XSB–PVM–Prolog: Adapting the PVM-Prolog Interface to the XSB System

Joaquim Gonçalves, Rui Marques, José C. Cunha
Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática
2825 Monte Caparica
(jmpg, rfm, jcc)@asc.di.fct.unl.pt

Abstract
PVM-Prolog is a programming interface from Prolog to the PVM system. In this report we present the specification of the PVM-Prolog interface, that was adapted to extend the XSB System.

1 Introduction
The PVM-Prolog is described in [1, 2, 3, 4]. In this report we focus only PVM interface component. This is an intermediate programming layer based on a set of built-in Prolog predicates for parallelism and distribution, based on the spawning of independent Prolog evaluators (as PVM tasks), in local and remote processors of a PVM virtual machine, including the passing of top goals to such Prolog tasks. PVM-like communication predicates allow Prolog terms to be asynchronously exchanged by Prolog tasks.

Each task must support a specific core inference engine for Prolog, if we are composing a homogeneous PVM-Prolog system. One of our prototypes relies upon the WAM model as the process engine, based upon a system called NanoProlog, developed at our Department by A. M. Dias [5]. It is possible to include any self-contained Prolog as the process engine in the PVM-Prolog system. The integration of other engines such as SICStus-Prolog [7] was already done, In this report we describe the PVM interface that was implemented for and XSB [6].

In order to provide the maximum possible flexibility to this interface, we decided to keep all the PVM functionalities accessible at the Prolog level. This is required for supporting the above mentioned hybrid heterogeneous applications. The set of predicates that directly correspond to the PVM functions constitute
a raw interface to PVM from Prolog, called PVM-Prolog0. These predicates are implemented as built-in predicates in any Prolog process engine, within a PVM-Prolog interface library, and they are described in the following.

Our main concern in the definition of this interface was to adapt the PVM functions according to the semantics of the Prolog language. Below we summarize the main aspects of our design:

- Messages are interpreted as Prolog terms, and extensions to PVM pack and unpack functions are provided in order to convert term representations in hybrid heterogeneous applications.
- All of the predicates exhibit a strict deterministic behavior, i.e. they all fail on backtracking.

## 2 PVM-Prolog0

### pvmO_mytid/1

**Prototype:**

```
pvmO_mytid(-Tid)
```

**Function:**

Returns the Tid of the calling process.

### pvmO_exit/1

**Prototype:**

```
pvmO_exit(-Info)
```

**Function:**

Tells the local pvm that this process is leaving PVM.

### pvmO_kill/2

**Prototype:**

```
pvmO_kill(+Tid, -Info)
```

**Function:**

Terminates a specified PVM process.

### pvmO_addhosts/4

**Prototype:**

```
pvmO_addhosts(+HostList, +NumHosts, -TidList, -Info)
```
Function:
Add hosts to the virtual machine.

pvm0_delhosts/4
Prototype:
pvm0_delhosts(+HostList, +NumHosts, -TidList, -Info)
Function:
Delete hosts from the virtual machine.

pvm0_spawn/7 Prototype:
pvm0_spawn(+PrologExec, +GoalList, +Options,
+Where, +Ntasks, -TidList, -Nspawnwd)
Function: Starts new XSB-PVM processes.
+Options: is the sum of some of the following values

PVM can choose any machine to start the task
pvm0_constant( taskDefault, 0, pvm_spawn )

+Where specifies a particular host
pvm0_constant( taskHost, 1, pvm_spawn )

+Where specifies a type of architecture
pvm0_constant( taskArch, 2, pvm_spawn )

start up processes under debugger
pvm0_constant( taskDebug, 4, pvm_spawn )

processes will generate PVM trace data
pvm0_constant( taskTrace, 8, pvm_spawn )

use complement host set
pvm0_constant( hostCompl, 32, pvm_spawn )

+Ntasks: Number of tasks to be started (>=1)

pvm0_parent/1
Prototype:
pvm0_parent(-Tid)
Function:
Returns the Tid of the process that spawned the calling process.

\texttt{pvm0\_tidtchost/2}
\textbf{Prototype:}
\begin{verbatim}
pvm0\_tidtchost(+Tid, -HostTid)
\end{verbatim}
\textbf{Function:}
Returns the host of the specified PVM process.

