1993

Intel NX to PVM 3.2 Message Passing Conversion Library

Trey Arthur

Michael L. Nelson
Old Dominion University

Original Publication Citation
Arthur, T., & Nelson, M. L. (1993). Intel NX to PVM 3.2 message passing conversion library. *NASA Technical Memorandum: 109038*. Hampton, VA: NASA Langley Research Center.

NASA Technical Memorandum 109038

Intel NX to PVM3.2
Message Passing Conversion Library

Trey Arthur
Computer Sciences Corporation
Hampton, Virginia

and

Michael Nelson
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia

October 1993

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virgina 23681-0001

Intel NX to PVM3.2
Message Passing Conversion Library
Version 2.0*

T. Arthur (j.j.arthur@larc.nasa.gov)†
M. Nelson (m.l.nelson@larc.nasa.gov)‡

October 14, 1993

Abstract

NASA Langley Research Center has developed a library that allows Intel NX message passing codes to be executed under the more popular and widely supported Parallel Virtual Machine (PVM) message passing library. PVM was developed at Oak Ridge National Labs and has become the defacto standard for message passing. This library will allow the many programs that were developed on the Intel iPSC/860 or Intel Paragon in a Single Program Multiple Data (SPMD) design to be ported to the numerous architectures that PVM (version 3.2) supports. Also, the library adds global operations capability to PVM. A familiarity with Intel NX and PVM message passing is assumed.

1. Introduction

At NASA Langley Research Center (LaRC), the center's vector supercomputers have become heavily saturated with users' jobs. Alternatives are being considered to off load some of these jobs to other systems. Among the alternatives considered is the transition of some applications from the vector supercomputers to parallel machines and workstations clusters. With the proliferation of high powered workstations, workstation clustering, in both batch and parallel use, offers an attractive solution to supercomputer saturation.

At NASA LaRC, the Parallel Virtual Machine (PVM) software provides the most popular parallel programming environment. PVM was developed at Oak Ridge National Laboratory

*This work was performed under NASA contract NAS1-19038

†Member of the Technical Staff, Computer Sciences Corporation, Hampton, VA
‡NASA Langley Research Center, Hampton, VA

and has become a defacto standard for message passing (ref. 4) . But before PVM had reached this level of popularity, many parallel applications had been developed on the Intel iPSC/860. There was a need to transition these Intel NX message passing (ref. 3) codes to PVM.

This document describes the Intel to PVM, version 3.2 (PVM3) libraries. A familiarity with Intel NX and PVM message passing is assumed. The libraries, *libi2pvm3.a* and *libfi2pvm3.a*, are written in C and contain wrappers for several Intel functions and routines. The executable, *pvmexec*, is a C program which starts the PVM daemons, runs the user application, waits for completion of the slaves, and terminates the PVM daemon processes. If *pvmexec* is run in the Distributed Queing System (DQS) environment (ref. 5), then the PVM daemons will not be started or stopped by *pvmexec*. *pvmexec* is able to detect if it is being run in DQS and will relinquish PVM daemon control to DQS.

The main purpose of the libraries is to allow the user with a code written for a Intel Message Passing Supercomputer in C or FORTRAN to quickly port the code to a workstation cluster using PVM3. To use the libraries in conjunction with the executable *pvmexec* (*pvmexec* is analogous to *cubeexec*[1]), the user must add two subroutine calls and convert asynchronous message passing calls (e.g., isend and irecv) to synchronous calls (e.g., csend and crecv).

Another use of the libraries is to give the PVM3 user access to many of the global libraries which are absent in the standard PVM distribution. To use the global routines without using *pvmexec*, the user should make a call to the **pvmsetup** routine (see section 4). After the task ids and number of slave processes are known, the **pvmsetup** routine is called so that the global routines can be used.

2. Building the libraries

This library is available via anonymous ftp from

blearg.larc.nasa.gov:/pub/pvm/i2pvm3.shar.Z

Before unpacking *i2pvm3.shar*, the user should make the directory *$HOME/pvm3*. To unpack the library, the following should be typed in the user's home directory:

% *sh i2pvm3.shar*

[1] *cubeexec* was developed by William J. Nitzberg from the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center to easily run executable code on the iPSC/860. *cubeexec* is not an Intel supported utility.

