

MPICH-Madeleine Installer's, User's and Developer's Guide

Nathalie Furmento

Guillaume Mercier¹

May 2006

¹runtime@labri.fr

Abstract

MPICH-Madeleine is a new free implementation of the MPI standard based on the MPICH implementation and the multi-protocol communication library called Madeleine. It aims to efficiently exploit clusters of clusters with heterogeneous networks. This manual presents an installer's, user's and developer's guide for MPICH-Madeleine.

The latest version of this document is available from the following URL: <http://runtime.bordeaux.inria.fr/mpi/manual/>.

Please note that the instructions given in this manual always reflect the latest implementation of MPICH-Madeleine which is available from the Subversion server hosted by the InriaGforge as explained in Section 2.1.2.

If you have any questions concerning MPICH-Madeleine, please, send an email to mpich-mad-users@lists.gforge.inria.fr.

Keywords: MPI, Communication, Heterogeneous networks.

Contents

1	What is MPICH-Madeleine?	3
1.1	Madeleine Basics	3
1.2	How is MPICH-Madeleine Implemented?	6
2	Installing MPICH-Madeleine	8
2.1	Getting the Source Code	8
2.1.1	Getting the Latest Stable Release	8
2.1.2	Getting the Latest Version	8
2.2	Required development tools	9
2.3	Prerequisites	9
2.4	Installing PM2	10
2.5	Installing MPICH-Madeleine	11
2.6	Summary of Environment Variables	13
2.7	Automatic Installation of MPICH-Madeleine	13
2.8	Troubleshooting	14
3	Running MPICH-Madeleine Programs	15
3.1	Basic Use	15
3.2	Using Several Networks	18
3.3	Use Case: Grid'5000 from an Individual Site	19
3.4	Use Case: Grid'5000 from Multiple Sites	20
3.4.1	Installation of the Tool Set	20
3.4.2	General comments	21
3.4.3	PM2	21
3.4.4	MPICH-Madeleine	23
3.5	Use Case: The Ping-Pong Application	24
3.5.1	Optimization Mechanisms	26
3.6	Instrumentation and Visualisation	26
3.6.1	MPE	26
3.6.2	MPICL	27
3.7	Troubleshooting	28
4	Debugging MPICH-Madeleine Programs	29
4.1	Debugger	29
4.2	Memory	30
4.3	Debugging the device	30

5	Developping MPICH-Madeleine	32
5.1	Modifying PM2	32
5.2	Modifying MPICH	33
5.3	Recompiling MPICH-Madeleine	33
	Bibliography	34
A	Testing the Installation of PM2	35
A.1	Debugging the PM2 modules	36
A.2	Debugging PM2 processes	38
A.3	Debugging Leonie	38

Chapter 1

What is MPICH-Madeleine?

MPICH-Madeleine is a free MPICH-based implementation of the MPI standard. If you are not familiar with MPI, you should first take a look at <http://www-unix.mcs.anl.gov/mpi/>. MPI is a high-level communication interface designed to provide high-performance communications on various network architectures including supercomputers and clusters of workstations (usually off-the-shelf PC's interconnected by high-speed links). Nowadays, clusters of workstations become increasingly popular thanks to the availability of many high-speed connection technologies (Gigabit-Ethernet, Myrinet, GigaNet, SCI). Furthermore, interconnecting such COW's to build heterogeneous clusters of clusters is now a hot issue. Unfortunately, no current MPI implementation supports this kind of architectures efficiently. Indeed, the only way to handle network heterogeneity is to use interoperable implementations of MPI: several MPI implementations (one per cluster) communicate with each other using an inter-MPI glue.

Our alternative proposal is to provide a true multi-protocol implementation of MPI on top of a generic and multi-protocol communication layer called *Madeleine* (version 3). *Madeleine III* is the communication sub-system of the *Parallel Multithreaded Machine* (<http://runtime.futurs.inria.fr/pm2>) runtime environment. It is especially targeted towards:

- Single clusters with several interconnection networks;
- Clusters of clusters (possibly with several interconnection networks).

1.1 Madeleine Basics

Let's explain the key concepts of *Madeleine* which are necessary to understand before using our implementation of MPI. Let's suppose that two clusters are available:

- the first cluster named `foo`, is composed of 3 nodes, named `foo0`, `foo1` and `foo2`, linked by a Myrinet network;
- the second cluster named `goo`, is composed of 4 nodes, named `goo0`, `goo1`, `goo2`, `goo3`, linked by a SCI network.

Both clusters also feature an Ethernet network (TCP) which links together all the machines of each cluster. The clusters can be seen on Figure 1.1.

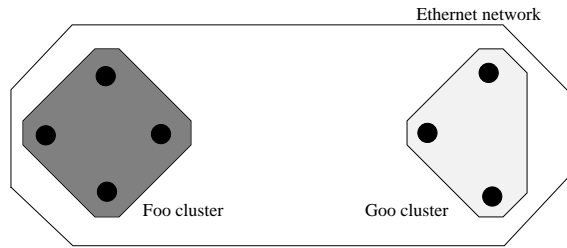


Figure 1.1: Example of interconnected clusters

Madeleine uses objects called *channels* in order to virtualize the available networks in a given configuration. There are basically two types of channels:

1. *physical channels* which are simple abstractions of real existing networks and;
2. *virtual channels* which are build above physical channels and can be used to create *heterogeneous networks*.

With our simple example, we can build three physical channels:

1. a channel build above the Myrinet network. This channel encompasses the nodes $\{f_{000}, f_{001}, f_{002}\}$;
2. a channel build above the SCI network. This channel encompasses the nodes $\{g_{000}, g_{001}, g_{002}, g_{003}\}$;
3. a channel build above the TCP network. This channel encompasses all the nodes of both clusters.

On top of these three different physical channels, we can build a virtual channel which encompasses all the nodes of the configuration. One may think that there is no difference with the TCP physical channel, but in fact the behavior of a program using the virtual channel will be totally different as Madeleine will automatically select the best available network to communicate between two nodes of this virtual channel.

Indeed, all communications occurring within the f_{00} cluster will use the Myrinet network, all communications occurring within the g_{00} cluster will use the SCI network. And if two nodes belonging to different clusters want to exchange messages, the TCP network will be used.

More complicate configurations can be expressed: if we suppose now that the node g_{003} features a SCI NIC, we can build a virtual channel over the physical channels corresponding to the Myrinet and SCI networks. In that case, we do not need (and most important: use) the TCP network. In fact, with that new configuration, from the application's point of view, all the nodes can communicate with each other. Indeed even if a node A is not *physically* connected to a node B , it can in any case send messages to it. Internally, the node g_{003} , which features both Myrinet and SCI NICs, will **forward** the Madeleine message from A to B .

How is this information given to Madeleine? The library uses configuration files (the following example files describe the configuration shown in Figure 1.2 with the node g_{003} featuring a SCI NIC):

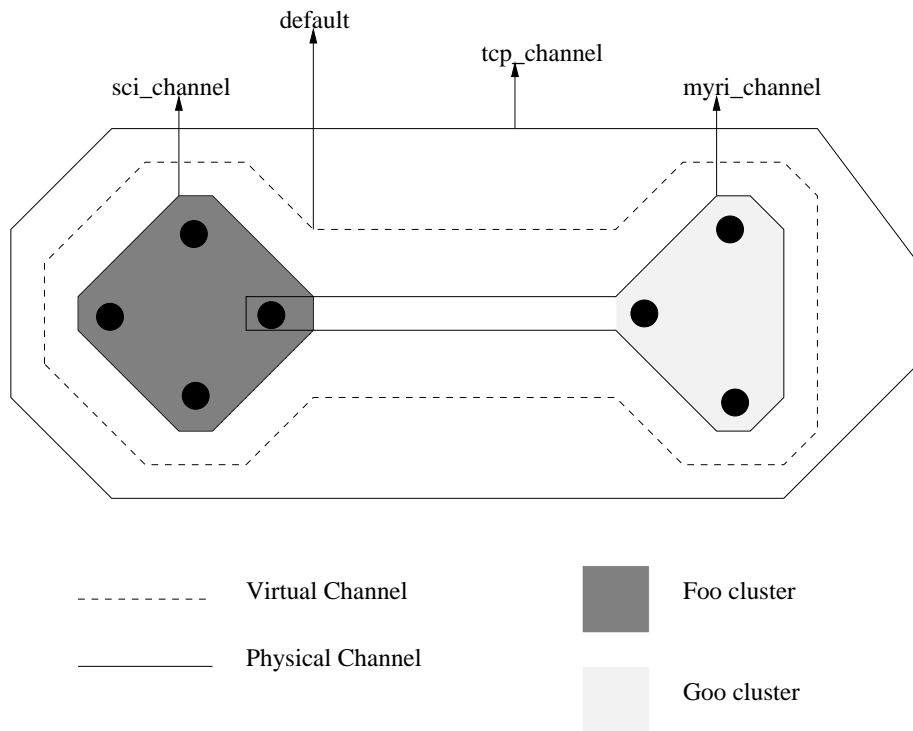


Figure 1.2: Example of configuration

- the first file, the network configuration file, named `localnet.cfg`, describes the different available networks.

```

networks : ( {
    name : tcp;
    hosts : ( foo0, foo1, foo2, goo0, goo1, goo2, goo3 );
    dev : tcp;
  }, {
    name : myrinet;
    hosts : ( foo0, foo1, foo2, goo3 );
    dev : mx;
  }, {
    name : sci;
    hosts : ( goo0, goo1, goo2, goo3 );
    dev : siscli;
  } );

```

A network is defined with its name (tag `name`), a list of machines it includes (tag `hosts`), and the identifier of the device connecting these machines (tag `dev`, should be one of the predefined identifiers recognized by Madeleine).

