:

Set up your environment for xllcreate and copy the PVM/PVMe example
(if not already done):

   1) a) Add the following lines to your .cshrc file:
         setenv LLC_ROOT /usr/local/apps/LLCREATE
         alias llcreate ${LLC_ROOT}/bin/llcreate
         alias xllcreate ${LLC_ROOT}/bin/xllcreate
      b) Remove from your .cshrc any old references to XL_ROOT and (x)loadlgen
```

```
   c) Source your .cshrc file so these settings take effect
      (* Don't forget this step ! *)

2) Copy the sample directory to your own area:

   /usr/bin/cp -r /usr/local/doc/Examples/pvm+pvme/Fortran ~/<your_directory>

   This will create a subdirectory under your home directory
   called <your_directory>.

3) Set your DISPLAY environment variable if not already set.

Build and submit the example:

4) In the directory ~/<your_directory>, type: make -f make.pvm

5) Call up the xllcreate program: xllcreate

6) Experiment with the entry panel, using the Help button to
   get started.  You may use "Help" throughout xllcreate as needed.

   For the program to properly execute, you will need to enter
   the following information:

     The program is PVM, and MPMD.
     The main executable is "manager", and the secondary "worker".
     The program does not take arguments, but does read
       from standard input. The appropriate file to redirect
       is called "manager.input.pvm" (You can change the input
       value in manager.input.pvm.)
     The program does not explicitly open any input files.
     The user.setup and user.cleanup scripts are optional, but
        user.cleanup will copy back an output file you won't
        otherwise see, as well as the pvm log file.

    *-------------------------------------------------------------*
    * NOTE: A short postscript document describing X-llcreate    *
    *       is available. See $LLC_ROOT/doc/LLC-doc.ps.          *
    *-------------------------------------------------------------*

7) Create the command file with "Create *.cmd File"

8) Submit the job!
```

```
Experiment with your own program...

   9) Assuming all goes well with this example, please feel free to try
      out the utility on your own favorite example.  Simply call up
      xllcreate from the directory containing the executable(s).


                              * * * * * * *
          Comments and suggestions will be greatly appreciated.
     Feel free to call ext. 5119 or send e-mail to karin@cam.nist.gov.
```

End: README.pvm

Program Listing: README.pvme

```
* IBM PVMe * * IBM PVMe * * IBM PVMe * * IBM PVMe * * IBM PVMe * * IBM PVMe *
--------------------------------------------------------------------------
|  This directory contains a simple Fortran example for PVM and PVMe
--------------------------------------------------------------------------
|
|  Filename           Style   Description
|  --------           -----   -----------
|  manager.f          MPMD    A basic "hello world" type program
|  worker.f
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.pvm
README.pvme  (this file) * IBM PVMe *
make.pvme
make.pvm
manager.f
manager.input.pvm
manager.input.pvme
user.cleanup
user.setup
worker.f

To run the sample program in batch under ** IBM PVMe ** ,
making use of the batch job preparation utility xllcreate:

Set up your environment for xllcreate and copy the PVM/PVMe example
(if not already done):

   1) a) Add the following lines to your .cshrc file:
         setenv LLC_ROOT /usr/local/apps/LLCREATE
         alias llcreate ${LLC_ROOT}/bin/llcreate
         alias xllcreate ${LLC_ROOT}/bin/xllcreate
      b) Remove from your .cshrc any old references to XL_ROOT and (x)loadlgen
      c) Source your .cshrc file so these settings take effect
         (* Don't forget this step ! *)
```

```
   2) Copy the sample directory to your own area:

      /usr/bin/cp -r /usr/local/doc/Examples/pvm+pvme/Fortran ~/<your_directory>

      This will create a subdirectory under your home directory
      called <your_directory>.

   3) Set your DISPLAY environment variable if not already set.

Build and submit the example:

   4) In the directory ~/<your_directory>, type: make -f make.pvme

   5) Call up the xllcreate program: xllcreate

   6) Experiment with the entry panel, using the Help button to
      get started.  You may use "Help" throughout xllcreate as needed.

      For the program to properly execute, you will need to enter
      the following information:

        The program is PVMe, and MPMD.
        The main executable is "manager", and the secondary "worker".
        The program does not take arguments, but does read
          from standard input. The appropriate file to redirect
          is called "manager.input.pvme".  (You can change the input
          value in manager.input.pvme, but THE INPUT VALUE MUST BE
          LESS THAN THE NUMBER OF NODES IN THE BATCH POOL!)
        The program does not explicitly open any input files.
        The user.setup and user.cleanup scripts are optional, but
           user.cleanup will copy back an output file you won't
           otherwise see.

       *-------------------------------------------------------------*
       * NOTE: A short postscript document describing X-llcreate     *
       *       is available. See $LLC_ROOT/doc/LLC-doc.ps.           *
       *-------------------------------------------------------------*

   7) Create the command file with "Create *.cmd File"

   8) Submit the job!

Experiment with your own program...
```

```
9) Assuming all goes well with this example, please feel free to try
   out the utility on your own favorite example.  Simply call up
   xllcreate from the directory containing the executable(s).


                          * * * * * * *
       Comments and suggestions will be greatly appreciated.
  Feel free to call ext. 5119 or send e-mail to karin@cam.nist.gov.
```

End: README.pvme

Program Listing: make.pvm

```
#------------------------------------------------------------------------
# This makefile is used to compile the simple PVM sample on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.
ARCH = RS6K
PVM_ROOT = /usr/local/apps/pvm3
PVMLIB = $(PVM_ROOT)/lib/$(ARCH)/libgpvm3.a $(PVM_ROOT)/lib/$(ARCH)/libpvm3.a
PVMFLIB = $(PVM_ROOT)/lib/$(ARCH)/libfpvm3.a

FC = f77
FFLAGS= -O
CC = cc
CFLAGS= -O

# EXEDIR sets the directory where executables are put.
# Typically, executables are kept in the PVM default binary directory
# $(HOME)/pvm3/bin/RS6K,  or in current working directory.
# If you choose not to move executables to the default location,
# you must let PVM know where the executable reside through the 'ep='
# extension in the hostfile.
# See example and PVM documentation for details.

