

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>  | <b>2</b>  |
| <b>2</b> | <b>PACX-MPI for Users</b>  | <b>3</b>  |
| 2.1      | Installing PACX-MPI . . . . .  | 3         |
| 2.1.1    | Receiving the PACX-MPI Library . . . . .   | 3         |
| 2.1.2    | Configuring PACX-MPI . . . . .   | 3         |
| 2.1.3    | Problems with <code>configure</code> . . . . .   | 4         |
| 2.1.4    | Compiling PACX-MPI . . . . .   | 5         |
| 2.1.5    | Installing PACX-MPI . . . . .  | 5         |
| 2.2      | How to compile an application to use PACX-MPI . . . . .                                | 5         |
| 2.2.1    | Compiling a C-Application . . . . .  | 5         |
| 2.2.2    | Compiling a Fortran-Application . . . . .  | 6         |
| 2.2.3    | Using the Profiling Interface . . . . .  | 6         |
| 2.2.4    | Configuring the Metacomputer . . . . .   | 6         |
| 2.3      | Encryption using SSL . . . . .   | 7         |
| 2.4      | Startup . . . . .  | 7         |
| 2.4.1    | Automatic Startup . . . . .  | 8         |
| 2.4.2    | Startup by hand . . . . .  | 8         |
| 2.4.3    | Server startup . . . . .   | 8         |
| 2.5      | Environment Variables . . . . .  | 9         |
| 2.6      | Creating a GLOBUS-compliant version of PACX-MPI . . . . .                              | 10        |
| 2.7      | Known Problem . . . . .  | 10        |
| <b>3</b> | <b>PACX-MPI for Developers</b>   | <b>11</b> |
| 3.1      | External Datastructures . . . . .  | 11        |
| 3.2      | Internal Datastructures . . . . .  | 11        |
| 3.2.1    | Internal Structures for <code>MPI_Comm</code> , <code>MPI_Group</code> et al . . . . . | 11        |
| 3.2.2    | <code>CMD_PACKET</code> . . . . .  | 13        |
| <b>A</b> | <b>Configuration of supported hardware</b>   | <b>15</b> |

# 1 Introduction

PACX-MPI is an implementation of the Message Passing standard MPI, optimized for Metacomputing. The major goal of the library is to make MPI applications run on a cluster of MPP's and PVP's without any changes in the sources and by fully exploiting the communication subsystem of each machine. To reach this goal, PACX-MPI makes use of the vendor MPI library on the systems, since this is currently the fastest portable API to the communication subsystem of each machine. This document discusses the usage and the installation of PACX-MPI. Current stable release is PACX-MPI 4.1. The technical background is only discussed briefly, please refer to the literature listed below. PACX-MPI was developed originally 1995 to couple an Intel Paragon and a Cray YMP. 1996 the project was extended to enable the coupling of two machines of the same type, e.g. two Intel Paragons or two Cray T3E's. Since 1997 the working group Parallel and Distributed Systems is working at the HLRS on the development of PACX-MPI.

## References

1. Edgar Gabriel, Michael Resch, Thomas Beisel, and Rainer Keller: *Distributed computing in a heterogenous computing environment*, in Vassil Alexandrov, Jack Dongarra (Eds.) in 'Recent advances in Parallel Virtual Machine and Message Passing Interface', pp. 180-188, Springer, 1998.
2. Michael Resch, Thomas Beisel, Holger Berger, Katrin Bidmon, Edgar Gabriel, Rainer Keller, and Dirk Rantzau: *Clustering T3Es for Metacomputing Applications* to appear at CUG98.
3. Edgar Gabriel, Michael Resch, and Roland Rühle: *Implementing MPI with Optimized Algorithms for Metacomputing*, Proc. Message Passing Interface Developer's and User's Conference (MPIDC'99), pp. 31-41, Atlanta, GA, March 10-12, 1999.

## 2 PACX-MPI for Users

In this part we give a brief overview on how to receive, configure and install the PACX-MPI library and how to compile and link an application, using the PACX-MPI library.

The given information in this document is valid for PACX-MPI version 4.1. If you want to get some information about an older version of PACX-MPI, please contact `PACX-MPI@h1rs.de`.