- the second file, the channel configuration file, named `config`, describes the channel mapping over the different nodes.

```

channels : ( {
    name : tcp_channel;
    net  : tcp;
    hosts : ( foo0, foo1, foo2, goo0, goo1, goo2, goo3 );
}, {
    name : sci_channel;
    net  : sci;
    hosts : ( goo0, goo1, goo2, goo3 );
}, {
    name : myri_channel;
    net  : myrinet;
    hosts : ( foo0, foo1, foo2, goo3 );
});
vchannels : {
    name      : default;
    channels : ( myri_channel, sci_channel );
};

```

A physical channel is defined with its name (tag name), the identifier of the network is based on (tag net which has to be defined in the network configuration file), and a list of machines it encompasses (tag hosts).

A virtual channel is defined with its name (tag name) and the list of physical channels it is build upon (tag channels).

Once a virtual channel is build, the physical channels below it are no longer visible by the application. However, it is possible to create several different Madeleine physical channels over the same physical network. Hence a physical channel can truly be seen as a *logical network* or a *network abstraction*.

1.2 How is MPICH-Madeleine Implemented?

As its name indicates, our work is based on the MPICH (<http://www-unix.mcs.anl.gov/mpi/mpich/>) implementation of MPI. Basically, the core of MPICH-Madeleine is based on a specific MPICH device, called `ch_mad`, which handles several Madeleine channels in parallel and thus allows the use of several networks at the same time within the same application. In order to do implement such a device, several threads are used: the MPI application code is executed by a particular thread and each channel is assigned its specific thread which will process all the incoming communications.

We also developed another device for handling intra-node SHMEM communications. This device uses the same concepts of `ch_mad`, that is, a thread is responsible for executing the polling of incoming communications. This device (which is included in the same directory as `ch_mad`) benefits from the advanced polling mechanisms available within the Marcel library. Figure 1.3 shows the architecture of MPICH-Madeleine.

The internal structure of MPICH should allow to design and plug a new device into the implementation without needing to modify the upper layers (mainly the Abstract Device Interface, the ADI). Unfortunately, it turned out that the development of a multi-thread MPICH device requires some modifications of the ADI and even of some of the higher level features of MPICH. This is why our Madeleine device cannot be downloaded on its own, but is part of a full, customized MPICH version.

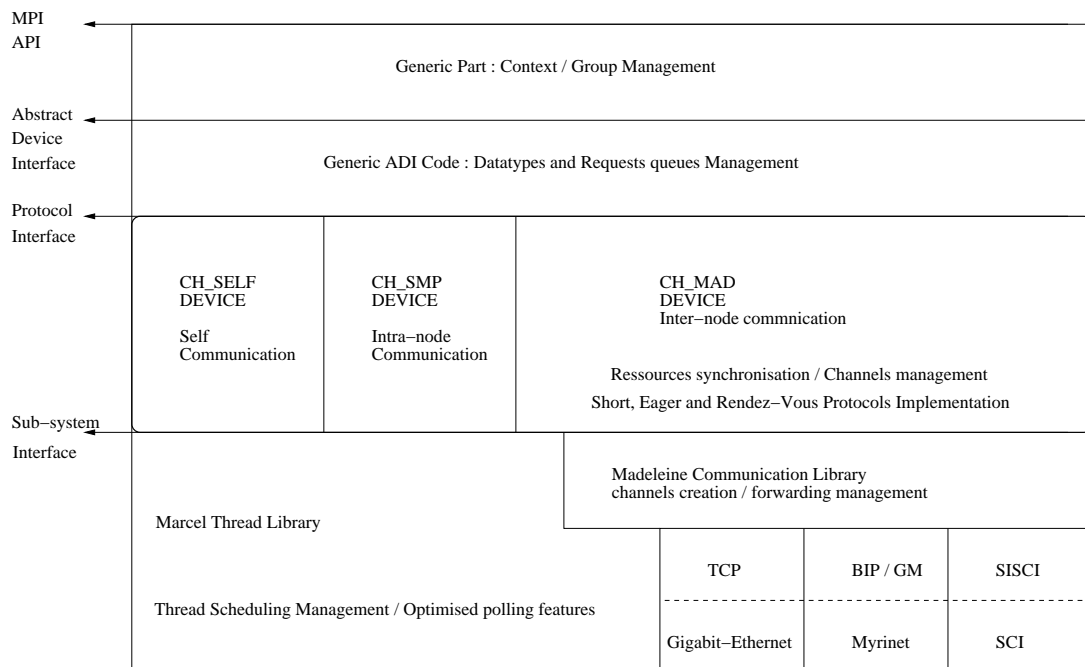


Figure 1.3: Architecture of MPICH-Madeleine

Chapter 2

Installing MPICH-Madeleine

2.1 Getting the Source Code

MPICH-Madeleine files are hosted by the project MPICH-Madeleine on the InriaGforge at <https://gforge.inria.fr/projects/mpich-mad/>.

2.1.1 Getting the Latest Stable Release

You need to download the package for MPICH-Madeleine as well as the one for PM2. The latest release for these two packages, `pm2` and `mpich-mad`, is available from the main page of the project MPICH-Madeleine at <https://gforge.inria.fr/projects/mpich-mad/> under the tab **Latest File Releases**.

Note that the latest release for the package `pm2` is also available from the main page of the project PM2 at <https://gforge.inria.fr/projects/pm2/>.

2.1.2 Getting the Latest Version

The source code for PM2 and MPICH-Madeleine is managed by a Subversion server hosted by the InriaGforge. To get the source code, you need:

1. To install the client side of the software Subversion if it is not already available on your system. The software can be obtained from <http://subversion.tigris.org/>.
2. To become a member of both projects `mpich-mad` and `pm2`. For this, you first need to get an account to the gForge server. You can then become a member of the projects by contacting one of the project administrators. The list of the project administrators is available from the main project page.

You can also choose to check out both project's SVN repository through anonymous access. In that case, you do not need to become a member, and you will have limited access to the repository.

More information on how to get a gForge account, to become a member of the specified projects, or on any other related task can be obtained from the InriaGforge at <https://gforge.inria.fr/>. The most important thing is to upload your public SSH key on the

gForge server (see the FAQ at <http://siteadmin.gforge.inria.fr/FAQ.html#Q6> for instructions).

The commands for getting the source code are given in the following sections.

2.2 Required development tools

The following development tools are required to compile MPICH-Madeleine:

- GNU C Compiler `gcc` (version 3.2 and higher).
- GNU `make` (version 3.81 and higher).
- GNU Bison (<http://www.gnu.org/software/bison/>) and Flex (<http://www.gnu.org/software/flex/>).

2.3 Prerequisites

If you plan to use Myrinet and SCI, you must get the appropriate drivers. Madeleine III supports Myrinet networks through the GM and MX drivers and SCI networks through the SISI driver. The default installation directories for these drivers are:

- `/opt/gm` for Myrinet/GM;
- `/opt/mx` for Myrinet/MX;
- `/opt/DIS` for SCI/SISI.

In case, your installation differs from these directories, you will need to set the following environment variables to point to your own installation directories:

```
% export MX_DIR=/softs/mx
% export GM_DIR=/softs/gm
% export SISI_PATH=/opt/DIS
```

The second step is then to create symbolic links from the sub-directory `.mpich-mad` in your home directory pointing to the drivers's distributions:

- A link to the GM distribution (including header files, libraries, ...) called `gm`, such as:

```
% mkdir $HOME/.mpich-mad
% cd $HOME/.mpich-mad
% ln -s /opt/gm gm
```

- AND/OR a link to the MX distribution (including header files, libraries, ...) called `mx`, such as:

```
% mkdir $HOME/.mpich-mad
% cd $HOME/.mpich-mad
% ln -s /opt/mx mx
```

- AND/OR a link to the SISCO distribution called DIS, such as:

```
% mkdir $HOME/.mpich-mad
% cd $HOME/.mpich-mad
% ln -s /opt/DIS DIS
```

The name of these links are **important**, so please respect them.