EXEDIR=$(HOME)/pvm3/bin/RS6K
EXEDIR=.

# LFLAGS gives paths to include for proper linking:

# FLIBS and CLIBS may give particular libraries required.
# (in this case, the pvm3 library and the Fortran pvm3 library)

FLIBS=$(PVMFLIB) $(PVMLIB)
CLIBS=$(PVMLIB)

# INCL gives required include paths.

INCL=-I$(PVM_ROOT)/include
```

```
#-------------------------------------------------------------------------
# This part is typically user-defined, giving the program names, etc.
#-------------------------------------------------------------------------

all: include manager worker

#
# Special consideration is given for the fact that xlf compiles do not
# search the include path for include files. So, the proper include file
# is linked into the source directory.  The fortran files are also touched
# to guarantee they are compiled with the proper header.
#
include:
	touch *.f
	ln -s -f $(PVM_ROOT)/include/fpvm3.h .

manager: manager.o
	$(FC) $(LFLAGS) $(FFLAGS) -o $(EXEDIR)/manager manager.o $(FLIBS)

worker: worker.o
	$(FC) $(LFLAGS) $(FFLAGS) -o $(EXEDIR)/worker worker.o $(FLIBS)

hostlist: hostfile
	cp hostfile $(EXEDIR)

.f.o:
	$(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
	$(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
	rm -f core $(EXEDIR)/manager.results $(EXEDIR)/manager*.*.out \
	$(EXEDIR)/manager*.*.err $(EXEDIR)/hostfile

new :
	rm -f core *.o $(EXEDIR)/manager $(EXEDIR)/worker *.results *.*.out \
	*.*.err $(EXEDIR)/hostfile *.cmd.tmp fpvm3.h manager.cmd pvmlog
```

End: make.pvm

Program Listing: make.pvme

```
#-------------------------------------------------------------------------
# This makefile is used to compile the simple PVMe sample on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#-------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.

FC = xlf
FFLAGS= -O
CC = xlc
CFLAGS= -O

# EXEDIR sets the directory where executables are put.
# Typically, executables are kept in the PVM default binary directory
# $(HOME)/pvm3/bin/RS6K,  or in current working directory.
# If you choose not to move executables to the default location,
# you must let PVM know where the executable reside through the 'ep='
# extension in the hostfile.
# See example and PVM documentation for details.

EXEDIR=$(HOME)/pvm3/bin/RS6K
EXEDIR=.

# LFLAGS gives paths to include for proper linking:

LFLAGS=-L/usr/lpp/ssp/css/libtb2 -lcss

# FLIBS and CLIBS may give particular libraries required.
# (in this case, the pvm3 library and the Fortran pvm3 library
# and proper include path for linking PVMe programs)

FLIBS=-lpvm3 -lfpvm3 -bI:/usr/lib/pvm3e.exp
CLIBS=-lpvm3 -bI:/usr/lib/pvm3e.exp

# INCL gives required include paths.

# INCL=

#-------------------------------------------------------------------------
```

```
# This part is typically user-defined, giving the program names, etc.
#------------------------------------------------------------------------

all: include manager worker

#
# Special consideration is given for the fact that xlf compiles do not
# search the include path for include files. So, the proper include file
# is linked into the source directory.  The fortran files are also touched
# to guarantee they are compiled with the proper header.
#

include:
	touch *.f
	ln -s -f /usr/lpp/pvm3/lib/fpvm3.h .

manager: manager.o
	$(FC) $(LFLAGS) $(FFLAGS) -o $(EXEDIR)/manager manager.o $(FLIBS)


worker: worker.o
	$(FC) $(LFLAGS) $(FFLAGS) -o $(EXEDIR)/worker worker.o $(FLIBS)


.f.o:
	$(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
	$(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
	rm -f core $(EXEDIR)/manager.results $(EXEDIR)/manager*.*.out \
	$(EXEDIR)/manager*.*.err $(EXEDIR)/pvmnodelist pvmnodelist

