LEHRSTUHL FÜR BETRIEBSSYSTEME

Univ.-Prof. Dr. habil. Thomas Bemmerl

MP-MPICH

USER DOCUMENTATION & TECHNICAL NOTES

Lehrstuhl für Betriebssysteme
Univ.-Prof. Dr. habil. Th. Bemmerl
RWTH Aachen

Martin Pöppe
Silke Schuch
Rainer Finocchiaro
Carsten Clauss
Boris Bierbaum
Joachim Worringen

last change: April 10, 2006

RWTH AACHEN UNIVERSITY

LEHRSTUHL FÜR BETRIEBSSYSTEME

Univ.-Prof. Dr. habil. Thomas Bemmerl

RWTH Aachen, Lehrstuhl für Betriebssysteme
Univ.-Prof. Dr. habil. Thomas Bemmerl
Kopernikusstr. 16
D-52056 Aachen
Germany

Phone: +(49)-241-8027634
Fax: +(49)-241-8022339
eMail: `contact@lfbs.rwth-aachen.de`
WWW: `http://www.lfbs.rwth-aachen.de`

Contents

Chapter 1

Introduction

Thank you for your interest in MP-MPICH. This document will give you an impression of what MP-MPICH is and then help you to install and use the different subsystems of MP-MPICH.

More information on MP-MPICH is usually available via the URL noted below. For specific questions, don't hesitate to write an email to the given support address.

1.1 What is MP-MPICH?

MP-MPICH is an MPI implementation for clusters of x86- and Sparc-based systems with one or more CPUs each. It allows for communication via shared memory, TCP/IP and SCI. MP-MPICH is derived from MPICH 1.2.0, complies with the MPI-1 standard just like MPICH does, and can be used with all tools which can be used with MPICH.

MP-MPICH consists of several parts which are more or less independent from each other:

SCI-MPICH. This is the part of MP-MPICH that enables the use of the fast SCI interconnect. The main benefit of SCI-MPICH is performance: message latencies (roundtrip / 2) of about 5μs, and the sustained bandwidth goes up to nearly 100% of the available SCI bandwidth (currently 74 MB/s for 32-bit PCI buses, about 170MB/s for 64-bit/66MHz PCI buses).

NT-MPICH. NT-MPICH is the port of MPICH to Windows NT and 2000/XP. It offers very good performance, a complete remote execution environment and many features like full MPE logging and graphics or MPI-IO for NTFS, which are not available in other MPI distributions for Windows. NT-MPICH also supports communication via SCI.

These two subprojects of MP-MPICH overlap as illustrated in 1.1.

Other subprojects that are not yet publically available are:

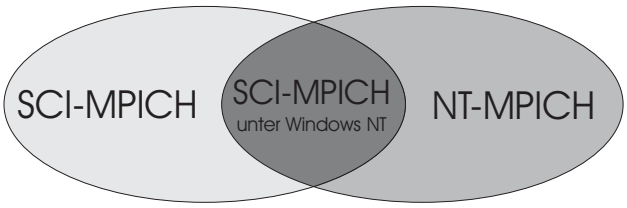


Figure 1.1: NT-MPICH and SCI-MPICH in the MP-MPICH environment

MetaMPICH. If multiple clusters are to be used as one single MPI system, MetaMPICH can be used to create such an environment which is transparent to the application. It connects the clusters with a freely configurable topology which is made up from TCP/IP or ATM connections between dedicated routing processes in each cluster.

1.2 Contact & Support

We have established a number of communication channels between the users and the developers of MP-MPICH:

- To stay informed on MP-MPICH or to check for updates, look at

  `http://www.lfbs.rwth-aachen.de/mp-mpich`

- If you have problems, comments, suggestions etc., please send an email to

  `mp-mpich@lfbs.rwth-aachen.de`

- For bug reports, please use our trac server which you can reach via

  `https://svn.lfbs.rwth-aachen.de/projects/mp-mpich`

  Click on create new session to continue. You have the possibility to view and create bug-tickets.

- We have installed a (low traffic) mailing list to inform the users of MP-MPICH of new releases. To subscribe to the mailing list, please visit

  `http://MailMan.RWTH-Aachen.DE/mailman/listinfo/mp-mpich`

We'll be happy to hear from you. However, support is not guaranteed - we'll do what we are able to. But we do not only want to hear from you if something is really wrong. We are always interested in user feedback of any kind: improvement suggestions, wishes, good experiences you made with MP-MPICH and everything else that you think we should know.

1.3 About this document

Despite using one single source code base for all platforms and networks, MP-MPICH consists of two mainly used flavours: SCI-MPICH, mainly used with Linux, and NT-MPICH, which uses TCP on Windows platforms. All information concerning these main streams are found in the appropriate chapters.

More general installation instructions, but only for Unix, can be found in the next chapter.

Chapter 2

Compiling and installing MP-MPICH from source

Though based on ANL MPICH 1.2.6, MP-MPICH has its own configure script which can be build with newer versions of `autoconf`. Many configure switches of the original are removed while some new are provided.

2.1 Configuring MP-MPICH

For detailed configure options run `./configure --help`.

Unique - and therefore mentioned here - is the option to configure MP-MPICH with several devices at the same time using the option `--with-device=ch_...[,ch_...,ch_...]`. At run-time, the device can be chosen by calling `mpirun` with the option `-machine ch_...`.

Chapter 3

SCI-MPICH

This part of the MP-MPICH documentation is intended to give you the basic knowledge to install and compile SCI-MPICH (chapter 3.2) and run applications (chapter 3.3), but also to supply more in-depth information on tuning parameters in order to let you get the best performance out of your SCI cluster (chapter 3.5). Of course, this kind of knowledge is also required to solve potential problems in setting up your SCI-MPICH environment and in running applications.The operation of SCI-MPICH under Windows NT is covered in chapter 3.7. To see which performance you can expect from SCI-MPICH, take a look at chapter 3.10. If you intend to tune or modify SCI-MPICH, chapter 3.11 gives an overview on the internal design of SCI-MPICH, but also on current limitations or known bugs.

3.1 What is SCI-MPICH?

SCI-MPICH is part of the MP-MPICH project: it is the part of MP-MPICH that enables the use of the fast SCI interconnect for inter-process communication. It achieves up to 235 MB/s bandwidth between processes on different nodes and message latencies below 5μs.

Like MP-MPICH, SCI-MPICH is based on the MPICH implementation of MPI. It consists of a new ADI-2 channel device called `ch_smi` which manages the communication via SCI. Next to the full MPI-1 functionality, it also offers the one-sided communication as specified by the MPI-2 standard and other features not common in MPI implementations through the MP-MPICH architecture.

This document is intended not only to help you installing and using SCI-MPICH, but also to assist you in understanding (to a certain degree) how it works and how you can achieve the best performance or a certain behaviour with it. **Therefore, *we strongly recommend to read this document completely, including the parts that go beyond the basic setup*.**

3.2 Installation

The installation does not differ much from the normal installation of MPICH. Only the differences are noted in this chapter. Refer to the original MPICH documentation for further information.

3.2.1 Requirements in Hard- and Software

SCI-MPICH can be used on a variety of systems. The preconditions that must be met are:

Hardware See 3.1 for supported CPU architectures. Support for other architectures should be easily possible, but has not yet been tested (see chapter 3.14 for more information on platform availability). SCI-MPICH is 64-bit safe.

CPU TypeStatus	Notes
IA-32 / x86 (Pentium, Athlon)	release - no known problems
IA-64 (Itanium)	alpha - only tested for SMP operation (no SCI)
x86-64 (Opteron)	beta - not intensively tested
Alpha	release - no known problems
Sparc	release - no known problems

Table 3.1: CPU types supported by SCI-MPICH

Additionally, you need of course at least one PCI-SCI adapter per node[1] for multi-node operation. SCI-MPICH supports all available PCI-SCI adapters from Dolphin and Scali, connected in arbitrary topologies.

Operating System SCI-MPICH can operate under Solaris, Linux 2.x and Windows NT 4.0 /2000 / XP / .NET. We have performed tests on all of these platforms to ensure correct functionality. However, as Windows and Linux on IA-32-platforms are our primary development platform, they are tested best. Again, ports to other operating systems should not be too difficult.

SCI-Drivers SCI-MPICH is based on the SMI library, which in turn uses the SISCI programming interface for SCI. Both existing SISCI implementations, from Dolphin or Scali, can be used. However, the Scali implementation lacks some features like DMA, reliable remote interrupts and a number of queries. The whole SCI-MPICH functionality is only available with the Dolphin implementation of the SISCI API.

[1] Multiple adapters are supported in beta-state.

If using Dolphin drivers with Linux 2.4, please look at question Q-19 in chapter 3.9.

Libraries A number of libraries is required to let SCI-mpich work:

- *SISCI Library (libsisci or libsisci_api)*: For access to the SCI hardware, the SISCI library is required (see above). It is supplied with your SCI drivers.
- *Pthread Library (libpthread)*: SCI-MPICH uses POSIX threads for certain operation modes.
- *Other system libraries*: Depending on the configuration, other system libraries (like *libdl*) may be required. They should be available on any of the supported operating system platforms.

Compiler SCI-MPICH consists of different parts in different languages:

- The main libraries under Linux and Solaris are plain ANSI-C and thus should be compilable with any current C compiler. The Fortran bindings are Fortran77 and should be compilable with any Fortran77 compiler. The calling and naming conventions are adapted to the compiler during configuration. Creation of Fortran bindings can be disabled via the `--disable-f77` option to configure.
- The Fortran90 bindings have not been tested lately, but generally work. A Fortran90 compiler is required to create the modules. Problems may occur with the most recent (untested) Fortran90 compilers.
- The C++ bindings are a separate package which are compiled by default (use configure option `--disable-c++` to avoid this) using the default or specified C++ compiler.
- On IA-32 platforms, the MPI operations for MPI_Reduce() / MPI_Accumulate() are optimised using MMX and SSE assembler instructions. Any recent assembler (like `as` or `gas`) should cope with the MMX instructions; to translate the SSE instructions, the assembler `nasm` is required to be in the `$PATH`. The configure script will set up things automatically.

We have tested the following compilers:

- **Gnu gcc** / g77 **2.95.3** under Linux 2.2/2.4, x86 and Alpha:

  o.k., optimisation `-O3` recommended

- **Gnu gcc** / g77 **3.2** under Linux 2.2/2.4, x86:

  o.k., optimisation `-O3` recommended

- **Gnu gcc** / g77 **3.4** under Linux 2.4, x86:

  o.k.

- **Gnu gcc** / g77 **3.5** under Linux 2.4, x86:

  o.k.

- **Portland Group pgcc** / pgf77 3.2.4 under Linux 2.4 x86:

  The `ch_smi` device does not work if compiled with `pgcc` (does not detect new messages – maybe does some invalid assumptions?). However, it is possible to compile the library with `gcc` and use `pgf77` / `pgf90` for the Fortran bindings (see the configure options `--with-cc` and `--with-fc`) as we have done.

- **Sun cc** / **f77** from **Workshop 5**, Sparc and x86

  o.k., optimisation `-xO3` recommended

- **Sun cc** / **f77** from **Forte 6** (aka Workshop 6), Solaris 7 / 8 Sparc

  o.k., optimisation `-xO3` recommended, 64bit-mode supported

- **Sun cc** from **Forte 6, x86** (Forte 6 on x86 does no longer include Fortran)

  o.k., optimisation `-xO3` recommended

- **Intel icc** / **ifc** Version 5.0.1 (Build 020124D0) for **Linux**

  o.k., optimisation does not make a difference for performance (default optimisation is sufficient). Make sure that you use at least the version of the Compiler as specified above; older versions which have the same version number, but lower build numbers, couldn't handle shared libraries correctly and created buggy code.

For some assembler files on x86 platforms, a recent version of the Gnu assembler as is required (on Linux, this the default assembler; on Solaris, it needs to be installed separately).

The C++ bindings can not be built with any of the Sun C++ compilers mentioned above; we did not yet evaluate the reason for this.

For Windows NT/2000, Microsoft Visual C++ version 6 can be used with the supplied project files.

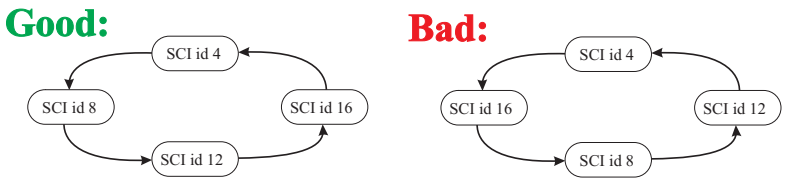
Tools To compile the SMI library on non-Windows platforms, you need the Gnu version of make (aka `gmake`). The startup script `mpirun` is a sh shell script which needs the commands `basename`, `dirname`, `awk`, `sed` and `sort` (these should be available on any Unix platform). Some optional startup modes of `mpirun` require `xterm` and a pager (like `less` or `more`).

3.2.2 Setting up the cluster nodes and the SCI interconnect

SCI-MPICH can run on any SCI connected cluster if all nodes on which processes are started can reach each other via SCI. Additionally, all processes need to be able to communicate via TCP/IP (required for startup synchronisation only).

However, it is recommended to configure the SCI interconnect and nodes as follows. These rules vary somewhat depending on the type of SCI driver you use (Dolphin or Scali).

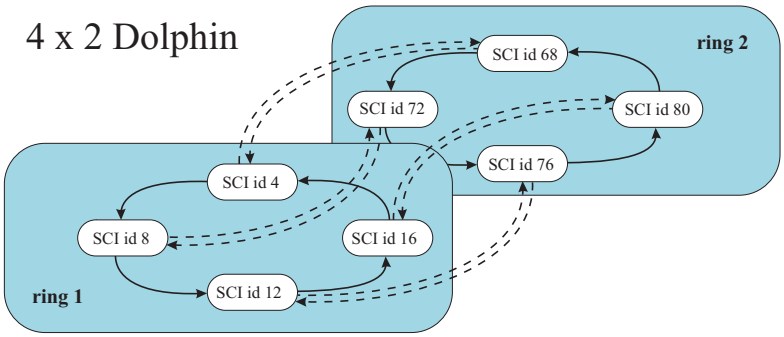
- Nodes with adjacent SCI ids should also be physically adjacent regarding the SCI cabling. This makes internal, topology-oriented optimisations of SCI-MPICH (especially for collective communication) effective. Example (node ids relate to a Dolphin-driven cluster):



- Dolphin SCI driver: The nodes within a single ring need to start with SCI id 4, while the following (in the direction of the cabling) nodes get assigned SCI ids incremented by 4 (8, 12, ...). One ring can include a maximum number of 15 nodes. For a two-dimensional torus topology, each additional ring of the torus starts with an additional displacement of '64' for the nodes SCI ids. This is the way to set up a 2D torus as required by the Dolphin SCI driver. The table below illustrates this scheme for a 2-dimensional torus with n rings. (node ids are given in hexadecimal notation).

ring number	first node in this ring	second node in this ring	...	maximum node id in this ring
0	0x0004	0x0008	...	0x003c
0	0x0044	0x0048	...	0x007c
...	...	...	...	...
n-1	(n-1)*0x40 + 0x4	(n-1)*0x40 + 0x8	...	(n-1)*0x40 + 0x3c

- Scali SCI driver: If you use a Scali-driven cluster, you need to ensure that the nodes' SCI ids are assigned in a similar manner. The slightly different scheme is shown in the following table for the example of a cluster with n rings and a maximum number of 15 nodes per ring (maximum number of rings is also 15). The node ids are given in hexadecimal notation.



ring number	first node in this ring	second node in this ring	...	maximum node id in this ring
0	0x100	0x0200	...	0xf00
1	0x1100	0x2100	...	0x1f00
...	...	...	...	...
n-1	0xn100	0xn200	...	0xnf00

The automatic assignment performed by the driver and its configuration daemon will assign different SCI ids; use the manual configuration tool to adjust this once. Currently, only 2D-torus topologies are supported with Scali.

- The numerical order of the SCI ids of the nodes should be the same as the alphabetical order of the nodes' hostnames. Example:

hostname	SCI node id (Dolphin)	SCI node id (Scali)
p3-00	0x0004	0x0100
p3-01	0x0008	0x0200
p3-02	0x000c	0x0300
p3-03	0x0010	0x0400

This will ensure the proper mapping between processes started by `mpirun` and their MPI ranks in the MPI_COMM_WORLD communicator. This is required for correct naming of output files and terminal windows. Please note that host names like "node1 node2 ... node10 node11 ..." will not sort correctly as node10 is alphabetically smaller than node2. Instead, you should use leading zeros like in the table above (like "node01 node02 ... node10 node11 ..."). You may install and use (local) host alias names to work around such a problem without reconfiguring the hostname of every node.

In case of a misconfigured cluster, SCI-MPICH will still execute correctly. However, i.e. the rank of a process which is appended to an output file will not be identical to the MPI rank the process was assigned during the execution, and the performance of collective operations may degrade.

3.2.3 Installing the Archives

The necessary steps to install the software depend on the archive you downloaded:

mp-mpich_1.x.y.tar.gz Just unpack the complete MP-MPICH archive to a suitable place.

NT-MPICH.zip The binary distribution for Windows NT includes the `ch_smi` device and the SMI library. Refer to chapter 3.8 for the necessary installation steps.

3.2.4 Updating an Existing SCI-MPICH Installation

If you want to update an existing SCI-MPICH installation, you may simply install the new version on top of the old version.

3.2.5 Configuring and Compiling

The information in this chapter describes the configuration and compilation of SCI-MPICH on Unix systems. For information on SCI-MPICH under Windows NT, see chapter 3.8. The steps in building the MPI library are identical to the original MPICH:

1. You need to set up the complete source tree according to your system environment using the `configure` script. Usually, it is sufficient to just invoke `configure` without any parameters.
2. Build the complete set of libraries (as specified via the configure pass) via a call to make in the MPICH top-level directory
3. Test MPICH by compiling some examples or test from the examples directory tree by calling make in the appropriate directory and launching the examples via `mpirun`. Refer to chapter 3.3 for the usage of `mpirun`.
4. If you want to install everything in a (global) directory, call make install. You can specify this directory with the configure option `--prefix=/path`. Please note that you first have to call make, before calling make install.

For SCI-MPICH, there are some special or additional options for the `configure` pass which need to be supplied.

- To specify the device to be built, you may supply the device option (but `ch_smi` is the default device to be built):

  `--with-device=ch_smi`

  All other device-specific options are directly appended as a single string to this option (use quotes if this string includes spaces), separated by a colon.

Example: `--with-device=ch_smi:"--device-option-1 --device-option -2"`

- The SISCI library is required for linking an SCI-MPICH based application. As default, the script looks in the standard system paths as well as in `/opt/scali` and `/opt/DIS`. To specify another path to the SISCI installation, use: `--with-sisci=/path/to/SISCI` (this is a device-specific option!) Please note that the SISCI library needs to be located under `/path/to/SISCI/lib`, the SISCI include files need to be located in `/path/to/SISCI/include`.

- The C compiler should be instructed to optimise the code because this significantly reduces the small-message latency. Use the `--with-optcc` option for this: Gnu cc: `--with-optcc=-O3` Sun cc: `--with-optcc=-xO3` We have observed a reduction of the small-message latency of more than 3 μs on PentiumIII-800 systems when using optimized code (see section 3.10.1)

Example for the invocation of the configure script:

```
./configure --with-device=ch_smi:--with-sisci=/usr/local/SCI/SISCI
-v --with-optcc=-x=3 --with-cc=gcc
```

All other configure options can be used as usual. Some of these standard options that might be interesting to SCI-MPICH users are:

- If you want the device to perform fully asynchronous, non-blocking communication which allows overlapping of computation and communication, you need to enable the usage of threads in the device with the option

  `--enable-async (this is a device-specific option!)`

  Please note that the minimum message latency will increase in this case.

- Per default, SCI-MPICH uses static libraries. If you want to use shared libraries instead, specify `--enable-sharedlib` Please consider the notes we made in chapter 3.10.2 concerning static vs. dynamic linking.

- To enable debugging/tracing output of the MPI library, specify `--enable-debug` To actually enable the tracing output, pass the option `-mpichdebug` to the program (not to `mpirun`). Example: `mpirun -np 2 buggy -mpichdebug` This option is also needed if you want to debug inside the SCI-MPICH libraries if they are built as shared libraries.

- Special debugging/tracing output of the device is enabled via `--enable-devdebug` This option also needs to be specified if the channel device (like `ch_smi`) should be able to generate extended debug/tracing output

- The `ch_smi` device can gather a large number of statistics which are helpful for performance analysis. This is enabled via

  `--enable-statistics (this is a device-specific option!)`

  (required for `STATISTICS` option in the `ch_smi` device configuration file)

- The configure script should automatically detect your operating system. However, if in doubt that the correct selection is performed, use the architecture option with one of the supported platforms:

  `--witch-arch=[solaris|solaris86|LINUX|LINUX_ALPHA]`

- If you have PBS installed on the cluster, SCI-MPICH supplies a special script to allow a the usage of PBS with normal "`mpirun` semantics"; this means you do not need to create a special PBS batch script and use `qsub` on this, but just use `qmpirun` instead of `mpirun` to start your MPI applications using PBS. All options of the two scripts are identical. To specify that you want to use PBS, do: `--with-pbs=queuename:hostname` Please note the use of the colon ':' instead of the usual '@' for the queue specification.

- To utilise PAPI performance monitoring via an MPI wrapper library, specify `--with-papi=path` The path needs to indicate a path where the required PAPI libraries can be found at `path/lib` and the binaries (server) need to be found in `path/bin`.

Notes:

- Because the `configure` script performs a number of tests, some of them related to SCI, you should configure & compile SCI-MPICH on the same (type of) machine as you will run SCI-MPCH applications on.

- The optimisation level of the compiler and the choice of debugging options has a significant influence esp. on the small-message latency (see section 3.10.1)

After configuring, call make to build the whole MPICH distribution, or call `make mpilib` to build only the library, but no additional tools and tests.

3.3 Usage

The information in this chapter describes the usage of SCI-MPICH on Unix systems. For information on SCI-MPICH under Windows NT, see chapter 3.8. For users familiar with MPICH, the use of SCI-MPICH won't cause any problems. If you are not familiar with MPICH or even with programming in MPI, you'll have to read the according documentation. This chapter does only deal with the differences between the use of the standard MPICH and SCI-MPICH.

3.3.1 Creating SCI-MPICH applications

To compile MPI applications, use the `mpicc`/`mpiCC`/`mpif77`/`mpif90` scripts (located in the bin directory) as usual. These scripts pass all options which they do not understand to the compiler frontends. Some switches of these scripts that may be useful are listed below.

Command Printing and Help

`-help` Print a short summary of available options and mode of operation.

`-show` To see the commands that the scripts really execute, add this option to the command line. All commands that would otherwise be executed are now printed to the terminal.

Linking options

`-static | -dynamic` Specify static or dynamic linking. If none of these switches is given, the linking mode as specified with configure will be used (default is dynamic linking). However, also if shared libraries have been built, it sometimes is desirable to link statically: small message latency decreases (see chapter 3.10.2), and potential problems with finding dynamic libraries on the nodes where the program is executed can be avoided. Linking dynamically without having built shared libraries will of course fail.

`-shlibpath | -noshlibpath` If linking with shared libraries, it is required that the application can find these libraries when launched. Using the switch `-shlibpath`, you can tell SCI-MPICH to encode the path to the shared libraries into the executable. This shared libraries will then always be read from the SCI-MPICH installation directory where the `mpicc` script (used for the compilation) is located. However, if you want that the libraries to link with are determined via system default paths or the `LD_LIBRARY_PATH` environment variable, specify `-noshlibpath` to enforce this behaviour.

MPI Profiling SCI-MPICH supports the MPI profiling interface which allows any library to intercept the MPI calls of an application. Different kinds of such libraries using this interface are available, offering different functionality which only need to be linked with the unmodified application. This can be done using the following options to any of the compiler frontend scripts mentioned above:

`-mpilog` Link with the logging library. A MPE log file will be generated which can be visualised using `jumpshot`. Requires the MPE libraries.

`-mpianim` Link with the animation library. The message transfers between the processes will be visualised in "real-time" (online). Requires the MPE libraries.

`-mpitrace` Link with the tracing library. Will print tracing output to stdout for each MPI function called.

`-mpipapi` Link with the PAPI wrapper library. Requires that the application is launched with the `mpirun` option `-mpipapi` (if not, startup will block)

`-mpistat` Link with the statistics wrapper library. This library will print some vital statistics concerning the utilisation of the MPI library by each process. This information will help you to quickly determine if an application is compute- or communication-bound, and to verify the effect of optimisations. The output for each process will look like this:

#type	calls	min[us]	max[us]	avg[us]	acc[s]	%of total
Application	1799	0.5	130523.0	13105.6	23.577	99.31
MPI collectives	1	132.5	132.5	132.5	0.000	0.00
MPI send (nonblocking)	796	1.3	2898.3	143.3	0.114	0.48
MPI recv (nonblocking)	796	1.2	176.7	11.0	0.009	0.04
MPI completion check	199	129.7	484.4	204.6	0.041	0.17
other MPI functions	6	0.7	23.6	6.9	0.000	0.00

Similar types of MPI functions are grouped and are timed together. For each group, the number of function calls, the minimal, maximum, average and accumulated duration and the percentage of the accumulated duration of the total execution time is printed to stdout for each process. The timing information 'Application' specifies the time that the process spent outside of the MPI library. This means each phase between two function calls is measured.

`-vampir` Link with the VAMPIR logging library (see chapter 3.4.2)

3.3.2 Running SCI-MPICH applications

To run the compiled application, use the `mpirun` script from `build/PLATFORM/ch_smi/bin`. The calling syntax is the same as for the original MPICH version:

```
mpirun [mpich-options] program_name program_parameters
```

Alternatively, you can call `mpiexec` (as recommended by the MPI-2 standard). This is only a name replacement, the script behind it and thus the functionality of `mpiexec` is identical to `mpirun`.

If you have configured SCI-MPICH for PBS (`--with-pbs option`), you should use `qmpirun` instead. Again, all options valid for `mpirun` can also be used with `qmpirun`.

The SCI-MPICH version of `mpirun` accepts all basic options as known from other MPICH implementations plus some special SCI-MPICH options. All applicable options are listed sorted by category.

Topology Definition

`-np N | -n N` Run with N processes. If this option is omitted,one process will be launched on every host in the machine file used (or on every host specified via the `-nodes` option). The option `-n` is an alias for `-np`.

`-machinefile FILE` Use FILE to describe the hosts on which to start the processes – see chapter 3.3.3

`-nodes NODE_0[,NODE_1,... ]` If you do not want `mpirun` to use the nodes (hosts) specified in any machine file, but want to specify the nodes to be used on the command line, you can use the `-nodes` parameter. This parameter must be followed by a comma-separated list of hostnames. This list must not contain spaces. The therewith specified nodes are used in the same manner as if they had been read from a machine file (see chapter 3.3.3 on page 22). Some more features of the `-nodes` option:

- To generate a list of nodenames which differ only in small parts of the name, it is possible to specify the hosts in a sh-like syntax, i.e.: `mpirun -nodes "p3-0{1,2,3,4}" exefile` Please note the usage of quotation marks around the node specifier - this is required to avoid the extension of this expression on the commandline which would result in an illegal syntax.
- If the `-n` option is omitted when using `-nodes`, `mpirun` will start one process on each specified node.
- The `-nodes` option is aliased by the `-node`, `-hosts` and `-host` options (other name, same functionality).
- It makes no sense to use `-nodes` together with the `-machinefile` parameter (`-nodes` has the highest priority).

`-smp` Running more than one process on a single node is only recommended for SMP systems and may have significant influence on the application performance (it can decrease or improve). In case of the Linux platform, it requires that the SCI drivers are compiled with SMP support enabled. To avoid unwanted SMP execution, the option `-smp` must be supplied to allow more than one process on any single node. Testing and Verbosity.

`-help` Print a short summary of available options and mode of operation.

`-v | -verbose (small letter 'V')` Be verbose on startup and print information on the startup process to stdout.

`-V | -Verbose (big letter 'V')` Be even more verbose on startup and additionally print information on the internal configuration of SCI-MPICH.

`-t | -testing | -show` Just show the commands `mpirun` would issue, but do not execute them.