### 2.1 Installing PACX-MPI

Currently, PACX-MPI 4.1 is tested and known to run on the following systems:

- Cray T3E
- NEC-SX4 and NEC-SX5
- Hitachi SR2201 and Hitachi SR8000
- IBM RS6000/SP
- SGI platforms
- SUN platforms
- Alpha platforms
- PC-clusters with LINUX

#### 2.1.1 Receiving the PACX-MPI Library

In order to use PACX-MPI, you first have to download a license agreement, available at <http://www.h1rs.de/structure/organisation/pds/projects/pacx-mpi>.

The license may be received for free. As soon as we receive the license agreement, we install an account for you at the same download page, so that you will always have the possibility to download the most recent version of PACX-MPI.

Now, unzip and extract the software with:

```
gunzip pacx4.1.tar.gz
tar -xvf pacx4.1.tar
```

This will create a directory called `PACX`.

#### 2.1.2 Configuring PACX-MPI

First, start `./configure --help` to see, what kind of switches and options have been implemented to support different versions of PACX. The most important among these are:

- `--with-mpi-dir=MPIDIR`
- `--with-mpi-lib-dir=dir`
- `--with-mpi-inc-dir=dir`
- `--enable-debug`
- `--enable-compression`

- `--enable-conversion`

The first option is probably the most important one: You may specify the location of your MPI-Library. It defaults to `/usr/local/mpich`. In order to change it, you may specify

```
./configure --with-mpi-dir=/usr/local/mpi
```

or something similar.

Now, if your version of MPI just installs `libmpi.a` in a special directory, without putting a link in `MPIDIR/lib`, you may have to specify this directory with `--with-mpi-lib-dir=dir` and adding this parameter to `./configure`. Same goes for `--with-mpi-inc-dir=dir`.

With `--enable-debug`, you may turn on debugging – it’s a default anyway. If you don’t care for debugging output or want to produce a smaller library, you may specify `--disable-debug`.

The switch `--enable-compression` is a way to increase transfer-rates on slow links, being off as default. If you enable this, you will additionally need the `lzo` library.

If you you want to turn on data conversion, e.g. to support heterogeneous clusters, use the `--enable-conversion`. It is **NOT** enabled as default.

Some standard configurations which are used by our developers are also stored in the File `CONFIGURE_EXAMPLES`.

### 2.1.3 Problems with configure

Having multiple compilers is still a problem for `configure`. If `configure` does stop when checking the compiler, just set the compiler you want to use at the command-prompt using environment variables. Important environment variables are:

```
export CC=<C-COMPILER>
export CXX=<C++-COMPILER>
export F77=<F77-COMPILER>
export YACC=<yacc-parser>
```

You may omit one of those, if you don’t experience any problems. Please make sure, that the Fortran compiler works with object code produced by your C-compiler. When `configure` exits without any errors, you’re all set up !

Some problems occur on IBM RS6000/SP systems, since you will have to use `cc` to compile the library and to pass the `configure` script, but later on you will need `mpcc` to create an executable. There is currently no clean way how to handle this in PACX-MPI. So please specify for `configure` the C-compiler to `cc` and change later on in the test directory in the Makefiles the variable `CC` to `mpcc`.

Some other problems may occur when using MPICH-1.1.2 as the native MPI version. MPICH-1.1.2 uses two different paths for architecture dependent and architecture independent files. Therefore, the user of MPICH-1.1.2 has to specify TWO different include-directories. With this feature, PACX-MPI has some problems, since we only support ONE include path for the native MPI. Possible solutions to overcome this problem are

1. Use MPICH-1.2: It is already available since a few month. In MPICH-1.2 the feature of two separate directories is removed again.
2. If you have to work for some reasons with MPICH-1.1.2, you should try to create only one include directory. For example, if you have an installation on linux with the `ch_shmem` device in `/usr/local/mpich` just execute:

```
cd /usr/local/mpich/build/LINUX/ch_shmem/include
ln -s ../../../../include/* .
```

### 2.1.4 Compiling PACX-MPI

To compile the library, you now have to type: `make`, which will create four libraries:

- `libpacx.a`: The main library of PACX-MPI, containing the basic C-interface.
- `libpacxf.a`: The Fortran-interface of PACX-MPI.
- `libppacx.a`: The Profiling Interface of PACX-MPI for C applications.
- `libppacxf.a`: The Profiling Interface of PACX-MPI for Fortran applications.

If you want to create only a part of the library, you may use the following options:

- `make cinterface` compiles only the basic routines and the C interface of PACX-MPI.
- `make fortran` compiles only the Fortran interface
- `make profile` compiles only the Profiling Interface of PACX-MPI

### 2.1.5 Installing PACX-MPI

If you now type `make install` the system will try to do the following steps:

- copy the four libraries mentioned above, e.g. `src/pacx/libpacx.a` to `${INSTALL_DIR}/lib/`.
- copy some of the header-files to `${INSTALL_DIR}/include/`.

