High Performance Clustering
**Acknowledgement**

The development of MPI Connect has benefited greatly from the work of people not connected to Scali. We wish especially to thank the developers of MPICH for their work which served as a reference when implementing the first version of MPI Connect. The list of persons contributing to algorithmic MPI Connect improvements is impossible to compile here. We apologise to those who remain unnamed and mention only those who certainly are responsible for a step forward.

Scali is thankful to Rolf Rabenseifner for the improved reduce algorithm used in MPI Connect.
# Table of contents

<table>
<thead>
<tr>
<th>Chapter 1 Introduction</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Scali MPI Connect™</td>
<td>6</td>
</tr>
<tr>
<td>Chapter 2 Using MPI Connect</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Setting up a MPI Connect environment</td>
<td>7</td>
</tr>
<tr>
<td>2.1.1 MPI Connect environment variables</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Compiling and linking</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1 Compiler support</td>
<td>8</td>
</tr>
<tr>
<td>2.2.2 Compiler flags</td>
<td>8</td>
</tr>
<tr>
<td>2.2.3 Linker flags</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Running MPI Connect programs</td>
<td>8</td>
</tr>
<tr>
<td>2.3.1 Naming convention</td>
<td>8</td>
</tr>
<tr>
<td>2.3.2 mpimon - monitor program</td>
<td>9</td>
</tr>
<tr>
<td>2.3.3 mpirun - wrapper script</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Useful tools</td>
<td>13</td>
</tr>
<tr>
<td>2.4.1 Debugging with a sequential debugger</td>
<td>14</td>
</tr>
<tr>
<td>2.4.2 Useful built-in-tools for debugging</td>
<td>14</td>
</tr>
<tr>
<td>2.4.3 Profiling MPI Connect applications</td>
<td>15</td>
</tr>
<tr>
<td>2.4.4 Profiling with ScaMPE</td>
<td>19</td>
</tr>
<tr>
<td>2.5 An example program</td>
<td>21</td>
</tr>
<tr>
<td>2.5.1 Hello-world.c - source in C</td>
<td>21</td>
</tr>
<tr>
<td>2.5.2 Hello-world.f - source in Fortran</td>
<td>22</td>
</tr>
<tr>
<td>2.5.3 Compiling</td>
<td>22</td>
</tr>
<tr>
<td>2.5.4 Linking</td>
<td>22</td>
</tr>
<tr>
<td>2.5.5 Running</td>
<td>22</td>
</tr>
<tr>
<td>2.6 MPI test programs</td>
<td>23</td>
</tr>
<tr>
<td>2.6.1 Producer - a producer-consumer MPI test program</td>
<td>23</td>
</tr>
<tr>
<td>2.6.2 Bandwidth - a bandwidth MPI test program</td>
<td>23</td>
</tr>
<tr>
<td>2.6.3 Bidirect - a bidirectional MPI test program</td>
<td>24</td>
</tr>
<tr>
<td>Chapter 3 Description of ScaMPI</td>
<td>25</td>
</tr>
<tr>
<td>3.1 General description</td>
<td>25</td>
</tr>
<tr>
<td>3.1.1 ScaMPI libraries</td>
<td>25</td>
</tr>
<tr>
<td>3.1.2 ScaMPI executables</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Starting ScaMPI application programs</td>
<td>25</td>
</tr>
<tr>
<td>3.2.1 Application start-up - phase 1</td>
<td>26</td>
</tr>
<tr>
<td>3.2.2 Application start-up - phase 2</td>
<td>26</td>
</tr>
<tr>
<td>3.2.3 Application start-up - phase 3</td>
<td>27</td>
</tr>
<tr>
<td>3.3 Stopping ScaMPI application programs</td>
<td>28</td>
</tr>
<tr>
<td>3.4 Communication protocol on standard-devices</td>
<td>29</td>
</tr>
</tbody>
</table>
3.5 Communication protocols on DAT-devices........................................ 29
  3.5.1 Inlining protocol........................................................................ 30
  3.5.2 Eagerbuffering protocol............................................................. 30
  3.5.3 Transporter protocol.................................................................. 31
  3.5.4 Zerocopy protocol .................................................................... 32
3.6 Communication resources on DAT-devices...................................... 32
  3.6.1 Connection............................................................................. 33
  3.6.2 Channel buffer........................................................................ 35
  3.6.3 Eagerbuffer buffer................................................................. 36
  3.6.4 Transporter buffer.................................................................... 37
3.7 Communication media .................................................................... 37
  3.7.1 Using Scali supplied DAT-adapters........................................... 38
  3.7.2 Using third-party/own DAT-adapters......................................... 38

Chapter 4  Tips & Tricks for ScaMPI...................................................... 39
  4.1 Application program notes............................................................ 39
    4.1.1 MPI_Probe() and MPI_Recv()............................................... 39
    4.1.2 Unsafe MPI programs .......................................................... 40
  4.2 Namespace pollution ................................................................... 40
  4.3 Error and warning messages....................................................... 41
    4.3.1 User interface errors and warnings....................................... 41
    4.3.2 Fatal errors ......................................................................... 41
  4.4 When things don’t work - troubleshooting .................................... 41
    4.4.1 Standard input and ScaMPI .................................................. 41
    4.4.2 Why doesn’t my program start to run?................................. 41
    4.4.3 Why doesn’t mpid start...................................................... 42
    4.4.4 SCI interconnect problems ................................................. 42
    4.4.5 Why does my program terminate abnormally?........................ 43
    4.4.6 How do I control shared memory usage?............................... 44
  4.5 How to optimize MPI performance .............................................. 45
    4.5.1 Performance analysis............................................................ 45
    4.5.2 Using MPI_Isend(), MPI_Irecv()........................................... 45
    4.5.3 Using MPI_Bsend()............................................................... 45
    4.5.4 Avoid starving mpi-processes - fairness............................... 45
    4.5.5 Using processor-power to poll............................................. 45
    4.5.6 Communication buffer adaption .......................................... 46
    4.5.7 Reorder network traffic to avoid conflicts............................ 46
  4.6 Benchmarking ............................................................................ 47
    4.6.1 How to get expected performance....................................... 47
    4.6.2 Memory consumption increase after warm-up....................... 47

Chapter 5 Support ............................................................................. 48
  5.1 Feedback .................................................................................. 48
  5.2 Scali mailing lists ...................................................................... 48
Chapter 1

Introduction

A Scali System is a set of interconnected nodes, where each node is a multi-
processor workstation running Linux. To help you use the full potential power
of such a system we have developed a suite of software.

1.1 Scali MPI Connect™

Scali MPI Connect™ is a high performance MPI implementation. The
programming environment for MPI Connect provides a variety of options and
tools for tuning and debugging. Any parallel MPI-conforming application can be
run with SMC and benefit from its performance.

This guide is written for users who have a basic understanding of MPI, and
some basic knowledge of the C and/or Fortran programming language. gcc and bash are used for all examples.
Chapter 2 Using MPI Connect

This chapter describes the setup, compile, link and run of a program using MPI Connect. Furthermore some useful tools for debugging and profiling are briefly discussed.

Please note that the “ScaMPI release notes” are available in the /opt/scali/doc/ScaMPI directory.

2.1 Setting up a MPI Connect environment

2.1.1 MPI Connect environment variables
The use of MPI Connect requires that some environment variables are defined. These are usually set in the standard startup scripts (e.g., bashrc when using bash), but they can also be defined manually.

- **MPI_HOME**
  Installation directory. For a standard installation, the variable should be set as:
  ```
  export MPI_HOME=/opt/scali
  ```

- **LD_LIBRARY_PATH**
  Path to dynamic link libraries. Must be set to include the path to the directory where these libraries can be found:
  ```
  export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:$MPI_HOME/lib
  ```

- **PATH**
  Path variable. Must be updated to include the path to the directory where the MPI binaries can be found:
  ```
  export PATH=${PATH}:$MPI_HOME/bin
  ```

Normally, the MPI Connect library’s header files mpi.h and mpif.h reside in the $MPI_HOME/include directory.

2.2 Compiling and linking

MPI is an “Application Programming Interface” (API) and not an “Application Binary Interface” (ABI). This means that you as a main rule should compile and link your application when starting to use MPI Connect. But since the mpich-implementation is widely used we have made MPI Connect ABI compatible depending on versions of mpich and MPI Connect. Please check “ScaMPI release notes” for details. If your application is dynamically linked with mpich you should only need to change the library-path(LD_LIBRARY_PATH) to get it running.
2.2.1 Compiler support

**MPI Connect** is a C++ library built using the GNU compiler. This implies that you have to link with the GNU runtime library. Depending on the compiler used, the way to link with the **MPI Connect** libraries varies. Check the "ScaMPI release notes" for information on supported compilers and how linking is done. Please note that the GNU compiler, or a similar version of the C++ compiler, must be installed on your system. The GNU compilers are included in the ScaFegcs package, available for download at [http://www.scali.com/](http://www.scali.com/).

2.2.2 Compiler flags

The following string must be included as compile flags (bash syntax):

```
-D_REENTRANT -I$MPI_HOME/include
```

2.2.3 Linker flags

The following string outlines the setup for the necessary link flags (bash syntax):

```
-L/opt/scali/lib $CRT_BEGIN -lmpi $CRT_END
```

The runtime setup CRT_BEGIN and CRT_END libraries are defined for some compilers. Please note that when linking a Fortran main program, the Fortran interface library `libfmpi` must be included before CRT_BEGIN.

2.3 Running MPI Connect programs

Note that executables issuing **MPI Connect** calls cannot be started directly from a shell prompt. **MPI Connect** programs can either be started using the MPI monitor program `mpimon`, the wrapper script `mpirun`, or from the Scali Universe GUI [5].

2.3.1 Naming convention

When an application program is started, **MPI Connect** is modifying argv[0]. The following convention is used for the executable, reported on the command line using the Unix utility `ps`:

```
<userprogram>-<rank number>(mpi:<pid>@<nodename>)
```

where:

- `<userprogram>` is the name of the application program.
- `<rank number>` is the application's mpi-process rank number.
- `<pid>` is the Unix process identifier of the monitor program `mpimon`. 

---

*Scali MPI Connect Users Guide Version 1.1*
Section: 2.3 Running MPI Connect programs

<nodename> is the name of the node where mpimon is running.

Note that MPI Connect requires a homogenous file system image, i.e., a file system providing the same path and program names on all nodes of the Scali System.

2.3.2 mpimon - monitor program
The control and start-up of an MPI Connect application is monitored by mpimon.

2.3.2.1 Basic usage
Normally mpimon is invoked as:

mpimon <userprogram> <program options> -- <nodename> [ [<count>] [...

where

  <userprogram> is name of application
  <program options> are options to the application
  -- is the separator ending the application options
  <nodename> [ <count> ] is name of node and the number of mpi-processes
to run on that node. The option can occur several times in the list. Mpi-processes will be given ranks sequentially according to the list of node-number pairs. The <count> is optional and if omitted a count of one is assumed

Examples:
Starting the program "/opt/scali/examples/bin/hello" on a node called "hugin":

mpimon /opt/scali/examples/bin/hello -- hugin

Starting the same program with two processes on the same node:

mpimon /opt/scali/examples/bin/hello -- hugin 2

Starting the same program on two different nodes, "hugin" and "munin":

mpimon /opt/scali/examples/bin/hello -- hugin munin

Starting the same program on two different nodes with 4 processes on each:

mpimon /opt/scali/examples/bin/hello -- hugin 4 munin 4

2.3.2.2 Advanced usage
The program mpimon has several options which can be used for optimising MPI Connect performance. Normally it should not be necessary to use any of these options. However, unsafe MPI programs might need buffer adjustments to solve deadlocks. Another reason for trying the "advanced usage" is if you are running multiple applications in one run.