2.4 Installing PM2

Here are the required steps to install PM2.

1. Choose an installation directory for PM2, and set the environment variable PM2_ROOT to this directory.

```
% export PM2_ROOT=$HOME/soft/pm2
```

2. Go in the parent directory of the directory PM2_ROOT and either

- (a) unpack the PM2 archive;

```
% cd $HOME/soft
% tar zxf pm2-2005-03-16.tar.gz
```

If your tar does not accept the option z, use:

```
% gunzip -c pm2-2005-03-16.tar.gz | tar xf -
```

- (b) or check out the latest version from the Subversion server

- using the anonymous access.

```
% cd $HOME/soft
% svn checkout svn://scm.gforge.inria.fr/svn/pm2/trunk pm2
```

- or using your gForge account.

```
% cd $HOME/soft
% svn checkout svn+ssh://<login>@scm.gforge.inria.fr/svn/pm2/trunk pm2
```

In both cases, you should end up with a directory named pm2 as indicated by the environment variable PM2_ROOT.

3. You need to add to your PATH the directory pm2/bin.

```
% export PATH=${PM2_ROOT}/bin:${PATH}
```

4. PM2 is installed.

2.5 Installing MPICH-Madeleine

Here are the required steps to install MPICH-Madeleine.

1. Go in some directory of your choice, and either

- (a) unpack the MPICH-Madeleine archive;

```
% cd $HOME/soft
% tar zxf mpich-mad-2005-03-16.tar.gz
```

If your tar does not accept the option z, use:

```
% gunzip -c mpich-mad-2005-03-16.tar.gz | tar xf -
```

- (b) or check out the latest version from the Subversion server

- using the anonymous access.

```
% cd $HOME/soft
% svn checkout svn://scm.gforge.inria.fr/svn/mpich-mad/trunk mpich-mad
```

- or using your gForge account.

```
% cd $HOME/soft
% svn checkout
  svn+ssh://&lt;login>@scm.gforge.inria.fr/svn/mpich-mad/trunk
  mpich-mad
```

In both cases, you should end up with a directory named `mpich-mad`.

2. Choose an installation directory for MPICH-Madeleine, and set the environment variable `MPICH_MAD_ROOT` to this directory.

```
% export MPICH_MAD_ROOT=$HOME/soft/mpich-mad-install
```

3. Configure MPICH-Madeleine by issuing the following commands:

```
% cd mpich-mad
% ./configure --prefix=$MPICH_MAD_ROOT --with-arch=LINUX --with-device=ch_mad:--pm2root=
```

- PM2 is using a system of flavors to configure its different options. By default, MPICH-Madeleine is compiled using the flavor `mpi-flav`. If you wish to use another flavor, you can specify it as follows:

```
--with-device=ch_mad:--pm2root="$PM2_ROOT":--flavor="name of your flavor"
```

- If you want to enable SHMEM support, add the option `--shmem-support`.

```
--with-device=ch_mad:--pm2root="$PM2_ROOT":--shmem-support
```

SHMEM support is dynamically deployed on SMP nodes on which several MPI processes are spawned. Therefore, you can compile MPICH-Madeleine by enabling SHMEM support even if you have a set-up with both SMP and UP nodes. It will not impact the performance.

- If you want to enable fast buffering capabilities, add the option `--fast-buffer`.

```
--with-device=ch_mad:--pm2root="$PM2_ROOT":--fast-buffer
```

- The default behavior is to compile all networks support (TCP, SISC, GM; MX if available). If you DO NOT want to enable one specific network, add the following option. The network name can be any among `tcp`, `gm`, `bip`, `mx`, `sisci`, ...

```
--with-device=ch_mad:--pm2root="$PM2_ROOT":--mad3=no-"network name"
```

4. Build MPICH-Madeleine. That step might take some time.

```
% make
```

5. Install the MPICH-Madeleine commands.

```
% make install
```

6. You need to provide a network configuration file called `localnet.cfg` (see Section 1.1). This file should be located in the sub-directory `etc` of the directory you have installed MPICH-Madeleine in i.e. `$MPICH_MAD_ROOT/etc`. This file is mandatory but users can select another location by defining the appropriate environment variable (see Section 3.1).

7. Note: do NOT get rid off the PM2 source after the installation. It is still needed by users to compile and run their applications!

8. Users can add to their `PATH` the element `mpich-mad-install/bin` in order to easily access the commands `mpicc` and `mpirun`.

```
% export PATH=${MPICH_MAD_ROOT}/bin:${PATH}
```

2.6 Summary of Environment Variables

Here the list of environment variables you need to set (preferably in your login files such as `$HOME/.bashrc`) before being able to compile and execute MPICH-Madeleine applications.

```
## Directory where PM2 is installed
export PM2_ROOT=$HOME/soft/pm2

## Directory where MPICH-Madeleine is installed
export MPICH_MAD_ROOT=$HOME/soft/mpich-mad-install

## Access to the PM2 commands
export PATH=${PM2_ROOT}/bin:${PATH}

## Access to the MPI commands
export PATH=${MPICH_MAD_ROOT}/bin:${PATH}
```

2.7 Automatic Installation of MPICH-Madeleine

The tool set presented in Section 3.4.1 provides a script that allows to get the source code for PM2 and MPICH-Madeleine (or update the existing code), and to compile that code. The script has to be called as in the following example:

```
% installMpichMad.sh $HOME/soft $HOME/soft/mpich-mad-install
```

where:

- `$HOME/soft` is the parent directory in which the source files for the softwares PM2 and MPICH-Madeleine are (or will be) installed;
- `$HOME/soft/mpich-mad-install` is the directory in which MPICH-Madeleine is (or will be) installed.

The script will execute the following steps:

1. If the directory `$HOME/soft/pm2` does not exist, get the latest version from the Subversion server; otherwise update the existing version.
2. If the directory `$HOME/soft/mpich-mad` does not exist, get the latest version from the Subversion server; otherwise update the existing version.
3. Compile MPICH-Madeleine in the directory `$HOME/soft/mpich-mad-install`.

To maximize the automation of the installation, the following script can also be used to automatically copy the archive of the tool set and launch the installation script of MPICH-Madeleine on a given machine.

```
#!/bin/bash

if [ "$4" == "" ]
then
    echo "Error. Usage: $0 <mpich-mad-tools.tar.gz>
        <machine to install MPICH-Madeleine on>
        <PM2 & MPICH-Madeleine source directory>
        <MPICH-Madeleine installation directory> [-user <inria gforge login>]"
    exit 1;
fi

DIR=`basename $1 .tar.gz`

scp $1 $2:
ssh $2 "tar zxvf $1 ; ./${DIR}/installMpichMad.sh $3 $4 $5 $6 ; uname -a"
```

Here an example on how to call the script:

```
% ./installMpichMadAndTools.sh g5k-tools.tar.gz oar.sophia.grid5000.fr ~/soft ~/soft/mpi
```

2.8 Troubleshooting

You might encounter the following errors while compiling MPICH-Madeleine.

- Bison version error:

```
Generating Makefile leoparse-config.mak
building leoparse_parser.h
source/tools/leoparse_parser.y:25: unrecognized: %error-verbose

source/tools/leoparse_parser.y:25: Skipping to next %
make[1]: ***
[/home/test/MPI/MPICH_MAD/MPICH-MAD-INST/pm2/leonie/leoparse/include/leoparse_parser.h
make: *** [dot_h] Error 2
```

This error happens because of an outdated version of the Bison software. To fix it, you can either:

- Install a newest version of the Bison software ; or
- Comment the `%error-verbose` line in the `leoparse/source/tools/leoparse_parser.y` file of the PM2 software package.

- Another Bison error:

```
yylloc undefined
```

This error can be resolved by installing a newest version of the Bison software or by commenting the extern keyword preceding `YYLTYPE` `yylloc` in the `leoparse/source/tools/` file of the PM2 software package.

Chapter 3

Running MPICH-Madeleine Programs

3.1 Basic Use

The sub-directory `example/basics` of the MPICH-Madeleine source directory contains basic examples that can be used to test your MPICH-Madeleine installation. We will go through this section by using the program `cpi.c`.