Application Standard-I/O

`-stderr FILE` Redirect the stderr stream into a file. The output of each process is written into a file named `FILE_x` where `x` is the MPI rank of the process. Existing output files are renamed to `'FILE~'`, empty output files are removed after termination of the application. If only one process created a non-empty output file, this file will be named `FILE` (and not `FILE_X`).

`-stdout FILE` Redirect the stdout stream into a file. The output of each process is written into a file named `FILE_x` where `x` is the MPI rank of the process. Existing output files are renamed to `'FILE~'`, empty output files are removed after termination of the application. If only one process created a non-empty output file, this file will be named `FILE` (and not `FILE_X`).

`-stdin FILE` If you need to supply input via stdin to any process, you need to store this input in a file an used the stdin option: each process will get input from the file `FILE`. All processes will get the same input as specified in the file.

`-console all | <rank|nodename,...,rank|nodename>` Do open a separate console I/O window for each specified process, so that the console input and output (stdin, stdout, stderr) of the processes is cleanly separated and process can read its own input. The environment variable `DISPLAY` must be set correctly on the host on which the smirun-command is issued. The environment variable `MPI_XTERM` or `XTERM` (with higher priority for `MPI_XTERM`) can be set to specify the command used to open the terminal window. The default value is the xterm command. It is possible to open a console window for all processes using the parameter all. If only certain processes should open their own console window, specify the MPI ranks or the hostnames of these processes in a comma-separated list as parameter.

`-pager` In conjunction with `-console`, stderr and stdout are piped to a pager. The environment variable `MPI_PAGER` or `PAGER` (higher priority for `MPI_PAGER`) can be used to specify the command through which the output is piped. If you do not want to have to press a key after each page of output, you may set an appropriate environment variable to cause the used pager (less by default, use the `-V` option of `mpirun` to determine) to behave like the tail command. A usual way to choose is to set the environment variable `LESS` to `"+F"` (i.e. `export LESS=+F`). Refer to the man page of you pager for detailed information on which startup commands are supported.

Working directory and environment

`-wd PATH | -workingdir PATH` Specify a working directory. The default directory in which the execution will take place is the directory in which the executable is located.

`-env VAR=VALUE` Specify an environment variable `VAR` to be set to `VALUE` in the context in which the processes will execute. To specify multiple variables, this option can be supplied multiple times.

Debugging SCI-MPICH supports the startup of all or single processes of the application under the control of an debugger. To do this, you'll need to be able to create X-windows from the host and the command prompt from which you launch the `mpirun` command.

`-gdb all | <rank|nodename,...,rank|nodename>` Use the debugger `gdb` to control all (parameter all) or only the specified processes (by specifying a list of ranks). Examples:
`mpirun -n 4 -gdb 2,3 buggy`
Start the program buggy with 4 processes, and have the processes with MPI rank 2 and 3 execute within the control of `gdb`.

`-dbx all | <rank|nodename,...,rank|nodename>` Like option `-gdb`, but use the debugger `dbx`.

`-tv` Use the debugger `TotalView` to control all processes of the application. Make sure that the environment variable `TOTALVIEW` points to the `TotalView` executable.

Device Configuration

`-devconf FILE` Use `FILE` to supply a device configuration for `ch_smi`. This information can also be contained in the `machinefile` (see chapter 3.5). Advanced Options. A number of options exist which are useful for tracking problems or for library developers, but are not relevant for users of SCI-MPICH who just want to run MPI applications. These options are:

`-mpipapi` If an application has been linked with the PAPI wrapper for performance monitoring, you need to specify this with `mpirun`, too. This will start the performance data server (on the system on which `mpirun` is executed), to which the MPI processes deliver their performance monitoring data. The server in turn will print its port number to the console and wait for a connection with the performance monitoring application (a GUI or a logfile writer).

`-smidebug` Let the SMI library generate debug output – useful if you have problems which seem to be SCI related. The startup of the SMI library can be traced, and the error messages are more verbose and appear in the full context.

`-smitrace` Let the SMI library generate tracing output – useful if you want to see how long distinct sections in a run take inside the SMI library

`-smistat` Let the SMI library generate statistical timing and function call information on shutdown (equivalent to the output of SCI-MIPCH with the `"statistics 1"` option in the device configuration file)

`-nowatchdog` Fully disable the watchdog (not only set it to an infinite timeout).

`-skip id` If a certain process should not be started by `mpirun`, it is possible to specify this process via the supplied id. Such a behaviour may be useful if this process should be started from a debugger. The supplied id can have different semantics:

- id is a hostname from the list of hosts that `mpirun` will start processes on. In this case, no process is started on this host.
- id is an integer number. In this case, the process which would have this MPI rank will not be started.

In either case, `mpirun` will print the command line which would have been used to launch the process(es). This command line should be used for the manual startup of the other process. **Please note:**

- The process should be started on the same host which `mpirun` would have started it on.
- The initial startup synchronisation of the processes which form an application has a limited timeout. If the manually started process is launched with a delay which is beyond this timeout (a small number of seconds, depending on the number of processes), it will fail to synchronise with the other processes.

`-nolocal` Do always use SCI memory even if all processes are running on a single node. In this case, the SMI library would normally use local shared memory among the processes.

Example:

```
mpirun -np 7 -machinefile /home/lassy/cluster/machines -xterm flood 50000
```

Starts the program flood on 7 nodes which are read from the file `/home/lassy/cluster/machines` and opens a separate xterm window for each process. The parameter 50000 will be passed to flood.

Notes:

- `mpirun` is a `sh` script.

- `mpirun` uses `rsh` to launch the processes on the remote hosts. Make sure that the systems are configured accordingly to enable `rsh` to execute without explicit authentication (entering the password). Usually, this is done by setting up `$HOME/.rhosts` accordingly. On systems (many Linux distributions), the general security configuration (via the PAM mechanism) needs to be adjusted to allow this.

- Instead of `rsh`, `ssh` may be used as well (see configure option `--with-rsh`).

- `mpirun` needs to have write access to the `/tmp` directory of the host on which it is started.

- The processes of an SCI-MPICH application need to synchronise in the startup phase using a TCP/IP port. The default port address that is used is 51069. If you need to change this address, set the environment variable `SMI_SYNCPORT` to an according value.

3.3.3 Host Specification

The `mpirun` script can use a machine file to determine the hosts on which to run the MPI processes. Consider the following rules and limitations when creating or using such a file or look at the file machines for an example:

1. Each line in the file contains exactly one hostname. Following this hostname, the number of processes on this host may be specified with a colon (':') as separator (compatible with machine files for the well-known `ch_p4` device of MPICH). Comment lines are not supported However, you may include device configuration information in the machine file which has to be located below the lines containing the host names, separated from the host section by a blank line. See figure 3.1 for some examples:

2. The `mpirun` script selects the required number of hosts from the machine file from top to bottom. It launches one process on each of the selected hosts found in an alphabetical order (sorted by hostname). No process is launched on the local host (on which `mpirun` was invoked) unless this host appears in the machine file and is selected. If the specified number of processes of the MPI application is bigger than the number of hosts in the machine file, the processes are scheduled on the available hosts in a round-robin manner.

3. If the option `-machinefile` is omitted from the `mpirun` call, it uses the following strategy to find a machine file: - look for `./machines` - if not found, look for `$HOME/.machines` - if still not found, look for `$MPIR_HOME/util/machines` - start all processes on the current machine (from which `mpirun` was started).

Notes:

- If multiple processes are started on a node, these processes will communicate not via SCI shared memory, but via local shared memory (SYS-V or memory mapped files), unless the `-nolocal` option was used.
- If multiple processes are started on a node, the `-smp` option needs to be supplied to `mpirun` to certify that you want to do this.
- It is generally not a good idea to launch more processes on a node than there are CPU's in the node. This will degrade performance significantly. If you really need to do this, use the device configuration option `polling` (see section 3.5.2 on page 39).
- It is of course possible to simultaneously launch multiple MPI processes of different applications on a single host by running `mpirun` several times. Again, you should not use more processes than CPUs per node.

3.3.4 Launching SCI-MPICH applications manually

If for any reason (i.e. for debugging), you do not want to launch an SCI-MPICH application via the `mpirun` script (or via `rexecshell` for Windows NT), you may also start it manually. The arguments which are required are as follows:

`-h hostname` Specification of the master node for startup

`-n number_of_processes` Total number of process to form the application

`-r rank_of_process` Rank of the individual process The rank 0 must be given to a process which runs on the master node as indicated by the `-h` option.

`-m id_number` Identification number (choose any number, but it must be identical for all processes)

Optional arguments are:

`-d filename` Use the specified device configuration file

`-l` Indicate that all processes are running on the same node (this will improve performance as only local shared memory will be used)

`-w` disable the watchdog (required for debugging)

`-s` enable debug output of the SMI library

`-v` verbose startup of the `ch_smi` device

The order of these options does not matter. These SCI-MPICH options must be terminated by a `"--"`. Parameters for the application itself can be appended.

- Valid machine file, specifying 3 processes on 3 different nodes:
```
cluster-node-00
cluster-node-01
cluster-node-02
```

- Valid machine file, specifying 3 processes on 2 different nodes:
```
cluster-node-01
cluster-node-00
cluster-node-01
```

- Valid machine file, also specifying 3 processes on 2 different nodes:
```
cluster-node-00
cluster-node-01:2
```

- Valid machine file, specifying 6 processes on 3 different nodes:
```
cluster-node-00:2
cluster-node-01:2
cluster-node-02:2
```

- Invalid machine file (no comments allowed!)
```
cluster-node-00:2
# cluster-node-01:2
cluster-node-02:2
```

- Valid machine file with device configuration appended:
```
cluster-node-00:2
cluster-node-01:2


async_progress 1
rndv_poolsize 4M
```

Figure 3.1: Examples for machine files

Notes:

- Parameters supplied via the command line override the related parameters in a device configuration file.
- SCI-MPICH can be configured to use threads for certain operations (configure option `--enable-devthreads`). Using threads usually hinders Linux from writing a core file in the case of a process failure because Linux can not dump core of multithreaded processes (this may change with future Linux versions).

Example: To run an application called `myprog` on nodes A and B and give each one the parameters `-foo 0` with activated SMI debug output and disabled watchdog:

- on A: `myprog -h A -r 0 -n 2 -m 17 -s -w -- -foo 0`
- on B: `myprog -h A -r 1 -n 2 -m 17 -s -w -- -foo 0`

3.3.5 Shutting down SCI-MPICH applications

To shut down a running SCI-MICH application, it is sufficient to terminate any single process of the application (i.e. by sending a SIGINT). Of course, if the application was launched via the `mpirun` script, the complete application will be aborted if the `mpirun` script is aborted, i.e. by pressing `CTRL-C` on the console. In any case, the watchdog will ensure that the whole application will terminate immediately and cleanly. The same is true if your application terminates abnormally.

The clean shutdown of a parallel application includes the termination of all participating processes and the deallocation of all occupied resources. SCI-MPICH uses the advanced features of the SMI library like watchdog and resource management to ensure that all processes of the application detect the (abnormal) termination of one or several processes and will terminate as well. All resources are deallocated no matter how the application terminates (normal, by crash, coredump, signal...).

Note:

- If an SCI-MPICH application is under the control of a debugger, the shutdown mechanism may fail to deallocate all resources automatically in case of an termination cause by the debugger (because the debugger doesn't use `SIGINT`). This may require a reboot of the system after a number of runs to deallocate certain resources.

  A better technique would be to detach the program from the debugger (if possible) and kill it with an appropriate signal (except `SIGKILL`) which allows the process to shut down properly.

3.4 Compatibility

SCI-MPICH has been tested for compatibility with different MPI tools and environments.

3.4.1 MPICH ABI

SCI-MPICH is binary compatible with MPICH as it offers the MPI API through the MPICH ABI. Object files of applications which are designed to be used with MPICH need only be linked with the SCI-MPICH libraries.

3.4.2 VAMPIR

SCI-MPICH was successfully tested with VAMPIR tracing library (`libVT`) from the VAMPIR distribution 2.5, November 1999 on Linux x86. Newer version should work as well, as the profiling interface as defined by MPI remains the same. To link with the VAMPIR tracing library and thus automatically create VAMPIR trace files, supply the option `-vampir` to the MPI compiler scripts (`mpicc`, ...). Make sure that the environment variable `VAMPIR_LIB` is set such that the required tracing library `libVT` can be found in `$VAMPIR_LIB`. VAMPIR is a product of Pallas GmbH; more information on the VAMPIR tracing environment can be found at `http://www.pallas.de`.

3.4.3 TotalView

SCI-MPICH does offer the required interface for the efficient usage of TotalView. This means that manual startup of a TotalView debug session is possible. However, the integrated startup of a parallel debug session with TotalView via `mpirun` is not yet implemented.

3.4.4 MPICH ch_p4 Startup

The startup via the `mpirun` script is compatible with the startup of applications linked with the original MPICH and the `ch_p4` device. This means, the same kind of machinefiles and other options can be used. However, even if the `-nolocal` option is not given, SCI-MPICH will only launch processes on the hosts specified in the machine file, not on the local host (which is the behaviour of `ch_p4` with the `-nolocal` option).

3.4.5 Scali ScaDesktop

This `ch_p4` compatibility does also allow the use of the Scali cluster frontend tool ScaDesktop to start applications. To do this, simply copy the files `mpirun`,

`mpirun.args` and `mpirun.ch_smi` from the SCI-MPICH installation (in the bin directory) to `/opt/scali/contrib/mpich/bin`, overwriting the existing files `mpirun` and `mpirun.args` (you may want to make backup copies of these). From the cluster-view of the `scadesktop` tool, select `Run -> MPICH program...` and select an SCI-MPICH executable. To pass additional options to `mpirun`, prepend them to the filename in the dialog. Options to the executable itself need of course to be appended.

3.4.6 OpenPBS

SCI-MPICH can easily be used with the OpenPBS (or PBSpro) queuing system. It supplies a special frontend script for this with `mpirun`-like semantics. All you need to do is to specify the PBS system when configuring SCI-MPICH (see chapter 3.2.5) and use the `qmpirun`-script to start your applications (see chapter 3.3.2).

3.5 Device Configuration

SCI-MPICH can be configured to optimally fit your hardware setup or the requirements of your MPI application. To do so, it reads a device configuration file on startup. You can supply the name of such a file via the `-devconf` option of `mpirun`. If the option `-devconf` is omitted from the `mpirun` call, the following strategy to find device configuration information is used:

1. look for `./ch_smi.conf`
2. look for `$(HOME)/.ch_smi.conf`
3. look for device configuration information in the machines file which is used (see chapter 3.3.3)
4. use the built-in default settings

All of the parameters are supplied in the form
`PARAMETER VALUE`
`PARAMETER` must be one of the identifiers listed below (case does not matter). `VALUE` must be an integer (usually the number of bytes or a boolean switch). If `VALUE` means a number of bytes, you can append the letter 'k' or 'M' to indicate that this value is expressed in Kilobytes (210) or Megabytes (220). Look at the file `ch_smi.conf` for an example.

For each parameter which is not specified in the configuration file, the default value as specified in this chapter will be used. Using these default values is fine for most cases, but to activate certain features or settings (which may improve or degrade performance, depending on the application and hardware), other settings than the default values may be required. An example for this is the activation of the asynchronous progress feature: this is useful for applications which are aware of

the possibility to overlap computation and communication, but other applications may show a reduced performance with this feature activated.

Lines which start with a '#' or ';' character are considered as comments and are ignored, as well as empty lines. If an application is started with the verbose option (-v option for mpirun), SCI-MPICH will print warning messages for lines which are syntactically incorrect or which do specify illegal values.

3.5.1 Protocol-specific Configuration

The three different protocols for communication (short, eager, rendezvous) use a certain layout and dimension of their memory pools/queues. These definitions can be set at the startup of the application by the following parameters.

Note:

- It may happen that SCI-MPICH is not able to allocate all the buffers of the specified size In this case, the size and number of the buffers for the eager and rendez-vous protocol are decreased automatically. The actual buffer configuration is printed by process 0 if the application is started with the -v option.

SHORT Protocol

The size of the buffers which is used for control packets and short messages can be adjusted. If you application sends a lot of small messages which are in the low range of the size for the eager protocol, you may increase SHORT_bufsize to send these messages via the short protocol which is somewhat faster. This is true for messages smaller 1024 bytes. However, very small messages (only a few bytes) will take slightly longer in this case.

SHORT_bufsize	The size of the buffers for short messages & control packets. default: size of the largest atomic SCI data move (64 or 128 bytes)

Notes:

- The maximum message size for short messages does not equal SHORT_bufsize, but is several bytes smaller. It is printed on startup if you specify the -V option to mpirun.
- This value is aligned to powers of 2 internally and is at least 64 bytes.

- You need to consider that increasing `SHORT_bufsize` alone will decrease `SHORT_nbrbufs` because SCI-MPICH imposes certain built-in limits on the memory usage of the short protocol. To keep the number of buffers constant, you need to specify a value for `SHORT_nbrbufs`.

`SHORT_nbrbufs` The number of buffers for short messages & control packets.

default: $\frac{Pagesize}{SHORT\_bufsize} - 1$

Lower values lead to more frequent remote updates of the related counter which means less performance. Higher values lead to more memory consumption.

Notes:

- A value smaller than the default size does not save any memory since at least one memory page is reserved for the buffers between each two processes.

Memory usage of the short protocol for each process:
`NBR_PROCESSES*SHORT_bufsize*SHORT_nbrbufs (+ alignment)`

EAGER protocol

`EAGER_bufsize` The maximum size of a message to be transferred via the eager protocol.

default: 16 kB

Other values (lower or higher) are possible. Lower values lead to less performance since the rendezvous protocol is used for messages > `EAGER_bufsize`, higher values lead to more memory consumption.

Notes:

- This value is aligned to multiples of the stream buffer size internally.
- Extending the eager buffer size beyond 64kB usually makes no sense because the rendez-vous protocol starts to be faster for messages of a certain size.

`EAGER_nbrbufs` The number of buffers (each `EAGER_bufsize` bytes long) which are allocated for each sender-receiver pair.

default: 4

Other values (lower or higher) are possible. Lower values lead to (potentially) less performance because the rendezvous protocol is used if no free eager buffer is available on the receiving process. Higher values lead to more memory consumption.

Note:

- You may set this value to 0 to disable the eager protocol if you are really tight on memory.

`EAGER_maxcheckdev` The number of calls to `CheckDevice()` if no free eager buffer is available before the rendez-vous protocol is used (this is more an internal parameter)

default: 1

Other values >= 0 are possible. The performance impact of this parameter heavily depends on the communication pattern. If you set `EAGER_nbrbufs` to 0, you should set this parameter to 0, too.

`EAGER_immediate` If a process receives an unexpected message (a message for which no receive buffer has yet been posted) via the eager protocol, it has the choice of leaving the message in the incoming eager buffer until a matching receive has been posted, or it can free the incoming eager buffer by temporarily copying the message in an internal buffer. The first method avoids the additional copy operation, but blocks the incoming eager buffer which may require the sending process to send the next message via the (slower) rendez-vous protocol. The option `EAGER_immediate` selects one of these methods

0 leaves the message in the incoming buffer
1 immediately frees the incoming buffer
default: 0

`EAGER_dynamic` Besides the 'traditional' static eager protocol with a fixed number of fixed-sized buffers for each sender-receiver pair, a new variant of the eager protocol called dynamic eager can be activated by this parameter. This protocol version uses a fixed-sized ring buffer with a variable number of buffers inside, leading to a more effective usage of memory resources. The size of the ring buffer is calculated as `EAGER_bufsize*EAGER_nbrbufs` and will be reduced if the memory resources are not sufficient.

0 uses static eager protocol
1 uses dynamic eager protocol
default: 0

Memory usage of the eager protocol for each process:
`NBR_PROCESSES*EAGER_bufsize*EAGER_nbrbufs (+ alignment)`

RENDEZ-VOUS protocol

`RNDV_poolsize` The pool of locally shared memory from which the buffers for the rendez-vous message transmission are dynamically allocated.

default: 1 MB

Other values (lower or higher) are possible. Lower values lead to less performance since partial message transfers will occur more often (at least for every message > `RNDV_memorysize`). Higher values lead to more memory consumption. This value is aligned to page size.

Note:

- If SCI-MPICH can not allocate a buffer of the specified size, it halves the buffer size and try again until the buffer size is smaller than the eager buffer size.

`RNDV_blocksize` The block size for the pipelining of writes and reads.

default: 24 kB

Other values (lower or higher) are possible. The smaller the value, the more protocol overhead will be created. Increasing the value may reduce the pipelining effect and thus the effective bandwidth. The optimum value depends on the relation of local and remote memory bandwidth and thus needs to be manually determined for each system. However, the default value usually delivers more than 95% of the peak performance for all kinds of systems we tested.
This value is aligned to multiples of the stream buffer size internally.

Notes:

- A value of 0 turns off the interleaving

`RNDV_receipt` The size of the remote receive buffer, expressed as a multiple of the `RNDV_blocksize` value.

default: 12

Other values (lower or higher) are possible. Not using the maximum available memory for this receipt buffer make better use of the existing shared memory resources and may provide better overall performance in situations with many concurrent rendez-vous transfers towards one process. The peak point-to-point performance may be improved by using higher values.

A value of 0 means that the maximum available memory should be used for the reception of a rendez-vous message.

`RNDV_blocking` The blocking variant of the rendez-vous protocol uses a traditional ring buffer with flow control via shared read- and write-pointers instead of the control-message based flow control in the standard rendez-vous protocol. This results in a higher point-to-point bandwidth especially for messages of data which is not cached or which exceeds the size of the cache. However, it will hinder the processing of incoming messages during the transfer of a rendez-vous message (it blocks) which may reduce effective throughput in certain situations. This means, it is best to use this protocol for communication pattern where the receiver waits for only this message to arrive. Therefore, this protocol is always used for ready sends (via `MPI_Rsend()`) with the rendez-vous protocol. For cases where other send modes do also benefit from the blocking behaviour, it is possible to use this protocol for all PIO-based rendez-vous transfers.

0 uses non-blocking rendez-vous protocol for all transfers except ready-sends
1 uses blocking rendez-vous protocol for all PIO-transfers
default: 0

Notes:

- The blocking protocol may exhibit very high performance degradation for situations where one process has to process a very big number of outstanding/concurrent rendez-vous messages. This may be avoided by defining a bigger memory pool via the `RNDV_memorysize` parameter.

`RNDV_MAXRECVS` To limit the degree of parallel message transmission, which may have a negative impact on performance and resource availability, the maximum number of concurrent rendez-vous receive operations a process performs is limited. This limit can be changed using the `RNDV_MAXRECVS` parameter. Incoming receive requests which exceed this limit will be postponed until the ongoing receive operations have finished.

default: 0 (unlimited)

`RNDV_MAXSENDS` Same as `RNDV_MAXRECVS` for the maximum number of concurrent send operations of a process.

default: 4

Memory usage of the rendez-vous protocol for each process:
`RNDV_memorysize`

Single-sided Communication

SCI-MPICH supports the complete set of single-sided (aka 'one-sided') communication as specified by the MPI-2 standard.

`SSIDED_ENABLED` Disable single-sided (aka one-sided) communication. Usually, there is no reason to do so.

0 disables single-sided communication
1 enables single-sided communication
default: 1

`SSIDED_RMTPUT_PRIVATE` Threshold value: `MPI_Get` accesses above this value will be executed as a remote put operation to speed up the transfer. This value applies if the window is located in private memory at the target.

default: 148

`SSIDED_RMTPUT_SHARED` Threshold value: `MPI_Get` accesses above this value will be executed as a remote put operation to speed up the transfer. This value applies if the window is located in shared memory at the target.

default: 122

For more information on the protocols, look for documentation at our WWW site.

3.5.2 Global Device Configuration

Additionally, there is a number of options which have effect not only on a single protocol, but on multiple protocols or the whole device

Generation of runtime-statistics If you want to create runtime statistics for a variety of functions in the device, use the `STATISTICS` option. Each process will print a table of function call statistics and counters to stdout after proper termination. To use this option, the configure script needs to be invoked with the `--enable-devdebug` switch You then can control the generation of the statistics with the following option:

`STATISTICS` Switch to turn statistics on or off with the supplied value:

0 turns them off 1 turns them on default: 0

Notes:

- The use of this function is mainly for internal debugging and tuning and is probably of no meaning to the average user.

Watchdog SCI-MPICH uses the Watchdog of the SMI library to shut down the whole application in the case that one or more processes have crashed or are fully blocked. For debugging, this behaviour is not desired, thus you can control the watchdog via this parameter.

`WATCHDOG` Turn the watchdog off or enable it with a defined time-out

0 turns the watchdog off
$>$ 0 turns the watchdog on and sets the time-out to the given value (in seconds). The watchdog will consider a process as dysfunctional when it does not respond within the time-out.
default: 5

Self-send shortcut In the case of a process sending a message to itself, a shortcut has been implemented which uses a single-copy variant for the eager and rendez-vous protocol (copying directly between the send- and receive user buffer). As this acceleration is usually desired, it is turned on by default.

`SENDSELF` Switch to turn on single-copy variant of the eager and rendez-vous protocol for messages send to itself.

0 turns them off
1 turns them on
default: 1

True asynchronous sends Depending on the MPI-Applications you are running, it may be desirable to use an implementation of MPI_Isend which does really work asynchronously to allow the overlap of computation and communication. SCI-MPICH allows this kind of communication, using DMA or a thread for PIO. However, depending on the application it may not always improve performance. Therefore, you can control the use of asynchronous communication via the following option:

`ASYNC_PROGRESS` Switch to turn the usage of asynchronous Isends on or off with the supplied value:

0 turns them off
1 turns them on
default: 0

Notes:

- This option is only available if the configure-option `--enable-devthreads` or `--enable-libthreads` was used.

Use of DMA for message transfer The PCI-SCI adapter boards support DMA between shared memory segments. We have implemented support for DMA transfers – check for yourself if it helps with your application. Use the DMA_MINSIZE options to select a threshold from which DMA is efficient on your system.

`USE_DMA_PT2PT` Switch to turn on DMA transfer mode for messages send via asynchronous MPI send calls.

0 turns DMA off
1 turns DMA on, using non-mapped remote segments
2 turns DMA on, using mapped remote segments (slower)
default: 0

Notes:

- This option only has an effect if asynchronous transfers are enabled (see option `ASYNC_PROGRESS`) or if registering and zero-copy operations are enabled (see options `REGISTER` and `ZEROCOPY`).
- Enabling DMA implies setting `SENDSELF` to 1, to (this is done internally independent from the user setting of `SENDSELF`).
- DMA transfer between processes on the same node is not possible. In this case, SCI-MPCH uses the traditional PIO protocol variant.
- DMA with non-mapped remote segments is way faster than with mapped remote segments because the overhead for mapping a remote segment is very significant and increases nearly linearly with the size of the remote segment. There are no reasons not to use non-mapped remote segments with DMA. The old mode is for reference only.

DMA transfer threshold Using DMA for message transfers can be significantly faster than PIO. This is especially true if the message buffers can be used as source and target for memory transfers (see options `REGISTER` and `ZEROCOPY`). However, setting up a DMA transfer includes some overhead, especially if registering and zero-copy is not supported by the platform. For transfers up to a certain message size, depending on the capabilities and performance characteristics of the platform and the communication pattern of the application. Therefore, you can adjust the minimum size for a message to be transferred via DMA. There are two different thresholds, one for the synchronous (blocking) and one for the asynchronous (non-blocking, see option `ASYNC_PROGRESS`) version of the rendezvous protocol. Doing DMA for the asynchronous protocol variant may pay off for smaller message size, because the focus for this protocol is to transfer data in the background, but not necessarily as fast as possible (as it is the case for the blocking protocol). The short and the eager protocol do never use DMA.

`SYNC_DMA_MINSIZE` Threshold value for the size of a message which is to be transferred using DMA synchronous (blocking) rendez-vous protocol...

default: 512 kB

`ASYNC_DMA_MINSIZE` Threshold value for the size of a message which is to be transferred using DMA by the asynchronous (non-blocking) rendez-vous protocol.

default: 32 kB

Zero-copy transfers For messages sent with the rendez-vous protocol, SCI-MPICH is able to perform zero-copy transfers (direct copy from user send-buffer to user receiver buffer without intermediate copies) for optimised performance. This is done by directly connecting to the remote receive buffer. However, a number of conditions must be met to allow for zero-copy transfers, read the notes below. If zero-copy is enabled, but can not be performed because not all necessary conditions are met, SCI-MPICH will fall back to the standard protocols.