The complete syntax for the program:

mpimon [ <mpimon option> ]... <program & node-spec> [ -- <program & node-

Specs> ]...
where

\(<\text{mpimon options}>\) are options to mpimon; See "Appendix A" for complete list of options.

\(<\text{program & node-spec}>\) is an application and node specification consisting of

\(<\text{program spec}> -- <\text{node spec}> [<\text{node spec}>]\)...

and

\(<\text{program spec}>\) is an application and application-options specification

\(<\text{node spec}>\) is specifying which node and how many instances

\(<\text{nodename}> [<\text{count}>]\).

If <count> is omitted, one mpi-process is started on each node specified.

Examples:

Starting the program "/opt/scali/examples/bin/hello" on a node called "hugin" and the program "/opt/scali/examples/bin/bandwidth" with 2 processes on "munin":

\texttt{mpimon /opt/scali/examples/bin/hello -- hugin -- /opt/scali/examples/bin/bandwidth -- munin 2}

Changing one of the mpimon-parameters:

\texttt{mpimon -channel_entry_count 32 /opt/scali/examples/bin/hello -- hugin 2}

\subsection{2.3.2.3 Giving numeric values to mpimon}

Numeric values can be given as \texttt{mpimon} options in the following way:

\[ [<\text{prefix}>]<\text{numeric value}>[<\text{postfix}>]\]

with

\(<\text{prefix}>\) selects numeric base when interpreting the value

"0x" indicates hex-number (base = 16)

"0" indicates octal-number (base = 8)

if <prefix> is omitted, decimal-number (base = 10) is assumed

and

\(<\text{postfix}>\) selects a multiplication factor

"K": means a multiplication with 1024

"M": means a multiplication with 1024 \times 1024

Examples:

Input:

123

Value as interpreted by mpimon (in decimal):

123
2.3.2.4 How to give options to mpimon

There are three different ways to give options to mpimon. The most common way is to specify options on the command line invoking mpimon. An other way is by defining environment-variables and the last way of doing it is by defining options in configuration file(s).

2.3.2.4.1 Command line options

Options should follow mpimon-programname. See Appendix-A for a complete of options.

2.3.2.4.2 Environment-variable options

Setting a mpimon-option with environment means defining variables in the following manner:

SCAMPI_<uppercase-option> where SCAMPI_ is a fixed prefix followed by the option converted to uppercase.

Examples:

SCAMPI_CHANNEL_SIZE=64K means setting -channel_size to 64K

2.3.2.4.3 Configuration-files options

mpimon reads up to 3 different configuration-files when starting. First the systemwide configuration(/opt/scali/etc/ScaMPI.conf) is read. If the user has a file on his/hers home-directory, that file(~/ScaMPI.conf) is then read. Finally if there is a configuration file on current directory, this file(./ScaMPI.conf) is read. The files should contain one option per line given as for command line options.

2.3.2.5 Priority between different ways of defining options

Priority is defined as follows (from lowest to highest):

- System-wide configuration-file(/opt/scali/etc/ScaMPI.conf)
- Configuration-file on home-directory(~/ScaMPI.conf)
- Configuration-file on current directory(./ScaMPI.conf)
- Environment-variables
- Command line-options
2.3.2.6 Network options

**MPI Connect** is designed to handle several networks in one run. There are two types of networks, built-in standard-devices and DAT-devices. The DAT-devices are selected by giving the option “-networks <net-list>” to mpimon. <net-list> is a comma-separated list of DAT-adapters[4]. **MPI Connect** uses the list when setting up connections to other mpi-processes. It starts off with the first net in the list and sets up all possible connections with that net. If this fails the next in list is tried and so on until all connections are ok or all adapters in <net-list> are tried.

A connection has a set of parameters that affect usage of memory and behaviour. They can be customized with mpimon options. The syntax of these options is: `<option> <net>:<value>` where `<option>` is one of the following: -channel_entry_size,-channel_entry_count,-channel_size, -channel_inline_threshold, -eager_size, -eager_count, -transporter_size, -transporter_count. The `<net>` should be a DAT-adapter name. A default value for all nets can be given without the colon.

More information on each of the options can be found in next chapter.

2.3.3 mpirun - wrapper script

**mpirun** is a wrapper script for **mpimon**, giving MPICH style startup for **MPI Connect** applications. Instead of the **mpimon** syntax, where a list of pairs of node name and number of mpi-processes is used as startup specification, **mpirun** uses only the total number of mpi-processes.

Using **scaconftool**, **mpirun** attempts to generate a list of operational nodes. Note that only operational nodes are selected. If no operational node is available, an error message is printed and **mpirun** terminates. If **scaconftool** is not available, **mpirun** attempts to use the file /opt/scali/etc/ScaConf.nodeidmap for selecting the list of operational notes. In the generated list of nodes, **mpirun** evenly divides the mpi-processes among the nodes.

2.3.3.1 mpirun usage

```
mpirun <mpirunoptions> <mpimonoptions> <userprogram> [programoptions]
```

- `<mpirunoptions>` **mpirun** options
- `<mpimonoptions>` Options passed on to **mpimon**
- `<userprogram>` Name of application program to run.
- `<programoptions>` Program options passed on to the application program.

The following mpirunoptions exists:
- `-cpu <time>` Limit runtime to `<time>` minutes.
Section: 2.4 Useful tools

-np <count>  Total number of mpi-processes to be started, default 2.
-npn <count> Maximum number of mpi-processes pr. node, default np <count>/nodes.
-pbs Submit job to PBS queue system
-pbsparams <"params"> Specify PBS scasub parameters
-p4pg <pgfile> Use mpich compatible pgfile for program, mpi-process and node specification. pgfile entry: <nodename> <#procs> <progname> Program name given at command line is additionally started with one mpi-process at first node
-v Verbose.
-gdb Debug all mpi-processes using the GNU debugger gdb.
-maxtime <time> Limit runtime to <time> minutes.
-machinefile <filename> Take the list of possible nodes from <filename>
-noconftool Do not use scaconftool for generating nodelist.
-noarchfile Ignore the /opt/scali/etc/ScaConf.nodearchmap file (which describes each node).
-H <frontend> Specify nodename of front-end running the scaconf server.
-mstdin <proc> Distribute stdin to mpi-process(es). <proc>: all (default), none, or mpi-process number(s).
-part <part> Use nodes from partition <part>
-q Keep quiet, no mpimon printout.
-t test mode, no mpi program is started
<params> Parameters not recognized are passed on to mpimon.

2.4 Useful tools

Debugging with a separate debugging session for each mpi-process requires no parallel debugger. However, debugging several mpi-processes in separate debugging sessions, may become a time consuming and tedious task.
2.4.1 Debugging with a sequential debugger

MPI Connect applications can be debugged using a sequential debugger. By default, the GNU debugger gdb is invoked by mpimon. If another debugger is to be used, specify the debugger using the mpimon option `-debug <debugger>`.

To set debug-mode for one or more mpi-processes, specify the mpi-process(es) to debug using the mpimon option `-debug <select>`. In addition, note that the mpimon option `-display <display>` should be used to set the display for the xterm terminal emulator. An xterm terminal emulator, and one debugger, is started for each of the mpi-processes being debugged.

For example, to debug two mpi-processes with rank 0 and 1 using the default gdb debugger:

```
mpimon -display my_pc:0.0 -debug 0,1 <program & node spec>
```

Initially, for both mpi-process 0 and mpi-process 1, an xterm window is opened. Next, in the upper left hand corner of each xterm window, a message containing the application program’s run parameter(s) is displayed. Typically, the first line reads Run parameters: run <programoptions>. The information following the colon, i.e., run <programoptions> is needed by both the debugger and the MPI Connect application being debugged. Finally, one debugger is started for each session. In each debugger’s xterm window, do whatever debugging action that is appropriate before the mpi-process is started. Then, when ready to run the mpi-process, paste the run <programoptions> into the debugger to start running.

2.4.2 Useful built-in-tools for debugging

2.4.2.1 Using built-in segment protect violation handler

If you have an application that terminates with a SIGSEGV-signal it is often useful to be able to freeze the situation instead of exiting which is normal behaviour. The built-in SIGSEGV-handler can be made to do this by defining the environment-variable SCAMPI_INSTALL_SIGSEGV_HANDLER. Legal options are:

1. The handler dump all registers and start looping. Attaching with a debugger will then give the possibility to examine the situation giving the segment protect violation.
2. the handler dumps all registers but all processes will exit afterwards.

All other values will disable the installation of the handler.
2.4.2.2 Using built-in sanity-check of data.

If you are unsure of the quality of your network it is nice to be able to run **MPI Connect** with extra sanity-checking of data. This is a slower mode where we make a checksum of all data both at the sender and the receiver and compare. This mode is controlled by the environment-variable `SCAMPI_DATACHECK_ENABLE` and has the following options:

1. when error is detected report and loop
2. when error is detected report and exit

All other values disable the sanity check.

2.4.3 Profiling MPI Connect applications

When developing MPI programs it is difficult to do performance analysis. There are different tools available that can be useful in detecting / analysing performance bottlenecks:

- **MPI Connect** has built-in proprietary trace and profiling tools
- Freeware that uses the standard MPI profiling interface such as MPE which is developed by the mpich-implementors. It is part of a mpich-distribution and we have also made it available as a part of ScaMPI-distribution
- Commercial tools that collect information during run and postprocesses and presents afterwards. One example of this is Vampir from Pallas GmbH.

See [http://www.pallas.de](http://www.pallas.de) for more information.

The main difference between these tools is that the **MPI Connect** tools can be used with an existing binary while the other tools require reloading with extra libraries.

2.4.3.1 Using MPI Connect built-in trace

To use built-in trace-facility you need to set the mpimon-option `-trace "<options>"` specifying what options you want to apply. The following options can be specified: (`<...-list>` is a semicolon-separated list of Posix-regular-expressions.)

- `b` Trace beginning and end of each MPI_call
- `-s <seconds>` Start trace after `<seconds>` seconds
- `-S <seconds>` End trace after `<seconds>` seconds
- `-c <calls>` Start trace after `<calls>` MPI_calls
- `-C <calls>` End trace after `<calls>` MPI_calls
- `-m <mode>` Special modes for trace
  - `<mode> = “sync”`: Synchronize with MPI_Barrier before starting collective call
- `-p <selection>` Enable for process(es): 'n,m,o..' = (list) or 'n-m' = (range) or 'all'
- `-t <call-list>` Enable for MPI_calls in `<call-list>`.
Section: 2.4 Useful tools

**MPI_call** = 'MPI_call' | 'call'

- **-x** <call-list> Disable for MPI_calls in <call-list>.
- **-f** <format-list> Define format: 'timing', 'arguments', 'rate'
- **-v** Verbose
- **-h** Print this list of options

By default only one line is written per MPI-call. The "-b" option is useful when trying to pinpoint what MPI-call that are started but not completed (deadlocks). The "-s/-S/-c/-C" -options are nice to have if you have an application that runs ok for a longer period and then stop or if you want to have a closer look at some part of the execution of the application.