- On top of the environment variables defined in Section 2.6, you need to define the following environment variable which is needed by PM2.

```
% export LEO_RSH="ssh -n -f"
```

- Compile your application with `mpicc`.

```
% cd mpich-mad/examples/basic
% $MPICH_MAD_ROOT/bin/mpicc -o cpi cpi.c
```

- You need to make sure you can connect to the machines defined in your channel using the SSH protocol. MPI applications are launched using the PM2's mechanisms to remotely connect to the machines, this requires a SSH access to the machines without password or pass-phrase. You might need to change the definition of the environment variable `LEO_RSH` to match the configuration of your SSH connection. Here an example to test your configuration is working:

```
% echo $LEO_RSH
ssh -n -f
% $LEO_RSH jack uname -a
% Linux jack 2.6.4-fkt #14 SMP Thu Sep 23 2004 i686 GNU/Linux
%
```

- We suppose for now the network configuration file `localnet.cfg` is present in the directory `$MPICH_MAD_ROOT/etc` as explained in Section 1.1. Look at the contents of that file to see which machines you can use for your application.

Specify the machines and execute the application by:

1. Using the standard MPI option `-np`

- The parameter `-machinefile` can be used to specify the file containing the list of machines you wish to use.

```
% cat mymachines
node-16.bordeaux.grid5000.fr
node-41.bordeaux.grid5000.fr
node-47.bordeaux.grid5000.fr
% $MPICH_MAD_ROOT/bin/mpirun -machinefile mymachines ...
```

- When using a OAR reservation, the application will be launched on the machines reserved by OAR. You can also specify a list of machines by setting the environment variable `MPI_HOST_FILE` to a file containing the list of the machines you wish to use.

```
% cat $OAR_NODEFILE
node-16.bordeaux.grid5000.fr
node-41.bordeaux.grid5000.fr
node-47.bordeaux.grid5000.fr
% cat $MPI_HOST_FILE
node-16.bordeaux.grid5000.fr
node-41.bordeaux.grid5000.fr
%
```

- Start your application using the command `mpirun` taking in parameter the number of processes to launch and the name of the application. The number of processes does not have to be the number of machines in the machine file (as specified above). MPICH-Madeleine will use a subset of these machines, or will start more than one process per machine accordingly.

```
% $MPICH_MAD_ROOT/bin/mpirun -np 2 cpi
```

2. Using directly the MPICH-Madeleine configuration files

- Create a channel configuration file named `config` as explained in Section 1.1. Let's suppose you are using a channel with two hosts. Your file should be similar to:

```
channels : ({
    name : dalton;
    net  : tcp;
    hosts : ( joe, jack );
});
```

- Start your application using the command `mpirun` taking in parameter the location of your channel configuration file and the name of the application.

```
% $MPICH_MAD_ROOT/bin/mpirun -config config cpi
```

- You should obtain an output similar to:

```
Process 0 of 2 on &lt;your machine>
pi is approximately 3.1415926544231318, Error is 0.0000000008333387
wall clock time = 1.094123
Process 1 of 2 on &lt;your machine>
```

Remarks:

- The default behavior of MPICH-Madeleine is to launch processes only on the nodes specified in the channel configuration file. No process is spawned on the local machine (unless it is listed in the channel configuration file)
- MPI expects to locate the executable on the local machine, and at the same location on the remote machines. In our example, the location of the executable is `$PWD/cpi` e.g. `/home/mpich-user/mpich-mad/example/basics/cpi` and this location must be valid on any machine running the application. However, with MPICH-Madeleine, the local machine is not necessarily part of the execution, and it might be unrealistic to suppose the absolute path to your home directory is exactly the same on any machine you wish to run your application on.

MPICH-Madeleine provides a mechanism allowing to overcome this restriction. When the environment variable `MPICH_MAD_NON_LOCAL_EXECUTABLE` is set to the value 1, MPICH-Madeleine will not require the executable to be present on the local machine, and will be able to locate this executable on all the remote nodes even though the absolute path to your home directory is different. There are two conditions which still need to hold:

1. The relative path from your home directory to the installation directories of PM2 and MPICH-Madeleine must be the same on all the nodes, e.g. `soft/pm2` and `soft/mpich-mad-install`.
2. The executable needs to be present in a directory PM2 will search through. We advice you to install the executable of your application in the directory `$MPICH_MAD_ROOT/pm2/mp`

Here an example of how to use this mechanism:

- On all the machines M_1, M_2, \dots, M_n running the application:

```
% mkdir $MPICH_MAD_ROOT/pm2/mpi-flav/examples/
% cd mpich-mad/examples/basic
% $MPICH_MAD_ROOT/bin/mpicc -o $MPICH_MAD_ROOT/pm2/mpi-flav/examples/cpi cpi.c
```

- On the machine M starting the application:

```
% export MPICH_MAD_NON_LOCAL_EXECUTABLE=1
% ls cpi
ls: cpi: No such file or directory
% $MPICH_MAD_ROOT/bin/mpirun -config config cpi
```

- The default network file used for launching jobs is `$MPICH_MAD_ROOT/etc/localnet.cfg`. This file has to be provided by the person installing MPICH-Madeleine. However, users can select another network file by defining the environment variable `MPI_NET_FILE` with the location of the new file, e.g.:

```
% export MPI_NET_FILE=$HOME/networks.cfg
```

- If the machines you wish to use to run your application are interconnected with more than 1 network (as for example, the nodes `foo0`, `foo1` and `foo2` in Section 1.1), then MPICH-Madeleine will by default select the first network (here TCP) to connect the nodes. To use another network, you can set the environment variable `MPICH_MAD_PROTOCOL` to the name of this network.

```
% export MPICH_MAD_PROTOCOL=mx
```

- If you wish to start your application under the debugger, you can use the option `-gdb`. All the processes of the MPI application will be launched through `gdb`, the GNU debugger.

```
% mpirun -gdb -config config test
```

3.2 Using Several Networks

How can I handle several networks at the same time in my MPI application? The important question is: "Should users know that the cluster they are using features several different networks?".

Well it depends, that is why the MPI user can deal with this issue in two ways:

1. "In Madeleine I trust": in that case, the heterogeneity of the networks is completely hidden to the MPI application.

==> You can do so by creating a unique virtual channel encompassing all the nodes of your clusters. In that case, MPICH-Madeleine will use this channel and you will never need to know what is going on underneath, in particular which network will be selected by MPICH-Madeleine to communicate between any nodes of your application.

2. "I am a big boy/girl now": for whatever reasons, your favorite MPI application has to deal with the different available networks. You wish for instance to group your processes by clusters.

==> You can do so by creating as many channels as needed (i.e. one channel by cluster). These channels can either be physical or virtual. As physical channels offer better performance, they should be preferred.

You can then access the different channels through *MPI Communicators*. When you create a configuration file, channels are sorted in the order they are declared. So, if the node `x` belongs to three different channels, `x` can send messages over the `channel[i]` by performing a send operation over the communicator `MPI_USER_COMM[i]`.

The communicator `MPI_COMM_WORLD` uses the *default channel*, which definition is mandatory and should cover all the nodes of a given configuration.

You can find an example of such an application at <http://runtime.futurs.inria.fr/mpi/ressources/sample.c>.

3.3 Use Case: Grid'5000 from an Individual Site

To execute a MPICH-Madeleine application on the Grid'5000 platform, you first need to reserve nodes using the resource manager OAR. Let's say the application needs six nodes to execute, the nodes can be reserved by using the following command:

```
% oarsub -l nodes=6,walltime=5 -I
```

- The option `-I` of the command `oarsub` requests an interactive mode and will give you a shell on the first reserved node. From that node, you can access the other nodes of your reservation.
- The optional option `walltime` allows to specify the length of the reservation. Here we request five hours.