`ZEROCOPY` Switch to allow or disable zero-copy transfers.

0 turns off zero-copy
1 turns on zero-copy
default: 1

Notes:

- This option only has an effect if DMA transfers are enabled. (option `USE_DMA`)
- Using persistent communication improves the performance of zero-copy transfers (`MPI_Send_init()` or `MPI_Recv_init()` see MPI documentation).
- The SCI driver (IRM/SISCI) must support registration of user-allocated memory. This is currently under development and will soon be available in the official Dolphin sources (see option `REGISTER`).

One of the following conditions for the memory of the send/receive buffer must be met to allow zero-copy:

1. The memory for the send/receive buffers should have been allocated via `MPI_Alloc_mem()` and should be freed with `MPI_Free_mem()`.
2. If the two functions from 1. can not be used, but `malloc()` / `free()` or a similar allocation mechanism is used, de-allocation and subsequent re-allocation of communication buffers must be avoided. Allocating a buffer once, using it for communication, and finally freeing it is fine.

3. If condition 2. can not be assured, it is recommended to recompile the application with the macro `MPI_MALLOC_WRAP` defined (option `-DMPI_MALLOC_WRAP` for a C/C++ compiler). This will activate macros in `mpi.h` which replace all occurrences of `malloc()` and `free()` with equivalent calls to `MPI_Alloc_mem()` and `MPI_Free_mem()`.

Registering user-allocated buffers This option is relevant in conjunction with zero-copy. It determines if SCI-MPICH should try to register user-allocated buffers (the send- and receive buffers handed over via the MPI calls) with SCI. If the registering succeeds (depending on available resources), the buffer can directly be used as a DMA source or target, allowing zero-copy transfers even for buffers not allocated via MPI. If the registering fails, SCI-MPICH will use intermediate buffers for transferring the data.

`REGISTER` Switch to allow or disable registering of user-allocated memory.

0 turns off registering
1 turns on registering
default: 1

Note:

1. The functionality required for registering memory is not provided in the standard Dolphin drivers so far. We are working on an inclusion of this feature.

On-demand connection Normally, all processes of an SCI-MPICH application connect all their memory segments to each other during `MPI_Init()`. For a large number of processes, this can take a considerable amount of time. However, not all of these connections are actually used during the execution of the application since it occurs that not each process communicates with each other using every available protocol.

To speed up the startup, SCI-MPICH offers a on-demand-connection option. Using this option, the memory segments are configured, but not actually connected on startup. Instead, the connection is performed on demand during the execution of the application.

`DEMAND_CONNECT` Switch to turn on or off fast startup using delayed segment connections.

0 turns on-demand connection off
1 turns on-demand connection on
default: 1

Note:

1. This option has become meaningless for current SCI-MPICH versions which enforce a fully dynamic resource management (see option `RESOURCE_SCHED`).

2. If the fast startup is used, SCI-MPICH can no longer dynamically adjust the size of the memory segments to the available resources. This means if a segment is to be connected on demand and the connecting process has no sufficient SCI resources available to perform the connection, the application will abort when releasing other resources does not help. Therefore, fast startup should only be used for memory configurations that are known to work.

Synchronisation of memory transfers To avoid thrashing on the PCI bus of nodes in the system, SCI-MPICH synchronises remote memory accesses between processes. There are different modes how this synchronisation can be performed which are selectable via the setting of `MEMCPYSYNC_MODE` in the configuration file. Additionally, a threshold value for the synchronisation with respect to the size of a transfer operation can be set.

`MEMCPYSYNC_MODE` Select how memory transfer operations are synchronised between the processes.

0 no synchronisation
1 synchronise only incoming transfers on the destination node
2 synchronise only outgoing transfers on the source node
3 synchronise both incoming and outgoing transfers
default: automatic setting of mode according to topology and the type of the PCI-SCI host adapter (recommended!).

`MEMCPYSYNC_MIN` Set a threshold value for the size (in bytes) of a inter-node memory transfer operation to be synchronised. Any value bigger than zero is valid. This value specifies the largest size of a memory transfer which is not synchronised.

default: 2048

Notes:

1. The current implementation of `MEMCPY_SYNC` does not consider multiple PCI-SCI adapters in one node, but synchronises on a per-node basis (not on a per-adapter basis as it would be required for efficient use of multiple PCI-SCI adapters).

2. This synchronisation can significantly improve performance on systems with Dolphin LC-2 link chips (D31x and D32x series of PCI-SCI adapters). In this case, disabling synchronisation (by choosing mode 0) is not recommended for systems with more than 4 nodes.

   For systems with LC-3 link chips (D33x series of PCI-SCI adapters), this synchronisation is not required, but usually reduces performance if used. The automatic configuration turns the synchronisation off for LC-3 based systems.

3. Memory transfers performed by the short-protocol and intra-node transfers are never synchronised.

PCI-SCI adapter scheduling Systems with multiple PCI busses can benefit from using multiple PCI-SCI adapters in parallel. There are different ways to use the multiple adapters which are selectable on startup.

`ADAPTER_MODE` Specify the usage and assignment of multiple PCI-SCI adapters in a node.

0 all processes use the same default PCI-SCI adapter
1 use one adapter for incoming and one for outgoing transfers (called IMPEXP mode, for import-export)
2 each process use a different adapter for all his transfers (called SMP mode)
default: 0

Note:

1. This option is still experimental.

PCI-SCI transfer verification For experimental reasons, PCI-SCI transfer verification can be turned off. This will lead to slightly lower latencies - but may of course lead to crashes or incorrect results.

`NO_VERIFY` Specify PCI-SCI transfer verification.

0 enables verification
1 disable verification
default: 0

Checking for new messages Per default, SCI-MPICH polls for the arrival of new messages. This is the fastest way to detect the arrival of a new message and guarantees the very low latency of down to 5μs for inter-process, inter-node message exchange. However, in some environments blocking (on remote interrupts) while waiting for a new message is preferred to polling as it leaves the CPU available for

other task while an MPI process is just waiting for a new message. SCI-MPICH offers an option to chose between polling and three blocking variants.

`MSGCHK_TYPE` Specify the type of checking for new messages.

0 use polling only (low message latency, high CPU usage)
1 use interrupts only (high message latency, lower CPU usage)
2 use interrupts with subsequent polling
3 use interrupts with a number subsequent blocked checks
default: 0

`MSGCHK_DELAY` For types 2 and 3 only: Specify the delay (amount of time in us) to poll (type 2) or block (type 3) after an interrupt did come in.

default: 50

`MSGCHK_REPEAT` For type 3 only: Specify the number of time to block and check for new messages after an interrupt was processed.

default: 10

Note:

1. The latency for small messages will increase (up to a factor of 10), and the bandwidth for big messages will decrease (up to 10%) using a non-polling mode. However, the actual performance impact on an application depends very much on the communication pattern.
2. This option is only available if the configure-option `--enable-devthreads` or `--enable-libthreads` was used.
3. Turning on of the non-polling modes automatically turns on the asynchronous progress mode (see option `ASYNC_PROGRESS`)

Optimised collective operations The standard task of a MPI communication device is to process point-to-point send and receive operations. These operations are also used by the upper layers of the MPI library to perform collective operations (like `MPI_Barrier`, `MPI_Alltoall`, ...). However, a communication device has the option to perform some or all collective operations, too, if it has better means to do so (or if the standard implementation performs bad with this specific device). SCI-MPICH also has some optimised collective operations which can be selected with this switch.

Currently, the following collective operations are replaced:

`MPI_Allgather, MPI_Allgatherv` with custom SCI-MPICH version (performs good for even number of processes)

`MPI_Allreduce, MPI_Reduce, MPI_Scan` with algorithm by Rolf Rabenseifner of HLRS Stuttgart (performs better for bigger messages, switching between standard algorithm for small messages and custom algorithm is done automatically), also in an SCI-optimised variant, or with custom SCI-MPICH pipelining (even better for big messages). Additionally, the reduce operations are MMX/SSE-optimised on IA-32 platforms.

`MPI_Reduce_scatter` with custom SCI-MPICH version (performs great for short- and eager-sized messages, still good for large messages using asynchronous transfers)

`MPI_Alltoall, MPI_Alltoallv` with custom SCI-MPICH version for increased performance and better resource cache rate.

`MPI_Barrier` with shared-memory barrier - very low latency.

`MPI_Bcast` with pipelined message transfer for rendez-vous messages.

`MPI_Scatter, MPI_Scatterv` with custom SCI-MPICH version with good improvement for small messages.

`COLL_CUSTOM_ENABLE` Specify if the custom collective operations of SCI-MPICH should be used or not.

0 use standard MPICH collective operations
1 use custom collective operations of SCI-MPICH
default: 1

`USE_DMA_COLL` Specify if the custom collective operations of SCI-MPICH should use DMA for message transfer or not (currently only applies to the pipeline protocols).

0 use PIO only
1 use DMA if possible
default: 1

Various options are available to fine-tune the behaviour of the implemented custom-collective operations. These are mostly of interest for experiments and research.

`COLL_BARRIER` Enable or disable the use of the shared memory

default: 2

Various options are available to fine-tune the behaviour of the implemented custom-collective operations. These are mostly of interest for experiments and research.

`COLL_BARRIER_FANIN` Fan-in (and fan-out) of the shared-memory barrier. This related to the inter-node synchronisation; for intra-node synchronisation, the fan-factor is always set to the number of processes running on the node.

default: 2

Various options are available to fine-tune the behaviour of the implemented custom-collective operations. These are mostly of interest for experiments and research.

`COLL_REDUCE_LONG` Threshold for differentiation between "short" or "long" vectors for reduction.

default: size of eager buffers (`EAGER_BUFSIZE`)

`COLL_REDUCE_LONG_TYPE, COLL_REDUCE_SHORT_TYPE` Choice of the communication algorithm for reduction of short or long vectors. Valid values are:

0 tree
1 rabenseifner
2 pipeline
default: short: 0 – long: 2

`COLL_REDUCE_FANIN` Fan-in for the reduce-communication-tree (mostly for short vectors)

default: 4

`COLL_ALLREDUCE_TYPE` Choice of the communication algorithm for global reduction (applies for all vector lengths). Valid values are:

0 default (reduce followed by broadcast)
1 rabenseifner
2 pipeline
3 allgather
4 rabSCI
default: 4

`COLL_ALLGATHER_BARRIER` Message-size related lower threshold for using a barrier to synchronise / separate the communication phases of the allgather-communication.

default:

`COLL_ALLTOALL_MIN` Message-size related lower threshold for using the custom all-to-all communication. Usually, the custom-algorithm is faster for rendez-vous messages.

default:

`COLL_ALLTOALL_TYPE` Choice of the routing algorithm to select communication partners during the all-to-all communication. Valid values are:

0 plain
1 1-D
2 2-D
3 scampi
4 mpich
default: 1

`COLL_ALLTOALL_BARRIER` Message-size related lower threshold for using a barrier to synchronise / separate the communication phases of the all-to-all-communication.

default:

`COLL_PIPE_MIN` Message-size related lower threshold for using the custom pipeline algorithm.

default: eager buffer size

`COLL_PIPE_DMA_MIN` Message-size related lower threshold for using the custom pipeline algorithm with concurrent DMA- and PIO transfers.

default: 2 * `COLL_BCAST_BLOCKSIZE`

`COLL_PIPE_BLOCKSIZE` Blocksize for pipelining.

default: 32kB

`COLL_PIPE_NBRBLOCKS` Number of pipeline-buffers to use.

default: 8

`COLL_PIPE_DYNAMIC` Allow or disallow dynamic adoption of pipeline-buffer size, depending on the message size.

default: 1

`COLL_SCATTER_MAX` Maximum size for messages to use the custom scatter algorithm. default: `EAGER_BUFSIZE`

MPI-2 memory allocation SCI-MPICH supports the MPI-2 functions `MPI_Alloc_mem()` / `MPI_Free_mem()` which can be used to allocate "special" memory for message buffers which gives better performance than memory allocated via `malloc()` / `free()`. The memory manager for this allocation does only use "special" SCI-optimised memory areas if the requested size exceeds a lower threshold (because allocating smaller buffers does not pay off in performance, but uses up resources

and leads to memory fragmentation). The value for this threshold can be set via `ALLOC_MINSIZE`. Buffers sized below this threshold are allocated via standard `malloc()` internally.

`ALLOC_MINSIZE` — This parameter should be increased if your application makes increased usage of `MPI_Alloc_mem()` for small buffers, leaving not enough resources for bigger buffers which would best benefit from SCI-optimised memory.

default: 64 kB

`ALLOC_POOLSIZE` — The allocation of SCI-optimised memory is performed from a pool of such memory. The size of this pool can be set via `ALLOC_POOLSIZE`.

If an allocation request can not be satisfied from within this pool, SCI-MPICH will try to allocate the SCI-optimised memory from a memory area only created for this request. Creating such an area costs significantly more time than allocating from the pool, thus the pool should be at least twice as big as the usual allocation requests to avoid performance decrease.

In case that a pool of the specified size can not be allocated, SCI-MPICH reduces the size accordingly down to the size of the first allocation request.

default: 4 MB

Optimised non-contiguous communication SCI-MPICH offers optimised communication techniques for sending/receiving non-contiguous derived datatypes which offer up to twice the bandwidth than the standard MPICH technique (which is also used in all other MPI implementations we know).

Using this technique for (h)indexed and struct may, under certain conditions, lead to incorrect receive data if the datatype used for receiving is different then the datatype used for sending. For this reason, the optimisation is disabled for these types by default (full MPI conformance). If the situation described above does not apply to your application, you can turn on the optimisation for these types, too (limited MPI conformance).

`NC_ENABLE` — Specify if the custom non-contiguous (n-c) communication technique of SCI-MPICH should be used or not.

0 use standard MPICH n-c communication
1 use optimised n-c communication w/ full MPI conformance
2 use optimised n-c comm. w/ limited MPI conformance
default: 1

`NC_MIN` Specify the minimal size (in bytes) of a contiguous part of a non-contiguous datatype to be transferred directly. Data blocks below this size will be gathered locally to improve the performance of remote SCI operations.

default: 8

Resource Scheduling SCI-MPICH manages the SCI resources (local and remote SCI memory, DMA connections and registered memory regions) dynamically to ensure that even with limited SCI resources, the application can use a maximum processes. To enhance the performance of this dynamic resource management, these resources are not immediately destroyed if no longer required for a specific message transfer, but are cached for possible reuse. Only if a new resource can not be created, a number of the cached, but currently unused resources is destroyed to free system resources for the new resource to be created. This implies that there must exist a policy on which a decision which of the cached resource to choose for destruction. A number of different policies (called scheduling strategies) does exist which can be selected via a device parameter called `RESOURCE_SCHED`.

`RESOURCE_SCHED` Specify the scheduling strategy for the dynamic resource management. Possible values:

0 `IMMEDIATE` - don't cache at all, but destroy free'd resources immediately (mainly for testing purposes)
1 `LRU` - destroy the least recently used resource.
2 `LFU` - destroy the least frequently used resource.
3 `BEST_FIT` - destroy the resource which fits best (size).
4 `RANDOM` - randomly select a resource to destroy.
5 `NONE` - do not destroy any cached resources, but instead terminate the application if not enough resources exist (mainly for testing purposes).
default: 1 (`LRU`)

Different policies can be applied concerning the caching of resources. These policies can be controlled via certain parameters:

`CACHE_REGISTERED` Keep registered memory regions registered after the data transfer is complete? Caching is usually fine and increases performance, but may lead to data corruption if a `malloc()`'ed region was registered, then `free()`'d, and another, newly `malloc()`'ed region with addresses in the same range is to be registered. If such a situation can be guaranteed not to occur (by using the MPI allocation functions), it is safe to set this option to '1'.

default: 0 (do not cache registered regions)

`CACHE_CONNECTED` If caching of registered memory is enabled, it is also safe to enable caching of connections to remote memory regions by setting this option to '1'.

default: 0 (do not cache connections to remote regions)

Performance Modelling In cases where an application should scale better than it actually does, the fun with parallel programming starts. There are a lot of methods to improve the scaling of a parallel application, foremost by algorithmic improvements. However, if no further possibilities for algorithmic improved can be found, it is required to locate the performance bottlenecks that hinder scaling. Instrumenting the application and evaluating the produced traces is one way to do this. When the bottlenecks are located, it can be estimated how much a platform with higher performance would improve the scaling of the application.

SCI-MPICH offers another interesting way that is called performance modelling: by controlling certain performance characteristics of the MPI communication (below the actual value that is reached on the used platform), better estimations based on these experimental runs can be made on how much impact this specific characteristic has on the overall application performance.

A number of characteristics can be controlled, which is explained below. Naturally, it is not possible to increase the performance (raise the bandwidth, reduce the latency) beyond the limits implied by the used system platform.

`PERF_GAP_LTNCY` The gap latency is the delay of the sender between two subsequent transmissions of a control message (which may also be a short message) to the receiver. `PERF_GAP_LTNCY` defines the number μs of that are added to the natural delay.

default: 0

`PERF_SEND_LTNCY` The send latency is the duration that the sender needs to send a control message (which may also be a short message) to the receiver. `PERF_SEND_LTNCY` defines the number of μs that are added to the natural duration.

default: 0

`PERF_RECV_LTNCY` The receive latency is the duration that the receiver needs to read a new control message (which may also be a short message) from the incoming queue. `PERF_RECV_LTNCY` defines the number of μs that are added to the natural duration.

default: 0

`PERF_BW_LIMIT` To limit the bandwidth to a maximum value, specify this upper limit (in bytes per second) with `PERF_BW_LIMIT`.

default: 0 (which means 'no limitation')

`PERF_BW_REDUCE` To generally reduce the bandwidth, specify the reduction with `PERF_BW_LIMIT`. The given value will be used to reduce the bandwidth to value % of the natural bandwidth. If the bandwidth was already limited by `PERF_BW_LIMIT`, no further reduction will be applied.

default: 0 (which means 'no reduction')

3.6 Memory Allocation

For SCI-MPICH to achieve good performance with features like zero-copy, it is important to have control over the memory allocation performed by the user. Therefore, explicit and implicit memory allocation via the MPI library is supported.

3.6.1 Explicit Memory Allocation via SCI-MPICH

The MPI-2 standard defines functions to allocate and free memory which are supported by SCI-MPICH: `MPI_Alloc_mem()` allocates a block of memory which can subsequently be free'd with `MPI_Free_mem()` (see MPI-2 standard for exact prototypes). The `MPI_Alloc_mem()` function accepts an additional `MPI_Info` parameter. An `MPI_Info` type can contain an arbitrary number of (key, value) pairs (called attributes) to specify certain demands or conditions. SCI-MPICH supports attributes with two keys and different values as defined in table 3.2.

3.6.2 Implicit Memory Allocation via SCI-MPICH

If an application allocates memory via `malloc()`, SCI-MPICH does not have any control over this memory area which hinders certain optimisations concerning the registering of the memory for zero-copy DMA transfers. Therefore, the include file `mpi.h` contains a macro which maps all `malloc()` calls to an internal allocation function which in turn uses `MPI_Alloc_mem()` to allocate private memory with proper alignment. Likewise, the free() call is mapped to `MPI_Free_mem()`. This mapping will result in a negligible overhead for the allocation, but will help to improve communication performance.

However, conflicts between these mapped `malloc()`/`free()` calls and the original calls need to be avoided (because they will surely lead to problems, maybe even corrupted data or application crashes). Such a conflict would be to allocate memory within a code module which has `mpi.h` included, and deallocating it in another module which did not include `mpi.h`.

This behaviour of SCI-MPICH can be controlled upon the pre-compile configuration (using the `--disable-mallocmap` or `--enable-mallocmap` switches), in the source code by defining `MPI_ENABLE_MALLOCMAP` or `MPI_DISABLE_MALLOCMAP`

Key	Value	Purpose
type	private	Enforce allocation of private (non-shared) memory. This allows to let `MPI_Alloc_mem()` behave like `malloc()`.
	shared	Enforce allocation of shared (SCI shared) memory. The call will fail if not enough shared memory is available.
	default	`MPI_Alloc_mem()` will decide if shared or private memory will be allocated (based on the device configuration settings `ALLOC_MINSIZE` and the available shared memory resources). This call only fails if also a standard `malloc()` would fail due to a out-of-memory condition.
alignment	<integer>	Align the returned address of the buffer to the specified integer value. Can be used with any type (private or shared) of memory. Relevant to have private memory properly aligned for pinning/registering (an alignment of 64 is recommend, 8 is required).

Table 3.2: Recognised attributes of MPI_Info objects for memory allocation

or upon compilation via the compiler frontend scripts using the `-mallocmap` or `-nomallocmap` switches. The default is always to enable the mapping of `malloc()` and `free()`.

3.7 MPI-2 One-Sided Communication

SCI-MPICH supports one-sided communication[2] which is part of the MPI-2 standard. You can use the according calls for local- and remote-shared-memory communication.

The one-sided communication, regarding the setup of windows, synchronisation and data transfer can be used as defined by the MPI-2 standard. You need to consider that the effective performance that will result from these operations does strongly depend on the type of memory (private or shared) that you used for the creation of the window. Currently, none of the asserts as defined by the MPI-2 standard are supported (this means they will be safely ignored by SCI-MPICH).

[2]One-sided communication is also called single-sided communication, which means the same.

3.8 SCI-MPICH under Windows

The source code for SCI-MPICH is identical for Unix systems and Windows (NT, 2000 and XP). However, the compilation and usage is different and are described in this chapter.

3.8.1 Compiling the Library

[no special information available yet - please refer to chapter 4.]

3.8.2 Running Applications

[no special information available yet - please refer to chapter 4.]

3.9 Common Questions and Problems

There are some frequently asked questions and problems concerning SCI-MPICH which we try to answer and solve in this chapter. The quality of this chapter heavily depends on the feedback we get from you - the user. Don't hesitate to bother us with questions and problem reports - but please read the documentation carefully in advance!

3.9.1 General questions concerning SCI-MPICH

Q1: Can I use the `ch_smi` device with MPICH release x.y.z?
A1: The `ch_smi` device is integrated in MP-MPICH which has developed far away from the MPICH from ANL. A "re-integration" into the current MPICH release is not feasible; instead, the relevant changes in MPICH should be ported to MP-MPICH, if necessary.

Q2: Is SCI-MPICH thread-safe?
A2: No. As MPICH itself, on which SCI-MPICH is based, is not thread-safe, there is no real need for this. However, the `ch_smi` device in SCI-MPICH can be configured to use threads internally for true asynchronous communication. The device is fully thread-safe, but it still is complicated enough to ensure true thread-safety of the complete MPI library - the device is the most critical component concerning thread-safety.

Q3: How can I make use of my SMP nodes?
A3: With the current SCI drivers, you can specify more than one process per node in a SCI-MPICH application. The intra-node communication is performed via local shared memory This means SYS-V style `shmget()`, `shmat()` on Solaris and `mmap()` of files for Linux because of performance problems with

SYS-V calls on certain Linux platforms. However, due to performance related side-effects of SMP usage, you need to confirm the usage of an SMP configuration with the `-smp` option to `mpirun`.

Q4: Can I use SCI-MPICH in a heterogeneous environment - that means between systems running different operating systems or between x86- and Sparc-based systems?

A4: We successfully tested SCI-MPICH on a mixed Solaris-x86 and Linux cluster using the usual `mpirun` script. Communication of Unix- and NT-systems via SCI-MPICH is also possible, but there is no convenient startup-mechanism available (yet) - the processes need to be started individually via the command line. Connecting Sparc-based systems with x86-based systems will probably cause endian-related problems - we could not test it yet.

Q5: Does SCI-MPICH support extensions and tools like C++ bindings, MPE, ROMIO, Vampir, Totalview, ... ?

A5: SCI-MPICH is based on MPICH; thus you can use any extension and tool which can be used with MPICH 1.2.0 (also see chapter 3.4). However, we could not test with all available tools. Please report back success or failure stories.

3.9.2 Problems compiling the libraries and MPI applications

Q1: When I run configure for the SMI library, the output at the end says

```
> *** no SCI support -> SMP support only ***
```

although I have specified the exact path to the SISCI files using the `--with-sisci` option. It is the path where the Dolphin installation script has installed everything. What's wrong?

A1: The Dolphin installation script unfortunately installs the SISCI API library to a subdirectory of the `/path/to/SISCI/lib` directory. You need to move it one level up for the configure script to find it. Hopefully, Dolphin will fix that (they are informed).

Q2: I have fixed the parameters for configure, and all required files are placed at the correct locations, but configure still gives the same error messages, i.e. says "no SCI support".

A2: The configure script caches its findings in the file `config.cache`. Remove this file, or just call `make distclean`, and rerun the configure script to enforce a new configuration run from scratch.

Q3: During the make process of SCI-MPICH, I get an error message saying

`Could not link a C program with static MPI libraries`

What does that mean?

A3: Some code in the `ch_smi` device makes uses of functions that are only available in dynamically linked executables. Trying to statically link such this code into an executable fails. If you really want statically linked executables, you need to configure with the `--disable-sharedlib` option. This will disable the relevant code fragments in the device. However, this does of course mean that some functionality is not available (device-specific, optimised collective operations, in this case). You do not need to do anything about this message unless you really want statically linked executables. However, future versions of MP-MPICH (to which SCI-MPICH belongs) will rely even more on dynamic linking.

Q4: When I start my successfully linked MPI application via the `mpirun` script, the execution does not start, but fails with an error message like ...

A4: When the application process is launched on a remote node, it can not find the shared SCI-MPICH libraries that it is linked with. There are different ways to solve this problem:

- Do not link dynamically, but statically using the `-static` option when calling `mpicc` to link your application.
- Encode the path to the shared libraries into the executable by supplying the `-shlibpath` option when calling `mpicc` to link your application.
- Make sure that the environment variable `LD_LIBRARY_PATH` contains the path to your SCI-MPICH library directory. This may be achieved by editing your shell profile, i.e. the `.bashrc` file if you are using the bash shell. Refer to the operating system / shell documentation for more information.
- Install the SCI-MPICH libraries into a directory which is searched by default by the operating system when it's looking for shared libraries (`/usr/local/lib` is usually a good place). Use the `--prefix` option for the configuration of SCI-MPICH to specify this location, and run `make install` after building SCI-MPICH. Refer to the operating system documentation for more information.

3.9.3 Problems running MPI applications

Q1: I have problems which seem to be SCI related, or my application does not start at all.

A1: Generally spoken, an SCI-MPICH application is also an SMI application because it is based on the SMI library. This means, first make sure that

SMI applications run fine on your system. Refer to the SMI manual for instruction on how to check this.

Q2: I do supply a device configuration file to SCI-MPICH. But I have the impression that SCI-MPICH does not behave as it should according to the settings in this file.

A2: SCI-MPICH performs validation checks on each of the settings. Also, it may occur that the available resources do not match the demand specified by the device configuration file. To see the setup that SCI-MPICH really uses, use the verbose startup option (`-v` for the `mpirun` script). However, the memory buffer related information printed with this option only apply to process 0 - other processes may have larger or smaller memory buffers etc. because they have different resources on their nodes. In contrast, the information concerning the general setup are identical throughout all processes.

Q3: When I start an application across multiple nodes, I get a message like
`[0] could not create base segment`
during the startup, and my application seems to hang.

A3: A possible reason for this behaviour is that you are using an SMI library which is not configured for SCI and thus tries to communicate via local shared memory - which does not work across multiple nodes. Check which SMI library the executables are actually using when launched on the different nodes (check `LD_LIBRARY_PATH` for dynamic linking!); then check if the configuration of the SMI library you are using is for SCI. SMI will tell you if it can use SCI when you configure it.

Q4: When I start an application on a single node, I get a message like
`[0] could not create base segment`
during the startup, and my application seems to hang.

A4: A possible reason for this is that for some reason the System-V shared memory resources are exhausted on the specified node. Check with the `ipcs` command and remove all unneeded segment with `ipcrm`. SCI-MPICH usually cleans this up, even when the application should crash.

Q5: When I start an application across multiple nodes, I get a message like connection refused during the startup, and my application does not run.

A5: The start script `mpirun` launches the processes on the remote hosts using a remote shell (rsh). Make sure that you cluster is configured to allow remote shells and logins without authorisation from the node which executes the `mpirun` script. The easiest way usually is to edit the `$(HOME)/.rhosts` file. However, some Linux distributions are configured with high security by default and may disallow `rshrlogin` at all. Contact the system administrator to change this. We are planning to implement a new startup mechanism which no longer is based on these mechanisms.