From time to time it is feasible to trace only one or a few of the processes. Specifying the "-p" options allows you to pick the processes you want to trace. All MPI-calls are enabled for tracing by default. If you want to look only on a few calls you could do that by specifying a "-t <call-list>" option or if you want to exclude some call you add a "-x <call-list>" option. The "-t" will disable all tracing and then enable those calls that matches the <call-list>. The matching is done using "regular-posix-expression"-syntax. "-x" will to the opposite; First enable all tracing and then disable those call matching <call-list>.

Examples:

"-t MPI_Irecv" : Trace only immediate recv (MPI_Irecv)
"-t isend;irecv;wait" : Trace only MPI_Isend, MPI_Irecv and MPI_Wait
"-t MPI_[b,r,s]*send" : Trace only send-calls (MPI_Send, MPI_Bsend, MPI_Rsend, MPI_Ssend)
"-t i[a-z]**" : Trace only calls beginning with MPI_I

As you can see calls can be specified with or without the "MPI_"-prefix. You can also use upper- or lower-case when specifying calls.

The default format of the output has the following parts:

\[
<\text{absRank}> : <\text{MPIcall}><\text{commName}>_<\text{rank}><\text{call-dependant-parameters}>
\]

- **<absRank>** is rank within MPI_COMM_WORLD
- **<MPIcall>** is name of MPI-call
- **<commName>** is name of communicator
- **<rank>** is rank within communicator used

This format can be extended by using the "-f"-option. Adding "-f arguments" will give some more information concerning length of messages. If "-f timing" is given you get some timing-info between the <absRank> and <MPIcall>-fields. It has the following format:

\[
+<\text{relSecs}> S <\text{eTime}>
\]

where
Section: 2.4 Useful tools

<relSecs> is elapsed time in seconds since returning to the application from MPI_Init
<eTime> is elapsed execution time for current call

"-f rate" will add some rate-information. The rate is calculated by dividing the number-of-bytes transferred by the elapsed time to execute the call. All parameters to -f can be abbreviated and can occur in any mix. The verbose-option(-v) will print information about which options you have selected.

Normally you will not get any error-messages concerning the options you have given. But if you add -verbose as command-line option to mpimon, errors will be printed.

Rate-measurements on Alpha-processors will be inaccurate due to low-resolution-timers.

2.4.3.2 Using MPI Connect built-in timing

To use the built-in timing you need to set the mpimon-option -timing "<options>" specifying what options you want to apply.

The following options can be specified:

(-<...-list> is a semicolon-separated list of Posix-regular-expressions.)

-s <seconds> Print for intervals of <seconds> seconds
-c <calls> Print for intervals of <calls> MPI_calls
-m <mode> Special mode for timing
  <mode> = "sync": Synchronize with MPI_Barrier before starting collective call
  <mode> = "all": All processes synchronized
  <mode> = "n,m,o..": Synchronize with MPI_Barrier between processes n,m,o...
-p <selection> Enable for process(es) 'n,m,o..' = (list) or 'n-m' = (range) or 'all'
-f <call-list> Print after MPI_calls in <call-list>: MPI_call = 'MPI_call' | 'call'
-v Verbose
-h Print this list of options

Printing of timing-information can be either at a fixed time-interval if you specify "-s <seconds>" or for a fixed number-of-calls-interval if you use "-c <calls>". You can also get output after specific MPI-calls if using "-f <call-list>"; See above for details how to write <call-list>.

The output has two parts; First a timing-part followed by a buffer-statistics-part. The first part has the following layout:

All lines starts with <rank>: where <rank>: is rank within MPI_COMM_WORLD. This part is included to facilitate separation of output (grep).
The rest of the format has the following fields:

\(<\text{MPIcall}>\)<\text{Dcalls}>\<\text{Dtime}>\<\text{Dfreq}> \<\text{Tcalls}>\<\text{Ttime}>\<\text{Tfreq}\>

where

\(<\text{MPIcall}>\) is name of MPI-call
\(<\text{Dcalls}>\) is number of calls to \(<\text{MPIcall}>\) since last printout
\(<\text{Dtime}>\) is sum of execution-time for calls to \(<\text{MPIcall}>\) since last printout
\(<\text{Dfreq}>\) is average time-per-call for calls to \(<\text{MPIcall}>\) since last printout
\(<\text{Tcalls}>\) is number of calls to \(<\text{MPIcall}>\)
\(<\text{Ttime}>\) is sum of execution-time for calls to \(<\text{MPIcall}>\)
\(<\text{Tfreq}>\) is average time-per-call for calls to \(<\text{MPIcall}>\)

After all detail-lines (one per MPI-call which has been called since last printout), there will be a line with the sum for all calls followed by a line giving the overhead introduced when obtaining the timing-measurements.

The second part containing the buffer-statistics has two types of lines; one for
receives and one for sends.
"Receive-lines" has the following fields:

\(<\text{Comm}>\)<\text{rank}> recv from \(<\text{from}>\)\(<\text{worldFrom}>\):<\text{commonFields}>\>

where

\(<\text{Comm}>\) is communicator being used
\(<\text{rank}>\) is rank within \(<\text{Comm}>\)
\(<\text{from}>\) is rank within \(<\text{Comm}>\)
\(<\text{worldFrom}>\) is rank within MPI_COMM_WORLD

"Send-lines" has the following fields:

\(<\text{Comm}>\)<\text{rank}> send to \(<\text{to}>\)\(<\text{worldTo}>\):<\text{commonFields}>\>

where

\(<\text{Comm}>\) is communicator being used
\(<\text{rank}>\) is rank within \(<\text{Comm}>\)
\(<\text{to}>\) is rank within \(<\text{Comm}>\)
\(<\text{worldTo}>\) is rank within MPI_COMM_WORLD

The \(<\text{commonFields}>\) are as follows:

\(!\text{count}>!\text{avrLen}>!\text{zroLen}>!\text{inline}>!\text{eager}>!\text{transporter}>!\>

where

\(<\text{count}>\) is number of sends/receives
\(<\text{avrLen}>\) is average length of messages in bytes
\(<\text{zroLen}>\) is number of messages sent/received using zero-bytes-mechanism
\(<\text{inline}>\) is number of messages sent/received using inline-mechanism
Section: 2.4 Useful tools

- `<eager>` is number of messages sent/received using eagerbuffer-mechanism
- `<transporter>` is number of messages sent/received using transporter-mechanism

More details on the different mechanisms can be found in the description chapter.

Timing-measurements on Alpha-processors will be inaccurate due to low-resolution-timers.

2.4.3.3 Using MPI Connect built-in cpu-usage

To use built-in cpu-usage-timing you need to set the environment variable SCAMPI_CPU_USAGE.

The information displayed is collected with the system-call "times"; see manpages for more information.

The output has two different blocks. The first block contains cpu-usage by the submonitors on the different nodes. One line is printed for each submonitor followed by a sum-line and an average-line. The second block consists of one line per process followed by a sum-line and an average-line.

2.4.4 Profiling with ScaMPE

The ScaMPE libraries are adapted versions of the MPE libraries from MPICH. An executable program linked with one of the ScaMPE libraries `libtmpi`, `liblmpi` or `libampi` collects performance data during runtime. Normally, the libraries are installed in the directory `/opt/scali/contrib/lib`, and the upshot tool, described below, is installed in `/opt/scali/contrib/bin`.

The main components of ScaMPE are:
- A set of routines for creating logfiles for examination by the visualization tool upshot.
- Trace or real time animation of MPI calls.
- A shared display parallel X graphics library.

2.4.4.1 Linking an MPI Connect application

Profiling using one of the ScaMPE libraries is achieved by linking with the appropriate ScaMPE library before the standard MPI Connect library `libmpi`.

- Trace MPI calls - library `libtmpi`
  To trace all MPI calls, apply `-ltmpi`. Each MPI call is preceded by a line that contains the rank in MPI_COMM_WORLD of the calling process, and followed by another line indicating that the call has completed. Most send
Section: 2.4 Useful tools

and receive routines also indicate the values of count, tag, and partner (destination for sends, source for receives). Output is to standard output stdout.

- Generate log file - library liblmpi
  To generate an upshot style log file of all MPI calls, apply -llmpi. When the application is about to finish, an information message is printed to stdout, and the trace data is written to a log file for post-processing. The name of the log file, with suffix .alog, is created based on the argument provided in argv[0]. Note that when an application program is started, MPI Connect is modifying argv[0], as described in section 2.3. However, the log file name is always created as executablename-<ScaMPI postfix>.alog. For example, if the program being profiled is sendrecv, the generated log file is sendrecv-<ScaMPI postfix>.alog.

- Real time animation - library libampi
  To produce a real-time animation of the program, apply -lampi -lmpe -lm -lX11. Note that this requires the MPE graphics in libmpe, and that X11 Window System operations are used. To link the X11 libraries (libX11), it may be necessary to provide a specific path for the libraries. In addition, note that to resolve some mathematics references used, the standard library libm must be included in the link command line. For a description of the MPE graphic routines, see the MPICH documentation [11].

Notes for Fortran users
For a Fortran program, it is necessary to include the Fortran wrapper library libfmpi ahead of the profiling libraries. This allows C routines to be used for implementing the profiling libraries for use by both C and Fortran programs. For example, to generate an upshot style log file in a Fortran program, the libraries are included in the order -lfmpi -llmpi -lm.

2.4.4.2 Examine the generated log file - upshot
To examine a log file generated using liblmpi, the parallel program visualization tool upshot can be used to analyse the program performance. Note that upshot uses the environment variable $DISPLAY to select the display to use.

Start the visualization tool:

/opt/scali/contrib/bin/upshot
When started, browse and select the appropriate log file to be analysed. For more information, see the document named README_UPSHOT in the directory /opt/scali/contrib/doc/ScaMPI.

If upshot is not available, any other visualization tool, e.g., nupshot, that understands the log file format can be used instead. For more information, see the MPICH documentation.

2.5 An example program

When the ScaMPItst package has been installed, the source code and the executable code, for both the hello-world example program and a number of test programs, are located under the /opt/scali/examples/src and the /opt/scali/examples/bin directories. A description of each program in the package can be found in the README file, located in the /opt/scali/doc/ScaMPItst directory.

As examples, the MPI program named hello-world is used. It exists as a C program in the file hello-world.c, and as a Fortran program in the file hello-world.f. They are compiled and linked using GNU compilers. Before compilation, it is assumed that the BASH shell environment variable has been properly defined. In addition, MPI Connect must have been installed and function correctly.

