ScaMPI User’s guide

Acknowledgement

The development of ScaMPI 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 ScaMPI.
The list of persons contributing to algorithmic ScaMPI improvements are 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 ScaMPI.

Table of contents

Chapter 1 Introduction

A Scali System is a set of SCI interconnected nodes, where each node is a multiprocessor workstation or PC, running either Solaris or Linux. To get the full computational power of a Scali System it is necessary to use ScaMPI. ScaMPI is Scali's high performance MPI implementation. The programming environment for ScaMPI provides a variety of options and tools for tuning and debugging. In addition, ScaMPI is integrated with powerful third party software.

ScaMPI utilises shared memory on intra node communication, and the fast SCI interconnect on inter node communication. Any parallel MPI-conforming application can be run with ScaMPI, and benefit from the SCI performance.

1.1 Purpose of the ScaMPI User's guide

This document describes ScaMPI, the Scali implementation of the Message Passing Interface version 1.1 [1].

Its purpose is:
- to supply the user with enough information to use ScaMPI.
- to give the interested reader an overview of the ScaMPI implementation.

The guide is *neither* a tutorial in using MPI, *nor* a guide on how to design efficient MPI programs.

1.2 Scope of the ScaMPI User's guide

This document has the following layout:
- Chapter 2 - explains how to get started running your first MPI program with ScaMPI.
- Chapter 3 - describes how to compile, link, run, debug and profile ScaMPI programs.
- Chapter 4 - describes the internal design and functionality of ScaMPI.
- Chapter 5 - provides hints on what to do if problems are encountered.
- Chapter 6 - explains what to do if you need assistance from Scali.
- Chapter 7 - gives an overview of related documentation.
- Appendix A - outlines how to install ScaMPI.

1.3 Required knowledge

This guide is written for users which have a basic understanding of MPI [1, 2, 3], and some basic knowledge of the C and/or Fortran programming language.

1.4 Abbreviations and Acronyms

Abbreviation	Meaning
ABI	Application Binary Interface.
API	Application Programming Interface.
DQS	Distributed Queueing System from Florida State University.
GUI	Graphical User Interface.
HPC	High Performance Computing.
HPF	High Performance Fortran.
MPI	Message Passing Interface.
MPICH	A Portable Implementation of MPI.
MPE	Multi Processing Environment.
PVM	Portable Virtual Machine.
SCI	Scalable Coherent Interface.
SSP	Scali Software Platform (the generic name of all Scali software packages).

Table 1-1: Abbreviations

1.5 Basic terms

Term	Description
bash	GNU Borne-Again-SHell.
Cluster	A cluster is a set of interconnected nodes with the possibility to act as a single unit.
g77, gcc	GNU Fortran and C compilers.
Host	A single node of a Scali System, i.e., a multiprocessor workstation or PC.
Node	A single computer in an interconnected system of one or more computers.
Process	Instance of application program with unique rank within MPI_COMM_WORLD.
Scali System	Scali System = Scali software + compute nodes + high speed interconnect.
stdin, stdout	Standard Unix I/O streams.
Socket	Endpoint for Internet communication.
Unix	Refers to all UNIX and lookalike OSes supported by the SSP, e.g., Solaris and Linux.
xterm	Terminal emulator for the X-window system.

Table 1-2: Basic terms

By default, **gcc** and **bash** are used for all examples.

1.6 Typographic conventions

Term	Description.
Bold	Program names, options and default values.
Italics	User input.
#	Command prompt in shell with super user privileges.
%	Command prompt in shell with normal user privileges.

Table 1-3: Typographic conventions

Chapter 2 Getting started

This chapter outlines how to setup, compile, link and run a simple MPI program, to help a user unfamiliar with ScaMPI to get started. For a more completete description, please see chapter 3. This introduction also presents some of the ScaMPI test programs delivered by Scali.

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.

Topics covered:
- Section 2.1 - explaines how to build and run a simple MPI program on a Scali System.
- Section 2.2 - describes some of the MPI test programs delivered by Scali.

2.1 An example program

The example program is compiled using one of the GNU compilers. Before compilation, it is assumed that the BASH shell environment variable has been properly defined. In addition, ScaMPI must have been installed and function correctly. If not yet successfully installed, please see Appendix A for installation instructions, or consult your System Administrator.