Q6: My application runs fine with N processes. But if I start it with N+1 processes, a message like
`[0] SMI ERROR: _smi_init_mp failed`
or
`[0] SMI ERROR: _smi_barrier_init() failed`
is printed and the application terminates.

A6: These error message indicate that the size of the internal shared memory regions is not sufficient for the number of processes you are using. The default size of these regions is 256kB. It is recommended to double the size in case it is to small. This can be done via an additional parameter when configuring the SMI library. To increase the size to 512kB, add this parameter when invoking configure in the SMI library: `--enable-intsgmtsize=$((512*1024))`

Q7: My application runs fine with N processes. But if I start it with N+1 processes, a message `SCIConnectSegment()` failed. is printed and the application terminates.

A7: This error message indicates that the SCI kernel driver (IRM) has not enough identifiers for virtual connections available. The amount of these identifiers can be increased by two ways:

- Statically by changing the source code and recompile & reinstall the driver: Locate the file `irmConfig.h` in the driver sources (usually in `DIS/src/IRM/drv/src`) and set the definition of `DEFAULT_MAX_VC` to the value desired; then recompile & reinstall.
- Dynamically by passing a parameter to the module (Linux only!): the module parameter is `max-vc-number`, and you can pass it directly or via a line in the configuration file `pcisci.conf` which should look like: `max-vc-number=1024`

Q8: When I start an application across multiple nodes, I get one or more message(s) like
`perror: : no such a device`
`[0] SMI ERROR: SCIInitialize() failed`
and the application terminates.

A8: This message indicates that the SCI driver (IRM and/or SISCI) is not correctly installed or is not running on one or more nodes. Please make sure that drivers are running on all nodes by using the `scidiag` tool (to verify the IRM is running) and some SISCI examples like `scibench2` on all nodes.

Q9: When I start an application across multiple nodes, I get one or more message(s) like
`[0] SMI ERROR: SCIInitialize() failed`
and the application terminates.

A9: This message may indicate that the version of the SISCI library which was

linked into the executable does not match the installed SCI driver on the concerned nodes. Relinking the executable with the SISCI library from the distribution of the installed SCI driver should help. You should always take care that all nodes in the cluster run identical versions of the SCI software.

Q10: I use Linux with a 2.4 kernel and the default Dolphin SCI drivers. I also have installed the bigphysareapatch. When I start an application across multiple nodes, it works o.k. at the beginning, but after a few runs I get a message(s) like

```
[0] SMI ERROR: Not enough SCI resources available:
               SCICreateSegment() failed
*** Application aborted internally by process 0.
```

A10: Although you have a kernel with `bigphysarea`-Patch running and thus should have enough SCI resources available to run the application, the SCI driver can not allocate a local segment. The reason is that the default Dolphin driver configuration for 2.4 kernels is to not use the `bigphysarea` allocation scheme, but instead a technique introduced with the 2.4 kernel to allocate the required memory on boot. This does, however, not work reliably, at least if not everything is configured correctly. The easiest solution is to enable the `bigphysarea`-allocation in the driver again. To do so, you need to change two source files of the IRM SCI driver:

1. In `src/IRM/drv/src/prolog.h`: make sure that

   ```
   #define BIGPHYS_DISABLE 1
   ```

   is not active (commented).

2. For driver distributions of version < 1.15 only:
   In `src/IRM/drv/src/LINUX/os/memalloc.c`: make sure that also in this file

   ```
   #define BIGPHYS_DISABLE 1
   ```

   is not active (commented).

and recompile the driver.

Q11: When I start an application across multiple nodes, I get one or more message(s) like

```
[2] SMI_ERROR: SCI session terminated -remote process crashed?
```

and the application aborts. But my SCI setup seems to be working fine, `scidiag` reports no problems.

A11: We have experienced such a problem with some Linux 2.2 systems. After applying the modifications to the kernel as described in the SMI Manual, chapter 2.3.1 "Preparing the System", the problem did no longer show up (reported by Hirokazu Kobayashi `koba@criepi.denken.or.jp`).

Q12: When I run an application across multiple nodes, I get messages like

```
SCIOpen i 1 dev /dev/SISCI/1 err 3
```

during the startup, or

```
SISCI : Error doing ioctl_mmap: error is 0x40000904
```

but the execution seems to terminate normally.

A12: These messages are just debug output of the SISCI API and may disappear as Dolphin Inc. releases a newer versions of the API. They do not influence the correct execution of the SCI-MPICH application.

Q13: My application expects input from `stdin` (interactive input from the console), but I can not supply this input to it, and the application aborts or uses some random value.

A13: There are two possible solutions:

1. Use the `mpirun -console` option for the affected processed to have process 0 open up its own xterm window. Usually, only process 0 reads from `stdin`, thus `-console 0` should do it. If this does not work, you may try `-console all`.
2. Supply the input via a file which you specify with the `mpirun -stdin` option.

Q14: When I run an application across multiple nodes, I get messages like

```
perror: No such device or address
```

Sometimes, the execution goes on normally, but sometimes, the application terminates with a message like

```
[4] Remote rendez-vous memory can not be imported
```

A14: The perror-message is output from the operating system which occurs if SCI resources are exhausted. If the execution goes on normally, SCI-MPICH was able to get sufficient resources by freeing some SCI memory which is currently not needed. However, if this is not possible, the application needs to be aborted because a message can not be delivered. To avoid this problem, you may try the following measures:

- Restart the drivers (or even reboot the cluster) to regain potentially "lost" SCI resources
- If this does not help, you need to reduce the size of the SCI memory buffers that each process of an SCI-MPICH application allocates using custom device configuration options (see chapter 3.5.1).

- To make sure right during the startup (in `MPI_Init()`) that the application has enough SCI resources even for the worst case communication scenario during its execution, use the device configuration option `DEMAND_CONNECT` and set it to 0 (zero). This will enforce the connection and initialisation of all shared memory resources right at the startup. If this succeeds, it is guaranteed that there will be no resource shortage during the execution of the application.
- If you have just not enough SCI shared memory for you purpose, you'll need to increase the maximum shared memory size that the PCI-SCI adapters offer. Please refer to the Dolphin documentation to see how this can be achieved (it involves setting the `prefetch space memory size` on the PCI-SCI adapter using `sciconfig` and potentially increasing the amount of memory reserved for SCI mapping by the OS).

Q15: When I run an application across multiple nodes, I get messages like

```
[0] SMI ERROR: SCIMap...Segment() failed
[0] SMI ERROR: could not map segment
[0] SMI ERROR: Could not create undivided region
[0] SMI ERROR: Could not create shared region!
```

and the application terminates.

A15: You are running out of resources; shared address space in this case. You need to increase the maximum size of shared memory address space and probably also the maximum number of shared memory segments on you system. You need to be root to do these changes on every machine in the cluster. For Solaris, add the following lines (or similar values) to `/etc/system` and reboot:

```
* we need more memory for creating shared regions etc (SCI)
set shmsys:shminfo_shmmax=536870912
set shmsys:shminfo_shmseg=128
* more lomem-pages for the SCI driver
set lomempages=2048
```

For Linux, you need to edit some values in `/usr/src/linux/include/asm/shmparam.h` as described in the SMI manual. The kernel needs to be recompiled and restarted.

Note: On Linux, we observed problems with some C library releases and how they deal with shared memory. Updating the C library might help.

3.9.4 Problems Achieving Good Performance

Q1: The point-to-point bandwidth via SCI (between to remote processes) I get with a PingPong-benchmark is much lower than the raw SCI remote-write bandwidth. What is wrong with my setup?

A1: This effect usually occurs if SCI-MPICH chooses the wrong copy function for remote-writes via SCI. Two optimised assembly functions are available for systems which have write-combining for the remote SCI memory enabled or disabled. The selection of one of these methods is done on startup of each process of the application based on the available system information. If you start the application with the `-v` option, the selected copying method of process 0 is printed to `stdout`. On IA-32 platforms which have write-combining enabled[3], it should be WC32 for systems using Intel PentiumIII or lower grade CPUs and WC64 for AMD Athlon and Pentium4-class CPUs[4]. MMX should show up only for disabled write-combining. On Sparc and Alpha platforms, it will always print `memcpy` as these platforms do not offer this kind of performance tweak. The configuration parameter `MEMCPY_TYPE` allows to override the default setting.

3.10 Performance

To check if your system delivers the performance that it should deliver, we give some measurements for point-to-point communication (latency and bandwidth for ping-pong communication, measured with `examples/perftest/mpptest`).

[contents to be added (benchmark results)]

3.10.1 Compiler Optimisation

The choice of the compiler optimisation level and the enabling of debugging options has a significant influence esp.on the small-message latency. The chart in figure 3 shows several measurements of point-to-point latencies (performed with `mpptest`, roundtrip/2) for different configurations of SCI-MPICH (as specified via the parameters to the configure script).

Configure parameters for the "-xO 0, full debug, w/threads" curve:

```
--enable-debug --enable-devdebug --enable-devthreads
```

(This configuration allows debugging, tracing and statistical performance summaries).

Configure parameters for the "-xO 0, no debug, w/threads" curve:

```
--enable-devthreads
```

Configure parameters for the "-xO 4, no debug, w/threads" curve:

```
--with-optcc=-xO4 --enable-devthreads
```

[3]On Linux, it is required to enable support for MTRR (memory type range registers) in the kernel configuration. The correct setup of the MTRR for remote SCI memory is then performed by the SCI driver. This can be verified via 'cat /proc/mtrr'.

[4]The choice between WC32 and WC64 depends on the size of the cache-lines of the CPU, which is identical to the size of the write-combine buffers. Currently, 32- and 64-Byte lengths are known and supported.

Configure parameters for the "-xO 0, no debug, w/o threads" curve:
`--with-optcc=-xO4`
(This configuration does not allow full asynchronous progress / communication).

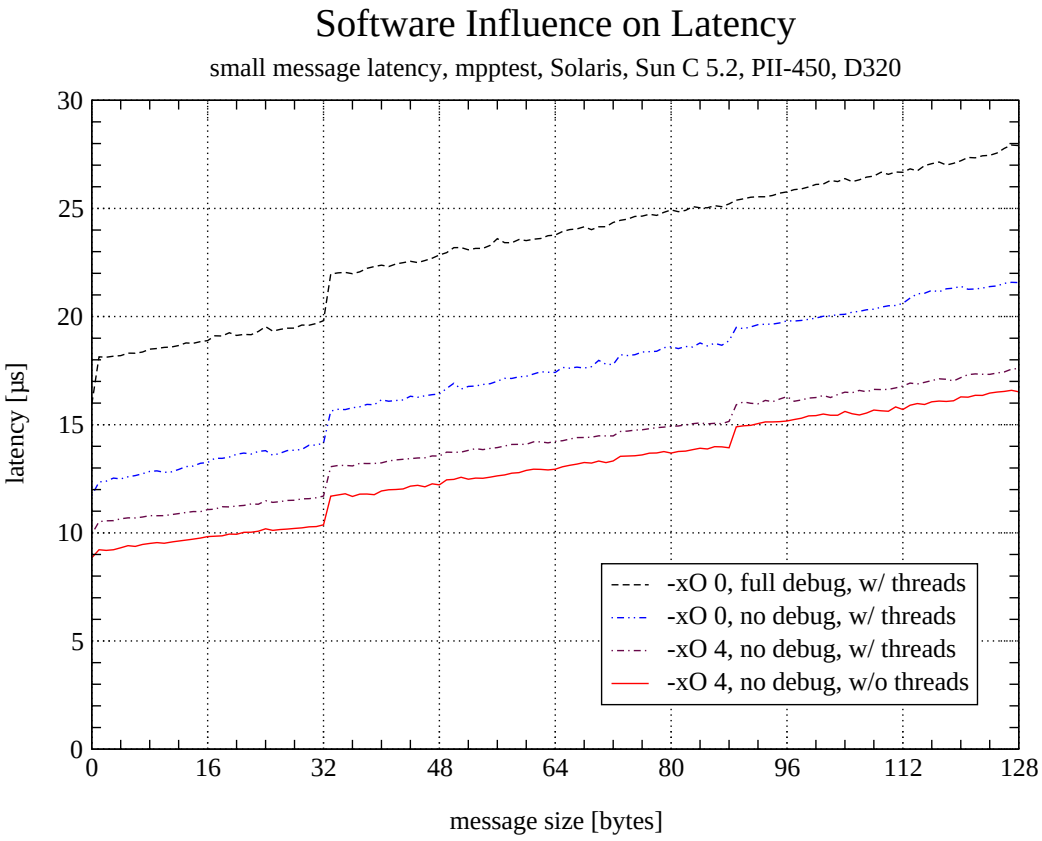


Figure 3.2: Small-message latency for differently configured SCI-MPICH variants

However, over-optimisation is not recommended, either, We have observed that optimising with `-O6` (using Gnu cc) will result in instable code, without improving the performance compared with a `-O3` optimisation. Recommended optimisation levels & flags are given in Table 3.3.

Concerning the Gnu Compiler (on IA-32 platforms), the results in figure 3.3 indicate that the version 3 of the compiler does only make a very little difference (if at all) when compared with the latest version 2 release.

3.10.2 Linking model

If you want to get the maximum performance from SCI-MPICH, especially concerning the message latency, you should choose to enforce static linking by linking

Compiler	Flags
Sun C 5.2 (Forte 6.1)	`-xO4`
Gnu gcc 2.95.2 (IA-32)	`-O3`
Microsoft Visual C	?
PGI pgcc 3.4-2	`-fast -g`[a]

[a]We have observed the strange behaviour that without the -g flag, communication deadlocks do occur. Therefore, `--enable-debug` should be specified during the configuration.

Table 3.3: Recommended optimisation flags

with the `-static` option. See figure 3.3 for the overhead in execution time for dynamic linking.

3.11 Internal Design

Although SCI-MPICH will run every MPI application with at least good performance, it is often possible to optimise the performance by considering the techniques SCI-MPICH uses internally. We try to give a broad overview on the internal design of SCI-MPICH to give the user enough understanding to optimise the MPI application or to configure the device for the best use with SCI-MPICH. For more in-depth information on SCI-MPICH, see the according references in chapter 3.15.

3.11.1 Verbose Startup

If you launch an SCI-MPICH application with the `-V` option of `mpirun`, you will get various information on the internal setup steps and configuration. Generally, the printed information does always only reflect the state of the process with the MPI rank 0 - the other processes may be set up differently concerning the number and size of the communication buffers. These resources are marked yellow in the figure below. Global configuration switches are identical on all processes and are marked red in figure 3.4.

3.11.2 Statistical Information on Device Activity

The communication device `ch_smi` of SCI-MPICH can gather various statistical information during the execution which may be very useful to see what the device has done how often, and how long such an operation took on average, minimal and maximal. Additionally, it is possible to get information on certain numerical states during the execution.

To be able to gather this information, SCI-MPICH must have been configured with the `--enable-devdebug` switch. If this has been done, you need to tell SCI-MPICH to actually do gather the statistical information by setting the `STATISTICS`

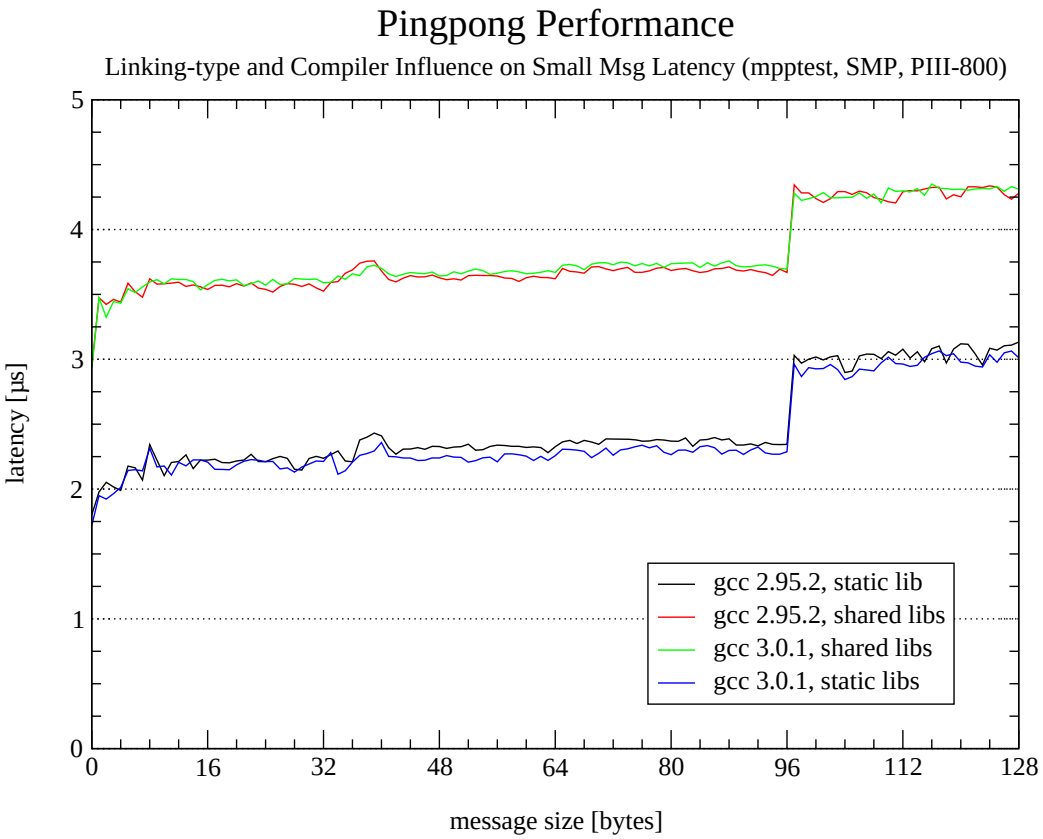


Figure 3.3: Comparison of different compiler versions and linking models. Measured on Linux x86, Dual Pentium-III 800, Serverworks ServerSet III-LE, SMP mode (node-internal)

option to '1' in the device configuration file. In this case, SCI-MPICH will print a table for each process with all information available after the application has completed successfully (which means it has called `MPI_Finalize()`) to the standard error output device.

These tables may contain any of the statistical values listed in Table 3.4. Three different types of statistical values to exist:

- *timer:* The occurrence, average, minimal and maximal duration, the accumulated time and the percentage of the total execution time are gathered for a timer value. All time values are specified in ìs except for the accumulated time which is specified in s.
- *counter:* Only the occurrence of the related event is counted for a counter value.
- *probe:* The occurrence and the average, minimal and maximal state of the

```
Fri Jul 6 16:23:01 CEST 2001:
running /home/joachim/linux_home/mp-mpich/
    examples/test/pt2pt/flood
on 2 LINUX smi processors
no device configuration file found, using
    defaults
```

Initial configuration information

```
using p3-01 as initialisation host
launching process 0 of 2 on p3-01
launching process 1 of 2 on p3-02
```

Process launch

Initialising SCI-MPICH:

```
SMI library version 2.6 (DEBUG)
 Lowlevel Init - ok.
 Internal Segments - ok.
 Barrier & Mutex Init - ok.
SMI Library initialized for 2 procs on 2
    nodes.
```

SMI Library Startup

```
Setting up protocols:
 SHORT: global - ok.
  RNDV: global - ok.
 EAGER: global - ok.
```

SCI-MPICH protocol initialisation

```
ADI version 2.10 - transport ch_smi 4.1,
    locks lock-free
 threads: disabled - stats: disabled - debug:
     disabled - csum: CRC32
Protocol memory configuration:
 SHORT: nbrbufs = 31, bufsize = 128 [96 net]
 EAGER: nbrbufs = 4, bufsize = 16384
 RNDV : blocksize = 24576, poolsize = 1048576
```

compile-time settings & memory setup

```
General configuration:
 Connect-on-demand: YES - Watchdog: 4 s -
    Sendself: YES
 Async Progress: NO - Polling: YES - DMA: NO
    - remote copy: MMX
 Adapter Mode: DEFAULT - Memcpy Sync: NONE -
    Collective: STANDARD
```

protocol parameters & global configuration

[Application Output]

```
SCI-MPICH finalized.
Application terminated.
```

shutdown information

Figure 3.4: Verbose startup information

state of a numerical value are counted for a probe value.

Only these values are listed in the tables for which the related events were triggered at least once. This means that all events related to values not listed in the table did not occur during the execution.

Value Label	Type	Description

Table 3.4: Available statistical values in SCI-MPICH

3.11.3 SCI resource requirements

SCI resources are limited in two ways:

- The size of the address window for remote memory: This parameter can be adjusted via the "page size" of the address translation on the SCI-PCI adapter board (using the `sciconfig` tool).
- The number of remote memory segments that can be established: Under Linux and Solaris, each SCI descriptor which is obtained via an `SCIOpen()` call, can be used for one local and one remote SCI memory segment. SCI-MPICH/SMI does in fact try to use the available SCI descriptors in an optimal way. However, the number of SCI memory segments that can be established is limited (maybe by some other parameters like kernel memory fragmentation or the number of open files allowed for a process).

SCI-MPICH tries to handle SCI resource shortage in a variety of ways:

- The number and size of preallocated EAGER buffers is reduced if not enough local SCI resources are available on startup.
- The size of the RENDEZ-VOUS memory pool is reduced if not enough local SCI resources are available on startup.

- Only the relevant part of remote memory for the EAGER protocol is imported.
- If 'on-demand connections' are enabled and a remote memory region can not be imported, other remote memory regions which are currently not in use are deallocated to free up some resources.

However, some communication patterns require full connectivity between all processes. The SCI resource requirements for each process of an application with P processes for this worst case scenario are calculated as follows:

Short protocol

$$
\begin{aligned}
Lmem_{short} &= (short_{bufsize} + 1) \cdot short_{nbrbufs} \cdot P \\
Rmem_{short} &= (short_{bufsize} + 1) \cdot short_{nbrbufs} \cdot (P - 1) \\
Lsgmt_{short} &= 1 \\
Rsgmt_{short} &= P - 1
\end{aligned}
$$

Note: The memory setup for the short protocol is fixed to the given device parameters, and the relevant memory segments are always at startup, independent from the device configuration.

Eager protocol

$$
\begin{aligned}
Lmem_{eager} &= eager_{bufsize} \cdot eager_{nbrbufs} \cdot P \\
Rmem_{eager} &= eager_{bufsize} \cdot eager_{nbrbufs} \cdot (P - 1) \quad \text{(managed dynamically)} \\
Lsgmt_{eager} &= 1 \\
Rsgmt_{eager} &= P - 1 \quad \text{(managed dynamically)}
\end{aligned}
$$

Rendez-vous protocol

$$
\begin{aligned}
Lmem_{rndv} &= rndv_{memorysize} \\
Rmem_{rndv} &= rndv_{memorysizeb} \cdot (P - 1) \quad \text{(managed dynamically)} \\
Lsgmt_{rndv} &= 1 \\
Rsgmt_{rndv} &= P - 1 \quad \text{(managed dynamically)}
\end{aligned}
$$

Total

$$
\begin{aligned}
Lmem_{total} &= ((short_{bufsize} + 1) \cdot short_{nbrbufs} + eager_{bufsize} \cdot eager_{nbrbufs}) \cdot P \\
&\quad + rndv_{memorysize} \\
Rmem_{total} &= ((short_{bufsize} + 1) \cdot short_{nbrbufs} + eager_{bufsize} \cdot eager_{nbrbufs} \\
&\quad + rndv_{memorysizeb}) \cdot (P - 1) \\
Lsgmt_{total} &= 3 \\
Rsgmt_{total} &= 3 \cdot (P - 1)
\end{aligned}
$$

Notes:

- Size of memory segments are always a multiple of the systems virtual memory page size and are thus rounded up if necessary.
- In case of SMP usage (multiple processes of an SCI-MPICH application on one node), the processes on one node need to share the available SCI resources.
- Obviously, the resource bottleneck is the remote memory.
- If an application runs out of SCI resources, you should first reduce the size of the rendez-vous memory pool (parameter `rndv_memorysize`; halving it i.e. from 1MB to 512kB usually does not hurt performance too much). If this does not suffice, reduce the number of the EAGER buffers and their size (device parameter `eager_nbrbufs` and `eager_bufsize`).

3.11.4 Message buffer sizes

[content to be added]

3.11.5 Asynchronous Transfers

[content to be added]

3.11.6 Source structure

All code related to SCI communication (the `ch_smi` device) is contained in `mp-mpich/mpid/ch_smi`. `smidef.h` is the central include file contains numerous vital settings and pre-defined default values. You can edit it to set default values as you like, but be careful that you know what you are doing.

The device initialisation and shutdown is performed in `smiinit.c`. Initialisation consists of calling the initialisation functions of all protocols and setting the pointers in the channel device structure. `smipriv.c` contains low-level functions, mostly to send and recv control messages. The three protocols short, eager and rendez-vous are implemented in the files `smishort.c`, `smi*eager.c` and `smi*rndv.c`. Collective operations and communicator setup are controlled from `smicoll.c`.

The source code uses lots of macros, using a tool like Source Navigator helps a lot to explore it. For a deeper explanation of the structure, protocols and algorithms, please look at the papers and Ph.D.-thesis on SCI-MPICH.

3.12 Known Bugs, Limits & Caveats

General:

- The default rendezvous-protocol does not work asynchronously - this leads to the behaviour that an `MPI_I*send()` is not completed until the corresponding `MPI_Wait()` is issued (this is the case for most MPI implementations!). To enable asynchronous message processing in SCI-MPICH, configure with `--enable-async` and specify the `ASYNC_PROGRESS` device configuration option.

Issues for the current release:

- successfully tested with MPICH test suite, (all test in `mpich/examples/test`); more testing is on the way.
- deadlocks may occur for large number of posted receives for which no send has taken place when using the
  `RNDV_BLOCKING 1`
  setting. This is protocol inherent.
- the following protocol configuration should not yet be used:
  `ASYNC_PROGRESS 1`
  `USE_DMA 1`
  `RNDV_BLOCKING 1`
- SCI-MPCH can be used with the recent versions of Scali SSP (tested with 2.1 and 3.0). However, DMA and remote interrupts are not available with the Scali drivers. This means that the related device options (i.e. `ASYNC_PROGRESS`) have no effect on these systems.

3.13 Release History

Release	Date	Notes
6.0	Mar 04	• SMI library included in mp-mpich • Support for IA-64 and Opteron • many, many fixes & improvements for stability
2.2	Oct 01	• increased security through checksums for short messages • new short-protocol for better space-time efficiency • efficient handling of derived datatypes (avoiding intermediate copy operations) • improved collective operations • zero-copy transfers supported for rendez-vous protocol • better DMA / asynchronous message transfers • `MPI_Alloc_mem()` / `MPI_Free_mem()` implemented (MPI-2) • single-sided communications (MPI-2) • tested and optimised for Alpha platform
2.1	22.Dec 00	• asynchronous transfers are fully functional now • new eager protocol variant (dynamic eager) for more efficient memory utilisation • new rendez-vous protocol variant (blocking rendez-vous) for higher point-to-point bandwidth • global synchronisation of memcpy operations for more efficient PCI transfers (= higher performance) • improved configuration and build process • improved application startup (`mpirun` script is nearly MPI-2 conform now) • uses shared libraries per default; modularisation of the ADI-2 communication devices (channel devices) • uses SMI library 2.5 • many bug-fixes and increased stability; higher configurability • fully 64-bit safe, thread-safe `ch_smi` device • support for D330

Release	Date	Notes
2.0	22.May 00	• "segment connection on demand" for much faster startup • only mapping the relevant parts of remote segments • fully flexible, adaptive sizing of message buffers of each process • full MPI 1.2 compatibility, incl. `MPI_Cancel` • improved SCI memcpy function (pure assembly) and alignment • numerous bug fixes
1.99	17. Dec 99	Major rework, preparation release for 2.0 • lots of bugfixes • works with D310 (32 bit) and D321 (64 bit) • uses the massively improved version 2.1 of the SMI library • does not depend on fixed segments for its protocols • Solaris x86 approved • Linux stability still weak -> user feedback required
1.1	23. Aug 99	Bugfix & performance release - now reaches nearly 100% of peak bandwidth for large messages
1.0	26. Jul 99	First public beta release, only tested well on Solaris x86

3.14 Approved Platforms and Configurations

Generally, SCI-MPICH can be run on any x86 system with Solaris, Linux or NT operating systems, interconnected with PCI-SCI adapters which can be accessed via the SISCI API. SCI-MPICH can also be run on SMP systems without SCI.