2.5.1 Hello-world.c - source in C

```c
#include <stdio.h>
#include "mpi.h"

void main(int argc, char** argv)
{
    int rank;
    int size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello-world, I'm rank %d; Size is %d\n", rank, size);
    MPI_Finalize();
}
```

Scali MPI Connect Users Guide Version 1.1
2.5.2 Hello-world.f - source in Fortran

program hello_world

implicit none
include 'mpif.h'
integer rank,size,ierr

call mpi_init(ierr);
call mpi_comm_rank(MPI_COMM_WORLD,rank,ierr);
call mpi_comm_size(MPI_COMM_WORLD,size,ierr);
write (*,'(A,I3,A,I3)') "Hello-world, I'm rank ",rank,
& "; Size is ",size
call mpi_finalize(ierr);
end

2.5.3 Compiling

% gcc -c -D_REENTRANT -I$MPI_HOME/include hello-world.c
% g77 -c -D_REENTRANT -I$MPI_HOME/include hello-world.f

2.5.4 Linking

% gcc hello-world.o -L$MPI_HOME/lib -lmpi -o hello-world
% g77 hello-world.o -L$MPI_HOME/lib -lfmpi -lmpi -o hello-world

2.5.5 Running

Start the hello-world program on the 3 nodes named nodeA, nodeB and nodeC.
% mpimon hello-world -- nodeA 1 nodeB 1 nodeC 1

The hello-world program should produce the following output:

Hello-world, I'm rank 0; Size is 3
Hello-world, I'm rank 1; Size is 3
Hello-world, I'm rank 2; Size is 3
2.6 MPI test programs

The ScaMPItst package contains a collection of MPI test programs for MPI Connect. The following sections give a brief description of some of the test programs, which can be used to measure basic MPI performance. To re-compile any of the test programs, you may use the included Makefile found in the appropriate /opt/scali/examples/src directory.

2.6.1 Producer - a producer-consumer MPI test program

Producer is a simple producer-consumer program. Mpi-processes with rank 0, 1, 2,..., n/2-1 send data while mpi-process n/2, n/2+1,...,n-1 receive data. Mpi-process 0 will send to mpi-process n-1, mpi-process 1 will send to mpi-process n-2, and so on.

The producer program parameters are:
- `-l i`  `i` is the loop count.
- `-n j`  `j` is the number of bytes to transfer for each send operation.

As a first test, run producer between any pair of two nodes, nodeX and nodeY:

```
% mpimon producer -l 1 -n 1024 -- nodeX nodeY
```

A single mpi-process is started on each node, and a single message of size 1024 bytes are transferred from the mpi-process on nodeX to the mpi-process on nodeY. The program should return TEST COMPLETE.

Repeat the test for all pairs of nodes. N is the number of nodes (must be an even number for this test).

```
% mpimon producer -l 1 -n 1024 -- <node1> <node2>...<nodeN>
```

The program should return TEST COMPLETE.

2.6.2 Bandwidth - a bandwidth MPI test program

Bandwidth is a program to measure bandwidth for various message sizes between two mpi-processes. First one-way bandwidth and the latency for a zero byte message are measured, then the ping-pong (two-way) bandwidth and latency are measured.

Measure the bandwidth between any pair of nodes, nodeX and nodeY, by running:

```
% mpimon bandwidth -- nodeX nodeY
```
### 2.6.3 Bidirect - a bidirectional MPI test program.

**Bidirect** tests uni- and bi-directional traffic between a given number of nodes.

The program may be run between two nodes, nodeX and nodeY, using the `run_bidirect` script as:

```
% run_bidirect nodeX nodeY
```

or between a given set of nodes, nodeX nodeY... nodeZ, using another script as:

```
% run_permutated_bidirect nodeX nodeY... nodeZ
```

The `run_permutated_bidirect` script will test uni- and bi-directional traffic between all permutations of node combinations.
Chapter 3

Description of ScaMPI

3.1 General description

ScaMPI consists of libraries to be linked and loaded with the user application program(s) and a set of executables which control the start-up and execution of the user application program(s).

3.1.1 ScaMPI libraries

libmpi  
Standard library containing the C API.
libfmpi  
Library containing the Fortran API wrappers.

3.1.2 ScaMPI executables

A number of executable programs are included in ScaMPI.

3.1.2.1 mpimon - monitor program

mpimon is a monitor program which is the user’s interface for running the application program.

3.1.2.2 mpisubmon - submonitor program

mpisubmon is a submonitor program which controls the execution of application programs. One submonitor program is started on each node per run.

3.1.2.3 mpiboot - bootstrap program

mpiboot is a bootstrap program used when running in manual-/debug-mode.

3.1.2.4 mpid - daemon program

mpid is a daemon program running on all nodes that can run ScaMPI. mpid is used for starting the mpisubmon programs (to avoid using Unix facilities like the remote shell rsh). mpid is started automatically when a node boots, and must run at all times.

3.2 Starting ScaMPI application programs

ScaMPI uses socket communication for control purposes. Schematically, start-up of application programs in a Scali System is performed as described in the following sections.
3.2.1 Application start-up - phase 1

- **Parameter control.**
  
  *mpimon* does as much control of the specified options and parameters as possible. The userprogram names are checked for validity, and the nodes are, using sockets, contacted to ensure they are responding and that *mpid* is running.

- **Connecting to nodes.**
  
  *mpimon* establishes a connection to the *mpid* daemon on each node specified, and transfers basic information to enable the daemon to start the submonitor *mpisubmon*.

![Figure 1-1: Application start-up - phase 1](image)

3.2.2 Application start-up - phase 2

- **Starting submonitors.**
  
  On each node, *mpid* starts the submonitor *mpisubmon*.

- **Transferring control information.**
  
  Each submonitor establishes a connection to *mpimon*. Control information are exchanged between each *mpisubmon* and *mpimon* to enable *mpisubmon* to start the specified userprograms (mpi-processes).

- **Allocating communication resources**
  
  Each process allocates communication resources.
3.2.3 Application start-up - phase 3

- **Starting mpi-processes.**
  On each node, `mpisubmon` starts all the mpi-processes to be executed. Processes start and enter `MPI_Init()`.

- **Mpi-processes synchronize.**
  Upon receipt of all control information, the processes will via the local `mpisubmon` inform `mpimon` that they are ready to run. When all processes are ready, `mpimon` will return a 'start running' message to all the processes.

- **MPI-processes return from `MPI_Init()` and start to run.**
  The user program(s) takes control.
3.3 Stopping ScaMPI application programs

Termination of application programs in a Scali System are performed as outlined below.

- **Mpi-processes enters MPI_Finalize().**
  
  Each process signals, via its local mbisubmon, to mpimon that it has entered **MPI_Finalize()**, and it is now waiting.

- **Mpi-processes synchronize**
  
  Processes wait for an "all stopped message" from **mpimon**. The message is transmitted via **mpisubmon** when all processes are waiting in **MPI_Finalize()**.

- **Mpi-processes leave MPI_Finalize().**
  
  Processes releases resources and terminates, each **mpisubmon** exits, and finally **mpimon** terminates.
3.4 Communication protocol on standard-devices

The standard devices uses a simplified protocol based on serial transfers. This can be visualized as data being written into one end of a pipe and read from the other. Messages arriving out-of-order are buffered by the reader. The names of these standard devices are: **smp** for intra-node-communication and **tcp** for node-to-node-communication. The size of the buffer inside the pipe can be adjusted by setting the following environment variables:

- `SCAFUN_TCP_TXBUFSZ` Set size of transmit-buffer.
- `SCAFUN_TCP_RXBUFSZ` Set size of receive-buffer.
- `SCAFUN_SMP_BUFSZ` Set size of buffer for intranode-communication.

3.5 Communication protocols on DAT-devices

In ScaMPI, the communication protocol (*inlining, eagerbuffering, transporter*) used to transfer data between a sender and a receiver depends on the size of the message to transmit, see figure below.

![Figure 1-4: Thresholds for different communication protocols](image)

The various communication protocols used, are briefly outlined in the following sections.
3.5 Communication protocols on DAT-devices

3.5.1 Inlining protocol
The inlining protocol is used when small messages are to be transferred.

When the inlining protocol is used, the application’s data is included in the message header. The inlining protocol utilizes one or more channel ringbuffer entries. The actual threshold for the inlining protocol can be set as described in section 3.6.2.

The inlining protocol is selected when:
\[ 0 \leq \text{message size} \leq \text{channel_inline_threshold}. \]

3.5.2 Eagerbuffering protocol
The eagerbuffering protocol is used when medium size messages are to be transferred.

Figure 1-5: Inlining protocol

Figure 1-6: Eagerbuffering protocol
The protocol uses a scheme where the buffer resources, being allocated by the sender, are released by the receiver, without any explicit communication between the two communicating partners.

The eagerbuffering protocol utilizes one channel ringbuffer entry for the message header, and one eagerbuffer for the application data being sent.

The eagerbuffering protocol is selected when:
channel Inline threshold < message size <= eager size.

### 3.5.3 Transporter protocol

The transporter protocol is used when large messages are to be transferred.

*Figure 1-7: Transporter protocol*
Initially (step 1), the protocol only transmits the message header. Once the receiver is ready to accept data (step 2), the sender is informed. Finally (step 3), the application's data is transferred from the sender to the recipient in the transporter ringbuffer.

The transporter protocol utilizes one channel ringbuffer entry for the message header, and transporter buffers for the application data being sent. The transporter protocol provides for fragmentation and reassembly of large messages, if necessary, for messages whose size is larger than the size of the transporter ringbuffer-entry (transporter_size).

The transporter protocol is selected when:
message size > eager_size.

### 3.5.4 Zerocopy protocol

The zerocopy protocol is special case of the transporter protocol. It has the same steps as a transporter except that data is written directly into the receivers buffer instead of being buffered in the transporter-ringbuffer.

The zerocopy protocol is selected when:
zerocopy_min < message size < zerocopy_max.

### 3.6 Communication resources on DAT-devices

All resources (buffers) used by ScaMPI reside in shared memory. To get a list of the resource settings, pass the `--verbose` option to `mpimon`.

ScaMPI operates on a buffer pool. The pool is divided into equally sized parts called chunks. ScaMPI uses one chunk per connection to other processes. The `mpimon` option "pool_size" limits the total size of the pool and the "chunk_size" limits the block of memory that can be allocated for a single connection.

To set the pool size and the chunk size, specify:

- **pool_size <size>** to set the buffer pool size
- **chunk_size <size>** to set the chunk size

Sections 3.6.2 - 3.6.4 outlines the various types of resources (channel, eagerbuffer, transporter) being used, and lists the `mpimon` options used to enforce specific buffering.
Note: It is normally not necessary to set buffer parameters since ScaMPI automatically adjusts them depending on pool_size / chunk_size and number of processes.

3.6.1 Connection

![Diagram of connections between processes]

Figure 1-8: Connections
A connection is a bidirectional data-transfer-mechanism. It has the following components:

![Connection building blocks diagram](image-url)

**Figure 1-9**: Connection building blocks
3.6.2 Channel buffer

The ringbuffer is divided into equally sized entries. The size varies for different architectures and networks; see “ScaMPI release notes” for details. An entry in the ringbuffer, which is used to hold the information forming the message envelope, is reserved each time a message is being sent, and is utilized by both the inline protocol, the eagerbuffering protocol, and the transporter protocol. In addition, one or more entries are utilized by the inline protocol for application data being transmitted.

To force the channel resource definitions, use `mpimon` and specify:

- `-channel_size <size>` to set the ringbuffer size (in bytes)
- `-channel_entry_size <size>` to set the size of each entry in the ringbuffer
- `-channel_entry_count <count>` to set the number of entries in the ringbuffer

To set the channel threshold definitions, use `mpimon` and specify:

- `-channel_inline_threshold <size>` to set threshold for inlining

Figure 1-10: Channel resource
3.6.3 Eagerbuffer buffer

An *eagerbuffer* buffer is allocated when medium size messages are to be transferred, and is utilized by the *eagerbuffering* protocol.