You can specify the `INSTALL_DIR` by using the option `--prefix` at the configure-step (see section 2.1.2). Default value is `$(HOME)/WORK/PACX`. In case the directories are not existing, they will be created by the install-script.

## 2.2 How to compile an application to use PACX-MPI

This section will focus on how to compile and link an application if you want to use PACX-MPI. If you are not interested in the details, you can change into the test-directory. There you get an example of how to compile an application written in C by typing:

```
make t1
```

and for a Fortran-application by typing:

```
make tf1
```

The new version 4.1 supports partially also the idea of using a script for compiling an application. To compile a C-application, the user has to type

```
${PACX-DIR}/bin/pacxcc -o t1 t1.c
```

or for a Fortran-application

```
${PACX-DIR}/bin/pacxfc -o tf1 tf1.f.
```

This might not work in all cases, e.g. when using the profiling interface of MPI.

### 2.2.1 Compiling a C-Application

For applications written in C it is important, that the program is using the MPI-header file provided by the PACX-MPI library. For this, you have to list the include-path of PACX-MPI before the include-path of the native MPI. Additionally you have to link the `pacx`-library to your application, too. An example may then look like this:

```
cc -o myprog mprog.c -I PACX_DIR/include -I MPI_DIR/include -L PACX_DIR/lib  
-L MPI_DIR/lib -lpacx -lmpi
```

with `PACX_DIR` being the directory where PACX-MPI is installed and `MPI_DIR` being the directory of the native MPI implementation.

### 2.2.2 Compiling a Fortran-Application

For Fortran-applications PACX-MPI does also provide an own header-file. The file `mpif.h` is created from the header file of the vendor MPI library during compile time by resetting some parameters. Thus, the assumptions of the chapter before about the order of the include path holds for Fortran applications, too.

Another important point for applications written in Fortran is, that the fortran-interface of the PACX-MPI library is linked before the native MPI-library. Therefore the directory, where the pacx-library is lying has to be listed before the directory of the native MPI library. Additionally you have to link both PACX-MPI libraries to your application, say `libpacxf.a` and `libpacx.a`:

```
f90 -o myprog myprog.f -I PACX_DIR/include -I MPI_DIR/include
-L PACX_DIR/lib -L MPI_DIR/lib -lpacxf -lpacx -lmpi
```

### 2.2.3 Using the Profiling Interface

To use the profiling interface for an application written in C, your compile and link path should look like follows:

```
cc -o myprog myprog.c -IPACX_DIR/include -I MPI_DIR/include -lmyprofile-lib
-L PACX_DIR/lib -L MPI_DIR/lib -lpacx -lmpi
```

If your application is written in Fortran and you want to use the profiling interface, the line is somewhat more complicated (will be simplified in the next release):

```
f90 -o myprog myprog.f -I PACX_DIR/include -I MPI_DIR/include
-L PACX_DIR/lib -L MPI_DIR/lib -lmyprofile-lib -lpacxf -lpacx -lmpi
```

### 2.2.4 Configuring the Metacomputer

To make the application run on a cluster of MPPs, PACX-MPI needs some information about the configuration. This includes the names of all machines and the number of nodes used on each machine. There are different possibilities to describe the configuration of a metacomputer. PACX-MPI uses configuration files to describe the metacomputer.

The configuration of the metacomputer is split into two parts: a configuration file (`.hostfile`) describes the participating machines, the number of nodes and optionally the startup - command for these machines. Therefore the configuration file contains for each machine a line which looks like below:

```
<hostname> <number of nodes> <startup-command>
```

The second configuration file (`.netfile`) describes details of the network configuration and is not necessary if the user is using default values. The default network connection is a standard TCP-connection on the TCP-ports defined in the MetaMPI header-file. To make use of the ATM-interface developed in the frame of this project, the user defines this in the network configuration file. This looks like below:

```
BEGIN <hostname1>;
HOST=<hostnumber>, PROTOCOL=<protocol>, <ATTRIBUTES>=<attributes>;
END;
```

For each machine having a non-default connection, the user defines a chapter which begins with `BEGIN` and ends with an `END` statement. Inside this section, the user can define the connection to other hosts, specifying the protocol and optionally some protocol-dependent attributes. The same specification has to be done for the corresponding host, although some consistency checks between both machines are made.