The following should be typed to compile the library:

% *cd pvm3/i2pvm3*
% *make*

This will compile the libraries and pvmexec.c. The libraries are moved to *$PVM_ROOT/lib/$PVM_ARCH*. The executable, *pvmexec*, is moved to *$PVM_ROOT/bin/$PVM_ARCH*. The include files that the user will need are installed in *$PVM_ROOT/include*.

To compile a program to run in the PVM environment, the following libraries should be linked in this order: *(libfi2pvm3.a), libi2pvm3.a* and *libpvm3.a*. The following is an example compile line for a C and FORTRAN program, respectively:

% *cc -O -o daria daria.c -L$PVM_ROOT/lib/$PVM_ARCH -li2pvm3 -lpvm3*
% *f77 -O -o daria daria.f -L$PVM_ROOT/lib/$PVM_ARCH -lfi2pvm3 -li2pvm3 -lpvm3*

3. Use of pvmexec

The libraries can be easily used in conjunction with the executable *pvmexec*. *pvmexec* starts up the daemon processes, runs the application, waits for the application to finish then kills the PVM daemons. If *pvmexec* is run in the Distributed Queing System (DQS) environment (ref. 5), then the PVM daemons will not be started or stopped by *pvmexec*. *pvmexec* is able to detect if it is being run in DQS and will relinquish PVM daemon control to DQS.

When using the library with *pvmexec*, the first executable line in the code should be a call to **pvminit()**. This routine receives messages from *pvmexec*. The final call in the user's program should be to **pvmquit()**. **Failure to call these routines by ALL processes will cause *pvmexec* to hang**. Once the **pvmquit()** routine has been called by all processes, *pvmexec* will kill the PVM daemons and exit. As noted before, the user must also convert asynchronous routines to synchronous routines.

pvmexec recognizes the three options -t, -v, and -V. Option -t is used to specify the number of processes to start, -v is verbose mode, and -V prints the version of *pvmexec*. An example for running four processes of the executable *node*:

% *pvmexec -v -t 4 node*

pvmexec will start daemons on all of the hosts in *hostfile*. *hostfile* is a PVM host file (ref. 1) and is read from the directory in which *pvmexec* is executed. If *hostfile* is not present, *pvmexec* will run the all PVM processes on the current workstation.

4. Use of libraries without pvmexec

The libraries can be used without using *pvmexec*, however, it is the user's responsibility to start and stop the PVM daemons (see (ref. 1) for more information). To use the libraries without *pvmexec*, make a call to **pvmsetup** after the task ids and number of slave processes are known. **NOTE**: if using **pvmsetup**, do NOT call **pvminit** or **pvmquit**. The following code fragment is an example on how to use **pvmsetup**.

C example:

```
mytid = pvm_mytid();
tids[0] = pvm_parent();
if( tids[0] < 0){
  tids[0] = mytid;
  pvm_spawn("spmd", (char**)0, 0, "", NPROC-1, &tids[1]);
  pvm_initsend( PvmDataDefault );
  pvm_pkint(tids, NPROC, 1);
  pvm_mcast(&tids[1], NPROC-1, 0);
}
else {
  pvm_recv(tids[0], 0);
  pvm_upkint(tids, NPROC, 1);
}

pvmsetup(tids,NPROC);
```

FORTRAN example:

```
call pvmfmytid( mytid )
call pvmfparent( tids(0) )
if ( tids(0) .lt. 0) then
  tids(0) = mytid
  call pvmfspawn( 'spmd', PVMDEFAULT, '*', NPROC-1, tids(1), numt )
  call pvmfpack( INTEGER4, tids, NPROC, 1, info )
  call pvmfmcast( NPROC-1, tids(1), 0, info )
else
  call pvmfrecv( tids(0), 0, info )
  call pvmfunpack( INTEGER4, tids, NPROC, 1, info )
end if

call pvmsetup( tids, NPROC )
```