To change the *eagerbuffer* resource definitions, use `mpimon` and specify:

- `-eager_size <size>` to set the buffer size (in bytes)
- `-eager_count <count>` to set number of buffers

![Diagram of Eagerbuffer buffer](image-url)
### 3.7 Communication media

ScalMPI is designed to take advantage of the DAT-standard, "Direct Access Transport" [4], when handling different communication media. Choosing between different media is done by giving a comma-separated list of DAT-adapters to mpimon specifying 

```
-net <list>
```

The `<list>` is interpreted from the diagram provided in Figure 1-12:

![Diagram of Transporter buffer](image)

**Figure 1-12: Transporter buffer**

A transporter buffer is allocated when large messages are to be transferred, and is utilized by the transporter protocol. The buffer consists of equally sized entries arranged as a ringbuffer.

To change the transporter resource definitions, use `mpimon` and specify:

- `transporter_size <size>` to set the buffer entry size (in bytes)
- `transporter_count <count>` to set number of entries in buffer
left-to-right when trying to establish connections; **mpimon** will take the first adapter in the list and try to make connections with it. If this fails, next in list is tried until all connections are made.

### 3.7.1 Using Scali supplied DAT-adapters

Assuming the packages containing DAT-adapters are installed correctly, you will find a utility "scanet" which can be useful. It will print a list of all adapters available on your cluster. Type "scanet -h" for more information on options.

### 3.7.2 Using third-party/own DAT-adapters

ScaMPI follows the specification concerning how to connect/use a uDAPL-implementation. This means that it we assume to find a library named "libdat.so" pointed to by the environment-variable LD_LIBRARY_PATH. After opening the library we assume to find 3 symbols, dat_ia_open, dat_ia_close and dat_registry_list_providers. A call to dat_ia_open should setup the jump-table needed to use the library. 

Note that ScaMPI is licensed software and you will need a "third-party-dat-license" to use a non-Scali dat-adapter.

If you are running a mix of Scali and non-Scali-adapters you must append line(s) in the dat-configuration-file (/opt/scali/etc/dat.conf) describing the non-Scali-adapters. The syntax follows "static registry"-specification from uDAPL[4].
Chapter 4  Tips & Tricks for ScaMPI

This chapter is the place to start when something seems to go wrong running your ScaMPI programs. If you have any problems with ScaMPI, first check the (not yet complete) list of common errors and their solutions. An updated list of ScaMPI Frequently Asked Questions are posted in the Support section at http://www.scali.com. If you cannot find a solution to the problem(s), please read this chapter before contacting support@scali.com.

Currently, the following sections are by no means complete. Problems reported to Scali will eventually be included in appropriate sections. Thus, please send your relevant remarks by e-mail to support@scali.com.

4.1 Application program notes

4.1.1 MPI_Probe() and MPI_Recv()
During development and test of ScaMPI, we have run into several application programs with the following code sequence:

```c
while (...) {
    MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, comm, sts);
    if (sts->MPI_TAG == SOME_VALUE) {
        MPI_Recv(buf, cnt, dtype, MPI_ANY_SOURCE,
                  MPI_ANY_TAG, comm, sts);
        doStuff();
    }
    doOtherStuff();
}
```

For MPI implementations that have one, and only one, receive-queue for all senders, the program’s code sequence works ok. However, the code will not work as expected with ScaMPI. ScaMPI utilizes one receive-queue per sender (inside each mpi-process). Thus, a message from one sender can bypass the message from another sender. In the time-gap between the completion of MPI_Probe() and before MPI_Recv() matches a message, another new message from a different mpi-process could arrive, i.e., it is not certain that the message found by MPI_Probe() is identical to one that MPI_Recv() matches.
To make the program work as expected, the code sequence should be corrected to:

```c
while (...) {
    MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, comm, sts);
    if (sts->MPI_TAG == SOME_VALUE) {
        MPI_Recv(buf, cnt, dtype, sts->MPI_SOURCE, 
                  sts->MPI_TAG, comm, sts);
        doStuff();
    }
    doOtherStuff();
}
```

### 4.1.2 Unsafe MPI programs

Because of different buffering behaviour, some programs may run with MPICH, but **not** with ScaMPI. Unsafe MPI programs may require resources that are not always guaranteed by ScaMPI, and deadlock might occur (since ScaMPI use spinlocks, these might seem to be livelocks). If you want to know more about how to write portable MPI programs, see for example[2].

A typical example that will **not** work with ScaMPI (for long messages):

```c
while (...) {
    MPI_Send(buf, cnt, dtype, partner, tag, comm);
    MPI_Recv(buf, cnt, dtype, MPI_ANY_SOURCE, 
              MPI_ANY_TAG, comm, sts);
    doStuff();
}
```

To get this example to work with ScaMPI, the `MPI_Send()` must either be replaced by using `MPI_Isend()` and `MPI_Wait()`, or the whole construction should be replaced using `MPI_Sendrecv()` or `MPI_Sendrecv_replace()`.

### 4.2 Namespace pollution

The ScaMPI library, being written in C++, have all its class names prefixed with `MPI_`. Depending on the compiler used, the user may run into problems if he/she has C++ code using the same prefix `MPI_`. In addition, there exist a few global variables that could cause problems. All these functions and variables are listed in the include files `mpi.h` and `mpif.h`. Normally, these files are installed in `/opt/scali/include`.

---

*Scali MPI Connect Users Guide Version 1.1*
Due to the fact that ScaMPI doesn’t have fixed its OS routines to specific libraries, it will be good programming practise to avoid using OS functions as application function names. Naming routines or global variables as send, recv, open, close, yield, internal_error, failure, service or other OS reserved names may result in an unpredictable and undesirable behaviour.

4.3 Error and warning messages

4.3.1 User interface errors and warnings
User interface errors are problems with the environment setup causing difficulties for mpimon when starting a ScaMPI program. mpimon will not start before the environment is properly defined. These problems are usually easy to fix, by giving mpimon the correct location of some executable. The error message provides a straight forward indication of what to do. Thus, only particularly troublesome user interface errors will be listed here.

Using the -verbose option enables mpimon to print more warnings.

4.3.2 Fatal errors
Upon a fatal error, ScaMPI prints an error message before calling MPI_Abort() to shut down all mpi-processes.

4.4 When things don’t work - troubleshooting
This section is meant as a starting point to help debugging. The main focus is on locating and repairing faulty hardware and software setup, but can also be helpful in getting started after installing a new system. For a description of the Scali Universe GUI, see the Scali System Guide.

4.4.1 Standard input and ScaMPI
The -stdin option specifies which mpi-process rank should receive the input.
You can in fact send stdin to all the mpi-processes with the all argument, but this requires that all mpi-processes read the exact same amount of input. The most common way of doing it is to send all data on stdin to rank 0:
mpimon -stdin 0 myprogram -- node1 node2... < input_file
Note that default for -stdin is none.

4.4.2 Why doesn’t my program start to run?
? mpimon: command not found.
include /opt/scali/bin in the PATH environment variable.

mpimon can’t find mpisubmon.
 Set MPI_HOME=/opt/scali or use the -execpath option.

The application has problems loading libraries (libsca*).
 Update the LD_LIBRARY_PATH to include /opt/scali/lib.

Incompatible mpi versions.
mpid, mpimon, mpisubmon and the libraries all have version variables that are
checked at start-up.
 Set the environment variable MPI_HOME correctly
 Restart mpid because a new version of ScaMPI is installed without restart of
mpid
 Reinstall ScaMPI because a new version of ScaMPI was not cleanly installed
 on all nodes.

Set workingdirectory failed
 ScaMPI assumes homogenous file-structure. If you start mpimon from a
directory that is not available on all nodes you must set
SCAMPI_WORKING_DIRECTORY to point to a directory that are available on
all nodes.

ScaMPI uses wrong interface for tcp-ip on frontend with more than
one interface
 Set SCAMPI_NODENAME to hostname of correct interface.

MPI_Wtime gives strange values
 ScaMPI uses a hardware-supported high precision timer for MPI_Wtime.
 This timer can be disabled by SCAMPI_DISABLE_HPT=1

4.4.3 Why doesn’t mpid start
mpid opens a socket and assigns a predefined mpid port number, see
/etc/services, to the end point. If mpid is terminated abnormally, the mpid port
number cannot be re-used until a system defined timer has expired.
 Use netstat -a | grep mpid to observe when the socket is released. When
the socket is released, restart mpid again.

4.4.4 SCI interconnect problems

4.4.4.1 SCI Routing

Program terminates with an ICMS_NO_RESPONSE error
message
This happens when one or more mpi-processes are unable to create a remote memory mapping to another node within a (long) period of time.

- Check if all relevant nodes are alive by issuing any command with scash, e.g., /opt/scali/bin/scash -p nodename.
- Check if SCI network routing is properly set with /opt/scali/sbin/scaconf tool (command: sciping OK), or use the Scali Universe GUI.

4.4.4.2 Bad clean up

- A previous ScaMPI run has not terminated properly.
  - Check for mpi-processes on the nodes using /opt/scali/bin/scaps.
  - Use /opt/scali/sbin/scidle
  - Use /opt/scali/bin/scash to check for leftover shared memory segments on all nodes (ipcs for Solaris and Linux).

4.4.4.3 Space overflow

- The application have required too much SCI or shared memory resources.
  - Your mpimon pool-size specifications are too large.

4.4.5 Why does my program terminate abnormally?

4.4.5.1 Core dump

- The application core dumps.
  - Use a debugger to locate the point of violation. The application may need to be recompiled to include symbolic debug information (-g for most compilers).
  - Define SCAMPI_INSTALL_SIGSEGV_HANDLER=1 and attach to the failing process with debugger.

4.4.5.2 SCI interconnect failures

- The program terminates with an ICMS_* message
  - An SCI problem has occurred, find out more using the SCI diagnostics helper: /opt/scali/bin/sciemsg <error-code>. Reloading of SCI drivers and rerouting your system may be necessary. Contact your local System Administrator if assistance is needed. The interconnect diagnostic in the Scali Universe GUI and the SCI documentation in the Scali System Guide may help you locate the problem. Problems and fixes will be included in the FAQ on http://www.scali.com. If there is a SCI problem needing attention, please contact support@scali.com.
4.4.5.3 General problems

? Are you reasonable certain that your algorithms are MPI safe?
☑ Check if every send has a matching receive.

? The program just hangs
☑ If the application has a large degree of asynchronicity, try to increase the channel-size. Further information is available in the ‘How do I control SCI and local shared memory usage?’ section. Are you really sure that your algorithms are MPI safe?

? The program terminates without an error message
☑ Investigate the core file, or rerun the program in a debugger.

4.4.6 How do I control shared memory usage?

? Adjusting ScamPI buffer sizes
Note that forcing size parameters to mpimon is usually not necessary. This is only a means of optimising ScamPI to a particular application, based on knowledge of communication patterns. For unsafe MPI programs it may be required to adjust buffering to allow the program to complete.

? How do I calculate shared memory usage?
The buffer space required by a communication channel is approximately:
chunk-size = (2 * channel-entry-size * channel-entry-count)
  + (transporter-size * transporter-count)
  + (eager-size * eager-count)
  + 4096 (give-or-take-a-few-bytes)
Total-usage = chunk-size * no-of-processes

4.4.6.1 Automatic buffer management
The pool-size is a limit for the total amount of shared memory.
The automatic buffer size computations is based on a full connectivity, i.e., all communicating with all others. Given a total pool of memory dedicated to communication, each communication channel will be restricted to use a partition of only (P = number of processes):
chunk = inter_pool_size / P
The automatic approach is to downsize all buffers associated with a communication channel until it fits in its part of the pool. The automatic chunk size is calculated to wrap a complete communication channel.
4.5 How to optimize MPI performance

There is no universal recipe for getting good performance out of a message passing program. Here are some do’s and don’ts for ScaMPI.