Once you are connected to the first reserved node, you need to create the MPICH-Madeleine configuration files. This is done by using the script `bin/generateConfigFiles.sh` present in the installation directory of MPICH-Madeleine. This script uses the environment variable `OAR_NODEFILE` set by OAR to create a network configuration file and a channel configuration file based on your reservation. The script accepts two parameters: the name of the network device to use, and the number of processes to start on each node. By default, the configuration will be based on the TCP network and on one process by node. The syntax of the command is as follows:

```
% $MPICH_MAD_ROOT/bin/generateConfigFiles.sh [-net <network device>]
[-weight <number of processes by node>]
```

Based on our reservation with six nodes and supposing the protocol MX will be used to connect the nodes, the configuration files can be created with the following command:

```
% $MPICH_MAD_ROOT/bin/generateConfigFiles.sh -net mx
```

The files will be as follows:

1. The channel configuration file named `appliMpi.cfg`:

```
channels : ({
  name      : grid5000;
  net       : grid5000;
  hosts     : (node-35.lyon.grid5000.fr,
               node-36.lyon.grid5000.fr,
               node-40.lyon.grid5000.fr,
               node-45.lyon.grid5000.fr,
               node-48.lyon.grid5000.fr,
               node-52.lyon.grid5000.fr);
});
```

2. The network configuration file named `networksMpi.cfg`:

```
networks : ({
  name   : grid5000;
  hosts  : (node-35.lyon.grid5000.fr,
            node-36.lyon.grid5000.fr,
            node-40.lyon.grid5000.fr,
            node-45.lyon.grid5000.fr,
            node-48.lyon.grid5000.fr,
            node-52.lyon.grid5000.fr);
  dev    : mx;
});
```

Let's execute our previous example, the program `cpu.c`:

```
% cd mpich-mad/examples/basic
% $MPICH_MAD_ROOT/bin/generateConfigFiles.sh mx
% export MPI_NET_FILE=$PWD/networksMpi.cfg
% $MPICH_MAD_ROOT/bin/mpirun -config appliMpi.cfg cpu
```

The output of the application will be similar to the following:

```
node-35.lyon.grid5000.fr
node-36.lyon.grid5000.fr
node-40.lyon.grid5000.fr
node-45.lyon.grid5000.fr
node-48.lyon.grid5000.fr
node-52.lyon.grid5000.fr
Madeleine/MX: hardware configuration
...
Process 0 of 6 on node-35-a.lyon.grid5000.fr
pi is approximately 3.1415926544231239, Error is 0.0000000008333307
wall clock time = 1.098861
Process 1 of 6 on node-36-a.lyon.grid5000.fr
Process 3 of 6 on node-45-a.lyon.grid5000.fr
Process 4 of 6 on node-48-a.lyon.grid5000.fr
Process 5 of 6 on node-52-a.lyon.grid5000.fr
Process 2 of 6 on node-40-a.lyon.grid5000.fr
```

3.4 Use Case: Grid'5000 from Multiple Sites

We have developed a tool allowing to reserve nodes simultaneously on Grid'5000 with the goal of executing an inter-cluster application. We will see in the following how to execute a PM2 application as well as a MPICH-Madeleine application on inter-cluster nodes. The tool `oargridsub` available on the cluster in Grenoble on the machine `frontale.grenoble.grid5000.fr` can also be used to that effect, it was actually used as a first basis for the first version of our tool.

3.4.1 Installation of the Tool Set

You need to download the latest version of our tool on each cluster, available from the Subversion server on the InriaGforge.

```
% svn co svn://scm.gforge.inria.fr/svn/mpich-mad/g5k-tools
```

3.4.2 General comments

The list of available clusters is obtained as follows:

```
% ./g5k-tools/myoargridsub.sh -l
Available clusters are:
    idpot          oar.idpot.grenoble.grid5000.fr
    sophia         oar.sophia.grid5000.fr
    parasol        oar.parasol.rennes.grid5000.fr
   .gdx           oar.orsay.grid5000.fr
    toulouse       oar.toulouse.grid5000.fr
    paraci         oar.paraci.rennes.grid5000.fr
    bordeaux       oar.bordeaux.grid5000.fr
    icluster2      oar.icluster2.grenoble.grid5000.fr
    tartopom       oar.tartopom.rennes.grid5000.fr
    lyon           oar.lyon.grid5000.fr
%

```

The general syntax of the command is obtained as follows:

```
% ./g5k-tools/myoargridsub.sh -h

Usage: myoargridsub.pl -l
      displays the list of clusters
myoargridsub.pl [-v] [-net network] [-pm2] <cluster list> <program name>
      [<argument 1> ... <argument n>]
-v controls the amount of debugging output
-net interconnection network to use between the nodes (Default: tcp)

Died at ./myoargridsub.pl line 28.
%

```

The format of the cluster list is as follows (`<cluster name>:[dev=<interconnection network>]:<resource list>[:<optional property list>]`). Here some examples of reservation requests:

```
local:nodes=10
sophia:dev=mx:nodes=2:myrinet=\\'yes\\',bordeaux:nodes=3
sophia:nodes=4,toulouse:nodes=1

```

When specifying interconnection networks for clusters, the reserved nodes on that cluster will be set up to communicate with that network, the global interconnection network (specified by the option `-net`) will be used to communicate between nodes from different clusters, e.g:

A reservation might be requested from any cluster of Grid'5000. You only need to make sure the appropriate softwares are installed on each of the clusters involved in the reservation.

3.4.3 PM2

We want to execute the application `mad_ping` between four nodes on the cluster `icluster2` in Grenoble and three nodes on the cluster in Sophia. As shown on Figure 3.2, `leonie`, the PM2 bootstrap code, will be executed on the front-end host, i.e. the machine requesting the reservation, and will interact with the nodes on the two requested clusters.

You first need to compile the application on both clusters. We suppose the softwares PM2 and MPICH-Madeleine have been installed as explained in Section 2 on the machines

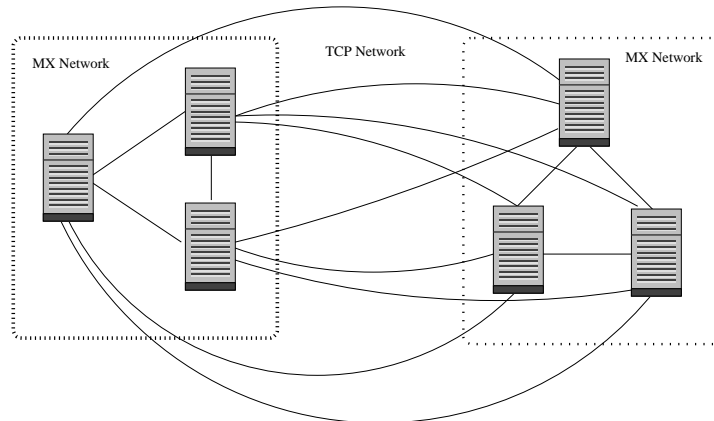


Figure 3.1: Clusters with different networks

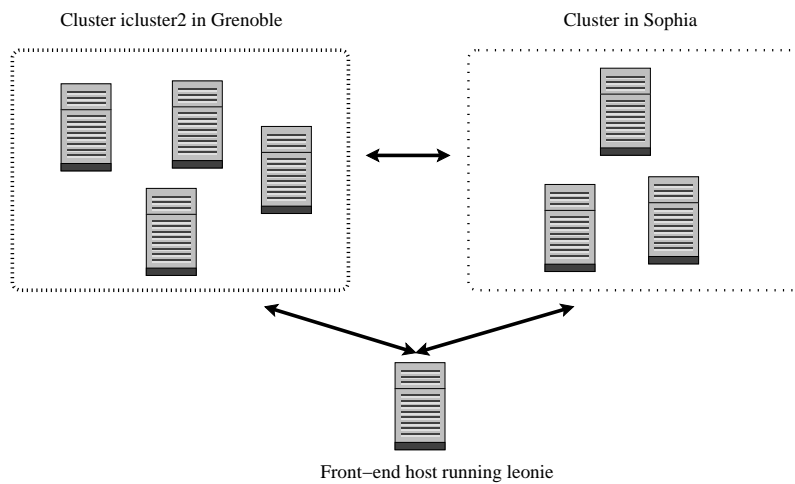


Figure 3.2: Inter-clusters Application

oar.icluster2.grenoble.grid5000.fr and frontale.sophia.grid5000.fr. You then need to execute the following commands on both machines:

```

% cd $PM2_ROOT
% make init
...
% cd mad3/examples/
% make mad_ping
...
%

```

You can now submit and execute your application on the two clusters. For that, you need to call the script `myoargridsub.sh` that is sitting in the directory `g5k-tools/`. The script will perform the following actions:

- Reserve the nodes on the requested clusters;
- Generate the configuration files for PM2;
- Start the application;

- On completion, kill the reservation on each of the clusters.

The parameters of the script are the string `-pm2` to indicate we want to execute a PM2 application (by default, the script executes a MPICH-Madeleine application), the description of the nodes you wish to request on the different clusters, the name of the application to start, and a optional list of arguments to be passed to the application.

```
% ./g5k-tools/myoargridsub.sh -pm2 icluster:nodes=4,sophia:nodes=3 mad_ping
Going to execute <mad_ping> on icluster:nodes=4,sophia:nodes=3
Requesting nodes=4 on cluster icluster2 oar.icluster2.grenoble.grid5000.fr
Reservation number 20198
Requesting nodes=3 on cluster sophia oar.sophia.grid5000.fr
Reservation number 21111
Hosts ita35.imag.fr,ita39.imag.fr,ita42.imag.fr,ita43.imag.fr,node-2,node-3,node-4
Submission hosts ita35.imag.fr,node-2
leonie --x --p -w --appli=mad_ping appli_325.cfg
##### ita35.imag.fr
##### ita39.imag.fr
##### ita42.imag.fr
##### ita43.imag.fr
##### node-2
##### node-3
##### node-4
Channel: channel
(ita35.imag.fr): My global rank is 0
The configuration size is = 7
....
test series completed
Exiting
Exiting
Exiting
Exiting
Exiting
Exiting
Exiting
Exiting
%
```

3.4.4 MPICH-Madeleine

We will now use one node from the cluster in Sophia and two nodes from the cluster in Toulouse to execute the application. The configuration is shown on Figure 3.3.