For the TCP-protocol, the following attributes are currently supported:

- **PORT**: set the start port number for this machine

**EXAMPLE:**

For a configuration of two machines called `host1` and `host2`, using 64 nodes on both machines the `hostfile` looks like follows:

```
host1 64
host2 64
```

If the hosts don't want to use the default PACX-MPI ports, but instead a sequence of ports starting with the port number 20000, the according `.netfile` looks like follows:

```
BEGIN host1;
  HOST=2, PROTOCOL=tcp, PORT=20000;
END;
BEGIN host2;
  HOST=1, PROTOCOL=tcp, PORT=20000;
END;
```

### 2.3 Encryption using SSL

PACX-MPI may be configured to use strong encryption provided by a SSL. There are several different versions, OpenSSL is recommended (<http://www.openssl.org>).

In order to use this extension, the user first has to make sure, that OpenSSL exists on his system. Then, `configure` is told to include checks for SSL with the flags `--enable-ssl --with-ssl-dir=<DIR>`.

The following compile of the PACX-MPI-library should run fine.

When compiling and linking the user program, the user should add the linker flags `-lssl -lcrypto`.

The only difference to get it to communicate over SSL is to specify `SSL` as `PROTOCOL` in the `.netfile` for the desired connection. The user should also provide a `cert.pem` and a `key.pem`

Unfortunately, the user currently gets asked for the correct key as a password, which then gets compared to the one in `key.pem` – twice, once for `PACX_in_server` and once for `PACX_out_server`. This is a restriction which hopefully gets abandoned in the future.

### 2.4 Startup

PACX-MPI handles two different ways of startups: for machines, where each node has the same network-address (e.g. Cray T3E, SGI Origin, NEC SX4), MetaMPI can set up directly the connection between the different machines, since the name of the machine can be mapped directly to a network address.

On other machines (e.g. IBM RS6000/SP, Fujitsu VPP 5000) where each node has a different network-address. Furthermore, the user sometimes cannot influence on which node his program will be executed. The network-address of the daemon-nodes is therefore not known before the application has been started, and the connection cannot be set up. For this case, another startup-procedure is necessary. The decision was, that each machine connects first a startup-server and sends the network-addresses of its daemons to this server. After all machines registered at the server, the server distributes the addresses again to all participating machines. Using this procedure, only the startup-server needs to have a fixed network-address. In the second step, the connection between the different machines can be established.

This protocol is a light-weighted version of the startup procedure proposed in the IMPI-draft.

For the first of the both possibilities, PACX-MPI offers on most platforms two different ways how to start an application, an automatic startup and a startup by hand. These two methods are described in details in the next two sections.

#### 2.4.1 Automatic Startup

For an automatic startup you have to add the startup command for each machine. The basic idea is, that you start your application on the first machine, and this machine will start your application on the other hosts. Basically your startup command in the hostfile has to look like this:

```
rsh host2 mpirun -np 102 ./myprog
```

but please remember, that you will often have to tell the host, where the rsh-command is residing, that you will have to go into your workig directory etc... So then your startup command will look like this:

```
/bin/rsh host2 cd $HOME; /usr/bin/mpirun -np 102 ./myprog
```

Thus the hostfile from the example above will contain two commands and look like in the example here:

```
host1      100
host2      100 (/bin/rsh host2.rus.uni-stuttgart.de cd ${HOME};
              /usr/bin/mpirun -np 102 ./myprog )
host3      100 (/bin/rsh host3.rus.uni-stuttgart.de cd ${HOME};
              /usr/bin/mpirun -np 102 ./myprog)
```

Another important point is, that you have to add on each machine two additional nodes to your application, which will handle the communication between the machines. This means, if you want to use 100 nodes on the first machine for your application, you will have to start it with 102 nodes. That's why in the hostfile above you see after the mpirun command as option for the number of processors 102, even if there are 100 nodes for your application specified before.

#### 2.4.2 Startup by hand

If you want to start every partition by hand, your hostfile has to look like in the section about creating a hostfile. Then you start your application on the first machine with 102 nodes:

```
mpirun -np 102 ./myprog
```

After a while you get from this machine a confirmation message, that you may start now your application on the second host, and which port-numbers will be used for the connection with the second host.

You may start now in another window your application on the second host, and both machines are establishing the connection.

Now you have to start the application on the third machine in another window, ...