4.5.1 Performance analysis.
Learn about the performance behaviour of your MPI application on a Scali System by using a performance analysis tool. The freely available ScaMPE profiling library may be used with ScaMPI. For more information, please see section.

4.5.2 Using MPI_Isend(), MPI_Irecv().
If communication and calculations does not overlap, using immediate calls, e.g., `MPI_Isend()` and `MPI_Irecv()`, are usually performance ineffective.

4.5.3 Using MPI_Bsend().
Using buffered send, e.g., `MPI_Bsend()`, usually degrade performance significantly compared to their unbuffered relatives.

4.5.4 Avoid starving mpi-processes - fairness.
MPI programs may, if not special care is taken, be unfair and may starve mpi-processes, e.g., by using `MPI_Waitany()` as illustrated for a client-server application in example 3.15 & 3.16 in the MPI 1.1 standard[1]. Fairness can be enforced, e.g., by use of several tags or separate communicators.

4.5.5 Using processor-power to poll.
ScaMPI is implemented using poll when waiting for communication to terminate. This is efficient when this period is short or if you don’t have anything else to use the processor power for. In threaded application with irregular communication patterns you probably have other threads that could make use of the processor. In this case performance may increase if you enable the backoff-polling-strategy built into ScaMPI. It functions like this: After waiting a short period (idle time) we start backing off using system call nanosleep to release processor. The nanosleep period starts at a minimum and it doubles for each call until it reaches a maximum. It is controlled by a set of environment-variables:

- `SCAMPI_BACKOFF_ENABLE` turns the mechanism on
- `SCAMPI_BACKOFF_IDLE=n` defines idle-period to n ms
  - Default 20 ms
- `SCAMPI_BACKOFF_MIN=n` defines minimum backoff-time in ms

---

Scali MPI Connect Users Guide Version 1.1
4.5 How to optimize MPI performance

SCAMPI_BACKOFF_MAX=n
Default 10 ms
defines maximum backoff-time in ms
Default 100 ms

4.5.6 Communication buffer adaption
If the communication behaviour of the application is known, explicitly giving
buffersize settings to mpimon, to match the requirement of the application, will
in most cases improve performance.
Example: Application sending only 900 bytes messages.
② Set channel-inline-threshold 964 (64 added for alignment) and increase
the channel-size significantly (32-128 k).
② Setting eager-size 1k and eager-count high (16 or more).
Note: If all messages can be buffered, the transporter-\{size, count\} can be set
to low values to reduce shared memory consumption.

4.5.7 Reorder network traffic to avoid conflicts
Many-to-one communication may introduce bottlenecks.
Zero byte messages are low-cost. In a many-to-one communication,
performance may improve if the receiver sends ready-to-receive tokens (in the
shape of a zero-byte message) to the mpi-process wanting to send data.
4.6 Benchmarking

Benchmarking is that part of performance evaluation that deals with the measurement and analysis of computer performance using various kinds of test programs. Benchmark figures should always be handled with special care when compared to similar results.

4.6.1 How to get expected performance

- **Improving performance for short runs.**
  By default, communication buffers are allocated when requested the first time. To eliminate this startup time from your measurement either run a warm-up phase before doing the actual measurement or use the parameter `-init_comm_world` to `mpimon` to allocate communication buffers between all pairs of mpi-processes.

- **Caching the application program on the nodes.**
  For benchmarks with short execution time, total execution time may be reduced when running it repetitive. For large configurations, copying the application to the local file system on each node will reduce startup latency and improve disc bandwidth.

- **The first iteration is (very) slow.**
  The mpi-processes in an application are not started simultaneously. Inserting an `MPI_Barrier()` before the timing loop will eliminate this. To reduce setup time after `MPI_Init()`, specify the parameter `-init_comm_world` to `mpimon`.

4.6.2 Memory consumption increase after warm-up

Remember that group operations (`MPI_Comm_{create, dup, split}`) may involve creating new communication buffers. If this is a problem, decrease the **chunk-size** as described in section 4.4.
Chapter 5  Support

5.1 Feedback
Scali appreciates any suggestions to improve both this Scali Library User’s Guide and the software described herein. Please send your comments by e-mail to support@scali.com.

The user of parallel tools software using ScaMPI on a Scali System, is encouraged to provide feedback to the National HPCC Software Exchange (NHSE) - Parallel Tools Library [10]. The Parallel Tools Library provides information about parallel system software and tools, and, in addition, it provides for communication between the software author and the user.

5.2 Scali mailing lists
We have developed mailing lists being available on the Internet. For instructions on how to subscribe to a mailing list (e.g., scali-announce or scali-user), please check out the Mailing Lists section at http://www.scali.com/.

5.3 ScaMPI FAQ
The ScaMPI Frequently Asked Questions are posted on our Web site at http://www.scali.com. Please check out the ScaMPI FAQ section at http://www.scali.com/. In addition, the FAQ is, when ScaMPI has been installed, available as a text file in /opt/scali/doc/ScaMPI/FAQ.

5.4 ScaMPI release documents
When ScaMPI has been installed, a number of small documents like FAQ, RELEASE NOTES, README, SUPPORT, LICENSE_TERMS, INSTALL are available as text files in the /opt/scali/doc/ScaMPI directory.
5.5 Problem reports

Problem reports should, whenever possible, include both a description of the problem, the software versions, the computer architecture, an example, and a record of the sequence of events causing the problem. Any information that you can include about what triggered the error will be helpful. The report should be sent by e-mail to support@scali.com.

5.6 Platforms supported

ScaMPI is available for a number of platforms. For up-to-date information, please check out the ScaMPI section at http://www.scali.com/. For additional information, please don’t hesitate to contact Scali at sales@scali.com.

5.7 Licensing

ScaMPI is licensed using Scali license manager system. In order to run ScaMPI a valid demo or a permanent license must be obtained. Customers with valid software maintenance contracts with Scali may request this directly from license@scali.com All other requests, including DEMO licenses should be directed to: sales@scali.com
Chapter 6  Related documentation

   The Message Passing Interface Forum, Version 1.1, June 12, 1995,  

[2]  MPI: The complete Reference: Volume 1, The MPI Core  
   Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker,  

   William Grop, Steven Huss-Lederman, Ewing Lusk, Bill Nitzberg, W.  

   http://www.datcollaborative.org


[6]  ScaMPI Data sheet  

[7]  Scali Free Tools  

[8]  Review of Performance Analysis Tools for MPI Parallel Programs  
   UTK Computer Science Department, http://www.cs.utk.edu/~browne/  
   perftools-review/.

[9]  Debugging Tools and Standards  
   hpdf/.

[10]  Parallel Systems Software and Tools  

    The MPICH home page, http://www.mcs.anl.gov/mpi/mpich/  
    index.html.

[12]  MPI Test Suites freely available  
    Argone National Laboratory, http://www-unix.mcs.anl.gov/mpi/mpi-  
    test/tsuite.html
Appendix A  Mpimon options

--
-automatic <selection>
-backoff_enable <selection>
-channel_entry_count <count>
-channel_entry_size <size>
-channel_inline_threshold <size>
-channel_size <size>
-chunk_size <size>
-debug <selection>
-debugger <debugger>
-disable-timeout
-display <display>
-eager_count <count>
-eager_factor <factor>
-eager_size <size>
-eager_threshold <size>
-execpath <execpath>
-help
-home <directory>
-herit_limits
-manual <selection>
-networks <networklist>
-pool_size <size>
-separate_output <selection>

-shmem
-statistics
-stdin <selection>
-timeout <timeout>
-timing <timing-spec.>
-transporter_count <count>
-transporter_size <size>
-trace <trace-spec.>
-verbose
-Version
-working_directory <directory>
-xterm <xterm>
-zeroCOPY_min <size>
-zeroCOPY_max <size>

Separator between user program options and host specifications
Set automatic-mode for process(es).
Set backoff-mode for process(es).
Set number of entries per channel.
Set entry_size (in bytes) per channel.
Set threshold for inlining (in bytes) per inter-channel.
Set buffer size (in bytes) per inter-channel.
Set chunk-size for inter-communication.
Set debug-mode for process(es).
Set debugger to start in debug-mode.
Disable process timeout.
Set display to use in debug-/manual-mode.
Set number of buffers for eager protocol.
Set factor for subdivision of eager buffers.
Set buffer size (in bytes) for eager protocol.
Set threshold (in bytes) for eager protocol.
Set path to internal executables.
Display available options.
Set installation-directory.
Inherit user definable limits to processes.
Set manual-mode for process(es).
Define priority order when searching network.
Enable separate output for process(es).
Filename:ScaMPIoutput_host_pid_rank
Application use Cray ShMem library.
Enable statistics.
Distribute standard in to process(es).
Set timeout (elapsed time in seconds) for run.
Enable built-in timing trace.
Set number of buffers for transporter inter-protocol.
Set buffer size (in bytes) for transporter inter-protocol.
Enable built-in trace.
Display values for mpimon-options.
Display version of monitor.
Set working directory.
Set xterm to use in debug-/manual-mode.
Lower limit for using zeroCopy.
Upper limit for using zeroCopy.
Appendix B  Obsolete mpimon options

These options are recognized by mpimon but are deprecated and should not be used. They exist for backwards compatibility and will be removed in future versions. Some of the options are ignored while others are translated to valid options.