As an example, the MPI program named **hello-world** is used. The example exists as a C program in the file **hello-world.c**, and as a Fortran program in the file **hello-world.f.** The step-by-step instructions for compilation, linking (with **libmpi** and **libfmpi**), and execution of the **hello-world** program are found in sections 2.1.1 - 2.1.6.

2.1.1 Setting up your BASH environment

Set MPI_HOME to point at the installation directory of ScaMPI, i.e., **/opt/scali**. In addition, include the path to the ScaMPI executables in the PATH statement.

```
% MPI_HOME=/opt/scali; export MPI_HOME
```

2.1.2 Hello-world.c - source in 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();
}
```

2.1.3 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
```

2.1.4 Compiling

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

2.1.5 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
```

2.1.6 Running

Start the **hello-world** program on the 3 nodes named hostA, hostB and hostC.

```
% mpimon hello-world -- hostA 1 hostB 1 hostC 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
```

2.2 MPI test programs

The ScaMPItst package contains a collection of MPI test programs for ScaMPI. Sections 2.2.1 - 2.2.3 gives 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. For further instructions on how to compile, link and run a program, please see section 2.1 and chapter 3.

2.2.1 Producer - a producer-consumer MPI test program

Producer is a simple producer-consumer program. Processes with rank 0, 1, 2, ..., n/2-1 send data while process n/2, n/2+1, ...,n-1 receive data. Process 0 will send to process n-1, process 1 will send to 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 process is started on each node, and a single message of size 1024 bytes are transferred from the process on nodeX to the process on nodeY. The program should return TEST COMPLETE.

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

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

The program should return TEST COMPLETE.

2.2.2 Bandwidth - a bandwidth MPI test program

Bandwidth is a program to measure bandwidth for various message sizes between two 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
```

2.2.3 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 s**cript 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 Using ScaMPI

This chapter describes the setup, compile, link, run, debug and profile of an MPI program using ScaMPI. The control and start up of any MPI program using ScaMPI is monitored by the monitor program **mpimon**.The MPI program(s) to be started, can be invoked as described in section 3.4.

Running ScaMPI on a Scali System with multi-user and resource management software implies that the startup mechanism is built on top of **mpimon**. Scali encourages use of state-of-the-art management software, it is a user friendly and effective way of utilizing a Scali System.

Topics covered:
- Section 3.1 - explains how to setup the ScaMPI environment.
- Section 3.2 - provides compile instructions for use with ScaMPI.
- Section 3.3 - provides guidelines used when linking.
- Section 3.4 - describes how to start and run ScaMPI applications.
- Section 3.5 - explains ways to debug an ScaMPI application.
- Section 3.6 - explains how to analyse performance of an ScaMPI application.

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

3.1 Setting up a ScaMPI environment

The System Administrator has (probably) set up your ScaMPI shell environment in the standard startup scripts. The environment variable MPI_HOME is set to point at the ScaMPI installation directory. The standard PATH variable should include the path to the ScaMPI executables and the path to the dynamic link libraries is set in the LD_LIBRARY_PATH. For a more detailed description, please see section 3.1.1.

Normally, the ScaMPI library's header files **mpi.h** and **mpif.h** reside in the $MPI_HOME/include directory.

3.1.1 ScaMPI environment variables

The use of ScaMPI 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.

Name	Description
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. May 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` An alternative to using LD_LIBRARY_PATH, is to give a flag (-R) to the linker which includes a specific path to the dynamic libraries. For details, check the release notes.
PATH	Path variable. May be updated to include the path to the directory where the MPI binaries can be found: `export PATH=${PATH}:$MPI_HOME/bin`

Table 3-1: Environment variables for Unix

3.2 Compiling

Please note that the ScaMPI library is an API (Application Programming Interface) and not an ABI (Application Binary Interface). Thus, all applications must be recompiled and linked with ScaMPI.

3.2.1 Compiler support

ScaMPI is a C++ library built using the GNU **egcs** compiler. Thus, depending on the compiler used by the user, the way to link the ScaMPI libraries varies. For details, please check the ScaMPI release notes. Please note that the GNU **egcs**/**gcc** 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/download**.