We try to supply information on certain configurations which have been tested successfully with SCI-MPICH. This does not mean that the usage of SCI-MPICH is limited to these configurations - every SCI platform should work fine, we just can not test them all. If you want to add an entry to this ever growing list, feel free to submit the relevant information to the address given in chapter 1.2. We will happily add your entry to this table.

CPU	Chipset	SCI	Nodes	Topology	IRM	SISCI	OS
PPro-200 SMP	440FX	D310	6	ringlet, no SMP use	1.9.8.1	1.9.2	Solaris 7
PII-450 SMP	BX	D310 D320	6	ringlet or D515 switch incl. SMP use	1.9.9	1.9.9	Solaris 7 Linux 2.2
PII-450 SMP	BX	D320	4	ringlet incl. SMP use	1.0 2.44	1.0 2.44	Win 2000 Win NT
UltraSparc on - Ultra 60 - Enterprise 450	n/a	n/a	1	SMP with up to 4 CPUS	n/a	n/a	Solaris 7, 32 and 64 bit mode
PIII-800 SMP	Server-Works LE	D320 D330	2	ringlet	DISsp 1.2	DISsp 1.2	Linux 2.2
PIII-Xeon 550 SMP	NX	D320	6	ringlet, switch D515	1.9.9	1.9.9	Solaris 7
hpcLine (PII-450)	BX	D312	16	2D-Torus	SSP 2.1	SSP 2.1	Linux 2.2

Table 3.5: Approved SCI-MPICH platforms

3.15 References

[1] Joachim Worringen, Thomas Bemmerl: *MPICH for SCI-connected Clusters.* Proc. SCI Europe '99, pp. 3-11, Toulouse, France, September 1999

[2] Joachim Worringen: *SCI-MPICH - The Second Generation.* Proc. SCI Europe 2000, pp. 10-20, Munich, Germany, September 2000

[3] Joachim Worringen, Friedrich Seifert, Thomas Bemmerl: *Efficient Asynchronous Message Passing via SCI using Zero-Copy.* Proc. SCI Europe 2001, pp. 10-20, Dublin, Ireland, October 2001

[4] http://www.lfbs.rwth-aachen.de/users/joachim/SCI-MPICH

Chapter 4

NT-MPICH

This chapter of the documentation will give you a brief explanation of and its use. If you intend to use SCI on Windows NT you should also read chapter 3.

4.1 What is NT-MPICH?

NT-MPICH is basically a port of MPICH to the family of Microsoft Windows operating systems. It contains different communication devices for Windows and a set of tools that allow the usage of MPI in a Windows environment.

4.1.1 Differences between NT-MPICH and the original MPICH.NT

Why do we offer an MPICH version for Windows NT, if the original MPICH is also available for NT, and what are the differences and advantages of NT-MPICH?

The answer to the first question is quite simple: We started the development of NT-MPICH when there was no NT support in the original MPICH distribution. Windows NT is supported by ANL since MPICH version 1.2. NT-MPICH started with version 1.1.2 of MPICH.

The answer to the second question is not that simple, because it requires knowledge about the MPICH distribution and the components it comprises. One difference is that NT-MPICH is more complete than MPICH.NT. E.g. it contains a port of both the MPE logging and graphics library.

The main reason why we continue to distribute an MPICH version for Windows NT is the lack of performance in the original MPICH.NT distribution. As it turned out after the release of MPICH.NT, the communication devices of NT-MPICH perform better on both shared-memory and ethernet connected machines. This is the main reason why we did not abandon NT-MPICH. Furthermore we wanted to support communication via SCI on Windows NT machines, which made it necessary to provide an MPICH distribution for Windows NT anyway.

4.2 System Requirements

The NT-MPICH distribution has been developed under Windows NT 4.0 SP6 with Visual C++ 6.0 and Fortran Power Station 4.0 on x86 machines. As far as we could evaluate, NT-MPICH runs on machines with Windows NT 4.0 SP4 or later, any Windows 2000 version, Windows XP professional and .NET server.

The import libraries created are VC++ 5.0 compatible so that VC++ 5.0 users can use them. All static libraries, however, need the linker coming with VC++ 6.0. If you own a non-IA-32 based NT system we would be very interested if NT-MPICH compiles and runs on it.

NT-MPICH is implemented as a DLL (Dynamic Link Library). So, to use the binaries you need a compiler that is able either to link VC++ 5.0 import libs or a tool that can make use of a DLL in a different way. To link any of the static libraries coming with NT-MPICH, the linker from VC++ 6.0 or later is required. For compiling the sources, VC++ 6.0 is required. If you have trouble compiling the code because of missing header files and/or libraries, your compiler uses an outdated Platform SDK. In this case you should download the current Microsoft Platform SDK from the Microsoft web site[1]. Furthermore, the `MPI_Reduce()` implementation of NT-MPICH makes use of MMX and SSE instructions available on newer IA-32 compatible processors. To compile this code, the Visual C++ Processor Pack (available from Microsoft[2]) has to be installed on your computer.

Since the Fortran bindings are located in static libraries, your Fortran compiler must be able to use VC++ 6.0 compatible static libraries.

For starting a parallel MPI application based on NT-MPICH, you will need a tool to start programs on remote computers. Unfortunately, NT has no standard mechanism to do so. However, there are several solutions for this problem on the internet. So if you already have installed such kind of tool, you are ready to compute. If you don't, we offer a sophisticated tool for Windows-based clusters called `RexecShell`. This tool comes together with a special service that requires at least Windows NT 4 with service pack 4 or later. It has also been tested with Windows 2000, XP and .NET.

4.3 Installation

MP-MPICH (and thus NT-MPICH) comes as a compressed archive, either as a binary distribution or as complete source code without binaries. For the installation of the binary distribution, please continue with chapter 4.3.2. The installation of the source distribution is described in the following chapter.

Next to installing NT-MPICH, you may want to install the cluster manager if you need a facility to launch applications on remote nodes. See chapter 4.3.2 for a

[1] `http://www.microsoft.com/msdownload/platformsdk/setuplauncher.htm`
[2] `http://msdn.microsoft.com/vstudio/downloads/ppack`

description of this process.

4.3.1 Installing the Source Distribution

NT-MPICH is provided as a binary and a source distribution. The source distribution can be used to create the binary distribution with the Microsoft Visual C++ (Version 5 or 6) compile environment. To use other compilers or environments, the required Makefiles / project files need to be created if the Visual C++-files can not be used.

Additionally, you'll need the Microsoft Platform SDK and Visual C++ Processor Pack installed as described in chapter 4.2.

Unpacking the archive Just unpack the downloaded file to a location of your choice. This can be your home directory if you are planning to work on the sources, or a temporary directory if you only want to compile NT-MPICH once (with your compiler in your environment) and then just use the binaries. If you unpack the archive, a directory named mp-mpich will be created which contains the complete sources. To compile anything of the sources you have just unpacked, you'll need to define a environment variable named `MPI_ROOT` which contains the path where you have unpacked the archive (the directory which contains the source tree). Subsequently, we will refer to this directory as `MPI_ROOT`. After compiling the sources, the resulting libraries and binaries can be installed into the "final" destination directory. If you like, you can also compile and run the applications directly from the source tree.

Compiling the sources To compile the source tree, just open `MPI_ROOT\mpich.dsw`. This will bring up a workspace including 18 distinct projects, namely

- `ch_ntshmem`: NT-MPICH with `ch_ntshmem` device for communication solely via shared memory.
- `ch_smi`: NT-MPICH with the `ch_smi` device for SCI (Scalable Coherent Interface) connected clusters.
- `ch_wsock2`: NT-MPICH with `ch_wsock2` device. This device supports both SMPs and networks of workstations.
- `CPP`: this projects creates a library that contains the C++ bindings for NT-MPICH. (see section 4.6)
- `mpe`: this includes the logging facilities of MPE.
- `mpe_client`: the graphics part of MPE, automatically used by the project mpe.

- `mpe_server`: this will compile to a DLL containing the MPE server code. Used by `mpe_server_frontend` and the `RexecShell` plug-ins.
- `mpe_server_frontend`: a Windows program that uses `mpe_server` to create a standalone MPE server.
- `nt_fortran_bindings`: a static library containing the Fortran bindings for all parts of MPICH (MPI, MPI-IO, MPE). See below for details.
- `romio`: a static library, containing a port of ROMIO for MPI-IO on NTFS. This will be linked to the final MPICH DLLs automatically.
- `shmem_device`: a static library containing the `ch_ntshmem` device. This will be linked with the `ch_ntshmem` project to build the final `mpich_smp.dll`.
- `slog_api`: a library containing all functions needed to create SLOG files. This is needed by the mpe project, but can also be used independently.
- `smi_device`: contains the `ch_smi` device; needed by the `ch_smi` project.
- `wsock_2 device`: the combined TCP/IP & shared memory device the `ch_wsock2` project.
- `wrappergen`: A port of the `wrappergen` utility that can be used to create custom MPI wrapper libraries.
- `ampi, lmpi, tmpi`: These projects will create wrapper libraries for the MPI functions. See section 4.9 on how to use them.

The most relevant projects are `ch_ntshmem`, `ch_wsock2`, `ch_smi`, `CPP`, `nt_fortran_bindings`, and `mpe_server_frontend`. All other projects are used internally by one of the aforesaid and thus compiled automatically if needed. The projects starting with `ch_` will create a DLL that exports all MPI, MPI-IO and MPE functions. The only difference between the DLLs lies in the communication device they provide.

To compile the project `ch_smi`, you also need the SMI library. (see the LfBS web page[3]). The project assumes that an environment variable `SMI_DIR` contains the path to the installation directory of SMI. Remember to set this variable before you start the Developer Studio.

After compiling one of the main projects, you will find the resulting import library and DLL under `MPI_ROOT\lib\mpichXXX.lib`, where XXX is either `'_smp'` for `ch_ntshmem` or `'_smi'` for the `ch_smi` project. The `ch_wsock2` configuration creates a library just called `'mpich.[DLL|lib]'`.

The resulting libraries are binary compatible. This means that it is possible to change the communication device without recompiling or relinking your code. E.g. if you want to run your code, linked with mpich.lib (the wsock2 device) on your

[3] `http://www.lfbs.rwth-aachen.de/users/joachim/SMI`

SCI system, just rename 'mpich_smi.dll' to 'mpich.dll' and make sure the executable can find this DLL. Now your program uses the ch_smi device instead of the ch_wsock2 device.

The project nt_fortran_bindings is used to support several different Fortran compilers. Since different combinations of calling- and naming convention are used by different compilers, this project comes in several configurations. Each configuration supports a different Fortran compiler, denoted by the name of the configuration. Each configuration creates a static library in the MPI_ROOT\lib directory. These libraries are prefixed by mpichf_. See section 4.5 for a correlation between those libraries and the according compilers.

There are three ways to compile the required projects:

- Interactively use the MS Visual Studio to compile the projects step-by-step.
- Use the batch-compile option of MS Visual Studio to trigger the compilation of all required options at once.
- Use the batch file MPI_ROOT\build.bat from the command line. This batch file will compile the complete NT-MPICH for SMP and TCP/IP communication, including all available language bindings and the cluster manager. Make sure that the executable msdev.exe is in your path for build.bat to work.

Please note that the RexecShell (see chapter 4.7.3) can not be compiled with MS Visual Studio, but requires the Borland C++ Builder. We therefore provide this single executable file RexecShell.exe within the source distribution.

Installing NT-MPICH After having compiled all sources, resulting in a number of binaries and libraries, NT-MPICH should be installed into a different directory, i.e. to let other users access the files and to preserve a working version of NT-MPICH independent from the development in the source tree.

This installation is performed by calling the batch file MPI_ROOT\make_dist.bat which copies all required binaries, libraries, include files and examples to a specified directory. The usage is very simple:

```
make_dist <target_directory>
```

If you ever recompile NT-MPICH in the source tree, possibly generating new or modified libraries, you may always call this batch file again to re-install NT-MPICH in the same location.

If you recompiled the cluster manager service rclumad you can install the new version using rclumad-update.bat in the MPI_ROOT\bin directory.

4.3.2 Installing the Binary Distribution

We provide an executable nt-mpich_setup.exe to install the required libraries and executables for NT-MPICH. You can choose different installation types Complete Installation, Cluster Frontend, Cluster Node and User Defined.

Complete Installation Install all available components of NT-MPICH. Choose this installation type on your development computer. If you want to develop your own MPI-programs it is necessary to choose this installation type on at least one of your machines.

Cluster Frontend If you don't have a preferred method to access remote nodes, we offer tools called `mpiexec` and `RexecShell` that can be used to start remote processes (which form the MPI application). These tools work in conjunction with a special remote execution service named `rclumad` that has to be installed on all machines that are to take part in the computation. Both `RexecShell` and `mpiexec` use this service to start processes on remote nodes.

This installation type is included in the Complete Installation and is needed at least once. The cluster frontend tool `RexecShell` is installed and `mpiexec` is also available.

Cluster Node As explained above `RexecShell` uses a system service. When you choose this installation type solely the cluster manager service `rclumad` is installed. This service has to be installed on each computer you are going to use to run NT-MPICH applications on via the supplied remote execution tools (`mpiexec` or `RexecShell`).

If your nodes are not part of a domain but of a workgroup you have to set the following registry key:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa]
"forceguest"=dword:00000000
```

User Defined If you want to use NT-MPICH without the provided remote execution tools you can just install the development part of NT-MPICH. You can choose one or more of the following components:

- tools for remote execution (Cluster Frontend)
- enable this computer to be a cluster node (Cluster Node)
- example source code, headers and libraries
- header and library files for development
- tools for profiling, logging and visualization (Cluster Frontend)
- Web pages for MPI and MPE

Configuring Cluster Nodes The cluster manager service contains a Control Panel Applet Cluster Manager which is installed together with the service. It will appear in the control panel of each system which has the service installed and can be used to set a few options for the Cluster Manager:

- set the minimal number of threads for servicing requests: the default value of 5 should usually do fine.

- activate the logging feature and specify the logfile location: if the service does not behave as you think it should, the log file may give hints on possible problems.

- start the service: if the service is not yet running, it can be started from here. Of course, the service can be configured to be constantly running etc. via the standard means of Windows 2000.

The logfile can be helpful if you encounter problems with `rclumad`. In case of problems, please send us this logfile together with a description of your problem (see chapter 1.2).

Uninstalling NT-MPICH NT-MPICH is listed at the table of installed software at the control panel. You can uninstall it from here.

4.4 Compiling MPI Programs written in C

Generally, two things are needed to create an executable from your MPI source code: the MPI include file mpi.h, and the library that contains the implementation of the MPI functions. This library is called `mpich.dll` for NT-MPICH.

4.4.1 Compiling the Examples and Running the Tests

NT-MPICH supplies a wide range of examples, located in the `MPI_ROOT\examples` directory. The subdirectories contain different project (.dsp) and workspace files (.dsw) which you can use to compile the examples. To compile the examples you'll need to define `MPI_ROOT` as described in chapter 4.3.1.

Additionally, we have generated a number of VBscript programs named `runtests.vbs` which run a whole test suite. For more information, please refer to the file `MPI_ROOT\examples\test\Readme_NT.txt`.

4.4.2 Create your own Applications

Creating a program that uses NT-MPICH is simple. Create a project that contains the code for your program. Make sure that the compiler can find the include file `mpi.h`, which is located in the `MPI_ROOT\include` directory. You also tell the linker to search the directory `MPI_ROOT\lib` for additional libraries. Finally link your program with one of the `mpichXXX.lib` files.

4.5 Compiling MPI Programs written in Fortran

NT-MPICH is also usable with Fortran. Unfortunately, things get complicated when Fortran code is involved. Due to the fact that there is no standard how Fortran compilers implement function calls internally, nearly each Fortran compiler uses a different approach. To support a large variety of compilers, the MPI Fortran bindings are not located in the DLL but in a static library. Because of the differences described above we offer several different versions of this library. You have to select one of them that is appropriate for your compiler and link your program with both this library and the `mpichXXX.lib`. The different libraries are:

- `mpichf_apf.lib`: Bindings for the Absoft FortranPro compiler.
- `mpichf_gnu.lib`: To be used with GNU g77 compiler.
- `mpichf_int.lib`: For the Intel Fortran compiler.
- `mpichf_lf.lib`: To be used with the Lahey LF95 compiler (see also section 4.8.1).
- `mpichf_sal.lib`: To be used with the Salford FTN77/95 compiler.
- `mpichf_vf.lib`: Intended for Compaq Visual Fortran, the Portland Group compiler and Fortran PowerStation 4.0 (see also section 4.8.2)

Please note that we do not possess all of the compilers mentioned. So we couldn't test all of the libraries. They have been created based on third-party information. The only compilers we tested are G77, Intel Fortran, Lahey LF95, Salford FTN77 and MS PowerStation. **So please let us know if you managed to use NT-MPICH with any other compiler.**

If you are using NT-MPICH with Fortran programs, make sure that you also read chapter 4.8 which covers some related issues.

4.6 Compiling an MPI Program written in C++

NT-MPICH also comes with the MPI C++ bindings. To use them, you have to bind your program with either `CPP.lib` or `CPPd.lib`, which is the debugging version of the library. We currently do not support the C++ profiling interface (`PMPI::...`). There is one issue you should note when using the C++ interface together with error handlers that throw exceptions (e.g. `MPI::ERRORS_THROW_EXCEPTIONS`). Do not use the compiler switch `/GX` (or `/EHc`) when compiling sources that contain try-catch blocks, since these switches will cause the compiler to assume that extern C functions will never throw an exception. We were successful using the switch `/EHsc-`, that explicitly disables the `/EHc` switch.

4.7 Starting NT-MPICH Applications

NT-MPICH applications are made up a number of processes running on one or more nodes. The problem with starting up such an application is to launch the processes on the different nodes with the correct parameters. There are different ways to do this which are described in this chapter. There is one important setting which is necessary no matter how you launch your application: you have to make sure that all processes are able to find the NT-MPICH DLLs file. This can be done in three ways:

- On all nodes, set the system environment variable `PATH` to include the path to all the NT-MPICH DLLs. They can be located in a local or a shared directory (the latter option makes updates easier).
- On all nodes, copy the NT-MPICH DLLs from `MPI_ROOT/lib` to a system library directory (like `C:\WINNT\SYSTEM32`)
- Copy the NT-MPICH DLLs into the same directory where the executable is located.

4.7.1 Generic Startup

Since Windows NT/2000 comes without standard tools to start processes on remote nodes, we decided not to rely on a special service to start NT-MPICH programs. Instead you can use whichever method you like to start the processes of your distributed application. All you need to know are the commandline parameters expected by an NT-MPICH application; these are presented later on in this chapter.

4.7.2 Application Startup using "plug-ins"

The tools `RexecShell` and `mpiexec` described in this chapter make use of plug-ins to start MPI applications, which may use different communication devices. Each device in the NT-MPICH distribution expects a different set of commandline parameters for startup (see section 4.7.6 for details). Other applications may use environment variables for the same purpose. So to be most flexible, `RexecShell` and `mpiexec` don't know anything about the applications they start. In fact they rely on the support of independent DLLs that are responsible for creating the correct parameters and environment settings for different types of applications. These DLLs are called plug-ins. So in order to create the correct startup environment for your application, you have to select the appropriate plug-in.

4.7.3 Startup with RexecShell

`RexecShell` is a graphical frontend and makes it quite simple to start your distributed application and monitor the execution. To start an application, perform

the following steps:

1. Open the Configure dialog of `RexecShell`.

2. From the drop down list labelled "Active Plug-in", select the plug-in appropriate to the device you intend to use. E.g. for the `ch_wsock2` device (`mpich.dll`), you would select the `ch_wsock` plug-in.

3. Select the hosts (aka nodes) you want to use for this run. All available hosts are listed in the left list box. Nodes coloured black are available (which means that a connection to the cluster manager service could be established) and can be selected. Hosts which are visible in the network, but have no cluster manager service running are listed red and can not be selected for executing the application. You can select a host multiple times to have it execute the related number of processes of the application.

4. Specify the executable, the working directory in which it should execute and additional parameters, if needed. The path specified here will be transferred to UNC. Make sure that this path is valid and reachable from each hosts that you have specified to run an application process!

5. Configure the selected plug-in (see chapter 4.7.5).

6. Decide if you want to lock the hosts/nodes and if you want to have the user profile be loaded on the remote host.
   Locking the hosts is a very limited type of scheduling: it makes sure that no other user can execute processes via `RexecShell` or `mpiexec` as long as your processes are running. Loading the user profile on the remote hosts can take quite some time and slow down the application startup, but is sometimes necessary to prevent startup failure (depending on the default system configuration of the remote hosts).

7. Enter your credentials for the remote nodes on the account tab. The account information entered here is stored and transmitted securely using the Windows 2000 encryption mechanisms.

8. Close the configure dialog and click on the Start button, or select the accordant menu entry. For each process, a window for the console output of the process (stdout and stderr) will open up, and your (parallel) application should start up.

4.7.4 Startup with mpiexec

If you prefer a commandline tool to start your application, you can use `mpiexec`. It has been modelled after the description in section 4.1 of the MPI-2 standard. This

chapter first describes the generic options of `mpiexec` and then explains some important techniques required to make good use `mpiexec`'s advanced features. Generic Syntax. A generic call of `mpiexec` looks as follows:

```
mpiexec [options] [plug-in options] [--] program <parameters>
```

`mpiexec` assumes that after removing all options recognised by itself and the plug-in, the first remaining parameter is the name of the program to start. This implies that you can put the options in any position within the command line, as long as you make sure that after removing them the command line looks like

```
mpiexec program <parameters>.
```

To avoid problems if you want to pass parameters to the program that are similar to `mpiexec`'s (or the plug-in's), you can put a `--` (dash dash) in front of the executable name. This will instruct `mpiexec` to ignore anything behind `--` when looking for options. The available options for `mpiexec` are described in the next paragraph; the specific options for each plug-in are described in chapter 4.7.5. Available options. Valid options for `mpiexec` are:

`-?` Print help message.

`-account name` Load and use the account 'name' from registry (the account must have been stored using the `-store` option).

`-accounts` List all accounts stored in registry.

`-configfile file` Load `RexecShell` config file 'file'. This supersedes all other command line parameters.

`-debug` Print debug messages (implies `-loud`)

`-domain name` Use 'name' as domain for account.

`-gc` Graphical configure. Don't pass the commandline to the plug-in but use a dialog box instead.

`-help` See `-?`.

`-host n1[,n2[,...[,nk ]]]` Use nodes n1...nk. Supersedes `-machinefile`.

`-lock` Lock all nodes for exclusive use.

`-loud` Print informational messages during execution.

`-machinefile file` Use 'file' as machinefile (default: machines.txt).

`-n num` Start 'num' processes (default: 1).

`-password pass` Use 'pass' as password for account.

`-path path` Use 'path' to prefix executable name (default: `mpiexec`'s working directory).

`-plugin file` Use 'file.dll' as actual plug-in (default: `ch_wsock`).

`-plugins` List available plug-ins.

`-profile` Load user's profile on remote nodes.

`-store` Store the currently specified account in registry (to be used together with the `-account` option).

`-test` Do not start processes. Just print commands.

`-user name` Use 'name' as login name for account.

`-wdir dir` Set 'dir' as working directory for remote processes (default is `mpiexec`'s current working directory).

Specifying the hosts `mpiexec` supports several methods to select the hosts to use for the computation. The most common and well known way to do so is to use a so called machines file. This is a simple text file, containing a list of hosts to use. Per default, `mpiexec` looks for a file called `machines.txt` in the actual working directory, the home directory of the user, and in the installation directory of `mpiexec`. A different way to specify the hosts to use is the `-host` switch, that takes a comma separated list of hostnames as argument. If neither the `-host` switch is specified nor a machines file can be found, `mpiexec` tries to find usable hosts by querying the network. This is thought as fallback mechanism and might be quite time consuming. To speed up the query, `mpiexec` tries to read a list of excluded hosts from a file named `Excluded.rsh`, which should be located in the same directory as `mpiexec`. It expects a simple text file containing (NetBios) hostnames separated by `<CR>`. This file can be created by `RexecShell` (File|Edit exclude list), or manually, using a text editor.

Assigning processes to hosts It is not required that as many hosts are specified as processes are to be started. If less hosts than processes are specified, `mpiexec` automatically launches more than one process on the hosts specified. During this process it tries to achieve an equal processes/processor ratio on each host. That means that hosts with a higher number of processors will get more processes to execute. Of course it is legal to specify the same host multiple times. This can be used to enforce a given number of processes on each host.

Usage of plug-ins In order to support all different devices of the NT-MPICH distribution, `mpiexec` uses the same "plug-in" mechanism as `RexecShell` does. That means that the commandline and/or environment for the processes to start

is not created by `mpiexec` itself, but by a separate DLL that is loaded at runtime. Each plug-in recognises a different set of options. These options are also specified in the commandline of `mpiexec`, but will be parsed by the plug-in.

Path conversion While parsing the commandline, `mpiexec` tries to resolve all path and file names to their according UNC forms, in order to simplify their specification. E.g. assume that your current working directory is `K:\mpi`, where `K:` is mapped from `\\nt_server\global`, then the command `mpiexec -wdir .. myprog` is identical to

```
mpiexec -wdir \\nt_server\global \\nt_server\global\mpi\myprog
```

Handling of console input/output When a process is started under control of `mpiexec`, its standard output/error handles are redirected to the node `mpiexec` is running on. This allows `mpiexec` to display all console output created by the program. The standard input handles are also redirected, but only the first process started will be allowed to interact with the user. This means that everything you type in the console window `mpiexec` is running in, will be sent to the first process, which should be the one with MPI id 0. All other processes will not get any input from stdin. So if you develop an interactive application, remember to use process 0 to interact with the user and distribute the input using MPI functions.

4.7.5 Available plug-ins

The NT-MPICH distribution currently contains the following plug-ins:

ch_shmem: This plug-in works in conjunction with the `ch_ntshmem` (SMP only) device of NT-MPICH. It recognises the following command line parameters:

`-alog` Use the `ALOG` logging format. Sets the environment variable `MPE_LOG_FORMAT=ALOG`

`-clog` Use the `CLOG` logging format. Sets `MPE_LOG_FORMAT=CLOG`.

`-display host` Set the environment variable `DISPLAY=host`

`-mpe` Start the internal mpe server and set the DISPLAY variable accordingly.

`-num n` Start n processes.

`-slog` Use the `SLOG` logging format. (default). Sets `MPE_LOG_FORMAT=SLOG`.

There is one special issue with this plug-in. Since the `ch_ntshmem` device starts all processes itself, `mpiexec` has to start only one process. So instead of the `-n` switch of `mpiexec` use the `-num` option of the `ch_shmem` device. E.g. to start `mandel.exe` and the `ch_ntshmem` device with two processes, use

```
mpiexec -plugin ch_shmem -num 2 mandel.
```

Do not use

```
mpiexec -plugin ch_shmem -n 2 mandel!
```

ch_smi: This plug-in is to be used with the `ch_smi` device. Like the `ch_shmem` plug-in it supports the switches `-alog`, `-clog`, `-display`, `-mpe`, and `-slog`.

Additional parameters are:

`-devconf file` Use 'file' as configuration file for `ch_smi` (see also section 3.5).

`-port num` Use port 'num' during startup.

`-smidebug` Let the SMI library generate debug output – useful if you have problems which seem to be SCI related. The startup of the SMI library can be traced, and the error messages are more verbose and appear in the full context.

`-v` Be verbose during startup.

ch_wsock: This plug-in creates command lines for the `ch_wsock2` device and is the default in `mpiexec`. Like the two plug-ins above it supports `-alog`, `-clog`, `-display`, `-mpe`, and `-slog`. Additionally `-port num` can be used to overwrite the default startup port.

mpi_pro: Together with the `mpich_nt` plug-in (see below), this takes a special place in NT-MPICH, since it can be used to start programs using MPI/Pro from MPI Software Technology, Inc. So if you use not only NT-MPICH but also MPI/Pro or MPICH.NT you can avoid to install both, the cluster manager and the startup services for MPICH.NT and MPI/Pro, but use the NT-MPICH cluster manager to start all your applications. The parameters the `mpi_pro` plug-in supports are:

`-mpi_debug` Print debug messages.

`-mpi_port num` Use port 'num' during startup.

`-mpi_verbose` Print verbose messages.

mpich_nt: This plug-in can be used to start programs using the MPICH.NT implementation from ANL. It supports the following parameters:

`-polling` Use polling while waiting for messages.

`-port num` Use port 'num' during startup.

`-singlethreaded` Don't use a dedicated message thread.

`-verbose` Print messages during the run

4.7.6 Starting NT-MPICH programs manually

If you don't use `RexecShell` or `mpiexec` to start MPI programs (i.e. when you want to set up a parallel debugging session), you need to know the commandline parameters for the different devices:

ch_ntshmem: `<progname> -np <#of processes> [-consoles]`.
The switch `-consoles` makes `ch_ntshmem` create a new console for each process it creates. This can be useful for debugging purposes.

ch_wsock2: `<progname> [options] -- <your own parameters>`.
The `ch_wsock` device assumes that on each node you want to use, a process is started by an external mechanism (e.g. `RexecShell`). One of the processes coordinates the startup process. The following explanations refer to this process as master. The master always gets MPI id 0. The ids of the clients depend on the startup order of the clients and will be selected by the master. Valid options are:

`-n num` 'num' is the number of processes that take part in the computation. If this switch is given, the process assumes to be the master. (master only)

`-m node` 'node' is the name of the host, the master is running on. If this switch is present, the process assumes not to be the master. (clients only)

`-p port` Use the port number 'port' for communication between master and clients. (optional, master and client)

`-b address` Use IP address 'address' for communication between the processes. This is intended for multi homed hosts to specify a NIC to use.(optional, master and clients)

Example: Assume you want to use 3 processes, one on the host NT1 and two on NT2. One of the processes on NT2 is the master. Use the following commands:

On NT2:

```
myprog -n 3 -- param      (master, gets MPI id 0)
myprog -m NT2 -- param    (client, MPI id unspecified)
```

On NT1:

```
myprog -m NT2 -- param    (client, MPI id unspecified)
```

ch_smi: For a description of the start parameters for `ch_smi` refer to section 3.3.4 of this document.

4.8 More about NT-MPICH and Fortran

Using Fortran with libraries is a complicated thing. If you encounter difficulties regarding Fortran and NT-MPICH, please refer to the following sub-chapters.

4.8.1 Using the Lahey LF95 compiler

The linker shipping with the LF95 compiler behaves quite different from the linker issued with Microsoft's Visual C++. It expects symbols exported from a DLL to be decorated. Since the Microsoft linker undecorates the names, your program will report missing symbols in `mpich.dll` on startup. There are two possible solutions for this problem:

1. If you have the Microsoft linker (link.exe) installed, add the switch `-ml` MSVC to your linking options. E.g.

   ```
   lf95 fpi.obj -lib mpichf_lf.lib,mpich.lib -ml MSVC
   ```

   will link the object file with `mpich_lf.lib` and `mpich.lib`, using Microsoft's linker (if available). This is the recommended solution.

2. If you don't possess `link.exe` or you can't use `-ml MSVC`, you have to specify aliases for all MPI functions your program uses. To simplify this task, the `MPI_ROOT\lib` directory contains a response file (`lf_imports.txt`) that contains all the aliases. You have to add this to your linker options. So in contrast to the example above your commandline should now look as follows:

   ```
   lf95 fpi.obj -lib mpichf_lf.lib -implib mpich
        @MPI_ROOT\lib\lf_imports.txt
   ```

   Please note that `mpich.lib` is not given behind `-lib`, but you use `-implib mpich` instead.

4.8.2 Using the Compaq Visual Fortran compiler and compatibles

The Compaq Visual Fortran, MS Fortran PowerStation and Portland Group Fortran compilers handle character strings in a different way than all other compilers supported by NT-MPICH. In combination with the calling convention they use, we had to implement special bindings for all MPI functions that might take a character string as argument. While it is possible to use functions like `MPI_SEND` with character strings, you have to take care when you do so. At least the Microsoft and Compaq compilers show some strange behaviour. You can not call the same MPI function within the same subroutine when one call uses a character sting as argument and the other uses a different data type. In this case the compiler does not distinguish between the two calls and one will be linked to the wrong binding routine, causing a crash during execution. To avoid this problem you should place calls that use character strings as arguments to MPI functions into separate subroutines or, even better, into separate source files. The following example explains the issue:

Assume you have a function that broadcasts a string to all processes. You first broadcast the string length, and, after that, the string itself. The code could look as follows:

```
SUBROUTINE BCNAME(FNAME)
CHARACTER*200 FNAME
INTEGER IFLEN
INCLUDE 'mpif.h'
IFLEN = LEN(FNAME)
CALL MPI_BCAST(IFLEN,1, MPI_INTEGER,0, MPI_COMM_WORLD, IERR)
CALL MPI_BCAST(FNAME,IFLEN, MPI_CHARACTER,0,
               MPI_COMM_WORLD, IERR)
END
```

This code calls `MPI_BCAST` once with an integer argument and once with a character string. The compiler would create wrong code. To work around this problem, you should modify your program as follows:

Create a new subroutine:

```
SUBROUTINE SEND_LEN(IFLEN)
INTEGER IFLEN
INCLUDE 'mpif.h'
CALL MPI_BCAST(IFLEN,1, MPI_INTEGER,0, MPI_COMM_WORLD, IERR)
END
```

Replace the first call to `MPI_BCAST` with a call to the new routine.

```
SUBROUTINE BCNAME(FNAME)
CHARACTER*200 FNAME
INTEGER IFLEN
INCLUDE 'mpif.h'
IFLEN = LEN(FNAME)
C Replaced the call to MPI_BCAST with a
C call to the new subroutine
CALL SEND_LEN(IFLEN)
CALL MPI_BCAST(FNAME,IFLEN, MPI_CHARACTER,0,
               MPI_COMM_WORLD, IERRPLL)
END
```

To avoid warnings when compiling the code you should put either `SEND_LEN` or `BCNAME` into a separate source files. This also makes it impossible for the compiler to mix up the two calls.

4.8.3 Using non-supported Fortran Compilers

If your Fortran compiler is not found among the list of supported compilers in chapter 4.5, this means that either we could not get information about the behaviour of that compiler or did not know that this compiler exists. However, you can try to use one of the existing libraries. If that does not work because either you get linking errors or your program crashes after calling MPI functions, we will have to create a special library for you. In order to do so, please consult the documentation of your compiler and tell us the following:

- Which naming convention is used (how are subroutines named by the compiler)
- Which calling convention is used (caller or callee clears stack)
- How are common blocks exported and what is their name in the object file.
- Which is the representation of logicals (which value is `.true.`, which is `.false.`).
- How are character strings passed to subroutines.

If you can't find this information in the documentation please do the following: Create a small test program that contains at least one subroutine and a call to an MPI function. Please also use the symbol `MPI_BOTTOM` somewhere in the program (e.g. call `mpi_send(MPI_BOTTOM,...)`). Create an object file from this program. Do not link it! Then send an e-mail to `mp-mpich@lfbs.rwth-aachen.de`, containing the name of your compiler, the object file, the source of the program and the error messages your compiler/linker produces when you try to create an MPI program. We will then try to figure out how to create a working library for your compiler.

4.9 Using the Profiling Interface

In order to allow users of MPI to write wrappers for MPI routines, the MPI specification requires that every routine `MPI_xxx` must be callable by the alternative name `PMPI_xxx`. That makes it easy for users to provide their own version of MPI routines that e.g. create log files or debug output. In NT-MPICH this is achieved by exporting each function with two different names from the DLL. Thus programs written in C can directly access the `PMPI_xxx` routines in the DLL. Just link your program with your replacement `MPI_xxx` routines and everything should work as specified.

Unfortunately, for programs using the Fortran bindings things are more complicated. Since the Fortran bindings are placed in a static library, there is no (easy) way to create alias names for functions. To solve this problem, for each Fortran

library there is a matching profiling library that has to be linked in addition to the other libs required. The name of the profiling library starts with 'p'. So e.g. in order to use the profiling functions with VisualFortran you have to link `mpich.lib`, `mpichf_vf.lib` and `pmpichf_vf.lib`.

The NT-MPICH distribution contains three sample libraries, `ampi.lib`, `lmpi.lib` and `tmpi.lib`. These can be used to instrument your program. The first one will use the MPE graphics feature to show an animated graphics that visualises the communication pattern of the application. `lmpi.lib` will use the MPE logging features to create a logfile, describing the calls of MPI functions. The last library just prints a trace of the MPI calls. To use these libraries, just link your program with one of them. If your program doesn't seem to use the replacement calls, you have to change the order in which the libraries are linked. E.g. the VC linker only uses the replacement, if you put the tracing library in front of `mpichXXX.lib`. So your commandline should look like

```
link $(OBJS) -llmpi.lib -lmpich.lib <more parameters>.
```

The project `cpi.dsp` in `MPI_ROOT\examples\basic` shows how the libraries can be used. Users who don't use MSVC might encounter linking problems with these libraries. Since they use stdio functions (especially `fprintf()`) you have to link the program with Microsoft's C runtime library. Otherwise linking errors can occur.

4.10 Using MPE

NT-MPICH contains a complete MPE implementation, including MPE logging and MPE graphics. To compile a program using MPE no special efforts are required. All functions are contained in the `mpich.dll`.

4.10.1 MPE Graphics

If you intend to use MPE graphics, please read the following notes.

The original MPE relies on the X-Windows protocol to provide its graphical services. Since it is uncommon that Windows based computers support this protocol, we re-implemented the graphics code. NT-MPICH contains a program (`mpe_server_frontend.exe`) that is able to display MPE windows created by distributed applications. This server program has to run on the node that is to display the graphical output. As communication mechanism between your processes and the server, RPCs are used. Thus it is not required that one of your MPI processes is running on the host displaying the window. The MPE server is completely independent from the MPI communication.

Please note that the MP-MPICH plug-ins of `RexecShell` are able to display MPE graphics. So if you use `RexecShell` or `mpiexec` there is no need to start the external MPE server. Just enable the "Use internal server" (`-mpe` switch for `mpiexec`) feature of the plug-in.

We tried to preserve the semantics of all MPE graphics functions in our implementation. Unfortunately, there is no formal specification of the MPE functions' semantics. We only had the man pages and of course the original implementation as reference. We tried to match the original semantics as far as possible, but nevertheless it is possible that our implementation behaves differently. If you notice any differences that break your code, please let us know.

There are also functions we changed intentionally. These are:

- `MPE_Capture_file`: The original MPE uses the tool `xwd` to create a snapshot of the MPE window. Obviously this is impossible under NT. So on NT, `MPE_Capture_file()` creates bitmap (.BMP) files. We also augmented the functionality of the function. Originally, the snapshot of the window is written to disk by the process that called `MPE_Capture_file()`. This makes it necessary to transfer the bitmap from the server to the client, which (depending on the image size) can be very time and network bandwidth consuming. To work around this, NT-MPICH allows to instruct the MPE server to write the image to disk. To do so, just prefix the `filename` parameter with the string `"server:"`. E.g. the call

  ```
  MPE_Capture_file(handle,"server:c:\\temp\\mpe",5);
  ```

  would cause the MPE server to write bitmap files named `C:\temp\mpeXX.bmp`, while

  ```
  MPE_Capture_file(handle,"c:\\temp\\mpe",5);
  ```

  would cause the client to create the files on its hard disk. Especially if you use a network drive to write the images to, it is more efficient to make the server write the files.

- `MPE_Add_RGB_color` and `MPE_Create_color_array`: The original MPE implementation only supports up to 256 different colours. We removed this limitation if your MPE server's display supports more than 256 colours. Thus, if you use a True- or HighColor display, you are allowed to create as many colours as you like, only limited by your main memory.

4.11 Jumpshot

The MPICH distribution comes with two JAVA programs used to display MPE log files. Jumpshot is used to visualise `CLOG` files, while Jumpshot-3 can be used for files in the new `SLOG` format. Both work on Windows NT (tested with JDK 1.1.8 and 1.2).

4.11.1 Running Jumpshot or Jumpshot-3

We created batch files to start the programs. They are located in the bin subdirectory. To work as intended these batch files rely on an environment variable `JAVA_HOME` that should contain the path to the root directory of the JDK (or the runtime environment), such that the JVM is located under `%JAVA_HOME%\bin\java`. Jumpshot does not work together with the Microsoft Virtual machine.

4.11.2 Compiling Jumpshot or Jumpshot-3

In order to simplify the build process on Windows we created a set of batch files that replace the UNIX makefiles. For Jumpshot the file `MPI_ROOT\jumpshot\src\code\make.bat` is used. It also assumes that the variable `JAVA_HOME` is set as described above. Due to strange dependencies between the source files of Jumpshot you might get some errors while compiling the first two files. This is expected and no reason to worry.

To build Jumpshot-3 you can use the file `make.bat` in `MPI_ROOT\jumpshot-3\src`. Both files will create a set of JAR files that can be used together with the batch files described in the previous paragraph.

Please note that the `CLOG` format changed from MPICH version 1.1.2 to version 1.2. This means that you cannot use the Jumpshot from 1.1.2 together with `CLOG` files created by version 1.2 and vice versa.

4.12 Frequently asked questions

In this section we will try to give answers to some frequently asked questions. Please help us to improve this part of the documentation by giving us feedback, bug reports and problem solutions. We also like success stories. To contact us, please send an e-mail to `mp-mpich@lfbs.rwth-aachen.de`.

Q1: Which of the DLLs should I use?

A1: As a rule of thumb you should use `mpich.dll` (the `ch_wsock2` device). This is the most sophisticated and provides both SMP shared memory and TCP/IP network support. You should use `mpich_smp.dll` only if you don't have the TCP/IP protocol installed on your machine (which is very uncommon). Of course you should use `mpich_smi.dll` if you have an SCI connected cluster.

Q2: Can I run my programs in heterogeneous environment?

A2: Currently NT-MPICH does not support heterogeneity. You should only use machines with processors of the same vendor (e.g. Intel x86).

Q3: Is it possible to mix Unix and NT workstations in the same cluster?

A3: Theoretically, yes – but currently, only for communication via SCI. All other NT-MPICH communication devices are "custom made" for the Win32 platform. Currently there is no compatible Unix device for socket communication via TCP/IP, but we are planning to create a such a compatible device (any volunteers?). But even if you use compatible devices, starting the applications in the heterogeneous environment is still a delicate topic.

Q4: I can't link my Fortran program with NT-MPICH. What's wrong?
A4: Please read section 4.5 and 4.8 of this document.

Q5: How can I display the MPE graphics on a Unix workstation?
A5: You can't. The MPE graphics implementation provided by NT-MPICH uses a dedicated server to display its windows. This server is a Win32 DLL that uses DCE RPCs for communication with its clients.

Q6: I use `RexecShell` to start my program. Unfortunately all processes report a rank of 0. How can I solve this?
A6: You most likely didn't select an appropriate plug-in. Please make sure that you selected the plug-in matching the communication device your program uses. If you don't know this, the commandline switch `-mpiversion` will cause the program to print an identification string.

Q7: I want to use `RexecShell`, but it takes a long time to start the program because there are many hosts in my domain. What can I do?
A7: There are two possibilities to exclude hosts from `RexecShell`.

- Add hosts not to be used to the file Excluded.rsh by using File->Edit Exclude List...
- Use a file named `machines.txt` (see `mpiexec` chapter 4.7.4). This file contains all hosts to use. This file has to be located in the directory specified by the environment-variable USERPROFILE or in the same directory as `RexecShell.exe`. If this file exists, `RexecShell` will ask on whether to use it (and do not search the network) or to search for hosts in the domain.

Q8: How can I use remote execution with a host which is not in my domain?
A8: You can call `mpiexec` with the parameters domain and host:

```
mpiexec -domain <domain1>
        -host <host1,host2,domain2/host1,...>
```

or add `<domain2/host1>` to `machines.txt`. With `RexecShell`, you can include hosts with File->Edit Include List...

Q9: I want to use `RexecShell` but all nodes are in red colour, I even can't run processes on the local node.

A9: This behaviour mostly occurs when the nodes are not part of a domain. You can start the MPI-processes manually as mentioned in chapter 4.7.6. Or see the next answer.

Q10: Can I use NT-MPICH without having a domain?

A10: If your nodes are not part of a domain but of a workgroup you have to set the following registry key:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa]
"forceguest"=dword:00000000
```

Q11: I can't run any remote processes. If I want to launch a remote process, I get a message like Creation of Process failed. Das System kann die eigene Datei nicht finden.

A11: Per default, the executable of the application needs to be located under the same path on all nodes which were chosen to execute the application. This means that it must be copied to the same path on a local drive of each node, or must be located on a shared drive which is accessible by all nodes under the same path. Alternatively, `RexecShell` can be used to configure each host separately with respect to executable, path, account etc. To do this, open the context menu on a name in the 'Selected hosts' list to invoke the 'host configuration' dialog, and change the default configuration as required (see figure 4.1).

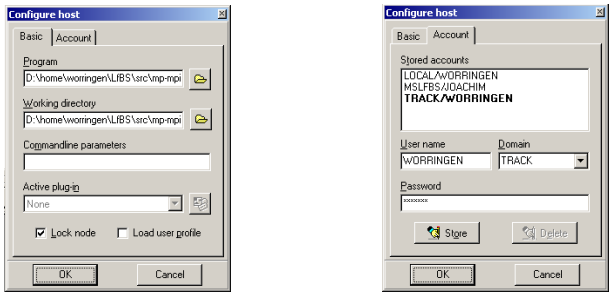


Figure 4.1: Host configuration dialog: Basic and Account configuration data

Q12: I can't run any remote processes. They seem to start but terminate immediately without any messages or error codes.

A12: This happens when the process can't find all required DLLs. Most likely this is the `mpichXXX.dll`. Please note that per default the user's environment is not loaded by `mpiexec` and `RexecShell`. This is done to speed up the process launch – loading the environment can take a lot of time. This means that the user part of the `PATH` variable is not set. Possible solutions:
- Add the location of the NT-MPICH DLLs to the system `PATH` variable (on each node!).

- Put a copy of the DLL into the windows system directory (on each node!).
- Put a copy of the DLL into the same directory of the program you want to start.
- Instruct `mpiexec` / `RexecShell` to load the user environment (see chapter 4.7).

Q13: Does NT-MPICH support Windows 2000?
A13: All versions of Windows 2000 are supported.

Q14: Does NT-MPICH support Windows XP?
A14: NT-MPICH works with Windows XP Professional.

Q15: Does NT-MPICH support the Microsoft .NET platform?
A15: Users from Microsoft have reported that NT-MPCH, including the remote execution service, runs fine on .NET severs. However, we could not verify this ourselves.

Q16: Does NT-MPICH support Windows 9x?
A16: Win 9x is not a supported platform for NT-MPICH. We do not try to be compatible with it. From our experience we can say that the complete shared-memory code does not work with Win9x. This means that you cannot start more than one process per node. Furthermore the startup tools `rclumad`, `mpiexec` and `RexecShell` do not run under Win 9x. Some users, however, report that they were able to run multiple processes with a network connection.

Q17: I want to use MFC (or any other GUI) within my MPI program. How can I do this?
A17: With MPI, your processes do not necessarily execute on the same machine you are sitting in front of. However, a Windows application can present its GUI only on the local machine. Therefore, we generally do not recommend to use MFC (or any other GUI) within an MPI application. Instead, you should split up your application into a front-end (using MFC or any other GUI) and a back-end (doing the MPI-based processing on the data specified by the front-end). If you need data visualisation from within the back-end, you should use MPE graphics.
Moreover, MPI was designed to make parallel applications portable. Using a certain GUI code within an MPI application will render the application non-portable, contradicting the purpose of MPI. Nevertheless you can look at /examles/basic/PiMfc which is an example for Pi calculation with MFC GUI.

Q18: No host in my network can be chosen for remote execution. Errors like 'Cannot connect to server' or 'The RPC server is unavailable (1722)' appear. What's wrong?

A18: It is required that the Cluster Manager Service is running on each remote host to use the remote execution tools. If you use Windows XP with SP2 problems witz the internal Windows firewall can occur. Use the newest remote execution tools and adapt your firewall settings.

Q19: How can I start the Cluster Manager Service?

A19: You can install it with the NT-MPICH installer, choose "Enable this computer to be a cluster node". You need administrator rights to install the service. Cluster Manager Service is then restarted automatically on system start.

Chapter 5

MetaMPICH

5.1 Introduction

MetaMPICH is an extension to MP-MPICH which makes it possible to connect groups of MPI processes running on different hosts and architectures in a transparent manner to run a single MPI application. The difference to existing solutions (like using the P4 communication layer) is the use of special *router processes* which manage the communication between the different hosts (see chapter 5.3.1 for the concept and chapter 5.6.4 for information on the internal design), and the utilisation of multiple devices to break the boundaries of dedicated fast cluster interconnects. This design is well suited for massively parallel architectures where a large number of pure MPI processes communicate only via the internal high-speed network run next to a few router processes located on nodes with facilities for external communication as well as for smaller cluster systems, where all nodes have access to a secondary external net (in most cases ethernet). On the one hand, some MPP architectures need a design like this because only a few designated nodes can communicate externally. On the other hand, this design is also of advantage by easing the communication load for the MPI processes as they don't have to bother with the external communication.

To understand the following notes on how MetaMPICH manages the communication between MPI processes, a few terms have to be understood:

- MetaMPICH groups the processors on the separate systems into **meta hosts**, which are virtual hosts consisting of single systems or compute nodes which have a shared file system, user accounts and most probable a high performance network interconnect which allows high bandwidth communication with low latency between them. At least one of the nodes (or an additional node which runs no MPI processes) has to be a **frontend node** which can be accessed via `ssh` or `rsh` for process creation. The frontend nodes must be able to run MPI processes on the meta host nodes without interactive password input.

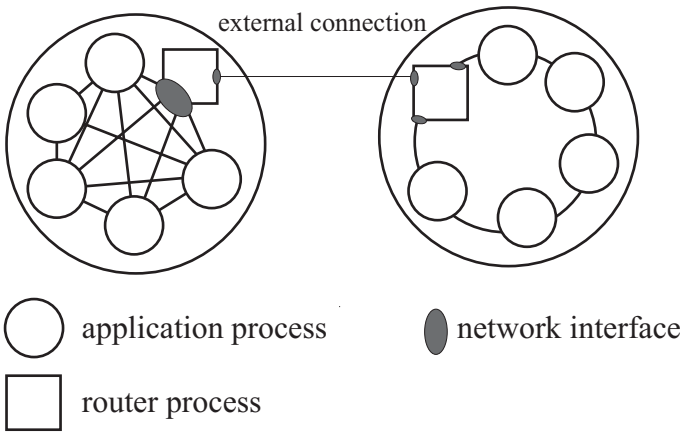


Figure 5.1: Meta system with router connection

- Besides the fast local **primary network** each meta host has network interfaces for one or more **secondary or external networks**. Only the minimal communication facilities are necessary, i.e. for each pair of meta hosts (X, Y) with $X = \{x1, x2, ...x_n\}\, Y = \{y1, y2, ...y_m\}$ there must exist a pair of nodes (x_i, y_j) which is able to establish a point-to-point connection. This way, MetaMPICH provides support for dedicated networks with non-routable private network addresses. Think of nodes coupled directly via multiple Gigabit Ethernet adapters connected back-to-back without a switch, and you get an idea of how flexible MetaMPICH can be configured.

- To overcome the limited access to external networks, MetaMPICH uses dedicated **router processes** in addition to the **application processes**. The former are transparent to the latter and therefore produce an overhead to the normal MPI processes of an application. Router processes are set up and run automatically on the accurate nodes by the `mpirun`-tools of MetaMPICH.

The following figures should help to point up the different communication architectures MetaMPICH is able to support. Figure 5.1 shows the classical router-to-router architecture with one single point-to-point connection. Internal the nodes are connected by a high performance cluster interconnect, for the external communication a TCP connection between two dedicated nodes is used.

A typical multi device architecture is shown in figure 5.2. No router processes are needed because all nodes can communicate directly using a secondary ethernet. The number of application processes increases by the number of saved router processes.

Even mixed architectures are possible. As you can see in figure 5.3, pairs of meta hosts can use either the secondary device or an external point-to-point router connection to communicate. Note that there can be *only one* secondary network in the whole meta system, so that if a meta host doesn't share this network, it has

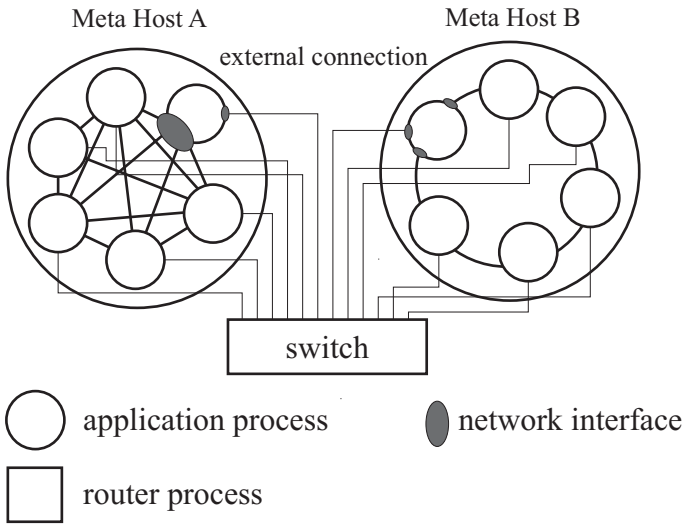


Figure 5.2: Meta system with multi device communication

to use router connections to all other meta host.

As one would guess from these simple examples, the configuration and set-up of meta systems is the central feature of MetaMPICH, because a complex communication architecture results in a complex configuration. This configuration is stored in a so called meta configuration file which contains everything MetaMPICH needs to run the processes via `mpirun` and connect the applications.

The following chapters show how to configure, compile and install MetaMPICH, and how to create the mentioned meta configuration files.

5.1.1 Feature list

- Operating Systems Linux, Solaris, Windows (with limitations, no `mpirun` support)
- Primary Networks: ch_smi (Scalable Coherent Interface and shared memory), ch_usock (TCP/IP), ch_shmem (shared memory)
- Secondary Networks: ch_usock
- External Networks (for router processes): TCP, ATM AAL5
- Manual configuration of asymmetric and symmetric router connections
- Automatic router configuration

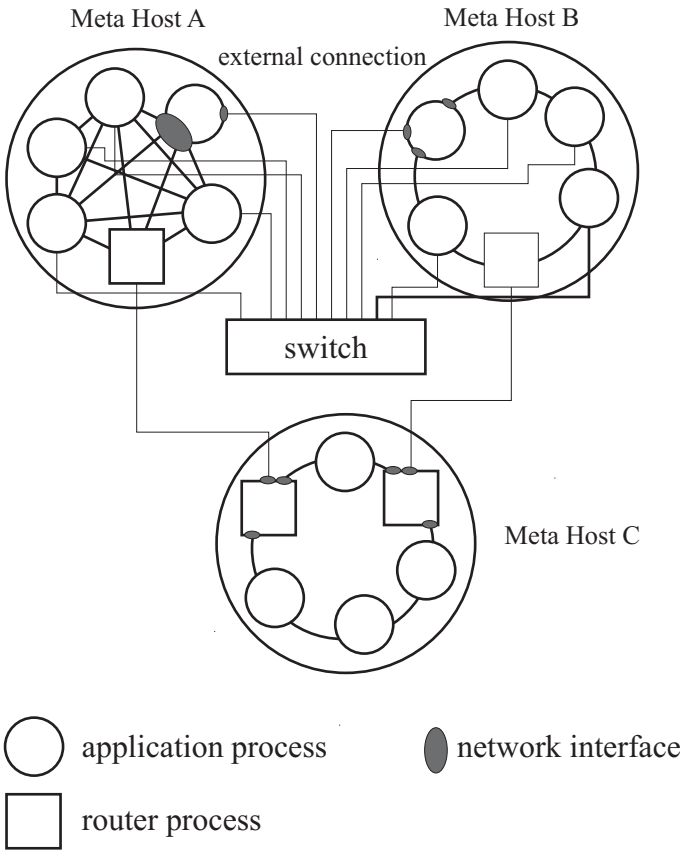


Figure 5.3: Meta system with router and multi device communication

5.2 Installation

The installation does not differ much from the normal installation of MPICH. Only the differences are noted in this chapter. Refer to the original MPICH documentation for further information.