<table>
<thead>
<tr>
<th>Obsolet option</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-barrier_fanin</td>
<td>IGNORED</td>
</tr>
<tr>
<td>-barrier_fanout</td>
<td>IGNORED</td>
</tr>
<tr>
<td>-environment</td>
<td>IGNORED</td>
</tr>
<tr>
<td>-init_comm_world</td>
<td>IGNORED</td>
</tr>
<tr>
<td>-inter_channel_entry_count &lt;count&gt;</td>
<td>-channel_entry_count &lt;count&gt;</td>
</tr>
<tr>
<td>-inter_channel_entry_size &lt;size&gt;</td>
<td>-channel_entry_size &lt;size&gt;</td>
</tr>
<tr>
<td>-inter_channel_inline_threshold &lt;threshold&gt;</td>
<td>-channel_inline_threshold &lt;threshold&gt;</td>
</tr>
<tr>
<td>-inter_chunk_size &lt;size&gt;</td>
<td>-chunk_size &lt;size&gt;</td>
</tr>
<tr>
<td>-inter_eager_count &lt;count&gt;</td>
<td>-eager_count &lt;count&gt;</td>
</tr>
<tr>
<td>-inter_eager_size &lt;size&gt;</td>
<td>-eager_size &lt;size&gt;</td>
</tr>
<tr>
<td>-inter_eager_threshold &lt;threshold&gt;</td>
<td>-eager_threshold &lt;threshold&gt;</td>
</tr>
<tr>
<td>-inter_pool_size</td>
<td>IGNORED</td>
</tr>
<tr>
<td>-inter_transporter_count &lt;count&gt;</td>
<td>-transporter_count &lt;count&gt;</td>
</tr>
<tr>
<td>-inter_transporter_size &lt;size&gt;</td>
<td>-transporter_size &lt;size&gt;</td>
</tr>
<tr>
<td>-intra_channel_count &lt;count&gt;</td>
<td>-channel_count smp:&lt;count&gt;</td>
</tr>
<tr>
<td>-intra_transporter_size &lt;size&gt;</td>
<td>-transporter_size smp:&lt;size&gt;</td>
</tr>
<tr>
<td>-intra_channel_entry_count &lt;count&gt;</td>
<td>-channel_entry_count smp:&lt;count&gt;</td>
</tr>
<tr>
<td>-intra_channel_entry_size &lt;size&gt;</td>
<td>-channel_entry_size smp:&lt;size&gt;</td>
</tr>
<tr>
<td>-intra_channel_inline_threshold &lt;threshold&gt;</td>
<td>-channel_inline_threshold smp:&lt;threshold&gt;</td>
</tr>
<tr>
<td>-intra_chunk_size &lt;size&gt;</td>
<td>-chunk_size smp:&lt;size&gt;</td>
</tr>
<tr>
<td>-intra_eager_count &lt;count&gt;</td>
<td>-eager_count smp:&lt;count&gt;</td>
</tr>
<tr>
<td>-intra_eager_size &lt;size&gt;</td>
<td>-eager_size smp:&lt;size&gt;</td>
</tr>
<tr>
<td>-intra_eager_threshold &lt;threshold&gt;</td>
<td>-eager_threshold smp:&lt;threshold&gt;</td>
</tr>
<tr>
<td>-intra_pool_size</td>
<td>IGNORED</td>
</tr>
<tr>
<td>-intra_transporter_count &lt;count&gt;</td>
<td>-transporter_count smp:&lt;count&gt;</td>
</tr>
<tr>
<td>-intra_transporter_size &lt;size&gt;</td>
<td>-transporter_size smp:&lt;size&gt;</td>
</tr>
</tbody>
</table>
Appendix C  uDAPL functions

The following functions must be implemented for ScaMPI to be able to use a uDAPL-implementation:

dat_cr_accept
dat_cr_query
dat_cr_reject
dat_ep_connect
dat_ep_create
dat_ep_disconnect
dat_ep_free
dat_ep_post_rdma_write
dat_ep_post_recv
dat_ep_post_send
dat_evd_create
dat_evd_dequeue
dat_evd_free
dat_evd_wait
dat_ia_close
dat_ia_open
dat_ia_query
dat_lmr_create
dat_lmr_free
dat_psp_create
dat_psp_free
dat_pz_create
dat_pz_free
dat_rmr_bind
dat_rmr_create
dat_rmr_free
dat_set_consumer_context
Appendix D  Using trace and timing

The Scali MPI communication library has a number of builtin timing and trace facilities. This is a feature built into the run time version of the library and no extra recompiling or linking of libraries are needed. All MPI calls can be timed and/or traced. A number of different environment variables controls this. In addition an implied barrier call can be automatically inserted before all collective MPI calls. All of this can give very detailed insight into the performance of your application.

Examples

The trace and timing are initiated by environment variables that either can be set and exported or set at the command line just before mpimon.

Timing

Timing will give you information about which MPI routines were called and how long the MPI calls took. This information is printed at intervals set by the user with the “-s n” option, where n if the number of seconds. Here is an example using a test application all2all.

```
%SCAMPI_TIMING=-s 10 mpimon /opt/scali/examples/bin/all2all -- localhost 2
```

<table>
<thead>
<tr>
<th>Time</th>
<th>Calls</th>
<th>Time</th>
<th>Time/cal</th>
<th>Calls</th>
<th>Time</th>
<th>Time/cal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: Init+10 s</td>
<td>#calls</td>
<td>time</td>
<td>tim/cal</td>
<td>#calls</td>
<td>time</td>
<td>tim/cal</td>
</tr>
<tr>
<td>0: MPI_Alltoall</td>
<td>10926</td>
<td>9.9s</td>
<td>903.6us</td>
<td>10926</td>
<td>9.9s</td>
<td>903.6us</td>
</tr>
<tr>
<td>0: MPI_Barrier</td>
<td>23</td>
<td>1.0ms</td>
<td>44.7us</td>
<td>23</td>
<td>1.0ms</td>
<td>44.7us</td>
</tr>
<tr>
<td>0: MPI_Comm_rank</td>
<td>1</td>
<td>7.2ms</td>
<td>7.2us</td>
<td>1</td>
<td>7.2ms</td>
<td>7.2us</td>
</tr>
<tr>
<td>0: MPI_Comm_size</td>
<td>1</td>
<td>1.3ms</td>
<td>1.3us</td>
<td>1</td>
<td>1.3ms</td>
<td>1.3us</td>
</tr>
<tr>
<td>0: MPI_Init</td>
<td>1</td>
<td>1.0s</td>
<td>1.0s</td>
<td>1</td>
<td>1.0s</td>
<td>1.0s</td>
</tr>
<tr>
<td>0: MPI_Wtime</td>
<td>45</td>
<td>42.7us</td>
<td>949.8ns</td>
<td>45</td>
<td>42.7us</td>
<td>949.8ns</td>
</tr>
<tr>
<td>0: Sum</td>
<td>10997</td>
<td>10.9s</td>
<td>991.2us</td>
<td>10997</td>
<td>10.9s</td>
<td>991.2us</td>
</tr>
<tr>
<td>0: Overhead</td>
<td>0</td>
<td>0.0ns</td>
<td>0.0ns</td>
<td>10997</td>
<td>10.4ms</td>
<td>943.5ns</td>
</tr>
<tr>
<td>--</td>
<td>Delta</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1: Init+10 s</td>
<td>#calls</td>
<td>time</td>
<td>tim/cal</td>
<td>#calls</td>
<td>time</td>
<td>tim/cal</td>
</tr>
<tr>
<td>1: MPI_Alltoall</td>
<td>10926</td>
<td>9.7s</td>
<td>886.8us</td>
<td>10926</td>
<td>9.7s</td>
<td>886.8us</td>
</tr>
<tr>
<td>1: MPI_Barrier</td>
<td>23</td>
<td>285.5ms</td>
<td>12.4ms</td>
<td>23</td>
<td>285.5ms</td>
<td>12.4ms</td>
</tr>
<tr>
<td>1: MPI_Comm_rank</td>
<td>1</td>
<td>7.0ms</td>
<td>7.0us</td>
<td>1</td>
<td>7.0ms</td>
<td>7.0us</td>
</tr>
<tr>
<td>1: MPI_Comm_size</td>
<td>1</td>
<td>1.3ms</td>
<td>1.3us</td>
<td>1</td>
<td>1.3ms</td>
<td>1.3us</td>
</tr>
<tr>
<td>1: MPI_Init</td>
<td>1</td>
<td>1.0s</td>
<td>1.0s</td>
<td>1</td>
<td>1.0s</td>
<td>1.0s</td>
</tr>
<tr>
<td>1: MPI_Wtime</td>
<td>43</td>
<td>43.9us</td>
<td>1.0us</td>
<td>43</td>
<td>43.9us</td>
<td>1.0us</td>
</tr>
<tr>
<td>1: Sum</td>
<td>10995</td>
<td>10.0s</td>
<td>1000.0us</td>
<td>10995</td>
<td>10.0s</td>
<td>1000.0us</td>
</tr>
<tr>
<td>1: Overhead</td>
<td>0</td>
<td>0.0ns</td>
<td>0.0ns</td>
<td>10995</td>
<td>10.0ms</td>
<td>911.0ns</td>
</tr>
</tbody>
</table>

We see that the output gives statistics about which MPI calls that are used and their frequency and timing. Both delta numbers since last printout and the total accumulated statistics. By setting the interval timing (in -s <seconds>) to a
large number only the accumulated statistics at the end is printed. The timings are presented for each process and with many processes this can yield huge amount of output. There are many options to SCAMPI_TIMING that will reduce this output. The selection parameter can time only those MPI processes you would like to monitor, in addition there are other ways to minimize the output screening away selected MPI calls and before or after a certain number of calls or between an interval of calls. Some examples are:

```bash
%SCAMPI_TIMING="-c 10000" mpimon ./all2all -- localhost 2
```

Print statistics only per 10000 MPI calls. Useful to start only after a certain point in the program has been reached.

```bash
%SCAMPI_TIMING="-s 10 -p 1" mpimon ./all2all -- localhost 2
```

Print statistics for processes 1 only.

```bash
%SCAMPI_TIMING="-s 5 -f MPI_Recv" mpimon ./all2all -- localhost 2
```

Print statistics only after MPI_Recv has been called. Very useful to start only after a certain point in the program has been reached.

For more information please see the section useful tools in the ScaMPI guide.

**Trace**

Trace will give you information about which MPI routines were called and possibly information about parameters and timing. Here is an example using a test application all2all:

```bash
%SCAMPI_TRACE="-p all" mpimon ./all2all -- localhost 2
```

Prints a trace of all MPI calls for this run that is relatively simple:
If more information are needed the arguments to SCAMPI_TRACE can be enhanced to ask for more information. The option "-f arg;timing" gives the arguments given to each MPI call. This includes the message size which is a very useful information when evaluating interconnect performance.

%SCAMPI_TRACE="-f arg;timing" mpimon /opt/scali/examples/bin/all2all -- n1 2

This output is truncated as it is 119kbytes long just for this simple example run on using only two MPI processes.

There are a number of parameters to select only a subset, either number of calls and intervals as described above under timing or selecting just some MPI calls or excluding some MPI calls.

%SCAMPI_TRACE="-t MPI_Barrier" mpimon ./all2all -- localhost 2

Only the MPI calls MPI_Barrier has been traced.
We can exclude the MPI_Alltoall call and trace just the other:
%SCAMPI_TRACE"-s 5 -x MPI_Alltoall" mpimon /opt/scali/examples/bin/all2all -- localhost 2
256k 1.68512 32.86.08152.16
1:  MPI_Wtime
0:  MPI_Wtime
384k 1.85342 54.03.24 31.38.67
1:  MPI_Barrier id: 12167
0:  MPI_Barrier id: 12167
0:  MPI_Wtime
1:  MPI_Wtime
0:  MPI_Wtime
512k 1.92256 7.405.33 67.523.54
0:  MPI_Keyval_free
1:  MPI_Keyval_free

Any combination of this is allowed.

%SCAMPI_TRACE"-f timing -x MPI_Alltoall" mpimon ./all2all -- node1 2
256k 1.69512 3.306.10 75.62315.24
1:  +7 s 2.1us MPI_Wtime
0:  +7 s 1.6us MPI_Wtime
384k 2.16342 6.302.52 85.0319.00
1:  +7 s 5.95us MPI_Barrier id: 12167
0:  +7 s 90.1us MPI_Barrier id: 12167
1:  +7 s 3.0us MPI_Wtime
0:  +7 s 1.6us MPI_Wtime
1:  +9 s 3.2us MPI_Wtime
0:  +9 s 1.5us MPI_Wtime
512k 2.04256 7.962.97 62.7925.58
1:  +9 s 4.9us MPI_Keyval_free
0:  +9 s 3.6us MPI_Keyval_free

Or :

% SCAMPI_TRACE"-f arg;timing -t MPI_Alltoall" mpimon ./all2all -- n1 2
192k 1.35512 2.633.66 71.1942.39
256k 1.82512 3.545.84 70.52141.01
1:  +6 s 5.1ms MPI_Alltoall ssz: 98304 x 4 = 384.0k rsz: 393216 x 1 = 384.0k Id: 11913
0:  +6 s 7.0ms MPI_Alltoall ssz: 98304 x 4 = 384.0k rsz: 393216 x 1 = 384.0k Id: 11913
1:  +6 s 6.0ms MPI_Alltoall ssz: 98304 x 4 = 384.0k rsz: 393216 x 1 = 384.0k Id: 11914
0:  +6 s 6.3ms MPI_Alltoall ssz: 98304 x 4 = 384.0k rsz: 393216 x 1 = 384.0k Id: 11914
1:  +6 s 6.2ms MPI_Alltoall ssz: 98304 x 4 = 384.0k rsz: 393216 x 1 = 384.0k Id: 11915
0:  +6 s 6.2ms MPI_Alltoall ssz: 98304 x 4 = 384.0k rsz: 393216 x 1 = 384.0k Id: 11915
1:  +6 s 6.5ms MPI_Alltoall ssz: 98304 x 4 = 384.0k rsz: 393216 x 1 = 384.0k Id: 11916

Using the scanalyze script.