new :
	rm -f core *.o $(EXEDIR)/manager $(EXEDIR)/worker $(EXEDIR)/*.results \
	$(EXEDIR)/*.*.out $(EXEDIR)/*.*.err $(EXEDIR)/pvmnodelist pvmnodelist\
	$(EXEDIR)/*.cmd $(EXECDIR)/*.cmd.tmp fpvm3.h
```

End: make.pvme

Program Listing: manager.f

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program manager
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     This is a trivial "hello world" program for illustrating the
c     use of the SP2
c
c     This manager program spawns copies of the program 'worker'.
c
c     Contact: Karin A. Remington, karin@cam.nist.gov
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      implicit none
c
c     Note: full path is given in INCL compilation flag in makefile:
c
      include 'fpvm3.h'
      integer maxnprocs
      parameter (maxnprocs = 32)
      integer tids(maxnprocs)
c
c     The tids array is used by PVM to hold the identifiers for the
c     spawned worker programs.  The values are used to direct messages to
c     the workers in the pvmfsend command.
c
c     The following integers are used in the PVM calls...
      integer info, i
      integer nprocs
      integer hostid, fromid
      integer rc, bufid, hostnm_
      character*7 my_hostname
c
c     The following integer is used to store the message returned by the workers...
      integer noderesult
c
c     Enroll the manager program in PVM and return an id for this process...
c     read input to determine number of processes to create...
c

      call pvmfmytid(hostid)
c
```

```
c     Get manager hostname:
c
      rc = hostnm_(my_hostname)
c
c     Read from standard input the number of processes to create...
c
      write(6,*)'Enter the number of PVM processes to create:'
      write(6,*)'NOTE: for PVMe, must be < number of nodes in pool!'
      read(5,*)nprocs

c
c     Open the file for output
c
      open(21,file='manager.results')
      write(6,100)
      write(21,100)
 100  format('------ PVM program: manager ----- ',/)

c
c     Spawn nprocs worker programs. Their task ids will be placed in the
c     tids array for later use in message passing.
c
      call pvmfspawn('worker', PVMDEFAULT, '*', nprocs, tids, info)
      if (info .ne. nprocs) then
        write(6,*) 'Error in spawning; Return code is ', info
        call pvmfexit(rc)
        stop
      endif
c
c     multi-cast (broadcast) the manager hostname to all processors...
c
      call pvmfinitsend(0,bufid)
      call pvmfpack(STRING, my_hostname, 7, 1, rc)
      call pvmfmcast(nprocs, tids, 100, rc)
c
c     Send each processor an integer value to print out, modify and return.
c
      do 10 i = 1,nprocs
       call pvmfinitsend(0, bufid)
       call pvmfpack(INTEGER4, i, 1, 1, rc)
       call pvmfsend(tids(i), 200, rc)
 10   continue

c
```

```
c     Now, the manager receives and prints the results of the workers.
c
      do 20 i = 1, nprocs
        call pvmfrecv(-1, 300, bufid)
        call pvmfunpack(INTEGER4, fromid, 1, 1, rc)
        call pvmfunpack(INTEGER4, noderesult, 1, 1, rc)
        write(6,200) fromid, noderesult
        write(21,200) fromid, noderesult
 20   continue

200   format('Result from task ',I10,': ',I4)

c
c     Close the output file:
c
      close(21)

c
c     Call pvmfexit to terminate the PVM program gracefully...
c
      call pvmfexit(rc)

      stop
      end
```

End: manager.f

Program Listing: manager.input.pvm

```
15
```

End: manager.input.pvm

Program Listing: manager.input.pvme

```
4
```

End: manager.input.pvme

Program Listing: user.cleanup

```
#!/bin/ksh

echo "${0##*/}: The user.cleanup script is running in directory:  "
pwd

cp $TMPDIR/manager.results $INITIALDIR

if [ "$CMD" = "pvm.script" ]
then
   echo "${0##*/}: Copying back pvm log file to ./pvmlog ... "
   cp /tmp/pvml.$MYUID $INITIALDIR/pvmlog
fi

#for host in $hostlist
#do
#   echo "Here is where I would copy back output from ${host} ..."
#done
echo "${0##*/}: Finishing the user.cleanup script."
```

End: user.cleanup

Program Listing: user.setup

```
#!/bin/ksh

echo "${0##*/}: Doing some mock setup work to test the user.setup facility..."
echo "${0##*/}: The setup script has the following variables set:"
echo "${0##*/}: USER: $USER"
echo "${0##*/}: TMPDIR: $TMPDIR"
echo "${0##*/}: hostlist:\n$hostlist"
echo "${0##*/}: The setup script is being executed in the following directory:"
pwd
echo "${0##*/}: Finishing the user.setup script."
```

End: user.setup

Program Listing: worker.f

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program worker
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     This is the worker program corresponding to the manager program:
c
c     manager.f
c
c     It receives an integer message from the manager, prints it,
c     modifies it by adding 100, and then sends the results back
c     to the manager.
c
c     Contact: Karin A. Remington, karin@cam.nist.gov
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      implicit none
c
c     Note: full path is given in INCL compilation flag in makefile:
c
      include 'fpvm3.h'
      integer nodeid, hostid, bufid, hostnm_
      character*7 my_hostname, manager_name
c
      integer rc, i
c
c     Enroll this program in PVM and return an id for this process...
c
      call pvmfmytid(nodeid)
c
c     Get worker hostname:
c
      rc = hostnm_(my_hostname)
c
c     Determine manager's id so that this worker can direct messages there...
c
      call pvmfparent(hostid)
c
c     Receive a broadcast containing the manager's hostname:
c
      call pvmfrecv(hostid, 100, bufid)
      call pvmfunpack(STRING, manager_name, 7, 1, rc)
c
```

```
c     Receive (and unpack) the start-up information from the manager...
c
      call pvmfrecv(hostid, 200, bufid)
      call pvmfunpack(INTEGER4, i, 1, 1, rc)

      write(6,100) nodeid, my_hostname, i, manager_name
 100  format('Stdout from node ',I10,' (',A,'):  Received i = ',I3,
     *                                         ' from host ', A ,'.')

c     Now, perform a trivial computation on the integer value:

      i = i+100

c
c     Send the computed information back to the manager.
c
      call pvmfinitsend(0, bufid)
      call pvmfpack(INTEGER4, nodeid, 1, 1, rc)
      call pvmfpack(INTEGER4, i, 1, 1, rc)
      call pvmfsend(hostid, 300, rc)
c
c     Call pvmfexit to terminate the PVM program gracefully...
c
      call pvmfexit(rc)