5.2.1 Requirements in Hard- and Software

Due to its internal design, MetaMPICH runs on virtually any platform on which the original MPICH can be used. It has been tested on Solaris (Sparc and Intel) and Linux (Intel). However, the target system must fulfill the following prerequisites:

- MPICH must provide an ADI-2 device for internal communication on this system.
- IPC message queues must be available systemwide (system calls `msgget()`, `msgsnd()`, `msgrcv()`, `msgctl()`). Some MPP systems do not provide this feature, but work-arounds are possible and have been realised for the Cray T3E (see chapter 5.5.3)
- POSIX threads must be available.
- The router processes must have access to a TCP/IP connection towards the other hosts. A number of ports must be available (the base for these portnumbers can be configured in the source code).
- To use the `mpirun` script, the Gnu version of `awk` (`gawk`) must be found in the path.

5.2.2 Compiling

To generate the required MPI library, just proceed as described in the MPICH documentation. The only differences are the additional options for the configure pass:

`--enable-meta` This option is required to build the special MetaMPICH version of the library. Next to some changes in the Makefiles, it sets the name of the library to be generated to `libmetampi.a`

`--enable-hetero` Per default, the option `--enable-meta` does build the library for use in a homogeneous environment where the binary representations of the datatypes are identical on each participating machine. If you are planning to use MetaMPICH in a heterogeneous environment (i.e. mixing machines with big and little endian or different floating point formats), this additional option will generate a version of the library in which all messages between the hosts are transmitted XDR encoded. Message within one host will not

be sent in native encoding, which means that using XDR will slow down the intra-cluster communication.

`--enable-meta-atm` This enables the ATM support for MetaMPICH.

`--enable-router-threads` This activates the multi-threaded router. It has to be tested on every hardware platform which version is performing better.

After configuring, call `make` to build the whole MPICH distribution, or call `make mpilib` to build only the library, but no additional tools and tests. The library which is built is called `libmetampich.a` (or .so) and can be also used without using the Metacomputing extension.

Note: If you are using the Gnu C compiler `gcc`, you should configure MPICH with a special compiler command to avoid a strange behaviour of `gcc`:

```
./configure [other options] -cc="gcc -I."
```

Otherwise, `gcc` might not be able to find certain include files.

5.3 Configuration

The meta hosts in a MetaMPICH setup are linked solely via point-to-point connections. In the current release of MetaMPICH, the routers on the different meta hosts communicate via TCP/IP or ATM. Future releases might use other communication means, but this won't affect the general communication design of MetaMPICH because every communication method supplies at least point-to-point connections.

The building blocks of the inter-meta host communication in a MetaMPICH setup are routers, connections and sockets. This concept is presented in chapter 5.3.1. The derived syntax of the configuration files which describes such a setup is explained in chapter 5.3.2 followed by a detailed example in chapter 5.3.3. Some more examples for typical setups are given in chapter 5.3.5.

5.3.1 Principles of MetaMPICH inter-meta host communication

Router A Router is a process that manages the communication from the own meta host towards one other meta host. It utilises the MPI functionality for internal communication and other communication means (currently, TCP/IP and ATM are supported) for external communication. At least one pair of routers is required between each pair of meta hosts in a MetaMPICH configuration because one router communicates with exactly one meta host.

Connection Two routers on different meta hosts communicate via connections. Each router needs at least one connection, but may have multiple connections. Thus, it is possible to define 1:1, 1:N and N:N setups between two meta hosts (see figure 5.4) to adapt to given hardware setup.

Socket Sockets are the base unit in the MetaMPICH communication architecture. Each connection uses at least one socket. If a router has access to multiple, independent transmission channels towards another meta host, it can be configured to use them concurrently (each socket is handled by its own thread) and transmit messages faster by splitting and combining them.

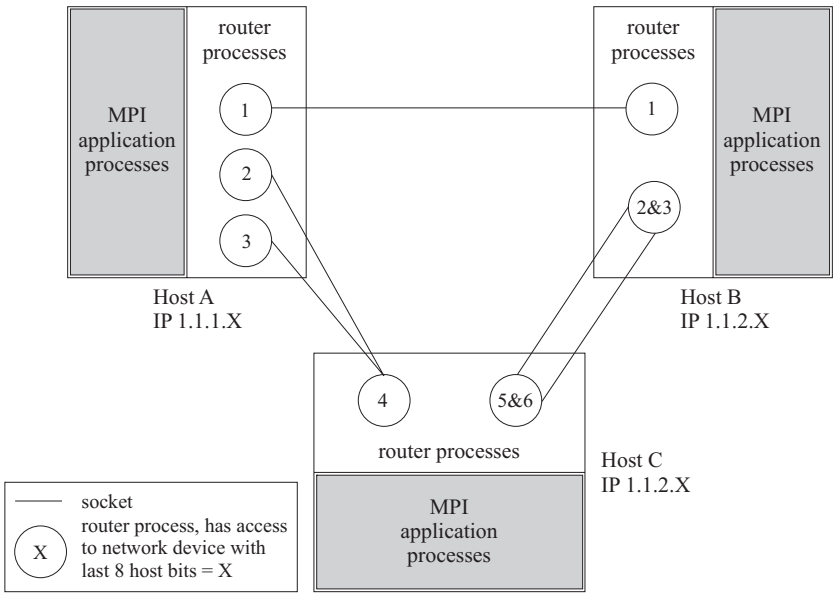


Figure 5.4: Advanced configuration: 3 hosts with N:1 and multi-socket connections

5.3.2 Syntax of the configuration file

To setup a MetaMPICH configuration, more information is required than can be provided by MPICH and the standard command line arguments. Thus it is necessary to use a configuration file which is passed to the MetaMPICH application at start-up (see chapter 5.4.2). This configuration file consists of three parts, which are separated by the keywords `OPTIONS` and `CONNECTIONS`.

Listing of the meta hosts

This section starts with a line of the following syntax with `nh` being the number of meta hosts in this setup:

`NUMHOSTS <nh>`

The following `nh` lines supply the names of each meta host and the number `nmpi` of MPI application processes that shall be run on this meta host. The name of the meta host can be chosen arbitrarily by the user and is used in the remaining lines of the configuration file to reference that meta host. The number nmpi does not include the router processes. This gives a syntax like:

`<meta-hostname> <nmpi> <mpi-root-path> <mpirun_args>`

`<nmpi>` is the number of application processes which should be run on the meta host. If you set this to 0, no user processes will be spawned, if you set it to `MAX`, the maximum number of processes will be started. To use `MAX`, you will have to specify the number of available processors (`<maxprocs>`)for each meta host node in the meta host definition, which is described in 5.3.2.

`mpi-root-path` is the root path for the mp-mpich installation on that specific meta host. This is useful if you connect heterogeneous systems which have different paths.

The `mpirun_args` are any meta host-specific options for the `mpirun` which is started on this meta host. The `mpirun` option `-np x` is always generated from the given number `nmpi`.

General options

This section starts with the Keyword OPTIONS. Now, some global parameters can be set at this point if the given default values need to be changed.

Option	Short description
HETERO	1 = use XDR
EXCHANGE_ORDER	0: little endian (default) 1: big endian
SPLITSIZE	Threshold for splitting messages over multiple connections
PORTBASE	Base for TCP ports. Needed to avoid conflicts between applications
NICDEF	assigns a network interface address to a name

Table 5.1: Options

`HETERO`

If your MetaMPICH-Cluster is heterogeneous concerning the binary representation of data (little endian mixed with big endian etc.), use this switch to let MetaMPICH use XDR-routines for data conversion. Of course, a heterogeneous

cluster won't work without this keyword which shows up by the routing processes being unable to synchronise (error message "wrong magic number" will be displayed). To successfully use this option, the Meta-MPICH library must have been compiled for heterogeneous use (option `--enable-hetero` for the configuration, see chapter 5.2.2).

(Default value: cluster is homogeneous, no data conversion)

`EXCHANGE_ORDER <byte_order>`

If you have a heterogeneous configuration, you may set up an exchange byte order here. This can be useful if one of the hosts is faster than the other, because in this case it can be forced to do the data conversion for the MetaMPICH router-router connections by setting this parameter to the order of the slower host. Note that this doesn't touch the XDR data conversion, but only the exchange of internal MetaMPICH data. Possible values for `byte_order:` are:

0: little endian (default)

1: big endian

`SPLITSIZE <split>`

This parameter is relevant for connections made of multiple sockets and defines the threshold for splitting messages. If a message to be transmitted is bigger than split bytes, it is sent via multiple sockets. The effective number of sockets n_{sock} used for transmitting a message of size msgsize is determined by the formula

$$n_{sock} = \text{msgsize} \;/\; \text{split} + \text{msgsize} \;\%\; \text{splitwith}$$

"/" being an integer division, "%" the modulo operation and the available number of sockets of the connection being an upper limit for n_{sock}. `SPLITSIZE` can be used to optimise the use of the available bandwidth. (Default value: 1500)

`PORTBASE <pb>`

The offsets used in section 3 of the configuration file are added to `PORTBASE` to form TCP/IP portnumbers. If the resulting portnumbers cannot be used, set this parameter to a suitable value.(Default value: 2500)

`ISEND_NUM <in>`

This parameter tells the routing process, how many unreceived `MPI_Isends` are to be expected. This is a problem because the router has to copy the data and stores it until the receiver does his work.

`NICDEF <nicname> <nicaddr>`

To make the subsections in the following connection-section more readable, you can assign a network interface address to an arbitrary identifier at this point. Currently TCP and ATM addresses are supported.

```
<nicaddr>:= TCP <ip-address>
          | ATM_PVC <atm-pvc>
          | ATM_SVC <atm-svc>
          | GETIP( <dns-hostname> )
```

ROUTERTIMEOUT <timeout>

In distributed environments it is nearly impossible to run the processes on the meta hosts synchronous. Therefore the routing processes wait <timeout> seconds for the other processes before they give up. Setting this value to 0 will let the routers wait forever, which will require to explicitly kill the processes if the startup fails.

Definition of Meta Computer Components

MetaMPICH supports the use of clusters of nodes as one so called meta host. These meta hosts have to be declared in the OPTIONS section. The syntax is:

```
<metahostdefinition> := METAHOST <metahostname> { TYPE <hosttype>;
       [<option>; ]
       [<option>; ... ]
       [DEVOPTS = <options>; ]
       TYPE = ch_smi | ch_usock | ch_shmem | ch_gm ;
       NODES= <nodelist>;
       [ROUTERS = <nodelist>;]
       EXTRAPROCS = <node> : <np> : <executable-path> : [<user>] : [<args>]
                    [, <node> : <np> : ..... ]
'}'
<nodelist> := <node>
       | <nodelist> , <node>

<node> := <nodename> [<maxprocs>] [< '(' <network-address>
                     [,<network-address]* ')'] [: <executable-path>]
       | <noderange> [<maxprocs>] [<network-address>]

<option> := 'ENVFILE'    '=' <path>
          | 'EXECPATH'   '=' <path>
          | 'EXECNAME'   '=' <filename>
          | 'CONFPATH'   '=' <path>
          | 'CONFNAME'   '=' <filename>
          | 'MPIROOT'    '=' <path>
          | 'ROUTEREXEC' '=' <path>
          | 'FRONTEND'   '=' <nodename>
          | 'USER'       '=' <username>
            | 'DEVOPTS'    '=' <literal>
<literal> := ''' <string> '''
```

The most important thing in this definition is the comma separated node list, which enumerates the cluster nodes which should be used in the application. The network interface addresses which will be used in the CONNECTIONS section have

to be declared after the nodes. Larger numbers of nodes can be enumerated by giving a name range like `p4-|01| - p4-|64|` using the delimiter `|` to mark the numeration area of the node names. It is also possible to exclude nodes or node ranges from the list by prepending a "`!`". It is important to understand that the definition is processed from left to right, so that an exclusion has to be defined after all other definitions concerning the nodes to be excluded to have an effect. The maximum number of processes can be specified with the node range as well as a start IP-address for the first node. The following nodes will get a subsequent IP-address. This feature is very helpful to define addresses of secondary network interfaces for the router processes, which are not in the DNS. Because nodes can be redefined in the list, it is possible to add differing properties of node range members later.

The optional `[: <executable>]` property allows to specify a different executable for a node. This is useful for special processes with different assignments in an application.

The keyword `ROUTERS` allows to separate router and node list and helps to make the configuration file more readable. In fact, it is just an alias for the `NODES` keyword and the router list is just appended to the node list internally.

The `<metahostname>` is an arbitrary name, but has to match a name in the first section.

Following is an example for such a definition:

```
METAHOST clusterB {
TYPE=ch_smi;
FRONTEND=p4-00;
EXECPATH=/home/martin/src/myprog ;
MPIROOT=/home/martin/mp-mpich_linux_smi ;
ENVFILE=/home/martin/metacfg/solaris_linux_smi_env ;
NODES= p4-|01| - p4-|64| 2 (192.168.2.1), p4-08,
         p4-02 5 (111.112.113.114),
   ! p4-08, ! p4-|32| - p4-|48|;
  }
```

It is possible to configure some important paths for each meta host to make it easy to use different instances of the MetaMPICH library on different platforms.

The `EXTRAPROCS` option makes it possible to run so called *e*xtra processes, which are run besides the application processes. This can be useful for special IO-processes or visualization tools. `EXTRAPROCS=` is followed by a comma separated list of process specifications, which have a similar syntax to procgroup file entries. For example

```
EXTRAPROCS=p4-03 : 1 : /home/martin/cpi2 : martin : '-verbose',
           p4-04 : 2 : /bin/ioserver : : '' ;
```

Option	Short Description
ENVFILE	This shell script is sourced before executing the application
EXECPATH	This is the working directory.
EXECNAME	The name of the executable (if not provided to the `mpirun` command).
CONFPATH	The path to this configuration file (if it differs on a remote metahost).
CONFNAME	The name of this configuration file (if it differs on a remote metahost).
MPIROOT	The root of the MPI-installation on the metahost.
TYPE	The type of the internal network. One of: `ch_usock`, `ch_shmem`, `ch_smi`
FRONTEND	Name of the cluster frontend, which must be accessible over the network via `ssh` or `rsh`. This option is used by `mpirun`.
USER	This user name is used to log in the meta host.
DEVOPTS	Command line options for the device-mpirun on the meta host.
EXTRAPROCS	This option allows to define extra processes on dedicated nodes which have to run in any case.
ROUTEREXEC	A dedicated default router process for the metahost is declared here, for example `$MPIR_HOME/bin/metarouter`.

Table 5.2: meta host options

would run one process of cpi2 on p4-03 and 2 processes of ioserver on p4-04. Note that these processes reduce the number of usable processors for the normal application processes.

Description of the host connectivity

This section is the most complicated, but also most important part of the configuration. After the keyword CONNECTIONS, it describes the setup of routers, connections and sockets for each pair of meta hosts from both sides. This leads to $nh * (nh - 1)$ subsections (one for each pair of meta hosts).

Each subsection starts with a line of the following syntax:

```
PAIR <from_host> <to_host> <nbr_routers> <router_executable>
```

`from_host` and `to_host` are names from the meta host list in section 1, defining the pair of meta hosts whose connectivity will be described now. `nbr_routers` is the number of router processes on meta host `from_host` handling messages towards meta host `to_host`. For each of these routers (one or more), a separate configuration is given in the `nbr_routers` lines following this line. Finally, a fully qualified filename for the executable of the router can be supplied. If instead a '-' (minus sign) is given, MetaMPICH knows that a separate router executable is not required on this platform and the router process will be spawned from the MPI application processes (see chapter 5.5 for more information on external or internal router process creation).

For the following lines, we define a simple syntax in figure 5.5 to help describing the setup of connections and sockets for each router.

```
<router_line>    ::= <src_address> '->' <dest_address>
<src_address>    ::= <address_list>
<dest_address>   ::= <dest_address> <dest_address> | <address_list>
<address_list>   ::= <address_list>,<address_list> | <address>
<address>        ::= IP | IP : OFFSET
```

Figure 5.5: Syntax of the router configuration entries

An IP is given in the usual form (i.e. 123.43.5.111); OFFSET is a single number (default value if omitted is 0). This offset, in fact, is added to the basic TCP/IP port number (see chapter 5.3.2, option `PORTBASE`) to distinguish multiple sockets between two hosts. Whenever more than one connection or socket is used between two hosts, they must use appropriate endpoints.

There are two modes for a router connection definition: an uni-directional definition is marked by the mapping operator `"->"`. This does not mean that the connection is uni-directional, but that this is only one direction of the definition. The other direction has to be defined in the appropriate section of the other meta

host. This is only needed for asymmetric router connections, where two or more routers on the one meta host are mapped to one on the other. If you use symmetric connection, the use of the bidirectional mapping operator "<->" is recommended to avoid errors.

Automatic Router Setup

The easiest way to set up router connections is to use the keyword AUTO_ROUTER after CONNECTIONS. Now MetaMPICH will try to set up a pair of routers for each pair of meta hosts. You will get nh-1 routers on each meta hosts. MetaMPICH will search for usable NICs in the meta host nodes. Because there are $nh - 1$ routers, $nh - 1$ NICs are needed on each meta host. First, the explicitly specified addresses of the nodes are used, and then, if they aren't enough, the node names are resolved to get more addresses. You can control this behaviour by either specifying AUTO_ROUTER NO_DNS, which will suppress the creation of addresses by name resolving, or AUTO_ROUTER ONLY_DNS, which will only use the node names and skip the defined addresses. If MetaMPICH cannot find enough network addresses, it will abort the application startup with an error message.

5.3.3 Example configuration and further explanations

To make things more clear, we discuss a sample configuration file for the setup given in figure 5.4. There are some caveats to take precautions when writing a configuration file.

We recognise that we have two possibilities to connect two host via more than one channel: we can use multiple routers with one or more connections each, or we use single routers with multiple sockets within a connection. It depends on the configuration of the host which solution we should use:

- If a single process can access the different channels, the multi-socket solution should be used to take advantage of the higher bandwidth even for single messages (connection between hosts B and C in our example).
- If the different channels can only be accessed from different processes on a host, the multi-router solution has to be used (resulting in the multiple connections between hosts A and C in our example).

Of course, you can use multiple routers, connections and sockets on a host even if it would not be necessary to verify configuration files or just for fun. Probably, some slow down will be noticed, but the overhead is very small due to the statical mapping of the routers and connections.

When a message is to be sent from one host (the source host) to another host (the destination host), the following rules are applied:

- If more than one connection exists between two hosts, the available connections are statically mapped to the MPI processes of the destination host.

NUMHOSTS 3 A 12 -xxgdb B 8 -v C 17		meta host declarations

OPTIONS
NICDEF addr_B TCP 1.1.2.1
METAHOST A {
NODES=hostA (1.1.1.1,1.1.1.2,1.1.1.3); }
METAHOST B {
NODES=hostB (addr_B,1.1.2.2,1.1.2.3); }
METAHOST C {
NODES=hostC (1.1.2.4,1.1.2.5,1.1.2.6); }
		Options

CONNECTIONS PAIR A B 1 - 1.1.1.1 -> addr_B1	subsection AB	section 2A
PAIR A C 2 - 1.1.1.2 -> 1.1.2.3 1.1.1.3 -> 1.1.2.3	subsection AC	
PAIR B A 1 - 1.1.2.1 1.1.1.1	subsection BA	section 2B
PAIR B C 1 - 1.1.2.2,1.1.2.3 -> 1.1.2.5,1.1.2.6	subsection BC	
PAIR C A 1 - 1.1.2.4 -> 1.1.1.2 1.1.1.3	subsection CA	section 2C
PAIR C B 1 - 1.1.2.5,1.1.2.6 -> 1.1.2.2,1.1.2.3	subsection CB	

Table 5.3: Configuration file for the MetaMPICH setup in figure 5.5 on page 109

- If more than one router process exists for the communication between the two hosts, the MPI processes of the source host are statically mapped to one of the available routers.
- If more than one socket is available in a connection, a message that is transferred via this connection is split in pieces and transferred in parallel via a number of sockets (see chapter 5.3.2, `SPLITSIZE` option).

Important issues:

- Socket numbers (the offsets following an IP) must be unique within each host-to-host connection.

5.3.4 Checking the configuration with checkmetacfg

There is a tool which checks a configuration files syntax in the `bin` directory named `checkmetacfg`. The tool outputs a detailed list of all meta hosts and their config-

uration and a list of routers on the meta host given on the command line. If you have trouble finding an error in your meta config file, you can use the `-yydebug` option to switch the parser in debug mode - this may give helpful information to useres with experience with bison/yacc.

5.3.5 More example configurations

To help setting up your own MetaMPICH cluster, we provide some more example configurations but without detailed explanation since they are self-explaining.

- Three hosts, each with one IP address, running one routing process towards each of the other two hosts. The hosts `sirius` and `magrathea` are connected via two parallel sockets (these two sockets could have different IP addresses, if the hosts had more than one NIC):

```
NUMHOSTS 3
frogstar 2
magrathea 2
sirius 2
OPTIONS
CONNECTIONS
PAIR frogstar magrathea 1 -
134.130.62.98 134.130.62.94
PAIR frogstar sirius 1 -
134.130.62.98:1 134.130.62.101:4
PAIR magrathea frogstar 1 -
134.130.62.94 134.130.62.98
PAIR magrathea sirius 1 -
134.130.62.94:0,134.130.62.94:2 134.130.62.101:1,134.130.62.101:2
PAIR sirius frogstar 1 -
134.130.62.101:4 134.130.62.98:1
PAIR sirius magrathea 1 -
134.130.62.101:1,134.130.62.101:2 134.130.62.94:0,134.130.62.94:2
```

- Two hosts with two NICs each, connected with one routing process each which utilises these NICs. Any number of NICs could be added as shown with two NICs in this example:

```
NUMHOSTS 2
worf 1
yar 1
OPTIONS
CONNECTIONS
PAIR worf yar 1 -
```

```
192.168.1.1:0,192.168.2.1:1 192.168.1.2:0,192.168.2.2:1
PAIR yar worf 1 -
192.168.1.2:0,192.168.2.2:1 192.168.1.1:0,192.168.2.1:1
SPLITSIZE 3000
```

- Two hosts with one IP address each, but one host is running two routers, the other host only one. This configuration makes sense if on the first host, multiple nodes have the same IP address:

```
NUMHOSTS 2
frogstar 1
magrathea 1
OPTIONS
CONNECTIONS
PAIR frogstar magrathea 2 -
134.130.62.98:0,134.130.62.98:1 134.130.62.94:0,134.130.62.94:1
134.130.62.98:2,134.130.62.98:3 134.130.62.94:0,134.130.62.94:1
PAIR magrathea frogstar 1 -
134.130.62.94:0,134.130.62.94:1 134.130.62.98:0,134.130.62.98:1
134.130.62.98:2,134.130.62.98:3
```

  Note: The last two lines printed above *have to be one line in the config file* (because it means that the router on `magrathea` connects to both routers on `frogstar`).

5.4 Usage

For users familiar with MPICH, the use of MetaMPICH instead won't cause any problems. If you are not familiar with MPICH or even with programming in MPI, you'll have to read the according documentation. This chapter does only deal with the differences between the use of the standard MPICH and MetaMPICH.

5.4.1 Creating MetaMPICH applications

Writing the application One of the design targets for MetaMPICH was transparency for the MPI application. Therefore, every MPI application should run on a working MetaMPICH configuration. However, to give the user the ability to optimise the application for the underlying MetaMPICH configuration, a new default communicator (next to `MPI_COMM_WORLD`) has been added. It is called `MPI_COMM_LOCAL` and includes all MPI processes on the meta host that the current MPI process runs on. Of course, *it does not include* the routing processes as they are transparent to the MPI application and can not be accessed from there.

Compiling the application An additional compiler option is required for compiling MPI programs for MetaMPICH. This option has to define the preprocessor macro META. Usually, an option like `-D` is used to define symbols on compile time. In this case, a compiler argument like `-DMETA` fulfills the task. Look at your compiler documentation if this option should not lead to the desired results. If you use the script `mpicc`, this symbol is already defined and you don't have to worry about this.

5.4.2 Running MetaMPICH applications

A new command line argument for `mpirun` is required to launch MetaMPICH applications. The simple syntax of the argument is

`-meta` *config_file*

This indicates MetaMPICH that the given application shall be run on the MetaMPICH configuration that is defined in the supplied configuration file. The according `mpirun` script gathers all required information to start the application from the config_file and finally starts the application on each participating meta host via remote shell (`rsh`) and the meta host-specific `mpirun` command. This "real" call of `mpirun` is done with the option `-metarun`, but the user will not have to deal with this internal `mpirun` option. Just make sure that the config file is accessible under the given name from every participating host and that the remote shell has the correct path to start `mpirun`.

The `mpirun` script uses a tool named `metapars` to choose the needed nodes from a meta host and determine the destinations of all routing processes. `metapars` generates the second-stage command lines using the normal non-meta `mpirun` commands to run the application.

The metarun option may be useful for testing or debugging purposes when you may need or want to manually start the application on the participating meta hosts. To do so, start the application like you would start an ordinary MPICH application and supply the options `-metarun` *config_file* and `-metahost` *metahost_name* as a parameter to the application itself (not for "`mpirun`"). The timing of this launch is not critical as the router processes will try to connect for a period of time while the real MPI processes wait for them.

Please notice that all other options for "`mpirun`" which would be used if the application was directly launched on a specific host have to be placed into the according meta host line (section 1 of the configuration file, see chapter 5.3.2). The options that are designated for the application have to be appended to the command line and are passed to the application as usual.

To give an example, regard the meta configuration file in figure 5.6 (only the section 1 of the configuration file is displayed since only this section is relevant for `mpirun`) named metatest. If you launch a command like

```
mpirun -meta /home/joe/metatest my_prog gridsize 1000
```

```
NUMHOSTS 2
foo 16 -xxgdb
bar 32 -v
```

Figure 5.6: Section 1 of a meta configuration file

`mpirun` will launch the following commands concurrently:

```
rsh node_in_foo mpirun -np 16 -xxgdb my_prog gridsize
   1000 -- -metahost foo -
metarun /home/joe/metatest
rsh node_in_bar mpirun -np 32 -v my_prog gridsize 1000 --
    -metahost bar -
metarun /home/joe/metatest
```

The correction of the number of processes (in this case, `MPI_COMM_WORLD` will contain 48 MPI processes plus the router processes) is done internally in the MetaMPICH library.

If a MetaMPICH application crashes, it is possible that an IPC message queue or semaphore which was created for the initialisation is not deleted which, on certain platforms, gives problems when starting the application again. In this case, the message queue or semaphore has to be removed manually using the `ipcs` and `ipcrm` commands. You may also try the `cleanipcs` script in `$MPIROOT/util`.

5.5 Platform notes

Some platforms have certain problems with MPICH or the MetaMPICH extension. We try to give hints to solve these problems (which often are problems with MPICH itself and not originating from the MetaMPICH extension).

5.5.1 Solaris 2.6(Intel)

- If the Gnu compiler `gcc` is used, MetaMPICH has to be configured not to support Fortran (switch `-nof77`) due to a symbol resolving problem in the concerned libraries.

- If the `ch_shmem` device is used, make sure that the maximum size of shared segments is set to a value big enough for your application (a multiple of the maximum message size, depending on the number of nodes in your cluster) by setting a parameter in `/etc/system`.

5.5.2 Linux 2.x(Intel)

- The setup of the shared memory needs a patch in `mpid/ch_shmem/p2p_shmalloc.h`. The following two blocks must be swapped when using Linux (this is already done in the MetaMPICH distribution, depending on the definition of `MPI_LINUX`):

```
/* If there is nothing special then
we will use mmap if we have it. */
#if !defined(SHMEM_PICKED) && defined(HAVE_MMAP)
# define USE_MMAP
# define SHMEM_PICKED
#endif
/* Reluctant default: System V shared
memory segments. Look out for ipcs. */
#if !defined(SHMEM_PICKED) && defined(HAVE_SHMAT)
# define USE_SHMAT
# define SHMEM_PICKED
#endif
```

5.5.3 Cray T3E

- Due to the lack of system wide IPC facilities, a special version of the pre-MPI-synchronisation routines had to be implemented. This is realised via `shmem_barrier_all()` in the files `src/env/pidsync.c` and `src/env/pidsync.h`, enclosed in `#ifdef _CRAY`. This is not noticeable by the user.

5.6 Internal Design

The goals of the MetaMPICH development were to enable the cooperation of heterogeneous, locally distributed systems within one MPI application, known as "meta-computing". As a first step towards an efficient meta-computing, the MPI implementation MPICH which is available on nearly every platform was to be extended into this direction considering the special setup of the test-configuration and to minimise the time to implement the software. The principle of the extensions included in MetaMPICH are illustrated in figure 5.7.