The powerful run time facilities ScaMPI trace and ScaMPI timing can be used to
monitor and keep track of MPI calls and their characteristics. The various trace
and timing options can yield many different views of your application's usage
of MPI. Common to most of these logs are the massive amount of data which

Scali MPI Connect Users Guide Version 1.1
58
can sometimes be overwhelming, especially when you run with many processes and use both trace and timing concurrently. In order to extract information from the huge amount of data a simple analysis tool is presented. This analysis tool accept output from ScaMPI applications run with some predefined trace and timing variables set.

There are two different set of tracing and timing

```
SCAMPI_TIMING="-s <seconds>"
SCAMPI_TRACE="-farg;timing"
```

Where 'n' is the number of seconds per printout from ScaMPI. This can be set to a large number in order to just collect final statistics.

The tool ScaAnalysis take the huge amount of timing and/or trace output and present a compact overview of the data.

### Timing

Below is given output from ScaAnalysis when applied to data generated from a run of all2all with SCAMPI_TIMING="-s 3".

```bash
% scanalyze all2all.timing 0
ScaMPI Timing analysis (file all2all.timing):
Total time used 8.0s
Communication time max 8.4s for process 1
2 MPI processes
0: 16.54.34
 0: Init+8 s
 0: MPI_Alltoall                      1    5.1ms    5.1ms       12399    7.1s   574.8us
 0: Sum                               1    5.1ms    5.1ms       12479    8.1s   652.7us
 0: Overhead                          0    0.0ns                12479   18.7ms    1.5us
 0: ________________________________________________________________
0: 16.54.34
 0: Init+8 s
 0: MPI_Alltoall                      0    0.0ns                12399    7.1s   574.8us
 0: MPI_Barrier                       0    0.0ns                   26    1.9ms   72.2us
 0: MPI_Comm_rank                     0    0.0ns                    1    7.0us    7.0us
 0: MPI_Comm_size                     0    0.0ns                    1    1.3us    1.3us
 0: MPI_Init                          0    0.0ns                    1    1.0s     1.0s
 0: MPI_Keyval_free                   1    4.5us    4.5us           1    4.5us    4.5us
 0: MPI_Wtime                         1    2.9us    2.9us          52   51.0us  980.9ns
 0: Sum                               2    7.4us    3.7us       12481    8.1s   652.6us
 0: Overhead                          0    0.0ns                12481   18.8ms    1.5us
 0: ________________________________________________________________
```

The list is presented for process 0 only as the second parameter given to ScaAnalysis is '0'. Omitting this parameter yield output for all processes.
Tracing and timing:

Below are an example taken from a real application running Fluent running Large Class 3 benchmark and SCAMPI_TRACE set to "-f arg;timing":

```
%scanaalyze fl5l3-16.log.2
ScaMPI Trace analysis (file fl5l3-16.log.2):
Found 85416 MPI calls, processing trace information
ScaMPI MPI calls:

<table>
<thead>
<tr>
<th>Calls</th>
<th>Total 85416 Avg.sz</th>
<th>&lt;128</th>
<th>128-1k</th>
<th>1k-8k</th>
<th>8k-32k</th>
<th>32k-256k</th>
<th>256k-1M</th>
<th>&gt;1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Recv</td>
<td>138850 33829 52163 453</td>
<td>19841 31011 30906 4476 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Send</td>
<td>138848 33830 52160 453</td>
<td>19841 31011 30907 4476 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Bcast</td>
<td>287354 4 287354 0</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Reduce</td>
<td>287550 4 287550 0</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>2208 7 2208 0</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Altoall</td>
<td>0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Allgather</td>
<td>0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Other MPI calls 4
Timing of MPI calls (in seconds)

<table>
<thead>
<tr>
<th>MPI Call</th>
<th>&lt;128</th>
<th>128-1k</th>
<th>1-8k</th>
<th>8k-32k</th>
<th>32k-256k</th>
<th>256k-1M</th>
<th>&gt;1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Recv</td>
<td>134.65 0.17 116.23</td>
<td>198.86 346.37 183.72 0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Send</td>
<td>4.17 0.02 3.03</td>
<td>5.77 10.02 190.41 0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Bcast</td>
<td>820.46 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Reduce</td>
<td>293.77 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>50.65 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Altoall</td>
<td>0.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Allgather</td>
<td>0.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The first part shows counting of different MPI calls, the average message size in bytes, number of MPI calls carrying a message of less than 128 bytes, number of calls carrying a payload of from 128 to 1k bytes, ditto for 8k to 32k bytes, 32k to 256k bytes, 256k to 1M bytes and finally the number of MPI calls with a message size of more than 1 Mbytes.

The seconds part shows the timing of these different MPI calls. The timing is a sum of the timing for all MPI calls for all MPI processes and since there are many MPI processes the timing can look unrealistic high. However, it reflects the total time spent in all MPI calls. If you are interested in timing more than tracing MPI calls the timing functionality is suggested. The trace functionality introduces some overhead and the total wall clock run time of the application goes up. The timing functionality is relatively light and can be used to time the application for performance benchmarking.
Appendix E  Troubleshooting ScaDAT

The Scali MPI now uses DAT as API to the drivers for the different interconnect drivers. The drivers are in DAT terminology called provider libraries, or dapl’s. This package implement common and generic part of uDAPL.

The ScaDAT package contains a DAT registry library (libdat.so), a common part that load and manage the specific interconnect/device provider libraries.

The ScaDAT package contains two provider libraries, a provider library implementation for TCP/IP sockets (libsockdapl.so) and a provider library implementation for Mellanox InfiniHost hardware (libibdapl.so).

3rd party DAT provider libraries (for example the open source reference implementation) can also be used with the registry by adding them to the dynamic registry config file /opt/scali/etc/dat.conf.

This package also contains DAT header files included in the uDAPL v1.0 specification available from http://datcollaborative.org.

Troubleshooting the ScaDAT package

There should be very little trouble with this package. The only requirements are that the fore mentioned libraries are in /opt/scali/lib with the proper permissions for shared objects, and that the fore mentioned dat.conf is in /opt/scali/etc, and properly formatted. These are the only files needed for normal operation.

If your attempting to use a 3rd party DAT provider library, make sure that it support the uDAPL v1.0 specification before contacting ScaliSupport, or 3rd party support.

Troubleshooting the ScaDSHM package

To verify that the shm0 device is operational, run a MPI test job on the node in question:
$ mpimon -networks shm0 /opt/scali/examples/bin/bandwidth -- node 2

Note that since shm0 is for intra node use only. An MPI job that uses only the shm0 device can run only on a single node. The nodename above is the same node name given twice, and must be one of the compute nodes in you cluster.
If the shm0 devices fails, the MPI job should fail with a "dat_ia_open ...fails" message.
To debug the shm0 device, verify that the kernel module is loaded by running lsmod(8) of the compute node in question. You should see a module named shmdat loaded. Then verify that the file/opt/scali/etc/dat.conf contain a line that look like this:
shmdat u1.0 nonthreadsafe default /opt/scali/lib/libshmdapl.so 1.0 ""
If it does, and /opt/scali/lib/libshmdapl.so exist and is readable and executable, and you're still experiencing problems, contact Scali support.
If the line is missing from /opt/scali/etc/dat.conf, or /opt/scali/lib/libshmdapl.so is missing, reinstall the ScaDSHM package. If the problem persist, contact Scali support.
The most common problem is that the shmdat module is not compiled for you kernel. The module shall be loaded at boot time, but to manually load the module issue the command as root:
# service shmdat restart

If that succeed, but it's not loaded at boottime, verify that the shmdat service is activated for your runlevel with chkconfig(8).
It's important that the kernel module is loaded with the servicescript, and not insmod directly, since the script will make sure that the device file, /dev/shmdat, is created with the right devicenumbers, and permissions.
If the module fail to load, an appropriate error message will be displayed. If you need to recompile the module, the normal messages will indicate that the kernel and module has a version mismatch, or that there are unresolved symbols in the module.
Compilation of the modules must be done on the frontend, and the package ScaDSHMAdap must be installed here. Compilation instructions is found in /opt/scali/doc/ScaDSHMAdap/README.
Note that the frontend must have the kernel headers, and kernelconfiguration for the kernel your using on the compute nodes.
If there is a problem with the compilation, you're most likely using a kernel that is unsupported by Scali, but contact Scali Support.
DSHM requires a datshm license to work and this is typically distributed together with your Scali software platform. Contact license@scali.com for more information regarding licensing of the DSHM module.
# List of figures

1-1 Application start-up - phase 1 ...................................................... 26  
1-2 Application start-up - phase 2 ...................................................... 27  
1-3 Application start-up - phase 3 ...................................................... 28  
1-4 Thresholds for different communication protocols .......................... 29  
1-5 Inlining protocol ........................................................................ 30  
1-6 Eagerbuffering protocol .............................................................. 30  
1-7 Transporter protocol .................................................................. 31  
1-8 Connections ............................................................................. 33  
1-9 Connection building blocks ......................................................... 34  
1-10 Channel resource ...................................................................... 35  
1-11 Eagerbuffer buffer ..................................................................... 36  
1-12 Transporter buffer ..................................................................... 37
Index

B
Benchmarking ScaMPI .................................................................47

C
Communication protocols in ScaMPI ............................................29
  Eagerbuffering protocol ..............................................................30
  Inlining protocol .......................................................................30
  Transporter protocol ..................................................................31, 32
Communication resources in ScaMPI ...........................................32
  Channel buffer ........................................................................34
  Eagerbuffer buffer ....................................................................36
  Transporter buffer .....................................................................37
Compiling
  ScaMPI ................................................................................7
  ScaMPI example program .........................................................22

D
Debugging
  ScaMPI ................................................................................13

E
Environment variables
  ScaMPI ................................................................................7

L
libfmpi .....................................................................................25
libmpi .........................................................................................25
Linking
  ScaMPI ................................................................................8
  ScaMPI example program .........................................................22

M
MPI ..........................................................................................6, 50
mpi.h .........................................................................................7, 40
mpiboot .....................................................................................25
MPICH ......................................................................................50
mpid .........................................................................................25
mpif.h .........................................................................................7, 40
mpimon .....................................................................................9, 25
Advanced usage ........................................................................9
Basic usage ...............................................................................9
mpirun .......................................................................................12
mpisubmon ...............................................................................25
Troubleshooting ScaMPI