5. Supported routines

Routine	Usage	Description
Sending		
csend()	csend(msgtype, buf, len, node, pid);	send a message
csendsi()	csendsi(msgtype, buf, len, node, pid);	send short integer message
csendi()	csendi(msgtype, buf, len, node, pid);	send an integer message
csendr()	csendr(msgtype, buf, len, node, pid);	send a real message
csendd()	csendd(msgtype, buf, len, node, pid);	send a double precision message
Receiving		
crecv()	crecv(msgtype, buf, len);	receive a message
crecvsi()	crecvsi(msgtype, buf, len);	receive short integer message
crecvi()	crecvi(msgtype, buf, len);	receive an integer message
crecvr()	crecvr(msgtype, buf, len);	receive a real message
crecvd()	crecvd(msgtype, buf, len);	receive a double precision message
Global		
gdhigh()	gdhigh(buf,num,work);	global double precision MAX
gdlow()	gdlow(buf,num,work);	global double precision MIN
gdprod()	gdprod(buf,num,work);	global double precision MULTIPLY
gdsum()	gdsum(buf,num,work);	global double precision SUM
gihigh()	gihigh(buf,num,work);	global integer MAX
gilow()	gilow(buf,num,work);	global integer MIN
giprod()	giprod(buf,num,work);	global integer MULTIPLY
gisum()	gisum(buf,num,work);	global integer SUM
gshigh()	gshigh(buf,num,work);	global real MAX
gslow()	gslow(buf,num,work);	global real MIN
gsprod()	gsprod(buf,num,work);	global real MULTIPLY
gssum()	gssum(buf,num,work);	global real SUM
gsync()	gsync();	synchronization
Other		
pvminit()	pvminit();	call when using *pvmexec*
pvmsetup()	pvmsetup(tids,nproc);	call when NOT using *pvmexec*
pvmquit()	pvmquit();	send quit signal to pvmexec
mynode()	int mynode();	returns logical process number
numnodes()	int numnodes();	returns number of processes
cprobe()	cprobe(msgtype);	wait for a message to arrive
infocount()	int infocount();	length of message in bytes
infonode()	int infonode();	node id for sending process
dclock()	double dclock();	returns wall clock in seconds

Table 1: Supported C routines

Routine	Usage	Description
Sending		
csend()	call csend(msgtype, buf, len, node, pid)	send a message
csendsi()	call csendsi(msgtype, buf, len, node, pid)	send short integer message
csendi()	call csendi(msgtype, buf, len, node, pid)	send an integer message
csendr()	call csendr(msgtype, buf, len, node, pid)	send a real message
csendd()	call csendd(msgtype, buf, len, node, pid)	send a double precision message
Receiving		
crecv()	call crecv(msgtype, buf, len)	receive a message
crecvsi()	call crecvsi(msgtype, buf, len)	receive short integer message
crecvi()	call crecvi(msgtype, buf, len)	receive an integer message
crecvr()	call crecvr(msgtype, buf, len)	receive a real message
crecvd()	call crecvd(msgtype, buf, len)	receive a double precision message
Global		
gdhigh()	call gdhigh(buf,num,work)	global double precision MAX
gdlow()	call gdlow(buf,num,work)	global double precision MIN
gdprod()	call gdprod(buf,num,work)	global double precision MULTIPLY
gdsum()	call gdsum(buf,num,work)	global double precision SUM
gihigh()	call gihigh(buf,num,work)	global integer MAX
gilow()	call gilow(buf,num,work)	global integer MIN
giprod()	call giprod(buf,num,work)	global integer MULTIPLY
gisum()	call gisum(buf,num,work)	global integer SUM
gshigh()	call gshigh(buf,num,work)	global real MAX
gslow()	call gslow(buf,num,work)	global real MIN
gsprod()	call gsprod(buf,num,work)	global real MULTIPLY
gssum()	call gssum(buf,num,work)	global real SUM
gsync()	call gsync()	synchronization
Other		
pvminit()	call pvminit()	call when using *pvmexec*
pvmsetup()	call pvmsetup(tids,nproc)	call when NOT using *pvmexec*
pvmquit()	call pvmquit()	send quit signal to pvmexec
mynode()	integer mynode()	returns logical process number
numnodes()	integer numnodes()	returns number of processes
cprobe()	call cprobe(msgtype)	wait for a message to arrive
infocount()	integer infocount()	length of message in bytes
infonode()	integer infonode()	node id for sending process
dclock()	double precision dclock()	returns wall clock in seconds

Table 2: Supported FORTRAN routines

If the PVM environment has machines with different byte ordering conventions, some additional code changes will be needed. This is because message passing on the Intel is based on sending messages in bytes. If the PVM environment has machines with different byte ordering conventions, the user will need to use a different set of communication routines.