      stop
      end
```

End: worker.f

H SPMD example (PVM/Fortran)

Program Listing: spmd.f

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program spmd
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     This is a trivial "hello world" program for illustrating an
c     SPMD programming style.
c
c     The first pvm task will spawn other copies of itself, and the
c     parent task will take the role of the 'manager' program.
c
c     Contact: Karin A. Remington, karin@cam.nist.gov
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      implicit none
c
c     Note: full path is given in INCL compilation flag in makefile:
c
      include 'fpvm3.h'
      integer maxnprocs
      parameter (maxnprocs = 32)
      integer tids(maxnprocs)
c
c     The tids array is used by PVM to hold the identifiers for the
c     spawned worker programs.  The values are used to direct messages to
c     the workers in the pvmfsend command.
c
c     The following integers are used in the PVM calls...
      integer info, i
      integer nprocs
      integer nodeid, parentid, fromid
      integer rc, bufid, hostnm_
      character*7 my_hostname, manager_name
c
c     The following integer is used to store the message returned by the workers...
      integer noderesult
c
c     Enroll the manager program in PVM and return an id for this process...
c     read input to determine number of processes to create...
c
```

```
      call pvmfmytid(nodeid)
c
c     Get hostname:
c
      rc = hostnm_(my_hostname)
c
c     Determine whether this is the first task by checking to see
c     whether it has a parent:
c
      call pvmfparent(parentid)
c
c     If parentid == PvmNoParent (-23), then this is the original task
c     which will act as the 'manager'; otherwise, the task will act as a
c     'worker'.
c
      if (parentid == PvmNoParent ) then
c
c        Read from standard input the number of processes to create...
c
         write(6,*)'Enter the number of PVM processes to create:'
         write(6,*)'NOTE: for PVMe, must be < number of nodes in pool!'
         read(5,*) nprocs
c
c        Open the file for output
c
         open(21,file='spmd.results')
         write(6,100)
         write(21,100)
 100     format('------ PVM program: spmd ----- ',/)

c
c        Spawn nprocs worker programs. Their task ids will be placed in
c        the tids array for later use in message passing.
c
         if ( nprocs > 0 ) then
c
c          Must ensure nprocs > 0 or spawn will fail.
c
           call pvmfspawn('spmd', PVMDEFAULT, '*', nprocs, tids, info)
           if (info .ne. nprocs) then
             write(6,*) 'Error in spawning; Return code is ', info
             call pvmfexit(rc)
             stop
```

```
          endif
c
c         multi-cast (broadcast) the manager hostname to all processors...
c
          call pvmfinitsend(0,bufid)
          call pvmfpack(STRING, my_hostname, 7, 1, rc)
          call pvmfmcast(nprocs, tids, 100, rc)
c
c         Send each processor an integer value to print out, modify and
c         return.
c
          do 10 i = 1,nprocs
            call pvmfinitsend(0, bufid)
            call pvmfpack(INTEGER4, i, 1, 1, rc)
            call pvmfsend(tids(i), 200, rc)
 10       continue
        endif
c
c       Now, this task can perform some work while the other tasks are
c       working (though in the trivial program, the work isn't too
c       meaningful.  This could normally serve to increase the number
c       of tasks by one.
c
        i = nprocs+1
        call worker(i, nodeid, my_hostname, my_hostname)
c
c       Write result from this computation:
c
        write(6,300) nodeid, i
        write(21,300) nodeid, i
c
c       Now, as manager, this process receives and prints the results
c       of the workers.
c
        do 20 i = 1, nprocs
          call pvmfrecv(-1, 300, bufid)
          call pvmfunpack(INTEGER4, fromid, 1, 1, rc)
          call pvmfunpack(INTEGER4, noderesult, 1, 1, rc)
          write(6,300) fromid, noderesult
          write(21,300) fromid, noderesult
 20     continue

 300    format('Result from task ',I10,': ',I4)
```

```
c
c        Close the output file:
c
         close(21)
c
c        End of the 'manager' part of the program...
c
      else
c
c        if (parentid != PvmNoParent ) then this process was spawned
c        by the 'manager' process, and will act as a worker.
c
c        Receive a broadcast containing the manager's hostname:
c
         call pvmfrecv(parentid, 100, bufid)
         call pvmfunpack(STRING, manager_name, 7, 1, rc)
c
c        Receive (and unpack) the start-up information from the manager...
c
         call pvmfrecv(parentid, 200, bufid)
         call pvmfunpack(INTEGER4, i, 1, 1, rc)

c
c        Now, do the work assigned:
c
         call worker(i, nodeid, my_hostname, manager_name)
c
c        Send the computed information back to the manager.
c
         call pvmfinitsend(0, bufid)
         call pvmfpack(INTEGER4, nodeid, 1, 1, rc)
         call pvmfpack(INTEGER4, i, 1, 1, rc)
         call pvmfsend(parentid, 300, rc)
c
c        End of the 'worker' part of the program...
c
      endif
c
c     Call pvmfexit to terminate the PVM program gracefully...
c     (manager and worker programs...)
c
      call pvmfexit(rc)

      stop
```

```
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine worker(i, nodeid, my_hostname, manager_name)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c         This is a trivial routine which only prints out the
c      item assigned by the 'manager' and then adds 100 to it.
c      In a more realistic setting, this routine would carry out
c      the useful work of the program.
c
c      Because the 'manager' program calls this routine, even if
c      no 'workers' are spawned, debugging of this part of the program
c      can be carried out in a serial setting.
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer i, nodeid

      character*7 my_hostname, manager_name

      write(6,100) nodeid, my_hostname, i, manager_name

 100  format('Stdout from node ',I10,' (',A,'):  Received i = ',
     *                                   I3,' from host ', A ,'.')

      i = i+100

      return
      end
```

End: spmd.f

I XPVM matrix transposition example (PVM/C and Fortran)