\texttt{pvm0\_perror/2}
\textbf{Prototype:}
\begin{verbatim}
pvm0\_perror(+MsgString, -Info)
\end{verbatim}
\textbf{Function:}
Prints message describing the last error returned by a PVM call.

\texttt{pvm0\_config/4}
\textbf{Prototype:}
\begin{verbatim}
pvm0\_config(-NumHosts, -NumArchs, -ConfigList, -Info)
\end{verbatim}
\textbf{Function:}
Returns information about the present virtual machine configuration.

- \texttt{ConfigList}: Each element is
  \begin{verbatim}
  config(NumHosts, HostName, HostArch, HostSpeed)
  \end{verbatim}

\texttt{pvm0\_tasks/3}
\textbf{Prototype:}
\begin{verbatim}
pvm0\_tasks(+Where, -Ntasks, -TaskList)
\end{verbatim}
\textbf{Function:}
Returns information about the tasks running on the virtual machine.

\texttt{pvm0\_getopt/2}
\textbf{Prototype:}
\begin{verbatim}
pvm0\_getopt(+What, -Val)
\end{verbatim}
Function:

Returns the value of libpvm options.

+What: Integer value which may be obtained with

\[
pvm0\_{\text{constant}}(\text{Constant, Value, Type})
\]

Message routing policy
\[
pvm0\_{\text{constant}}(\text{route, 1, pvm\_opt })
\]

Libpvm debug mask
\[
pvm0\_{\text{constant}}(\text{debugMask, 2, pvm\_opt })
\]

Auto error reporting
\[
pvm0\_{\text{constant}}(\text{autoErr, 3, pvm\_opt })
\]

Stdout destination for children
\[
pvm0\_{\text{constant}}(\text{outputTid, 4, pvm\_opt })
\]

Output message tag
\[
pvm0\_{\text{constant}}(\text{outputCode, 5, pvm\_opt })
\]

Trace data destination for children
\[
pvm0\_{\text{constant}}(\text{traceTid, 6, pvm\_opt })
\]

Trace message tag
\[
pvm0\_{\text{constant}}(\text{traceCode, 7, pvm\_opt })
\]

Message fragment size
\[
pvm0\_{\text{constant}}(\text{fragSize, 8, pvm\_opt })
\]

Allow messages to reserved tags and TIDs
\[
pvm0\_{\text{constant}}(\text{resvTids, 9, pvm\_opt })
\]

Stdout destination
\[
pvm0\_{\text{constant}}(\text{selfOutputTid, 10, pvm\_opt })
\]

Output message tag
\[
pvm0\_{\text{constant}}(\text{selfOutputCode, 11, pvm\_opt })
\]

Trace data destination
\[
pvm0\_{\text{constant}}(\text{selfTraceTid, 12, pvm\_opt })
\]

Trace message tag
\[
pvm0\_{\text{constant}}(\text{selfTraceCode, 13, pvm\_opt })
\]
pvm_catchout prints task ids with output
pvmO_constant( showTids, 14, pvm_opt )

pvmO_setopt/3
Prototype:
pvmO_setopt(+What, +Val, -OldVal)

Function:
Sets libpvm options.

+What: Integer value defining what to set, which may be obtained with pvmO_constant( Constant, Value, Type )

Message routing policy
pvmO_constant( route, 1, pvm_opt )

Libpvm debug mask
pvmO_constant( debugMask, 2, pvm_opt )

Auto error reporting
pvmO_constant( autoErr, 3, pvm_opt )

Stdout destination for children
pvmO_constant( outputTid, 4, pvm_opt )

Output message tag
pvmO_constant( outputCode, 5, pvm_opt )

Trace data destination for children
pvmO_constant( traceTid, 6, pvm_opt )

Trace message tag
pvmO_constant( traceCode, 7, pvm_opt )

Message fragment size
pvmO_constant( fragSize, 8, pvm_opt )

Allow messages to reserved tags and TIDs
pvmO_constant( resvTids, 9, pvm_opt )

Stdout destination
pvmO_constant( selfOutputTid, 10, pvm_opt )
Output message tag
pvm0_constant( selfOutputCode, 11, pvm_opt )

Trace data destination
pvm0_constant( selfTraceTid, 12, pvm_opt )

Trace message tag
pvm0_constant( selfTraceCode, 13, pvm_opt )

pvmCatchout prints task ids with output
pvm0_constant( showTids, 14, pvm_opt )

+Val: Integer value specifying new setting option, which may be obtained with pvm0_constant(Constant, Value, Type)

Don’t request or grant connections
pvm0_constant( dontRoute, 1, pvm_route ).