These routines help PVM determine how to send the message. To use these calls, replace *csend* with *csend*x where x is either *si, i, r* or *d* which stands for short integer, integer, real or double precision, respectively. For example, to send the real variable y to logical node 2, use this syntax: *csendr(msgtype, y, 4, 2, 0)*. Note that the message length is still in bytes, so the user only needs to add the appropriate appendix to *csend*. This message should be received by using the corresponding receive routine: *crecvr(msgtype, y, 4)*.

6. Unsupported routines

Many NX routines are absent from this library. The supported routines were chosen based on experience in porting from the Intel/i860 to the PVM environment. Many of the asynchronous routines are not supported because it is difficult to emulate these routines in PVM. The easiest solution to this problem is to have the user change asynchronous routines (e.g., isend, irecv) to synchronous communication (e.g., csend, crecv).

7. C Example

Given the following Intel C program :

```
#include <stdio.h>
#include <cube.h>

main()
{
  int iam, nproc;
  float x;

  iam = mynode();
  nproc = numnodes();
  if ( !iam ) {
    x = 20.0;
    csend(100, x, 4, -1, 0);
  }
  else {
    crecv(100, x, 4);
  }
  gssum(x,1,work);
  if ( !iam ) printf("check: x should equal %d\n",nproc*20.0);
  printf("iam= %d, x= %f\n",iam,x);
}
```

To run this program in a PVM environment using the libi2pvm3.a library, the following code changes would need to be made:

1) change the include file "cube.h" to "nx2pvm.h"
2) change the first executable line to "pvminit();"
3) change the last executable line to "pvmquit();"

Below is the modified C code:

```
#include <stdio.h>
#include <nx2pvm.h>

main()
{
  int iam, nproc;
  float x, work;

  pvminit();
  iam = mynode();
  nproc = numnodes();
  if ( !iam ) {
    x = 20.0;
    csend(100, x, 4, -1, 0);
  }
  else {
    crecv(100, x, 4);
  }
  gssum(x,1,work);
  if ( !iam ) printf("check: x should equal %d\n",nproc*20.0);
  printf("iam= %d, x= %f\n",iam,x);
  pvmquit();
}
```

The following is a makefile for compiling the program to run on a PVM environment:

```
#
INCLUDEDIR =    $(PVM_ROOT)/include
PVMLIB     =    $(PVM_ROOT)/lib/$(PVM_ARCH)
BDIR       =    $(PVM_ROOT)/bin
XDIR       =    $(BDIR)/$(PVM_ARCH)
CLIBS      =    -li2pvm3 -lpvm3

CFLAGS     =    -g

beavis:
        cc $(CFLAGS) -I$(INCLUDEDIR) -L$(PVMLIB) -o $@ beavis.c $(CLIBS)
        mv beavis $(XDIR)
```

8. FORTRAN Example

Given the following Intel FORTRAN program :

```
      program beavis
      include 'fcube.h'

      iam = mynode()
      nproc = numnodes()
      if(iam .eq. 0) then
        x = 20.0
        call csend(100, x, 4, -1, 0)
      else
        call crecv(100, x, 4)
      endif
      call gssum(x,1,work)
      if(iam .eq. 0) write(*,*) 'check: x should equal ',nproc*20.0
      write(*,*) 'iam = ',iam,', x= ',x

      end
```

To run this program in a PVM environment using the libfi2pvm3.a library, the following code changes would need to be made:

1) change the include file "fcube.h" to "fnx2pvm.h"
2) change the first executable line to "call pvminit()"
3) change the last executable line to "call pvmquit()"

Below is the modified FORTRAN code:

```
      program beavis

      include 'fnx2pvm.h'

      call pvminit()
      iam = mynode()
      nproc = numnodes()
      if(iam .eq. 0) then
        x = 20.0
        call csend(100, x, 4, -1, 0)
      else
        call crecv(100, x, 4)
      endif
      call gssum(x,1,work)
      if(iam .eq. 0) write(*,*) 'check: x should equal ',nproc*20.0
      write(*,*) 'iam = ',iam,', x= ',x

      call pvmquit()
      end
```