Program Listing: README.pvm

```
* ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM * * ORNL PVM *
--------------------------------------------------------------------------
|  This directory contains a mixed C/Fortran example for PVM
|  The SPMD program creates a distributed 3D matrix and calls
|  a Fortran subroutine to carry out a tranpose operation across
|  the processors.  This code also demonstrates the use of the
|  utility routine xpvm_connect() to connect this process to
|  an XPVM monitoring console, if one is running, and a simple
|  timer for measuring elapsed wall-clock time.
--------------------------------------------------------------------------
|
|  Filename:          Style   Description
|  --------           -----   -----------
|  setup.c            SPMD    Sets up 3D matrix and calls tranpose routine
|
|  Support routines:
|  --------                   -----------
|  transp.f                   Performs a 3D matrix transposition across
|                             two of the dimensions
|  transpold.f                Performs a 3D matrix transposition across
|                             two of the dimensions (naive implementation)
|  realtime.c                 Simple timer routine
|  user.cleanup               Script used to copy back trace and log files
|                             from batch session.
|  xpvmconnect.c              Utility subroutine for connecting a traced
|                             pvm program with the XPVM monitoring console.
|
--------------------------------------------------------------------------
|  Please direct any questions, comments or suggestions
|  to karin@cam.nist.gov.
--------------------------------------------------------------------------

This directory includes the following files:

README.pvm (this file) * ORNL PVM *
make.pvm
realtime.c
```

```
setup.c
transp.f
transpold.f
user.cleanup
xpvmconnect.c

------------------
BATCH use of XPVM:
------------------

To run the sample program in batch under ** ORNL PVM ** ,
making use of the batch job preparation utility xllcreate:

Set up your environment for xllcreate and copy the PVM example
(if not already done):

   1) a) Add the following lines to your .cshrc file:
            setenv LLC_ROOT /usr/local/apps/LLCREATE
            alias llcreate ${LLC_ROOT}/bin/llcreate
            alias xllcreate ${LLC_ROOT}/bin/xllcreate
      b) Remove from your .cshrc any old references to XL_ROOT and (x)loadlgen
      c) Source your .cshrc file so these settings take effect
            (* Don't forget this step ! *)

   2) Copy the sample directory to your own area:

      /usr/bin/cp -r /usr/local/doc/Examples/transp ~/<your_directory>

      This will create a subdirectory under your home directory
      called <your_directory>.

   3) Set your DISPLAY environment variable if not already set.

Build and submit the example:

   4) In the directory ~/<your_directory>, type: make -f make.pvm

   5) Call up the xllcreate program: xllcreate

   6) Experiment with the entry panel, using the Help button to
      get started.  You may use "Help" throughout xllcreate as needed.

      For the program to properly execute, you will need to enter
      the following information:
```

```
    The program is PVM, and SPMD.
    The program generates trace data (see pvm_spawn call in
       setup.c) and calls a routine to connect to XPVM if possible.

       To use XPVM, select the appropriate button, and make sure
       of the following BEFORE you submit your program:
          a) Your DISPLAY variable should be set within your
             .cshrc file, so all batch processes know where to
             display the XPVM console.
          b) Your console must allow the grand nodes permission
             to display.  Use 'xhosts +' to allow all node permission.

    The main executable is "setup" (no secondary program).
    The program takes two arguments, the first giving the dimension of
      the 3D matrix (it is assumed all dimensions are equal) and
      the second giving the number of processes to use. THE SECOND
      ARGUMENT MUST EVENLY DIVIDE THE FIRST, and BOTH ARGUMENTS MUST
      DIVISBLE BY 2.   (The latter requirement is due to the implemented
      transpose algorithm which performs simultaneous pairwise exchanges.
      To get around this  requirement, you may substitute transpold.f,
      which uses a less efficient  (but less restrictive) communication
      algorithm.)
    The program does not explicitly open any input files.
    The user.cleanup script is optional, but user.cleanup
       will copy back the pvm log file and the xpvm trace file.

   *-------------------------------------------------------------*
   * NOTE: A short postscript document describing X-llcreate    *
   *       is available. See $LLC_ROOT/doc/LLC-doc.ps.          *
   *-------------------------------------------------------------*

 7) Create the command file with "Create *.cmd File"

 8) Submit the job!


----------------------------------
XPVM on the interactive node pool:
----------------------------------


To use XPVM on the interactive node pool, first follow the intructions
for setting up an interactive PVM session.  It is probably easiest to
```

```
start XPVM  *after* you've successfully started PVM.

To proceed with XPVM:

1) Open TWO shell sessions (windows) running on one of the interactive nodes.

2) Check to ensure your DISPLAY is set and that the interactive grand node
has permission to open on your console.

3) Then, in one of the interactive shell sessions (windows),
execute /usr/local/bin/xpvm.  After it successfully starts, and
you've selected a tracefile (or accepted the default tracefile),
switch to the second window.

4) In the second interactive shell session (window), execute your program
as you normally would.  The included call to xpvm_connect() will cause
its trace information to be displayed in the XPVM display area.
When the program finished, you may switch to "playback" mode, and replay
the trace file and examine various "views" of the collected information.

5) When finished, you may choose to "quit" XPVM, which leaves the PVM
virtual machine intact, or "halt" XPVM, which "halts" the PVM daemon
as XPVM is exited.  You should choose "quit" if you intend to continue
using PVM after leaving XPVM, and "halt" if you are finished
with your PVM session.

                              * * * * * * *
          Comments and suggestions will be greatly appreciated.
     Feel free to call ext. 5119 or send e-mail to karin@cam.nist.gov.
```