Again, you first need to compile your MPICH-Madeleine application on both clusters. For now, we will use the application `cpi` as in Section 3.1. If you need to compile the application, execute the following commands:

```
% cd mpich-mad/examples/basic
% $MPICH_MAD_ROOT/bin/mpicc -o cpi cpi.c
```

You can now submit and execute the application on the two clusters. As before, the parameters of the script `myoargridsub.sh` are the description of the nodes you wish to request on the different clusters, the name of the application to start, and a optional list of arguments to be passed to the application.

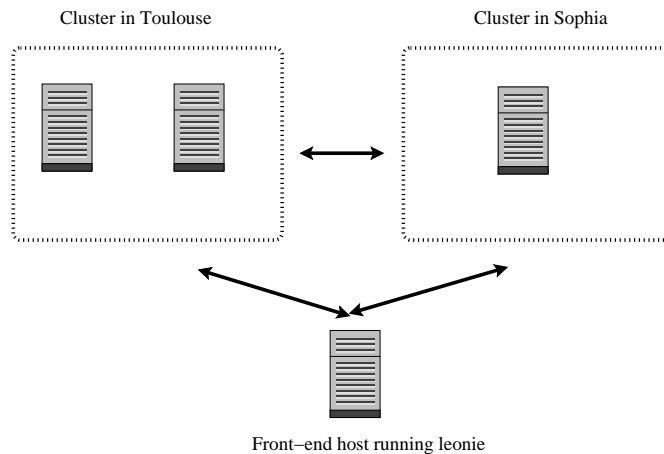


Figure 3.3: (Another) Inter-clusters Application

```

% ./g5k-tools/myoargridsub.sh toulouse:nodes=2,sophia:nodes=1
   /home/bordeaux/nfurmento/work/mpich-mad/examples/basic/cpi
Going to execute
   &lt;/home/bordeaux/nfurmento/work/mpich-mad/examples/basic/cpi>
   on toulouse:nodes=2,sophia:nodes=1
Requesting nodes=2 on cluster toulouse oar.toulouse.grid5000.fr
Reservation number 20198
Requesting nodes=1 on cluster sophia oar.sophia.grid5000.fr
Reservation number 21111
Hosts node-3.sophia.grid5000.fr,node-1.toulouse.grid5000.fr,node-2.toulouse.grid5000.fr
Submission hosts node-3.sophia.grid5000.fr node-1.toulouse.grid5000.fr
export MPI_NET_FILE=networks_411.cfg ; $HOME/work/mpich-mad-install/bin/mpirun
   -config appli_411.cfg
   /home/bordeaux/nfurmento/work/mpich-mad/examples/basic/cpi
##### node-3
##### node-1.toulouse.grid5000.fr
##### node-2.toulouse.grid5000.fr
Process 0 of 3 on node-3.sophia.grid5000.fr
pi is approximately 3.1415926544231318, Error is 0.000000008333387
wall clock time = 1.110340
Process 1 of 3 on cict-003.toulouse.grid5000.fr
Process 2 of 3 on cict-004.toulouse.grid5000.fr
%

```

3.5 Use Case: The Ping-Pong Application

This application is performing ping-pong between pairs of processors, by sending back and forth data of a specific derived datatype and a specific size between a processor P0 and P1. The application is based on the work presented in [1]. The syntax is as follows:

```

% ./pingpong.sh [-not | -opt] &lt;output file> &lt;channel config file>
   &lt;network config file> [&lt;arguments for ping pong ...>]

```

The first optional parameter can be dismissed for now and is explained in Section 3.5.1. The output file is the file in which to store the output of the application. The channel and network configuration files are the usual MPICH-Madeleine configuration files. The last

optional parameters are arguments to pass to the application ping-pong, the following list specifies the format of these arguments.

- The three following arguments must be specified together.
 - `-min n` specifies the minimum message size,
 - `-max n` specifies the maximum message size,
 - `-stride n` specifies the step from the minimum message size to the maximum one.
- The three following arguments are exclusive.
 - `-short` specifies a short message should be sent.
 - `-long` specifies a long message should be sent.
 - `-size n` specifies the size of the message to be sent.
- `-blocks n` specifies the number of blocks to split the message in. This argument does not have any effect for the struct datatype.
- `-tests str` specifies the datatypes to be tested. By default, all the three datatypes, vector, index and struct are tested. The string `str` should be a concatenation of any of these three datatypes.
- `-hindex` specifies the datatype `hindex` should be used instead of the datatype `index`.
- `-hvector` specifies the datatype `hvector` should be used instead of the datatype `vector`.
- `-setPairs pairs` specifies the list of pairs of processors to perform ping-pong with. `pairs` is expected to be a list of processor numbers delimited by the character `-` such as `1-0-3-2-4-5`. Ping-pongs will be performed between processors 1 and 0, 3 and 2, and 4 and 5. By default, ping-pongs are performed between each pairs of processors `P0` and `P1` such as `P0 != P1`.
- `-showRanks` shows the list of processors the application is using with their names and ranks.

When using a reservation with four nodes, similar to the one from Section 3.3, the ping-pong application can be compiled and executed as follows:

```
% cd mpich-mad/example/madeleine
% make PROG=pingpong MPIDIR=$MPICH_MAD_ROOT
% $MPICH_MAD_ROOT/bin/generateConfigFiles mx
% ./pingpong.sh ping.out appliMpi.cfg networksMpi.cfg
```

The output generated in the file `ping.out` will be similar to:

1089	13	517.439287	3	0-1
1000	15	314.886213	3	0-1
1000	17	646.868489	3	0-1
1089	13	4272.309697	3	1-2
1089	13	1125.088477	3	0-2
1000	15	658.126546	3	0-2
1000	17	1924.335284	3	0-2
1089	13	1421.845440	3	0-3
1000	15	560.744521	3	0-3
1000	17	2360.084277	3	0-3
1089	13	5618.979821	3	2-3
1000	15	589.500022	3	1-2
1000	17	1905.416642	3	1-2
1089	13	1604.428182	3	1-3
1000	15	457.957672	3	1-3
1000	17	2301.693570	3	1-3
1000	15	789.845773	3	2-3
1000	17	3383.997515	3	2-3

Each line matches one specific ping-pong between two processors P0 and P1, for a specific size s , a specific derived datatype $type$, the time of the ping-pong itself t and the number of blocks b the message is split in. The format of the line is as follows: s $type$ t b P0-P1.

3.5.1 Optimization Mechanisms

The ping-pong application requires MPICH-Madeleine to be installed in the directory `$HOME/soft/mpich`. It also offers support to use an optimized version of MPICH-Madeleine as presented in [1]. The use of the first (optional) parameter of the application allows to switch between an optimized and a non-optimized version of MPICH-Madeleine. When using the option `-opt`, the application will use the directory `$HOME/soft/mpich-mad-optimized-install` as the MPICH-Madeleine installation directory. With the option `-not`, the directory `$HOME/soft/mpich-mad-n` will be used. This requires that an optimized and a non-optimized version of MPICH-Madeleine have been compiled in the appropriate directories.

Note that this selection mechanism for the MPICH-Madeleine installation directory can easily be tuned to other requirements. Moreover, as stated in [1], the possibility of enabling or disabling the optimization mechanisms will soon be accessible directly from the MPI application and will not require to have two distinct installations of the software.

3.6 Instrumentation and Visualisation

3.6.1 MPE

The Multi-Processing Environment (MPE) which is part of MPICH can be used when executing applications with MPICH-Madeleine. The application must be compiled with the option `-mpitrace`. The execution will then generate log files tracing all MPI calls with a defined set of information for each event.

```

OPTIONS          =          -mpitrace

%: %.o
    mpicc ${OPTIONS} -o $@ $^ ${LIBS}

%.o: %.c
    mpicc -c ${OPTIONS} $<

```

MPE also provides logging libraries which are enabled by using the option `-mpilog` when compiling the application. The execution will generate log files which can be viewed using the `logviewer` graphical tool. Depending on your machine specifications, the log file may have to be converted into another format (`CLOG`, `SLOG`, `ALOG`) before being viewed.

```

% cat Makefile
OPTIONS          =          -mpilog

%: %.o
    mpicc ${OPTIONS} -o $@ $^ ${LIBS}

%.o: %.c
    mpicc -c ${OPTIONS} $<
% make pingpong
...
% mpirun -np 4 pingpong
...
% ls pingpong.clog
pingpong.clog
% logviewer pingpong.clog
$MPICH_MAD_GRID5000_HOME/share/jumpshot-2/bin/jumpshot is NOT present to process the clog
% clog2alog pingpong
% logviewer pingpong.alog
...