#### 2.4.3 Server startup

On architectures like the IBM SP2 or the Fujitsu VPPs the nodes of the parallel computer do not share a single internet address. Therefore it is not known before a programm has started under which address this programm can be contacted. On the other side this information is needed to couple computers with PACX-MPI. The solution is to have one fixpoint: a computer with a fixed network address that gathers the informations from the different partitions/clients after they have been started and distributes this information: the so called startupserver.

**Usage:** `startupserver #clients`

**Example:** You want to start two partitions. One on a SP2 with 32 nodes and another on a fujitsu with 16 nodes.

The clients need the information to use a startupserver in their hostfile. This is denoted by the keyword "Server" in the first row of the first uncommented line.

**Server** `startupservername.startupserverdomain Rank`

Rank denotes the Rank of the client: 0 for the first client, 1 for the second.

**Known Problems:** On some hosts `gethostname()` does not return the full qualified hostname. The hostname without the domain is of course not enough to connect from a remote site. A workaround is to set the environment variable `LOCALDOMAIN` to the value of your local domain. On some platforms this may already be done ( check manpage of resolver or your environment). This workaround works only with PACX version 3.7.6 or later.

**Compiling the startupserver:** Type in the top-level PACX-MPI directory

```
make startupserver
```

For your convenience, we also provide starting with this version a JAVA implementation of the startupserver. You can use this startup-server on any host with a java execution environment using the command

```
java -jar startupserver.jar
```

## 2.5 Environment Variables

PACX-MPI makes use of different environment variables during runtime. In this chapter we want to list the most important ones.

- **LOCALDOMAIN:** if the a call to `gethostname` does not return a full qualified hostname, PACX-MPI tries to determine the name of the domain by using this variable (see also the resolver man pages).
- **PACX\_DEBUG\_NODE:** if some additional information is needed about the behaviour of the application, and PACX-MPI has been configured with the `--enable-debug` option (default), then you can get additional output by setting this environment variable to the nodes, which should print debugging information. Setting it to -1 will make all process print debugging output. If You want to have several nodes printing debugging information, specify the nodes in a comma-seperated list; `PACX_DEBUG_NODE=1,2` would have MPI processes one and two output the debugging information. In most cases, the amount of information produced by this setting will be too much for the casual PACX-MPI user.
- **PACX\_TRACE:** if this variable is set to any value, you get information of traced functions of PACX-MPI.
- **PACX\_DEBUG\_DDD:** With this environment variable, one tells PACX-MPI to print the process-id of the MPI-process and wait the specified number of seconds, in order to attach with your favourite debugger.
- **PACX\_TCP\_BUFFER:** this environment variable sets the TCP-Buffer size of PACX-MPI.

- PACX\_HOSTFILE: set the name of the hostfile (default: `.hostfile`).
- PACX\_NETFILE: set the name of the netfile (default: `.netfile`).
- PACX\_PATH: search for `.hostfile` and `.netfile` in the following directories, separated by colons.
- PACX\_HOST\_ALIAS: The computer will also be recognised if this name is specified in the `.hostfile`. Important if job is running on a well known node of a HPC (like SP2), where every node has different hostname.
- PACX\_DYNAMIC\_PORTS: (De-)/Activate support for symanic port allocation. Where dynamic means the in respect with a given port range. This only works in combination with the startup daemon.
- GLOBUS\_TCP\_PORT\_RANGE: If PACX\_DYNAMIC\_PORTS is enabled then the GLOBUS\_TCP\_PORT\_RANGE will be checked for a valid port range.

## 2.6 Creating a GLOBUS-compliant version of PACX-MPI

PACX-MPI implements in this version a method to use the GLOBUS-startup mechanisms. To use this startup-method, the following things have to be done:

- Configure PACX-MPI with the option `-enable-globus`
- Compile the library
- Contact the GLOBUS-team to receive the so-called BNR-library, and a Makefile, which automatically links all required libraries to your application.
- Follow the general GLOBUS-mechanisms to start the application on all the machines. No `.hostfile` is required when using this method

We tested this solution with the GLOBUS-team on SGI-platforms up to now, but there is no reason for having any problems on other platforms. In case you have any questions regarding this method, don't hesitate to contact us.