End: README.pvm

Program Listing: make.pvm

```
#-------------------------------------------------------------------------
# This makefile is used to compile the simple PVM sample on the SP2.
# It can also be used as a template makefile for C or Fortran programs.
#-------------------------------------------------------------------------
#
# FC and CC specify the compilers to be used for Fortran and C.
# FFLAGS or CFLAGS can be set to -g if debugging is desired; the
# current -O setting allows compiler optimization.
ARCH = RS6K
PVM_ROOT = /usr/local/apps/pvm3
PVMLIB =  $(PVM_ROOT)/lib/$(ARCH)/libpvm3.a
PVMFLIB = $(PVM_ROOT)/lib/$(ARCH)/libfpvm3.a
PVMGLIB = $(PVM_ROOT)/lib/$(ARCH)/libgpvm3.a

FC = f77
FFLAGS= -O
CC = cc
CFLAGS= -O
CLOCK=realtime.o
XPVMC=xpvmconnect.o

# EXEDIR sets the directory where executables are put.
# Typically, executables are kept in the PVM default binary directory
# $(HOME)/pvm3/bin/RS6K,  or in current working directory.
# If you choose not to move executables to the default location,
# you must let PVM know where the executable reside through the 'ep='
# extension in the hostfile.
# See example and PVM documentation for details.

EXEDIR=$(HOME)/pvm3/bin/RS6K
EXEDIR=.

# LFLAGS gives paths to include for proper linking:
LFLAGS=-bI:/usr/lpp/xlf/lib/lowsys.exp

# FLIBS and CLIBS may give particular libraries required.
# (in this case, the pvm3 library and the Fortran pvm3 library)

FLIBS=$(PVMFLIB) $(PVMGLIB) $(PVMLIB)  /usr/lib/libxlf90.a
CLIBS=$(PVMGLIB) $(PVMLIB)
```

```
# INCL gives required include paths.

INCL=-I$(PVM_ROOT)/include

#-------------------------------------------------------------------------
# This part is typically user-defined, giving the program names, etc.
#-------------------------------------------------------------------------


all: setup

# setupold uses a more naive matrix transposition routine, found in transpold.f

setupold: setup.o transpold.o $(XPVMC) $(CLOCK)
 $(CC) $(LFLAGS) $(CFLAGS) -o $(EXEDIR)/setup setup.o transpold.o $(XPVMC) $(CLOCK) $(FLIBS)

setup: setup.o transp.o $(XPVMC) $(CLOCK)
 $(CC) $(LFLAGS) $(CFLAGS) -o $(EXEDIR)/setup setup.o transp.o $(XPVMC) $(CLOCK) $(FLIBS)

.f.o:
 $(FC) -c $(FFLAGS) $(INCL) $*.f
.c.o:
 $(CC) -c $(CFLAGS) $(INCL) $*.c

clean :
 rm -f core $(EXEDIR)/setup*.*.out \
 $(EXEDIR)/setup*.*.err

new :
 rm -f core *.o $(EXEDIR)/setup $(EXEDIR)/setupold  *.results *.*.out \
 *.*.err $(EXEDIR)/hostfile *.cmd.tmp fpvm3.h pvmlog* xpvm.trace*
```

End: make.pvm

Program Listing: realtime.c

```
/*
* realtime_():
*
* Returns the number of real time seconds from the last
* call to realtime_(); returns 0 on first call.
* May be called from FORTRAN as (double precision) realtime().
*
* Tested for SUN, ES1, SPS-2, RS-6000
*/

#ifdef SPS
#ifdef PAR
#include <time.h>
#else  PAR
#include <sys/time.h>
#endif PAR
#else  SPS
#include <sys/time.h>
#endif SPS

#define tim(tv) (double) (tv.tv_sec + 1.e-6 * tv.tv_usec)

double realtime();

double realtime_()
{
 return realtime();
}

#ifndef SPS
double realtime()
{
    static struct timeval tv;
    static struct timezone tz;

    gettimeofday(&tv,&tz);
    return tim(tv);
}
#endif SPS
```

```
#ifdef TEST

#include <stdio.h>

main()
{
 double realtime_(), t1, t2;
 char c;

 printf("Hit return to end timing... \n");
 t1 = realtime_();
 c =getchar();
 t2 = realtime_();
 printf("Elapsed time: %8.2f secs.\n", t2-t1);
}
#endif
```