The following is a makefile for compiling the program to run on a PVM environment:

```
#
PVMLIB  =       $(PVM_ROOT)/lib/$(PVM_ARCH)
BDIR    =       $(PVM_ROOT)/bin
XDIR    =       $(BDIR)/$(PVM_ARCH)
FLIBS   =       -lfi2pvm3 -li2pvm3 -lpvm3

beavis:
      cp $(PVM_ROOT)/include/fnx2pvm.h .
      f77 $(FFLAGS) -L$(PVMLIB) -o $@ beavis.f $(FLIBS)
      mv beavis $(XDIR)
```

9. Executing the examples

The program is compiled and linked by typing *make*. For compatibility with PVM, the executable *beavis* is moved to *$PVM_ROOT/bin/$PVM_ARCH*. To execute *beavis* over four machines, the file *hostfile* should be created with each machine name on a separate line (see (ref. 1) for details on how to set up a host file). To execute the code, the following should be typed:

```
% cd $PVM_ROOT/bin/$PVM_ARCH
% pvmexec -v -t 4 beavis
```

Analogous to PVM, all output to the screen is redirected to the file */tmp/pvml.*$< uid >$. To obtain the status of the job while it is running, in another window on any of the machines running PVM, the following should be typed:

```
% pvm
pvm> ps -a
```

10. Summary

This report describes the use of the NASA Langley Research Center library for conversion of Intel NX message passing codes to PVM3.2 message passing codes. If an application is a candidate for conversion, it must be of SPMD design and any asynchronous sends and receives must be converted to synchronous sends and receives. If the intended PVM environment is heterogeneous, some additional code modifications may be necessary.

This library should enable users to quickly port codes developed on the Intel iPSC/860 or Intel Paragon to other environments. This includes workstations clusters or even other parallel computers that provide PVM support. The use of *pvmexec* emulates the Intel NX environment and should minimize porting difficulties. The use of this library also adds global operations capability to PVM. Additions, modifications, or suggestions are welcome and can be sent to the authors.

References

1. Geist, A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Mancheck, R.; Sunderman, V.: *PVM 3 Users' Guide and Reference Manual.* ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, TN, May 1993.

2. Grant, B.K.; Skjellum, A.: *The PVM Systems: An In- Depth Analysis and Documenting Study - Concise Edition.* Lawrence Livermore National Lab, Livermore, CA, 20 August 1992.
3. *iPSC/2 and iPSC/860 User's Guide.* Intel Corporation, Order Number 311532-007, April 1991.
4. Nelson, M.: *PVM Provides Power in the Public Domain.* Parallelogram: The International Journal of High Performance Computing, May/June 1993, pp. 20-21.
5. Revor, L.: *DQS Users Guide.* Argonne National Lab, September 15, 1992.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
October 1993

3. REPORT TYPE AND DATES COVERED
Technical Memorandum

4. TITLE AND SUBTITLE
Intel NX to PVM3.2 Message Passing Conversion Library

5. FUNDING NUMBERS
WU 505-90-53-02

6. AUTHOR(S)
Trey Arthur
Michael L. Nelson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TM-109038

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified–Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
NASA Langley Research Center has developed a library that allows Intel NX message passing codes to be executed under the more popular and widely supported Parallel Virtual Machine (PVM) message passing library. PVM was developed at Oak Ridge National Labs and has become the defacto standard for message passing. This library will allow the many programs that were developed on the Intel iPSC/860 or Intel Paragon in a Single Program Multiple Data (SPMD) design to be ported to the numerous architectures that PVM (version 3.2) supports. Also, the library adds global operations capability to PVM. A familiarity with Intel NX and PVM message passing is assumed.

14. SUBJECT TERMS
Distributed Computing; Parallel Processing; PVM3; Intel NX Message Passing

15. NUMBER OF PAGES
13

16. PRICE CODE
AO3

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

NSN 7540-01-280-5500

Standard Form 298(Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102
NASA-Langley, 1992