## 2.7 Known Problem

When using MPICH with the p4-device as the local MPI-environment, Fortran applications won't work correctly. This is due to the fact, that PACX-MPI does NOT recover the calling arguments, and doesn't pass them correctly to the C-version of `MPI_Init`. Currently, we have no plans to fix this problem, since the MPI-2 standard requires MPI-implementations nevertheless to work without `argc` and `argv...`

On some IBM RS6000/SP systems we encountered problems with the signal-based MPI libraries, since they interrupted system-calls in PACX-MPI. While PACX-MPI could recover from these interrupts on some hosts, it did not work on all machines. Please use therefore the multi-threaded MPI library from IBM.

Results of Reduce operations are on some platforms not correct for the operation `MPI_LXOR` and `MPI_BXOR`.

## 3 PACX-MPI for Developers

When developing for PACX-MPI, one has to know the internal structures. The library tries to hide these data structures from the user as best as possible.

### 3.1 External Datastructures

Data which is stored in user space and has to be used internally in the PACX-MPI library, as well as the need to support the language binding for Fortran and C, suggests to use integers. The following data is given to the user as integer:

- `MPI_Comm`
- `MPI_Group`
- `MPI_Request`
- `MPI_Datatype`
- `MPI_Op`
- `MPI_File`
- `MPI_Info`

Other data which is not further processed internally, may be visible to the outside as structure. These structures are:

- `PACX_qos_enum`
- `PACX_qos_struct`
- `PACX_Status`

The definition for these may be found in the file `include/pacxobjects_extern.h`.

### 3.2 Internal Datastructures

The main internal structures are stored in the file `pacx.h` and `pacxobjects_intern.h`, while structures, visible to the user are stored in `pacxobjects_extern.h`.

#### 3.2.1 Internal Structures for `MPI_Comm`, `MPI_Group` et al

User data like the following

- `MPI_Comm`
- `MPI_Group`
- `MPI_Request`
- `MPI_Datatype`
- `MPI_Op`

are represented as Integers and used internally as index into separate arrays (`PACX_carray`, `PACX_garray`, ...) of size `PACX_carray_dim`, `PACX_garray_dim` and so on.

For consistency, we will first explain the `PACX_Group_struct`:

```

int          id; /* Position in the array */
BOOL        in_use;
BOOL        active; /* is it still allowed to start a communication ? */
int         counter; /* counts the number of references to this object */

MPI_Group   group; /* local part of the group */
int         nodes; /* total number of nodes in array */
int         * array; /* array[i] is rank of process i in PACX_COMM_WORLD, NOT always INCREASING */
int         local_nodes; /* number of nodes in local_array */
int * local_array; /* local_array[i] contains rank of local process in PACX_COMM_WORLD */
int         num_hosts; /* number of hosts involved in this group */
int         * roots; /* roots[i] is rank of process (in THIS COMM) on comp i, NOT always INCREASING */

```

- **id**: This states the position in the `PACX_garray` – this is constant all the time (may be removed).
- **in\_use**: This boolean flag shows, whether this entry is currently in use.
- **active**: This flag states, whether communication may be started within this group.
- **counter**: Counts the number of references to this very group.
- **group**: The `MPI_Group` of the local nodes of this group.
- **nodes**: Total number of nodes in the group (and the following array).
- **array**: The integer `array[i]` is the rank of process `i` in `PACX_COMM_WORLD`. Please note, that this array is **not** always increasing, because the order of the ranks may be 273 553 97 3 0 3 0 0 32188 427236 46220 246968 0 0 0 0 277 shuffled with the routine `PACX_Group_include`.
- **local\_nodes**: The number of local nodes in this array.
- **local\_array**: Like `array`, the `local_array[i]` contains the rank of local process in `PACX_COMM_WORLD`.
- **num\_hosts**: The number of hosts involved in this group.
- **roots**: The integer `roots[i]` is the rank of the local root process (in **this** group) on host `i`. Please note, that this array is also **not** increasing, the local root of host `i` may be a process with smaller rank, than the local root of host `i+1`.

The `PACX_Comm_struct` then looks like:

```

typedef struct {
    int          id; /* Position in the carray */
    BOOL        in_use;
    BOOL        active; /* is it still allowed to start a communication ? */
    int         counter; /* counts the number of references to this object */

    MPI_Comm    comm; /* The local part of the communicator */
    MPI_Comm    shadow; /* Shadow communicator for coll. operations */

    unsigned    comm_id; /* Global crc-id */
    int         level; /* Second part of the global id */
    PACX_Group_struct group; /* The group describing the configuration */
    int         type; /* intra- or inter communicator ? */