5.6.1 Modifications to MPICH

The functionality of MetaMPICH as an extension to MPICH is located in three source directories (`mpid/ch_gateway`, `mpid/ch_tunnel` and `src/routing`) and is separated from the original MPICH functionality. Although this means that one design target of MetaMPICH was the avoidance of modifications to the existing

MPICH source code to make the extension clean and simple, some minor changes in the existing code could not be avoided due to fundamental design limits. However, these changes only do concern the initialisation and finalisation of MPICH and the native ADI-2 devices. The most important changes are explained in this chapter; see chapter 5.8.2 for a complete listing of modified files.

Initialisation For proper initialisation of more than one ADI-2 device, some minor changes had to be done in `mpid/ch2/adi2init` and some more changes (for the setup of the meta configuration and environment) in `src/env/initutil.c`. These modifications include:

- `MPIR_Parse_Metaconfig(metacfg_file, devcfg)`

IN	char *metacfg_file	the name of the configuration file
OUT	MPID_Config **devcfg	device initialisation information

  New function to transfer the meta configuration from the configuration file into a global structure (`metampi_cfg`, defined in `include/metampi.h`)

- `MPIR_Copy_collops(comm)`

IN/OUT	struct `MPIR_Communicator` *comm	Communicator to get new collops

  New function to copy the field of function pointers pointing to collective operations. Each communicator owns such a field, but actually this have been the same pointers for every communicator. To be able to assign different collective operations to different communicators, this pointer field has to be copied. This function copies the function pointers of the supplied communicator back into the same communicator so they can be modified without interfering with other communicators.

- Parsing of meta-relevant parameters `in MPIR_Init()` (`src/env/initutil.c`, line 466), as there are:

  `-metarun filename` This is the parameter which makes an MPI application run in a meta environment. The configuration of this environment is described in the configuration file whose name is supplied.

  `-routing` On different platforms, different launching procedures for the application and routing processes are needed. This parameter indicates that this process shall become a routing process making it possible to specifically place a routing process on a certain node.

  `-barrier` For debugging and testing purposes, this parameter sets the global variable meta_barrier to true. This variable is used in various places to halt processes in an endless loop, allowing to attach with a debugger.

`-np nbr_of_proceses` This is a standard parameter of MPICH, but it is modified according to the real number of processes running in the meta environment (this number is gained from section 1 of the configuration file) before the initialisation of the devices. This is necessary on some platforms to let MPICH initialise and use data structures according to the global ranks of the processes in the meta environment.

- Creation of the additional communicators `MPI_COMM_ALL`, `MPI_COMM_META`, `MPI_COMM_HOST` and `MPI_COMM_LOCAL`. (`src/env/initutil.c`, line 803)
- Call of the routing routine `MPIR_Router()` for the processes which have identified themselves as routing processes (`src/env/initutil.c`, line 1537)

Native ADI-2 devices MetaMPICH was first implemented on Sun Solaris SMP-Workstations (Sparc, Intel) which use the `ch_shmem` device for communication between the processes of an MPI application. The modifications which were necessary in the `ch_shmem` device (and which might be necessary for using other ADI-2 devices in a similar way) are caused by the difference in the number of processes that the ADI-2 device needs to know and the ranks it has to serve. In the `ch_shmem` device, the following modifications had to be performed:

- a device-internal barrier needs to be set to the number of local processes (`mpid/ch_shmem/shmempriv.c`, line 152)
- the offset `MPID_shmem->globid` for the numbering of the processes on the local host has to be set to the rank of the first process on this host according to the meta configuration (`mpid/ch_shmem/shmempriv.c`, line 180)
- the number of processes to be spawned on this host has to be calculated according to the information from the meta configuration file and not from the argument n of the `-np` parameter (see above, modification in `mpid/ch_shmem/p2procs.c`, line 450)

When another ADI-2 device is to be used in a meta environment, similar changes which are all located in the initialisation section of the device might be necessary.

Message processing To determine the routing for a message that the MPI application sends, the first decision is based on the communicator used and its mapping. The mapping from the internal communicator ranks to global ranks is generated in `MPID_Set_mapping()`. For the global ranks, one mapping from the global ranks to the device to be used exists (`MPID_devset->dev`).

- `MPI_COMM_LOCAL` The mapping of this communicator is limited to processes running on the same host as the calling MPI application process. All messages are directly sent via the internal ADI-2 device.

- `MPI_COMM_WORLD` This communicator has a mapping that includes all global ranks without the router processes. In fact, `MPI_COMM_WORLD` gets replaced by the private MetaMPICH communicator `MPI_COMM_META`. Another private communicator `MPI_COMM_ALL` has the meaning that `MPI_COMM_WORLD` usually had (including *all* running processes, the application and the router processes) but is only used internally.

When the mapping indicates that the message has to be sent via the gateway device, the message is processed as described in chapter 5.6.3.

Shutdown Upon completion of the MPI application, all processes in `MPI_COMM_WORLD` call `MPI_Finalize()`. The router processes, however, do need to get told that they are expected to shut down, too, by calling `MPI_Finalize()`. For this purpose, `MPI_Finalize()` was extended: on each host, the application process with rank 0 sends a special message to all router processes (via the `MPI_COMM_HOST` communicator) telling them to shut down. They in turn synchronise with all other router processes that they are connected with before closing this connection.

5.6.2 Ranking System

The different communicators contain different groups of processes. To understand the internal design of MetaMPICH, it is necessary to understand the pattern by which the different ranks are assigned to the processes. As an example, regard the configuration file in figure 5.8 and the resulting meta configuration with the ranks of the processes in the different communicators in figure 5.9. We recognise the rules that are applied for assigning ranks:

1. The hosts are treated in the order of their appearance in section 1 in the configuration file. In the example, this means the processes on host A are given lower global ranks than the processes on host B relative to `MPI_COMM_WORLD` and `MPI_COMM_ALL`.

2. On each host, the application processes are given lower host-internal ranks than the router processes. For our examples, this means that the application processes have the ranks 0 through 3, the router processes the ranks 4 and 5 relative to `MPI_COMM_HOST`.

3. The relative order of the processes remains the same through all communicators: Rank(P_1, Comm A) $<$ Rank (P_2, Comm A) $\Rightarrow$ Rank(P_1, Comm B) $<$ Rank (P_2, Comm B)

4. The router processes are not part of `MPI_COMM_LOCAL`.

5. To the application processes, the communicator `MPI_COMM_WORLD` is equivalent to the communicator `MPI_COMM_META` (which they do not know of).

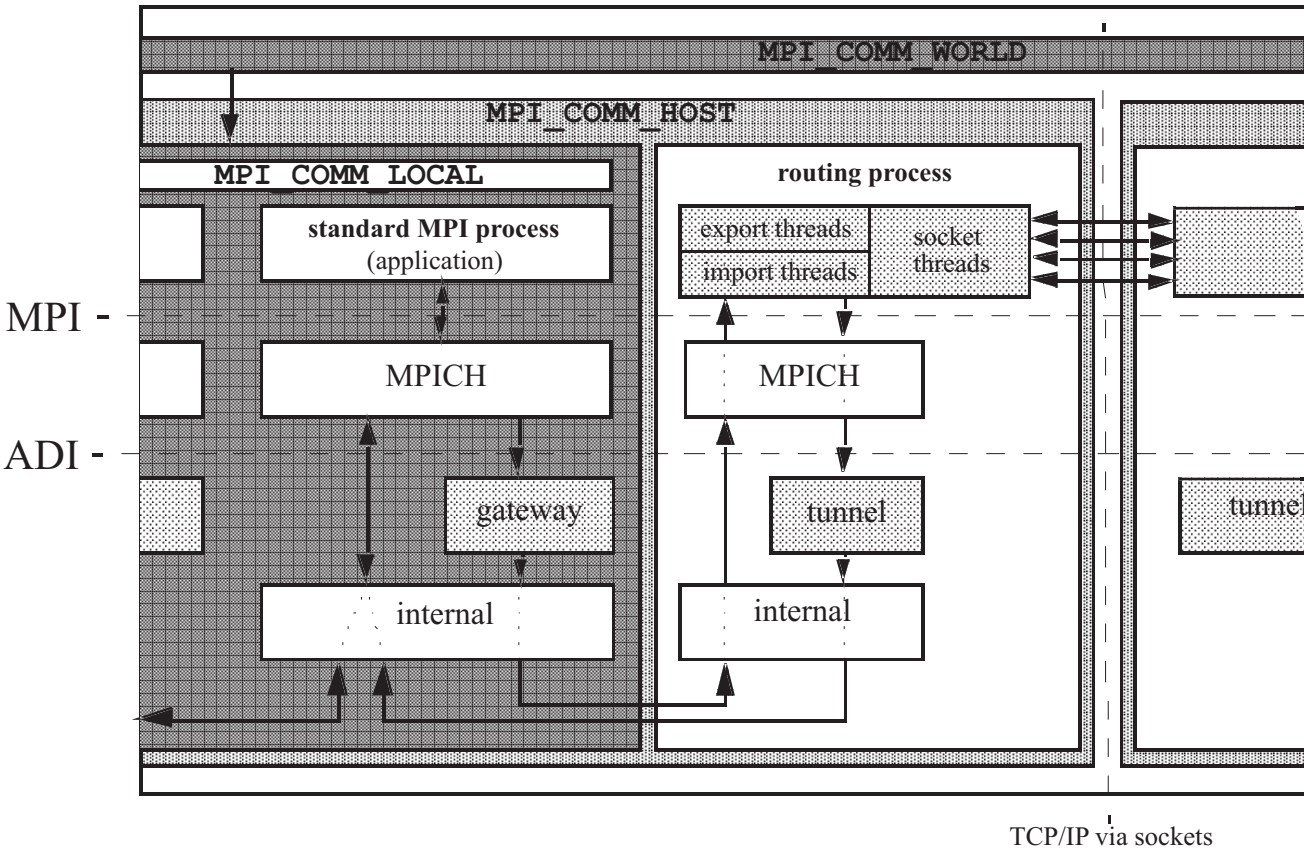


Figure 5.7: Concept of the meta extension to MPICH

```
NUMHOSTS 2
A 4
B 4
OPTIONS
CONNECTIONS
PAIR A B 2 -
1.1.1.1 -> 1.1.1.3
1.1.1.2 -> 1.1.1.4
```

Figure 5.8: example configuration file

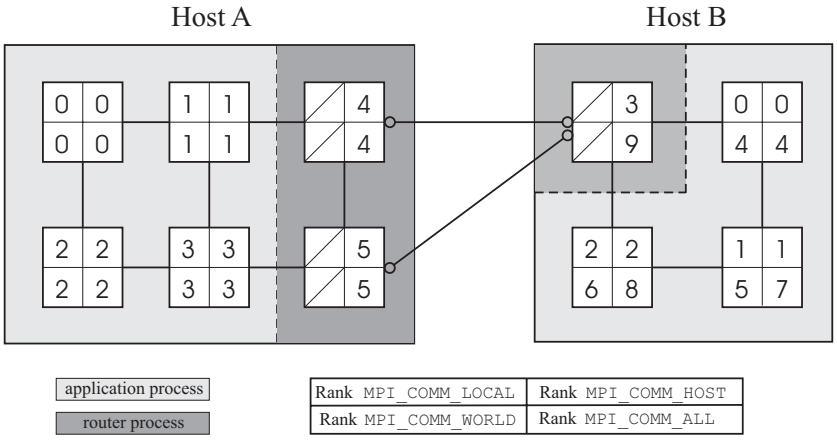


Figure 5.9: resulting meta environment

6. To the router processes, the communicator `MPI_COMM_WORLD` is equivalent to `MPI_COMM_ALL`.

5.6.3 Gateway- and Tunnel-Devices

The *gateway device* (`ch_gateway`) and the *tunnel device* (`ch_tunnel`) appear as ADI-2 devices towards the upper layers of MPICH, but actually work as wrappers with some added functionality to the real, internal ADI-2 device of the system. This allows the use of these pseudo-devices on any system featuring an ADI-2 device. The gateway device is used for sending messages from a MPI application process to a router process; the tunnel device is used for sending a message from the router process to the destination application process after it has been transmitted via the two router processes. The way that a message between two MPI application processes on different machines takes is noted below:

1. The message is sent inside the `MPI_COMM_WORLD` communicator. From the mapping of the global ranks to devices, the MPIR-Layer chooses to call the

appropriate send-function from the gateway device.

2. This function prepends a meta message header (defined as `Meta_Header` in `src/routing/mpi_router.h`) to the MPI message. This header contains all the information that is necessary to send a message between two MPI processes on the device level. This information contains:

   - the original global source and destination ranks of the message
   - the tag and the context_id of the original message
   - the mode in which the message was sent (blocking or non-blocking)
   - the byte size of the original message contents (which is appended to the header exactly like it has been sent from the source MPI process)
   - the "message representation" of the message contents which defines the encoding (none, byte-swap or XDR encoded)

3. The original MPI message with the prepended header is sent to the routing process. The global rank of the routing process is derived from a mapping of the global MPI ranks to the available router processes (`MPIR_Meta_cfg.granks_to_router`, established in `MPID_Gateway_init()`). This mapping may be different for the MPI processes of one host due to multiple available router processes towards a remote host. To send the message, the internal ADI-2 device of the system is used.

4. The export thread of the router process accepts the message, reads the global destination rank of the message (stored in the prepended header) and decides which connection will be used for this message (`get_conn_for_dest()` in `mpi_router.c`). The message is then transmitted via this connection, possibly handled by multiple threads if the connection is a multi-socket-connection.

5. The receiving threads of the router process on the other hosts recombine the message if necessary and handle it to the import thread of the router process which own the connection used. This import thread sends the message including the meta header to the global rank given in the meta header as a MPI message with `MPI_BYTE` datatype. In any case, this send is done via the tunnel device due to the mapping being set up accordingly.

6. The tunnel device gathers all information from the meta header before stripping it from the message. It then sends the original MPI message via the ADI-2 device using the rank of the original sender instead of his own rank as return address.

5.6.4 Router

Every MetaMPICH application running on at least 2 hosts needs at least one router process on each host. Theses router processes change the SPMD structure of the normal MPI application, because now there are different programs communicating via MPI. Although the router processes are not in the `MPI_COMM_WORLD` communicator, they are part of the MPI application. But the MPI application programmer does not have to cope with this structure modification because MetaMPICH tries to hide it.

The main loop of the router can be divided into two parts, each working for one direction of the communication. To avoid deadlocks and high latencies, these parts are implemented with threads. The problem which occurs is the non-threadsafeness of MPICH, so a lock has to be used to protect library function calls.

One thread per connection executes `select` on the sockets, waiting blocking for the next message. If data is available, one thread per socket begins to read the message, which is re-assembled when the parts have been received. Now the message is unwrapped and sent to the appropriate MPI process using the tunnel device.

The MPI part of the router uses `MPI_Iprobe` to detect available MPI messages which have been sent to the router by local MPI processes. Unfortunately, this has to be done by polling with the shared memory device. This is the reason why the routing process produces a CPU load of 100%. In addition, there is a continuous locking and unlocking of the MPICH library protection lock.

The router gets MPI messages from the gateway devices of application processes in a special message envelope which contains the original message. It unwraps this message and sends it to another router via TCP/IP (or another network, which can be implemented easily because of the modular design). A router MPI message can contain also a special command for the router itself, e.g. to terminate it.

The implemented TCP interface is prepared to use multiple network interfaces to transmit large messages. In order to do this it has to be configured with multiple IP addresses or and/or ports and a parameter called splitsize. This parameter sets the maximum size of non-split messages.

The router splits large messages into pieces and sends them over the multiple sockets using one thread per socket. The multi-socket configuration works even with only one network interface, when a port offset is appended to the IP address in the configuration file. This offset has to be used too, if multiple routers using one IP address are necessary.

Because a router has only connections to routers on one host, it is not a real router. Instead it distributes the messages on the routers of the other host. This is done by a simple modulo operation now, what can easily be changed. Think of a mapping which is built dynamically using statistics of the load of single router-router connections.

5.7 Performance

5.7.1 Evaluation platform

The performance numbers given in this chapter were all obtained on two Dual-PentiumPro Systems (200MHz, 256kB second level cache, 256 MB memory) running under Solaris 2.6, respectively. The network connection between these systems was realised via a quad-Ethernet NIC (Adaptec ANA6904 PCI, 10Mbit twisted pair full duplex) providing a direct, exclusive 4-channel connection without external interference. Additional, the standard NIC connected the systems via a switched 10Mbit Ethernet channel which was used as the fifth channel in our experiments.

5.7.2 Network performance

One of the main goals was that the available communication lines between the hosts should be used as efficiently as possible. Therefore, the routing processes can handle multiple sockets (to speak in terms of TCP/IP) to transfer a single message in parallel. This feature leads to a nearly linear speedup in peak transmission speed for messages via a connection versus the number of sockets which build this connection. This behaviour is illustrated in figure 5.10, measured with a pair of benchmark programs (called "speed_send" / "speed_recv") which transfer blocks of data with the given size from one host towards the other one using the according `send()` / `recv()` functions of the MetaMPICH routing process. However, not every message is send via multiple sockets. The number of sockets used for transmitting a message is depending on the size of the message and a threshold value called "split size". A simple formula serves to determine the number of channels $N_{channel}$ (which are represented by sockets for TCP/IP) from the message size M and the split size S:

$$N_{channel} = \text{Min } \{\text{Available channels, M div S + M mod s}\}$$

The value of split size has an important influence on the effective bandwidth and must be found as a trade-off between higher available bandwidth and higher synchronisation overhead for the threads when using more channels. The effect of choosing a value too small can clearly be recognised in figure 5.12 where a split size of 150 byte leads to performance loss more than 50% when transmitting a 256 byte message via two channels instead of only one.

To determine the optimal splitsize for a given network connection between two hosts, a small tool set named `optsplit_send/optsplit_recv` exists in `/src/routing/`. Look at the source code to learn how to use this tool.

5.7.3 MPI performance

To quantify the effective MPI bandwidth relative to the raw network bandwidth, the equivalent MPI program to the benchmark of chapter 5.7.2 was tested. It is

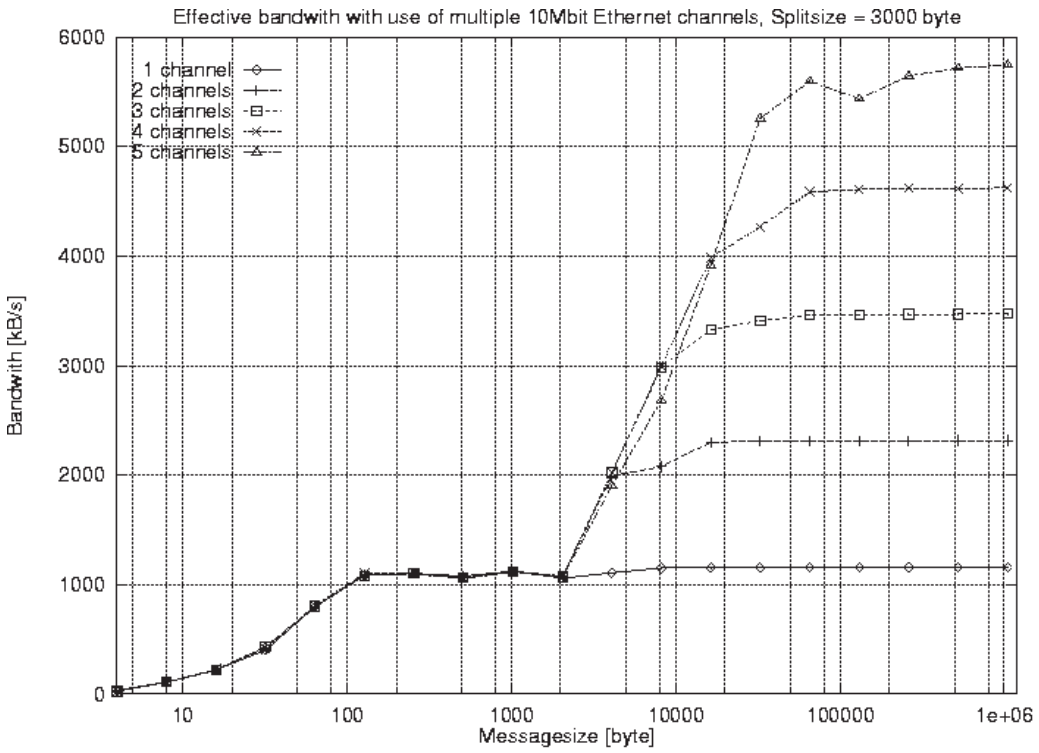


Figure 5.10: TCP/IP performance of the routing process

called "mpi_speed" and transfers the same blocks of data, but as a MPI application process in a meta configuration with one application and router process on each of the two hosts described in chapter 5.7.1 The results are shown in figure 5.12 and should be compared with figure 5.10 to see the overhead that is involved with the internal transfers of the MPI messages on each host and the routing in the two routing processes. Most part of this overhead is caused by the need of polling the internal MPICH device in the routing process which consumes most of the available CPU cycles. If the internal MPICH device can be used in a blocking mode, this problem should disappear. Another area for optimisation is locking of the MPI functions which is necessary because of MPICH not being thread-safe.

Of course, the performance of a MPI application is affected by running it on different hosts as the communication bandwidth between two processes on distinct hosts will always be a magnitude or more below the internal communication bandwidth on a typical host. Thus, efficient use of such a meta-computing environment requires the use of the `MPI_COMM_LOCAL` communicator where possible. Some efforts where made to adapt the usual collective operations to this environment. An example for such an operation is the barrier synchronisation. The meta variation of the barrier (attached to the `MPI_COMM_WORLD` communicator) performs a local

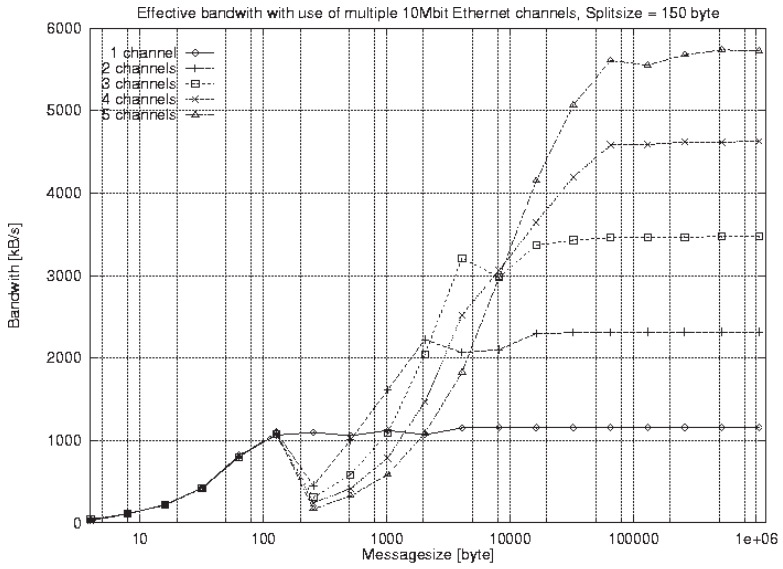


Figure 5.11: Compensation of higher Bandwidth by synchronisation overhead

synchronisation with all application processes on the host, then a barrier between all processes with local rank 0 on every host. A final local barrier is required to finish the global barrier. However, it made no sense to measure the effects of this technique on the used evaluation platform due to the lack of a significant number of MPI application processes running on the hosts.

5.8 Implementation details

5.8.1 Restrictions, limits and warnings

- The number of routing processes on a host is limited by the define for `META_MPI_MAX_RP` in `include/metampi.h`. Currently, it is set to 16 but may be set to any number if necessary.

5.8.2 Modified files

A number of source files from the original MPICH distribution had to be added modified to introduce the meta capabilities. These files are shown in table 5.4. The filenames are relative to the MPICH root directory. The state "mod" means that this file was modified, the state "new" indicates a new file or directory. Although it was the intention for the design of MetaMPICH not to modify the existing sources of MPICH but only add new source files, this could not be fully avoided. However, the modifications are very limited in size and number and are embraced with an

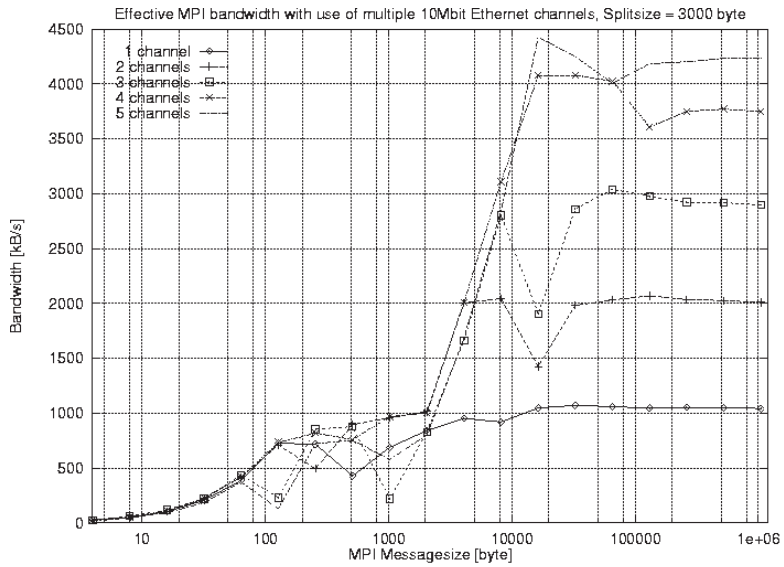


Figure 5.12: MPI Bandwidth illustrating inter-communication and routing overhead

`#ifdef META ... #endif` construct for easy location. If you compile MetaMPICH without defining the symbol META, you will get the original MPICH 1.1.0 code.

5.8.3 Error messages and solutions

5.8.4 Revision History

Release A1 July 17, 1998

- tested on Solaris and Linux
- does not include XDR

5.8.5 Release A2

September 11, 1998

- support for heterogeneous configurations added (using XDR)
- improved effective bandwidth for small messages

Filename	State	Description of the changes
configure	mod	- added the new source files and directories - modification of mpicc script
util/mpirun.sh.in	mod	recognition of option -meta to launch mpirun.meta
util/mpirun.meta.in	new	new script to start the MPI application on each host (needs gawk)
include/mpi.h	mod	added new `MPI_COMM_` communicator values for `ALL`, `META`, `LOCAL` and `HOST` and replaced the standard `WORLD` communicator by `META`
include/metampi.h	new	definition of the global meta configuration structure
include/pidsync.h	new	Synchronisation and leader election using IPC message queues (header file)
mpid/ch2/adi2init.c	mod	some correction for initialisation of multiple devices
mpid/ch_gateway/	new	this directory contains the gateway device
mpid/ch_tunnel/	new	this directory contains the tunnel device
mpid/ch_shmem/ p2pprocs.c	mod	adapted the number of processes to be forked
mpid/ch_shmem/ shmempriv.c	mod	adapted the number of participants on the shared memory barrier
mpid/ch_shmem/ shmemchkdev.c	mod	included support for heterogeneous communication in the shmem-packet
ssrc/routing/	new	this directory contains the routing process
src/env/initutil.c	mod	`MPIR_Init()`: - parsing of the configuration file - creation of the new communicators - start of the router process
src/env/finalize.c	mod	`MPI_Finalize()`: added global synchronisation of the MetaMPICH cluster prior to local shutdown
src/env/pidsync.c	new	Synchronisation and leader election using IPC message queues

Table 5.4: Modified files in MetaMPICH

Message	Additional Information	Possible Reasons	Comment
tcp_bind: error binding socket	errno	wrong IP in configuration file	the bind call cannot bind a socket to a given inet-address
tcp_accept: accept error	errno		
tcp_connect: non-existent connection handle	handle value		function is passed an invalid connection handle
tcp_establish_connection: non-existent connection handle	handle value		
tcp_send_message: non-existent conn handle	handle value		
tcp_receive_message: non-existent conn handle	handle value		
tcp_select: non-existent connection handle	handle value		
tcp_check_message: non-existent connection handle	handle value		
tcp_connect: server must not CONNECT			a connect is executed on the server side of a connection
tcp_connect: socket already connected			tcp_connect is called on an already connected connection
tcp_connect: connection not down			wrong connection state for this operation

Table 5.5: MetaMPICH error messages

List of Figures

Index