```

3.6.2 MPICL

The instrumentation library MPICL has been successfully tested with MPICH-Madeleine. The source code can be downloaded from the following address <http://www.csm.ornl.gov/picl/>. To install it, you need to:

```

% tar zxvf mpicl2.3.1.tar.gz
% cd mpicl2.3.1
% make MACH=mpich CC=$MPICH_MAD_ROOT/bin/mpicc FORT=$MPICH_MAD_ROOT/bin/mpif77 INCL=$MPICH_MAD_ROOT/bin/mpif77

```

The following Makefile can be used to compile your application with MPICL. We suppose that the environment variable `MPICL_DIR` is set to the directory where MPICL is installed.

```

ifneq (${MPICL_DIR},)
OPTIONS      +=      -I${MPICL_DIR}/INCLUDE -DMPICL
LIBS         +=      -L${MPICL_DIR}/LIBS/mpich -lmpicl -lmpich
endif

%: %.o
    ${MPICH_MAD_ROOT}/bin/mpicc -o $@ $^ ${LIBS}

%.o: %.c
    ${MPICH_MAD_ROOT}/bin/mpicc -c ${OPTIONS} $<

```

You then need to insert the following code in your application.

```

#ifdef MPICL
#include "pcontrol.h"
#endif // MPICL

// ...

int main(int argc, char *argv[]) {

    // ...

#ifdef MPICL
    /* enable tracing */
    MPI_Pcontrol(TRACEFILES, "", "tracefile.raw", 0);
    MPI_Pcontrol(TRACELEVEL, 1, 1, 1);
    MPI_Pcontrol(TRACENODE, 1000000, 0, 1);
#endif // MPICL

    // ...

}

```

The execution of the application will create a trace file `$HOME/tracefile.raw`. You can then view it using the ParaGraph visualization tool, which can be downloaded at <http://www.csar.uiuc.edu/software/paragraph/>.

3.7 Troubleshooting

If you encounter problems when running MPI applications over MPICH-Madeleine, the first step is obviously to run the application through the GNU debugger as explained at the end of Section 3.1

If this is not sufficient to detect the problems, we recommend then in a first step to verify PM2 is successfully working on your cluster. To do so, you can follow the different steps given in Section A. A second step would be to check the Madeleine device of MPICH-Madeleine. The code for the device is sitting in the directory `mpid/ch_mad`. Follow the instructions given in Section 5 if you need to modify or recompile this code.

Chapter 4

Debugging MPICH-Madeleine Programs

4.1 Debugger

If you wish to start your application under the debugger, you can use the option `-gdb` when calling `mpirun`. All the processes of the MPI application will be launched through `gdb`, the GNU debugger.

```
% mpirun -gdb -config config test
```

Marcel, the thread library of MPICH-Madeleine, defines `gdb` functions for printing thread.

```
(gdb) marcel-threads
0x3ff4fc00      user_task 43 I  1 machine nil
(gdb) marcel-printthread &__main_thread_struct
0xbffefc00      main 43 I  0 machine nil
(gdb)
```

One may as well switch to the context of non-running threads.

```
(gdb) r
Program received signal SIGINT, Interrupt.
[Switching to Thread 16386 (LWP 25769)]
0x4013c3e7 in sched_yield () from /lib/libc.so.6
(gdb) bt
#0  0x4013c3e7 in sched_yield () from /lib/libc.so.6
#1  0x0804dc6d in idle_func (arg=0x0) at marcel_archdep.h:41
#2  0x0804973a in marcel_create (pid=0x0, attr=0x3ffe0000, func=0, arg=0x0)
    at marcel_sched.h:35
(gdb) set-ctx &__main_thread_struct
switching to &__main_thread_struct(0x64c420)
(gdb) bt
#0  0x0804d5d8 in marcel_give_hand (blocked=0xbffefb6c)
    at source/marcel_sched.c:1173
#1  0x0804ef40 in marcel_sem_P (s=0xbffefb6c) at source/marcel_sem.c:47
...
(gdb)
```

More information on these functionalities are available in the 'Getting started with PM2' guide available at <http://runtime.futurs.inria.fr/pm2-doc/>.

4.2 Memory

The memory debugger `valgrind` has been integrated in MPICH-Madeleine. To start your application under `valgrind`, you need to compile MPICH-Madeleine with some specific options¹. To do so, you need to set the environment variable `PM2_VALGRIND` before configuring MPICH-Madeleine as explained in Section 2.5.

```
% cd mpich-mad
% export PM2_VALGRIND=on
% ./configure --prefix=$MPICH_MAD_ROOT --with-arch=LINUX
    --with-device=ch_mad:--pm2root="$PM2_ROOT"
...
Valgrind enabled
...
% make
% make install
```

Once your application has been compiled, you need to specify the option `-valgrind` when calling `mpirun`.

```
% mpirun -valgrind -config config test
```

When an error is detected by `valgrind`, the application is automatically attached to a debugger `gdb`. You can then use the functionalities explained in Section 4.1 above.

`valgrind` is started using the following default options `--v --show-reachable=yes --leak-check=yes --db-attach=yes`. If you wish to update these options and for example enable the XML output, you have to set the environment variable `PM2_VALGRIND_OPTIONS` to the required value.

```
% export PM2_VALGRIND_OPTIONS="--xml=yes --show-reachable=yes"
% mpirun -valgrind -config config test
```

If you wish to dump the output generated by `valgrind` into a file, it is possible to use the PM2 logging functionalities. At the end of the execution, the log files will be available on each of the machines used by the application in a file similar to `/tmp/pm2log-user-0`.

```
% export LEONIE_ARGS="--x -l"
% mpirun -valgrind -config config test
Session cleaned
%
```

4.3 Debugging the device

You can use the debug facilities of Leonie, the PM2 bootstrap code, to debug the device `ch_mad`. To do so, you first need to specify a debug option when configuring MPICH-Madeleine.

```
% cd mpich-mad
% ./configure --prefix=$MPICH_MAD_ROOT
    --with-arch=LINUX
    --with-device=ch_mad:--pm2root=$PM2_ROOT:--flavor_option=debug
```

¹needed by the thread library `marcel`

Then you might either want to debug leonie itself:

```
% export LEONIE_ARGS="--debug:mad3-log"  
% mpirun ...
```

Or the device:

```
% mpirun -np 2 $PWD/pingpong --debug:mad3-trace
```

You can look at Section 4.4 of the PM2 Manual [2] to find out more on the debug facilities of Leonie.

Chapter 5

Developping MPICH-Madeleine

You first need to set the following environment variables. We suppose that MPICH-Madeleine is installed in the directory `$MPICH_MAD_ROOT`, and PM2 in the directory `$PM2_ROOT`.

```
% export MPICH_MAD_ROOT=$HOME/soft/mpich-mad-install
% export PM2_ROOT=$HOME/soft/pm2
```

1. Set the environment variable `PM2_CONF_DIR`. This is the directory where PM2 stores the flavor configuration files.

```
% export PM2_CONF_DIR=$MPICH_MAD_ROOT/etc/pm2
```

2. Set the environment variable `PM2_BUILD_DIR`. This is the directory where PM2 compiles its libraries.

```
% export PM2_BUILD_DIR=$MPICH_MAD_ROOT/pm2
```

3. Set the environment variable `PM2_FLAVOR`. This is the default flavor used by MPICH-Madeleine.

```
% export PM2_FLAVOR=mpich-mad
```

5.1 Modifying PM2

MPICH-Madeleine defines two PM2 flavors : one for leonie called `leonie` and one for the MPICH-Madeleine code called `mpi-flav`. If you need to modify one of these flavors, you can either:

- use the graphical tool

```
% ezflavor
```

or;

- call the following command from the `$PM2_ROOT` directory.

```
% make config
```

5.2 Modifying MPICH

The code for the device Madeleine is located in the directory `mpid/ch.mad`.

5.3 Recompiling MPICH-Madeleine

To recompile MPICH-Madeleine, you need to execute the following commands from its source directory:

- In case you have modified PM2, you need to call:

```
% make mpiprecompile
```

- In case you have modified the device Madeleine of MPICH-Madeleine, you need to call:

```
% make mpidevlib
```

- Finally, you need to call:

```
% make install
```

Bibliography

- [1] **Optimisation Mechanisms for MPICH-Madeleine.** N. Furmento and G. Mercier. Technical Report 0306, INRIA, July 2005. <http://www.inria.fr/rrrt/rt-0306.html>.
- [2] **Getting started with PM2.** The PM2 Team. <http://runtime.futurs.inria.fr/pm2-doc/>.