```

```

/* This part is to handle the topology functions of MPI */
int          topo_type; /* type of topology: PACX_UNDEFINED, PACX_CART, PACX_GRAPH */
union topo {
  struct cart {
    int          ndims; /* number of dimensions */
    int          * dims; /* array containing the size of each dir. */
    int          * periods; /* periodicity of each dir. */
    int          reorder; /* reordering (true or false ) IGNORED! */
  } cart;
  struct graph {
    int          nnodes; /* MUST BE THE SAME AS group.nodes */
    int          * index; /* array containing the number of nodes in edges */
    int          * edges; /* array with representation of flat graph */
    int          reorder; /* whether we may reorder ! UNNECESSARY && IGNORED!*/
  } graph;
} topo;

PACX_qos_struct      qos; /* type of QoS-setting associated with comm. */

/*
** For the inter-communicators
*/

PACX_Group_struct r_group; /* remote group */
int          l_leader; /* rank of local leader in the peer-comm */
int          r_leader; /* rank of remote leader in the peer-comm */
int          tag; /* Tag used for the comm-setup */
PACX_Comm      peer_comm; /* Peer communicator used for the setup */
MPI_Comm      icomm; /* This is the intra-communicator used
to setup the inter-comm. Will be used
for local coll. operations (in contrary
to the shadow comm. which is used for
pt2pt operations for implementing coll ops */
  BOOL          mpi2; /* do we have mpi-2 intercomm (this means
are all processes in the same MPI_COMM_WORLD?)
if false, we have a MPI-1 intercomm */
} PACX_Comm_struct;

```

### 3.2.2 CMD\_PACKET

The so called CMD\_PACKET is used for the communication with the two daemons. First this packet and then additional data, if any is sent. The structure contains the following definitions:

```

typedef struct {
  pacx_uint32 magic1;      /* always 1234567 */
  pacx_uint32 magic2;      /* always 8909876 */
  pacx_uint32 cmd;         /* type of command, e.g. PACX_COMMANDS */
  pacx_uint32 exec;        /* Cmd to execute (send or receive ) */
  pacx_uint32 count;       /* Number of elements */
  pacx_uint32 datatype;    /* type of datatype, e.g. MPI_REAL ... */
  pacx_uint32 datalen;     /* Length of the datatype */
  pacx_uint32 tag;         /* message tag */

```

```

pacx_uint32 dest;          /* number of node to receive this packet */
pacx_uint32 source;       /* # of node starting the command */
pacx_uint32 op;           /* Operation (reduce for example) */
pacx_uint32 comm_id;      /* index of communicator */ /* ONLY UNSIGNED ?! */
pacx_uint32 comm_level;   /* for identical comms */
pacx_uint32 request_id;   /* For synchronous sends */
pacx_uint32 s_len;        /* # data bytes will arrive after cmd */
pacx_uint32 k_len;        /* # compressed data */
} CMD_PACKET;

```

The fields are in detail:

- magic1: Always the number 1234567.
- magic2: Always the number 8909876.
- cmd: The command issued – must be one of type PACX\_COMMAND (see file pacx.h).
- exec: The command to be executed – either \_\_PACX\_SEND or \_\_PACX\_RECV of type PACX\_COMMAND.
- count: The number of elements to send.
- datatype: The type of the following data to be send. This really is the so called `sendinfo` of the `PACX_darray` (which **always(?)** corresponds to the `id`).
- datalen: This is the length of the datatype – as far as I can see, this is only returned to the user in the `MPI_Status`. Other than that, it doesn't seem to be used anymore.
- tag:
- dest:
- source:
- op:
- comm\_id:
- comm\_level:
- request\_id:
- s\_len:
- k\_len:

When communication outside the host is performed, the `CMD_PACKET` will be converted into external data-representation. Formerly the XDR-routines of the RPC-package was used. Now, the routine `PACX_convert_32_big_little` is used for this purpose.