End: realtime.c

Program Listing: setup.c

```
/*-------------------------------------------------------------------------
/  File: setup.c
/-------------------------------------------------------------------------
/
/     This is a setup program for the 3-D matrix transposition problem
/     using pvm.
/
/     The size of the 3-D matrix is taken as an argument, along with
/     the number of processors to use.
/
/     This setup program spawns worker programs and
/     in the transposition.
/
/-------------------------------------------------------------------------
*/

/* Contact: Karin A. Remington, karin@cam.nist.gov */

/*
/  Note: full path to pvm3.h is given in INCL compilation flag in makefile:
*/
#include "pvm3.h"

#include<stdlib.h>
#include<stdio.h>
#include<malloc.h>

int xpvm_connect();

main(argc, argv)
int argc; char *argv[];
{
      int *tids;
      int i, j, k, slices, ntids, mytag;
      int size, sizesq, slicestart, index, slowproc;
      int type;
      int nodeid, parentid, rc;
      int nprocs, nworkers, narchs;
      struct hostinfo *hosts;
      char *my_hostname;
      double *f;
```

```
      double starttime, comtime, nodetime, maxtime, realtime();

/*
/     Enroll the manager program in PVM and return an id for this process...
/     and call pvmfconfig to return the # of nodes available (nprocs)...
*/
      nodeid = pvm_mytid();
      parentid = pvm_parent();
/*
/        Get hostname:
*/
      my_hostname = (char *)malloc(8);
      gethostname(my_hostname,7);
/*
/     If there is no parent for this process, take the role of manager
/     and start the other processes, sending each one an integer
/     which 'slice' of the matrix it is to handle.
*/
      if ( parentid == PvmNoParent) {
         if (argc < 2) {
           printf("Usage: %s size num-procs \n", argv[0]);
           exit(1);
         }
         size = atoi(argv[1]);
         ntids = atoi(argv[2]);
         slices = size/ntids;
         if ( slices * ntids != size ) {
             printf("Error: number of processes (%d) must divide size (%d).\n",
                     ntids, size);
             exit(1);
         }
/*
/        Allocate space for the array of task ids, and put
/        this manager tasks id in first slot:
*/
         tids = (int *) malloc(ntids*sizeof(int));
         tids[0] = nodeid;
/*
/        Spawn ntids - 1 copies. Their task ids will be placed in the
/        tids array for later use in message passing.
*/
         xpvm_connect();
         rc = pvm_spawn("setup", (char**)0, PvmTaskTrace, "",
                                            ntids-1, &tids[1]);
```

```
      if (rc != ntids-1) {
         printf("Error in spawning;  Return code is %i\n",rc);
         pvm_exit();
         exit(1);
      }

      for (i=1;i<ntids;i++){
         pvm_initsend(PvmDataDefault);
         pvm_pkint(&size,1,1);
         pvm_pkint(&ntids,1,1);
         pvm_pkint(tids,ntids,1);
         pvm_pkint(&i,1,1);
         pvm_send(tids[i], 100);
      }
      mytag = 0;

   }     /* End task management phase */

   if ( parentid != PvmNoParent ) {
      pvm_recv(parentid, 100);
      pvm_upkint(&size,1,1);
      pvm_upkint(&ntids,1,1);
      tids = (int *) malloc(ntids*sizeof(int));
      pvm_upkint(tids,ntids,1);
      pvm_upkint(&mytag,1,1);
      slices = size/ntids;
   }


/*
/     Allocate space for the array slice and generate entries...
*/
   sizesq = size*size;
   slicestart = mytag*slices*size;
   f = (double *) malloc(size*size*slices*sizeof(double));
   for (i=0;i<size;i++) {
      for (j=0;j<size;j++) {
         for (k=0;k<slices;k++) {
             index = k*sizesq + j*size + i;
             f[index] = slicestart + k*size + j;
         }
      }
   }
```

```
      starttime = realtime();
/*---------------------------------------------------*/
/*    Call to transpose routine:                     */
/*---------------------------------------------------*/
      transp(&mytag, &size, &slices, f, &tids[0], &ntids);
/*---------------------------------------------------*/
      comtime = realtime() - starttime;

      if ( parentid == PvmNoParent) {
/*
/        Receive the timing information from the children processes,
/        taking a max along the way:
*/
         maxtime = comtime;
         slowproc = 0;
         for (i=1;i<ntids;i++) {
            pvm_recv(tids[i], 200);
            pvm_upkdouble(&nodetime,1,1);
            if ( nodetime > maxtime ) {
               maxtime = nodetime;
               slowproc = i;
            }
         }

         printf("Maximum process time: %4.2f (from process %d)\n",
                maxtime,slowproc);
     } else {
         pvm_initsend(PvmDataDefault);
         pvm_pkdouble(&comtime,1,1);
         pvm_send(parentid, 200);
    }
/*
/    Call pvmfexit to terminate the PVM program gracefully...
*/
     pvm_exit();

}
```

End: setup.c

Program Listing: transp.f

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine transp(mytag, size, slices, f, tids, ntids)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     This subroutine performs a 3-D matrix transposition on the
c     data array f.  The call to transp() is made simultaneously by
c     the ntids processors which hold a "slice" of the array.
c     The processors do pairwise exchanges of the appropriate columns
c     of the array, followed by in-processor local transposition.
c     This version is written using PVM for message passing.
c
c     Contact: Karin A. Remington, karin@cam.nist.gov
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      implicit none

      include 'fpvm3.h'
      integer*4 mytag, ntids, tids(ntids)
      integer*4 size, slices
      double precision f(size, size, slices)

      integer*4 tagnode, tagcol, tagsend, tagrecv, msgsize, vecleng
      integer*4 i, is1, is2, indcol, is, colid
      integer*4 add1, add2, add3, halfsize, dest
      logical black


      integer info
      msgsize = size*slices
      vecleng = size

      halfsize = ntids/2

      if ( mytag < halfsize ) then
         add1 = 0
         add2 = halfsize
         add3 = halfsize
      else
         add1 = halfsize
         add2 = -halfsize
         add3 = 0
      end if
```

```
      if ( int( mytag/2 ) * 2 /= mytag ) then
         black = .false.
      else
         black = .true.
      endif

      if ( black ) then
         do 100 i = 2, halfsize
           dest = mod((mytag-1)+i,halfsize)+ 1 + add1
           indcol = (dest-1)*slices+1
           call pvmfinitsend(PVMDEFAULT, info)
           do 110 is = 1,slices
             call pvmfpack(REAL8, f(1,indcol,is), msgsize, 1, info)
110        continue
           call pvmfsend(tids(dest), mytag, info)
           call pvmfrecv(-1, dest-1, info)
           do 120 is = 1,slices
             call pvmfunpack(REAL8, f(1,indcol,is), msgsize, 1, info)
120        continue
100      continue
         do 130 i = 1, halfsize
           dest = mod((mytag-1)+i,halfsize)+ 1 + add3
           indcol = (dest-1)*slices+1
           call pvmfinitsend(PVMDEFAULT, info)
           do 140 is = 1,slices
             call pvmfpack(REAL8, f(1,indcol,is), msgsize, 1, info)
140        continue
           call pvmfsend(tids(dest), mytag, info)
           call pvmfrecv(-1, dest-1, info)
           do 150 is = 1,slices
             call pvmfunpack(REAL8, f(1,indcol,is), msgsize, 1, info)
150        continue
130      continue

      else
         do 200 i = 2, halfsize
           dest = mod((mytag+1)-i+halfsize,halfsize)+ 1 + add1
           indcol = (dest-1)*slices+1
           call pvmfinitsend(PVMDEFAULT, info)
           do 210 is = 1,slices
             call pvmfpack(REAL8, f(1,indcol,is), msgsize, 1, info)
210        continue
           call pvmfsend(tids(dest), mytag, info)
           call pvmfrecv(-1, dest-1, info)
```