(Default) Don’t request but allow
pvm0_constant( allowDirect, 2, pvm_route ).

Request and allow connections
pvm0_constant( routeDirect, 3, pvm_route ).

pvm0_sendsig/3
Prototype:
pvm0_sendsig(+Tid, +Signum, -Info)

Function:
Sends a signal to another PVM process.

pvm0_notify/5 Prototype:
pvm0_notify(+What, +MsgTag, +Cnt, +TidList, -Info)

Function: Request notification of PVM event such as host failure.

pvm0_mkbuf/2
Prototype:
pvm0_mkbuf(+Encoding, -Bufid)
Function:
  Creates a new message buffer.

Remark:
  In the context of XSB, Encoding must alway be 0.

pvm0_freebuf/2
Prototype:
  pvm0_freebuf(+Bufid, -Info)

Function:
  Disposes of a message buffer.

pvm0_getsbuf/1
Prototype:
  pvm0_getsbuf(-Bufid)

Function:
  Returns the message buffer identifier for the active send buffer.

pvm0_getrbuf/1
Prototype:
  pvm0_getrbuf(-Bufid)

Function:
  Returns the message buffer identifier for the active receive buffer.

pvm0_setsbuf/2
Prototype:
  pvm0_setsbuf(+Bufid, -OldBuf)

Function:
  Switches the active send buffer.

pvm0_setrbuf/2
Prototype:
  pvm0_setrbuf(+Bufid, -OldBuf)
Function:
    Switches the active receive buffer and saves the previous buffer.

pvm0_pkbyte/4
Prototype:
    pvm0_pkbyte(+ByteList, +Nitem, +Stride, -Info)

Function:
    Pack the active message buffer with a list of byte values.

Remark:
    +ByteList must always be a list.
    +Stride must always be 1.

pvm0_pkdouble/4
Prototype:
    pvm0_pkdouble(+DoubleList, +Nitem, +Stride, -Info)

Function:
    Pack the active message buffer with a list of double values.

Remark:
    +DoubleList must always be a list.
    +Stride must always be 1.

pvm0_pkfloat/4
Prototype:
    pvm0_pkfloat(+FloatList, +Nitem, +Stride, -Info)

Function:
    Pack the active message buffer with a list of float values.

Remark:
    +FloatList must always be a list.
    +Stride must always be 1.

pvm0_pkint/4
Prototype:
    pvm0_pkint(+IntList, +Nitem, +Stride, -Info)
Function:
   Pack the active message buffer with a list of integer values.

Remark:
   +IntList must always be a list.
   +Stride must always be 1.

pvm0_pklong/4
Prototype:
   pvm0_pklong(+LongList, +Nitem, +Stride, -Info)

Function:
   Pack the active message buffer with a list of long values.

Remark:
   +LongList must always be a list.
   +Stride must always be 1.

pvm0_pkshort/4
Prototype:
   pvm0_pkshort(+ShortList, +Nitem, +Stride, -Info)

Function:
   Pack the active message buffer with a list of short values.

Remark:
   +ShortList must always be a list.
   +Stride must always be 1.

pvm0_pkstr/2
Prototype:
   pvm0_pkstr(+string, -Info)

Function:
   Pack the active message buffer with a string.

pvm0_pkterm/2
Prototype:
   pvm0_pkterm(+Term, -Info)
Function:
Pack the active message buffer with a term.