| Test auf:           | i386         | ia64         | CrayT3E       | NEC-SX5      | Hitachi SR8k | Origin O2k  | IBM SP2   |
|---------------------|--------------|--------------|---------------|--------------|--------------|-------------|-----------|
| Operating System    | Linux-2.4.18 | Linux-2.4.17 | unicosmk2.0.5 | SuperUX 11.1 | HI-UX/MPP    | Irix64 6.5  | AIX-5.1   |
| Byte-Order          | little       | little       | big           | big          | big          | big         | big       |
| char                | 1            | 1            | 1             | 1            | 1            | 1           | 1         |
| short               | 2            | 2            | 4             | 2            | 2            | 2           | 2         |
| int                 | 4            | 4            | 8             | 4            | 4            | 4           | 4         |
| long                | 4            | 8            | 8             | 8            | 4            | 4           | 4         |
| long long           | 8            | 8            | 8             | 8            | 8            | 8           | 8         |
| unsigned short      | 2            | 2            | 4             | 2            | 2            | 2           | 2         |
| unsigned int        | 4            | 4            | 8             | 4            | 4            | 4           | 4         |
| unsigned long       | 4            | 8            | 8             | 8            | 4            | 4           | 4         |
| unsigned long long  | 8            | 8            | 8             | 8            | 8            | 8           | 8         |
| float               | 4            | 4            | 4             | 4            | 4            | 4           | 4         |
| double              | 8            | 8            | 8             | 8            | 8            | 8           | 8         |
| long double         | 12 (16)      | 16           | 8             | 16           | 16           | 16          | 8/16      |
| INTEGER             | 4            | 4            | 8             | 4            | 4            | 4           | 4         |
| REAL                | 4            | 4            | 8             | 4            | 4            | 4           | 4         |
| DOUBLE PRECISION    | 8            | 8            | 8             | 8            | 8            | 8           | 8         |
| COMPLEX             | 8            | 8            | 16            | 8            | 8            | 8           | 8         |
| DOUBLE COMPLEX      | 16           | 16           | 16            | 16           | 16           | 16          | 16        |
| LOGICAL             | 4            | 4            | 8             | 4            | 4            | 4           | 4         |
| CHARACTER           | 1            | 1            | x (1)         | 1            | 1            | 1           | 1         |
| MPI_FLOAT_INT       | 8            | 8            | 16            | 8            | 8            | 8           | 8         |
| MPI_DOUBLE_INT      | 12           | 16           | 16            | 16           | 16           | 16          | 16        |
| MPI_LONG_INT        | 8            | 16           | 16            | 16           | 8            | 8           | 8         |
| MPI_2INT            | 8            | 8            | 16            | 8            | 8            | 8           | 8         |
| MPI_SHORT_INT       | 8            | 8            | 16            | 8            | 8            | 8           | 8         |
| MPI_LONG_DOUBLE_INT | 16           | 32           | 16            | 32           | 24           | 32          | 16        |
| MPI_LONG_LONG_INT   | yes          | yes          | yes           | yes          | yes          | yes         | no?       |
| MPI_INTEGER1        | yes          | no           | yes           | no           | yes          | yes         | yes       |
| MPI_INTEGER2        | yes          | yes          | yes           | yes          | yes          | yes         | yes       |
| MPI_INTEGER4        | yes          | yes          | yes           | yes          | yes          | yes         | yes       |
| MPI_INTEGER8        | yes          | yes          | yes           | yes          | yes          | yes         | yes       |
| MPI_INTEGER16       | no           | no           | no            | no           | no           | no          | no        |
| MPI_REAL2           | no           | no           | no            | no           | no           | no          | no        |
| MPI_REAL4           | yes          | yes          | yes           | yes          | yes          | yes         | yes       |
| MPI_REAL8           | yes          | yes          | yes           | yes          | yes          | yes         | yes       |
| MPI_REAL16          | no           | yes          | no?           | yes          | yes          | yes         | yes       |
| CC                  | gcc-3.1      | ecc 7.0      | cc 6.5.0.1    | C Rev. 4.94  | cc           | cc 7.30     | xlc/mpcc  |
| FC                  | g77 3.1      | efc 7.0      | f90 3.5.0.1   | f90 Rev.253  | f90          | f90 7.30    | xlf/mpf90 |
| MPI used:           | mpich-1.2.3  | MPI/ex       | MPT-1.4.0.2p  | MPI/sx       | Hitachi-MPI  | SGI MPI 3.2 | poe       |
| MPI-Version         | 1.2          | 1.2          | 1.2           | 2.0          | 2.0          | 1.2         | 1.2       |
| MPI thread-safety   | None         | serialized   | None          | None?        | None?        | None?       | Yes       |
| MPI2 parallel IO    | yes (ROMIO)  | yes          | yes           | yes          | yes          | yes         | yes?      |
| MPI2 one-sided comm |              |              |               | yes          | yes          | no          | yes?      |
| MPI2 ext. coll.     |              |              |               | yes          | yes          | yes?        | yes?      |
| pthread-Version     | final        | final        | final         | final        | final        | final       | final     |

## A Configuration of supported hardware

This list of hardware-features was collected with the configure-script.