```
          do 220 is = 1,slices
            call pvmfunpack(REAL8, f(1,indcol,is), msgsize, 1, info)
 220      continue
 200    continue
        do 230 i = 1, halfsize
          dest = mod((mytag+1)-i+halfsize,halfsize)+ 1 + add3
          indcol = (dest-1)*slices+1
          call pvmfinitsend(PVMDEFAULT, info)
          do 240 is = 1,slices
            call pvmfpack(REAL8, f(1,indcol,is), msgsize, 1, info)
 240      continue
          call pvmfsend(tids(dest), mytag, info)
          call pvmfrecv(-1, dest-1, info)
          do 250 is = 1,slices
            call pvmfunpack(REAL8, f(1,indcol,is), msgsize, 1, info)
 250      continue
 230    continue
      endif

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Local (in processor) data transpose:
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      do 300 i = 0,ntids-1
        colid = i*slices+1
        do 310 is2 = 1, slices-1
          do 320 is1 = is2+1, slices
            call dswap(vecleng,f(1,colid-1+is1,is2),1,
     *                         f(1,colid-1+is2,is1),1)
 320      continue
 310    continue
 300  continue
      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     End subroutine transp
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



      subroutine  dswap (n,dx,incx,dy,incy)
c
c     interchanges two vectors.
```

```
c     uses unrolled loops for increments equal one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(1),dy(1),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c       code for unequal increments or equal increments not equal
c         to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dx(ix)
        dx(ix) = dy(iy)
        dy(iy) = dtemp
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c       code for both increments equal to 1
c
c
c       clean-up loop
c
   20 m = mod(n,3)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dx(i)
        dx(i) = dy(i)
        dy(i) = dtemp
   30 continue
      if( n .lt. 3 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,3
        dtemp = dx(i)
        dx(i) = dy(i)
        dy(i) = dtemp
        dtemp = dx(i + 1)
```

```
        dx(i + 1) = dy(i + 1)
        dy(i + 1) = dtemp
        dtemp = dx(i + 2)
        dx(i + 2) = dy(i + 2)
        dy(i + 2) = dtemp
   50 continue
      return
      end
```

End: transp.f

Program Listing: user.cleanup

```
#!/bin/ksh

echo "${0##*/}: The user.cleanup script is running in directory:  "
pwd

if [ "$CMD" = "pvm.script" ]
then
   echo "${0##*/}: Copying back pvm log file to ./pvmlog ... "
   cp /tmp/pvml.$MYUID $INITIALDIR/pvmlog
   if [ "$XPVMFLAG" = "yes" ]
   then
     echo "${0##*/}: Copying back xpvm trace file to ./xpvmtrace.$CLUSTER ... "
     cp $TMPDIR/xpvm.trace.$CLUSTER $INITIALDIR
   fi
fi

#for host in $hostlist
#do
#   echo "Here is where I would copy back output from ${host} ..."
#done

echo "${0##*/}: Finishing the user.cleanup script."
```

End: user.cleanup

Program Listing: xpvmconnect.c

```
#include <pvm3.h>
#include <pvmtev.h>

/* This routine allows a pvm program to connect to an existing   */
/* XPVM session.  This gets around the XPVM restriction that     */
/* processes spawned from within the XPVM monitoring console     */
/* cannot read input. Instead of spawning the process to be      */
/* monitored from within XPVM, the XPVM monitor is first started,*/
/* and then, from a separate shell session, the process to be    */
/* monitored is started.   Thi process makes a call to this      */
/* routine, which connects it to the running XPVM session.       */

/* Contact: Karin A. Remington, karin@cam.nist.gov */

int xpvm_connect()
{

Pvmtmask trace_mask;
int xpvm_tid;

/* Get XPVM Task ID */

if ( (xpvm_tid = pvm_gettid( "xpvm", 0 )) > 0 ) {

/* Set Self Trace & Output Destinations & Message Codes */
   pvm_setopt( PvmSelfTraceTid, xpvm_tid );
   pvm_setopt( PvmSelfTraceCode, 666 );
   pvm_setopt( PvmSelfOutputTid, xpvm_tid );
   pvm_setopt( PvmSelfOutputCode, 667 );

/* Set Future Children's Trace & Output Dests & Codes */
/* (optional) */
   pvm_setopt( PvmTraceTid, xpvm_tid );
   pvm_setopt( PvmTraceCode, 666 );
   pvm_setopt( PvmOutputTid, xpvm_tid );
   pvm_setopt( PvmOutputCode, 667 );

/* Generate Default Trace Mask */
   TEV_INIT_MASK( trace_mask );
   TEV_SET_MASK( trace_mask, TEV_MCAST0 );
   TEV_SET_MASK( trace_mask, TEV_SEND0 );
```

```
   TEV_SET_MASK( trace_mask, TEV_RECV0 );
   TEV_SET_MASK( trace_mask, TEV_NRECV0 );
/* Add Other Desired Events Here */

/* Set Self Trace Mask */
   pvm_settmask( PvmTaskSelf, trace_mask );

/* Set Future Children's Trace Mask */
/* (optional) */
   pvm_settmask( PvmTaskChild, trace_mask );

   return 0;
}
else {
   printf( "XPVM not running, cannot trace\n" );
   return -1;
}

}
```

End: xpvmconnect.c

J

Program Listing: hostfile.txt

```
grand17.nist.gov  wd=/a/danube/danube/nist/karin/testing ep=.:/danube/nist/karin/pvme/bin/RS6K
grand18.nist.gov  wd=/a/danube/danube/nist/karin/testing ep=.:/danube/nist/karin/pvme/bin/RS6K
grand19.nist.gov  wd=/a/danube/danube/nist/karin/testing ep=.:/danube/nist/karin/pvme/bin/RS6K
grand20.nist.gov  wd=/a/danube/danube/nist/karin/testing ep=.:/danube/nist/karin/pvme/bin/RS6K
```

End: hostfile.txt

Index