Remark:
if +Term is a complex term (i.e. a set o terms separated with ",",),
it must be enclosed between ().

eg: +Term is p(X),q(Y) then you must put it like (p(X),q(Y)).

pvm0_send/3
Prototype:
pvm0_send(+Tid, +MsgTag, -Info)

Function:
Immediately sends the data in the active message buffer.

pvm0_mcast/4
Prototype:
pvm0_mcast(+Tids, +Ntasks, +MsgTag, -Info)

Function:
Multicasts the data in the active message buffer to a set of tasks.

pvm0_recv/3
Prototype:
pvm0_recv(+Tid, +MsgTag, -Bufid)

Function:
Receive a message.

Remark:
+Tid = -1, means RECEIVE FROM ANY TID.
+MsgTag = -1, means RECEIVE ANY MSGTAG.
pvm0_nrecv/3
Prototype:
   pvm0_nrecv(+Tid, +MsgTag, -Bufid)

Function:
   Non-blocking receive.

Remark:
   +Tid = -1, means RECEIVE FROM ANY TID.
   +MsgTag = -1, means RECEIVE ANY MSGTAG.

pvm0_bufinfo/5
Prototype:
   pvm0_bufinfo(+Bufid, -Bytes, -Msgtag, -Tid, -Info)

Function:
   Returns information about a message buffer.

pvm0_upkbyte/4
Prototype:
   pvm0_upkbyte(-ByteList, +Nitem, +Stride, -Info)

Function:
   Unpack the active message buffer into a list of byte values.

Remark:
   +Stride must always be 1.

pvm0_upkdouble/4
Prototype:
   pvm0_upkdouble(-DoubleList, +Nitem, +Stride, -Info)

Function:
   Unpack the active message buffer into a list of double values.

Remark:
   +Stride must always be 1.
pvmO_upkfloat/4

Prototype:
pvmO_upkfloat(-FloatList, +Nitem, +Stride, -Info)

Function:
Unpack the active message buffer into a list of float values.

Remark:
+Stride must always be 1.

pvmO_upkint/4

Prototype:
pvmO_upkint(-IntList, +Nitem, +Stride, -Info)

Function:
Unpack the active message buffer into a list of int values.

Remark:
+Stride must always be 1.

pvmO_upklong/4

Prototype:
pvmO_upklong(-LongList, +Nitem, +Stride, -Info)

Function:
Unpack the active message buffer into a list of long values.

Remark:
+Stride must always be 1.

pvmO_upkshort/4

Prototype:
pvmO_upkshort(-ShortList, +Nitem, +Stride, -Info)

Function:
Unpack the active message buffer into a list of short values.

Remark:
+Stride must always be 1.
pvm0_upkstr/2 Prototype:
   pvm0_upkstr(-String, -Info)

Function:
   Unpack the active message buffer with a string value.

pvm0_upkterm/2  
Prototype:
   pvm0_upkterm(-Termo, -Info)

Function:
   Unpack the active message buffer with a term value.

pvm0_joingroup/2  
Prototype:
   pvm0_joingroup(+Group, -Inum)

Function:
   Enrolls the calling process in a named group.

pvm0_lvgroup/2  
Prototype:
   pvm0_lvgroup(+Group, -Info)

Function:
   Unenrolls the calling process from a named group.

pvm0_gsize/2  
Prototype:
   pvm0_gsize(+Group, -Size)

Function:
   Returns the number of members presently in the named group.
pvm0_gettid/3
Prototype:
    pvm0_gettid(+Group, +Inum, -Tid)

Function:
    Returns the tid of the process identified by a group name and instance number.

pvm0_getinst/3
Prototype:
    pvm0_getinst(+Group, +Tid, -Inum)

Function:
    Returns the instance number in a group of a PVM process.

pvm0_barrier/3
Prototype:
    pvm0_barrier(+Group, +Count, -Info)

Function:
    Blocks the calling process until all processes in a group have called it.

pvm0_bcast/3
Prototype:
    pvm0_bcast(+Group, +MsgTag, -Info)

Function:
    Broadcasts the data in the active message buffer to a group of processes.

Acknowledgments. This research was supported in part by Programa Ciência, project PROLOPPE contract PRA XIS / 3 / 3.1 / TIT / 24 / 94 and DECEERIP PAPIPRO, contract no. P-005.
References