Appendix A

Testing the Installation of PM2

This section presents a list of commands aiming to test if PM2 is fully working on your cluster. In case of problems, you can consult the PM2 web site at <http://runtime.futurs.inria.fr/pm2/>.

You first need to generate the flavors for PM2.

```
% cd $PM2_ROOT
% make clean
...
% make init
...
```

You can now compile and execute a sample Marcel application.

```
% cd marcel/examples
% export PM2_FLAVOR=marcel
% make clean
...
% make sumtime
...
<<< Generating libraries: done
  building sumtime.o
  linking sumtime
% pm2-load sumtime 1000
Sum from 1 to 1000 = 500500
time = 4.829ms
```

You can now compile and execute a sample Madeleine application.

```

% cd ../../mad3/examples/
% export PM2_FLAVOR=mad3
% make clean
% make mad_ping
...
<<< Generating libraries: done
    building mad_ping.o
    linking mad_ping
% pm2-conf localhost localhost
The current PM2 configuration contains 2 host(s) :
0 : localhost
1 : localhost
% pm2-load mad_ping
Directory /home/bordeaux/nfurmento/build/leonie/leonie/bin not found

Do you want to try to compile it by executing :
cd /home/bordeaux/nfurmento/work/pm2 ; make FLAVOR=leonie
[Y/n]
...

    linking leonie

*****
Restarting leonie --appli=mad_ping --flavor=mad3
--net=/home/bordeaux/nfurmento/soft/pm2/leonie/examples/networks.cfg
--d --x --p --l /home/bordeaux/nfurmento/.pm2/conf/mad3/.pm2conf.cfg
##### cict-034.toulouse.grid5000.fr
##### cict-034.toulouse.grid5000.fr
(cict-034.toulouse.grid5000.fr): My global rank is 0
(cict-034.toulouse.grid5000.fr): My global rank is 1
The configuration size is = 2
Channel: pm2
The configuration size is = 2
Channel: pm2
My local channel rank is = 0
Channel: pm2
My local channel rank is = 1
ping with = 1
pong with = 0
src|dst|size      |latency      |10^6 B/s|MB/s      |
  0 | 1          | 4         | 10.964    | 0.365    | 0.348
...
  0 | 1          | 2097152   | 33431.436 | 62.730   | 59.824
Exiting
test series completed
Exiting

```

A.1 Debugging the PM2 modules

The PM2 bootstrap code, `leonie`, is used by `pm2-load` and `MPICH-Madeleine` to launch the application on the requested processors. `leonie` accepts different parameters for debug purpose. It is possible to trace or log any of the modules used by PM2. The general syntax of `leonie` is:

```

% leonie [leonie parameters] configuration file [application parameters
to be passed over to the processes]

```

A simple call of `leonie` would be:

```
% leonie --x --p --appli=mad_ping appli.cfg
```

where the option `--x` indicates that session processes should not be started within a new graphical console (i.e. `xterm`), and the option `--p` indicates there should be no pause following the termination of the session processes. Start `leonie` without these options to fully understand their behavior. The call `leonie --help` shows the list of all the available options.

Debug parameters allow to trace specific modules. The general format of a debug parameter is `--debug:<MODULE_NAME>-<TRACE_LEVEL>`. For example, the debug parameter `--debug:ntbx-trace` will display all the trace messages within the module `ntbx` either made by `leonie` or by the processes started by `leonie` (depending on the parameter is specified as a `leonie` parameter or as a application parameter). **Note that the module must have been compiled with the option `debug`.** More debug parameters are available, you can print the list as follows:

```
% leonie --x --p --appli=mad_ping appli.cfg --debug:register
(ffxffffffff:-99:      ) register debug name: register [default] (show=5)
(ffxffffffff:-99:      ) register debug name: default [default] (show=2)
(ffxffffffff:-99:      ) register debug name: ma [default] (show=DEFAULT (2))
(ffxffffffff:-99:      ) register debug name: mar-mdebug [ma] (show=DEFAULT (2))
(ffxffffffff:-99:      ) register debug name: marcel-init [mar-mdebug] (show=DEFAULT (2))
(ffxffffffff:-99:      ) register debug name: log [default] (show=DEFAULT (2))
....

% leonie --x --p --appli=mad_ping appli.cfg --debug:mar-mdebug
(ffxffffffff:-99:      )                               &lt;main_thread is bffefe00>
(ffxffffffff:-99:      ) Init running level 3 (Init scheduler) start
(ffxffffffff:-99:      ) Init running level 0 (Init self)
....
```

The `leonie` parameter `-l` indicates the output of the debug should be redirected to a file in the default temporary directory. On a typical Unix system, the name of the file will be similar to `/tmp/pm2log-$USER-x`.

When executing a Madeleine application, the flavor (i.e. the configuration) `leonie` is used by `leonie` itself, and the flavor `mad3` is used for the application started by `leonie`. The following command will print the list of modules for a specific flavor:

```
% pm2-config --flavor=mad3 --modules
mad3 marcel tbx ntbx init
```

The debug parameters can be specified directly for `leonie`:

```
% leonie --x --p --debug:leonie-trace --appli=mad_ping appli.cfg
% leonie --x --p --debug:ntbx-trace --appli=mad_ping appli.cfg
```

or for the processes started by `leonie`:

```
% leonie --x --p --appli=mad_ping appli.cfg --debug:mad3-log
% leonie --x --p --appli=mad_ping appli.cfg --debug:mad3-trace
%
% leonie -l --p --appli=mad_ping appli.cfg --debug:mad3-trace
% leonie -l --p --appli=mad_ping appli.cfg --debug:mad3-log
```

or for both `leonie` and the processes started by `leonie`:

```
% leonie -l --p --debug:leonie-trace --appli=mad_ping appli.cfg --debug:mad3-trace
% leonie -l --p --debug:ntbx-trace --appli=mad_ping appli.cfg --debug:ntbx-trace
```

A.2 Debugging PM2 processes

When starting `leonie`, you can specify processes should be started under the debugger by using the option `-d` as in the following example:

```
% leonie -d --appli=mad_ping appli.cfg
```

This command will start the processes under the GNU debugger, each within a new graphical console. If you do not have the option of starting graphical tools, you should try the following command:

```
% leonie -d --x --appli=mad_ping appli.cfg
```

If your system allows users to create core files, this command will dump the execution of the faulty processes into a core file. You can then use the GNU debugger to examine the execution in more detail.

```
% pm2-which mad_ping
/home/bordeaux/nfurmento/build/mad3/examples/bin/mad_ping
% gdb /home/bordeaux/nfurmento/build/mad3/examples/bin/mad_ping ~/core.2994
GNU gdb Red Hat Linux (6.3.0.0-1.84rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu"...
Using host libthread_db library "/lib64/libthread_db.so.1".

Core was generated by `./home/bordeaux/nfurmento/build/mad3/examples/bin/mad_ping
--mad_leonie node-22.'.
Program terminated with signal 11, Segmentation fault.
...
#0 0x0000000000404084 in pseudo_main (_madeleine=0x5bfef0) at mad_ping.c:820
820      session = madeleine->session;
(gdb) bt
#0 0x0000000000404084 in pseudo_main (_madeleine=0x5bfef0) at mad_ping.c:820
#1 0x000000000044fe65 in marcel_sched_internal_create (cur=0x0, new_task=0x0,
attr=0x0, dont_schedule=0, base_stack=0)
at /home/bordeaux/nfurmento/work/pm2/marcel/include/scheduler-marcel/marcel_sched.h:4
#2 0x0000000000000000 in ?? ()
(gdb)
...
```

A.3 Debugging Leonie

You might need to start `leonie` itself under the debugger. To do so, you need to set the environment variable `LEO_DEBUG` to the value `1` before starting `leonie`.

```
% export LEO_DEBUG=1
% leonie --x --p --appli=mad_ping appli.cfg
GNU gdb 6.3-debian
...

(gdb)
```

The debugger then waits for some user input, you can for example set breakpoints or start the application. The file `$HOME/.leo_gdb_init` can be used to define a list of GDB commands to execute when starting the debugger. You can for example automatically start the execution of the application.

```
% echo "r" > ~/.leo_gdb_init
% leonie --x --p --appli=mad_ping appli.cfg
GNU gdb 6.3-debian
...

##### joe
##### joe
(joe): My global rank is 1
(joe): My global rank is 0
test series completed
...
Program exited normally.
(gdb)
```