Linux: ScaMPI is supported for use with the following compilers:
- GNU and EGCS **gcc**/**g++**/**g77** for i86pc
- Portland Group **pgcc**/**pgf77**/**pgf90** i86pc
- Compaq **ccc/fort** alpha

Solaris: ScaMPI is supported for use with the following compilers:
- GNU and EGCS **gcc**/**g++**/**g77** for UltraSPARC and i86pc

- Apogee **apcc**/**apCC**/**apf77**/**apf90** for UltraSPARC
- Portland Group **pgcc**/**pgf77**/**pgf90** i86pc
- Sun SunPro - **CC**/**f77**/**f90** for UltraSPARC and i86pc

3.2.2 Compiler flags

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

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

3.3 Linking

Due to the fact that it is required to link with the GNU runtime library, the syntax depends on the compiler being used. For details, please check the ScaMPI release notes.

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

```
"-L/opt/scali/lib -Wl,-R/opt/scali/lib $CRT_BEGIN -lmpi $CRT_END"
```

The runtime setup CRT_BEGIN and CRT_END libraries are defined for some compilers. -R is a flag asking the linker to include the path to the dynamic libraries. When specified, the environment variable LD_LIBRARY_PATH (see table 3-1) is not needed. Please note that when linking a Fortran main program, the Fortran interface library **libfmpi** must be included *before* CRT_BEGIN.

3.4 Running MPI programs

Note that executables issuing ScaMPI calls ***cannot*** be started directly from a shell prompt. ScaMPI programs can either be started from a shell prompt using the MPI monitor program **mpimon** (see section 3.4.1) or by using the wrapper script **mpirun** (see section 3.4.2), or from the Scali Desktop GUI [5]. In addition, ScaMPI programs can be started from a 3rd party software workload management system [15, 16] running on top of **mpimon** or **mpirun**.

Naming convention
Note that when an application program is started, ScaMPI 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>@<hostname>)

where:

<userprogram> is the name of the application program.

<rank number> is the application's process rank number.
<pid> is the Unix process identifier of the monitor program **mpimon**.
<hostname> is the name of the host where **mpimon** is running.

Remember that it is absolutely necessary to run an ScaMPI program on a homogenous file system image, i.e., on a file system providing the same path and program name on all nodes of the Scali System.

3.4.1 mpimon - monitor program

The control and start-up of an ScaMPI application is monitored by **mpimon**. The program **mpimon** has several options which can be used for optimising ScaMPI performance. Normally it should not be necessary to use any of these options. However, unsafe MPI programs [3] might need buffer adjustments to get rid of hangs. Trading performance by changing communication space is best avoided if there are no compelling reason to do so.

3.4.1.1 Basic usage

Normally the program is invoked as:

mpimon <userprogram> <programoptions> -- <hostname> [<count>] [<hostname> [<count>]]...

Parameter	Description
<userprogram>	Name of application program.
<programoptions>	Program options for the application program.
--	Separator, marks end of user program options.
[<hostname> [<count>]]	Name of host and the number of processes to run on that host. The option can occur several times in the list. Processes will be given ranks sequentially according to the list of host-number pairs.

Table 3-2: Basic options to **mpimon**

3.4.1.2 Advanced usage

The complete syntax for the program:

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

Parameter	Description
<program & host-spec>	**<program spec> -- <host spec> [<host spec>] ...**
<program spec>	**<userprogram>[<programoptions>]...**
<userprogram>	Name of application program.
<programoptions>	Program options for the application program.
--	Separator, signals end of user program options.
<host spec>	**<hostname> [<count>]**
<hostname>	Name of host. Given either as a host name or as an Internet address expressed in the Internet standard dot notation.
<count>	Number of processes to run on host. If **<count>** is omitted, one process is started on each host specified.

Table 3-3: mpimon parameters

Numeric values can be given as **mpimon** options in the following way:

Option	Description
<numeric value>	**<decimal value> | <decimal value><postfix>**
<postfix>	**<K>: <numeric value> = <decimal value> * 1024** **<M>: <numeric value> = <decimal value> * 1024 * 1024**

Table 3-4: Numeric input

A complete lists of available **mpimon** options::

mpimon option	Description
--	Separator, marks end of user program options.
-automatic <*selection*>	Set automatic-mode for process(es). Default: **none** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**
-barrier_fanin <*count*>	Set number of barrier fanin reads. Default: **8**
-barrier_fanout <*count*>	Set number of barrier fanout reads. Default: **8**
-debug <*selection*>	Set debug-mode for process(es). Default: **none** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**
-debugger <*debugger*>	Set debugger to start in debug-mode.
-disable-timeout	Disable process timeout.
-display <*display*>	Set display to use in debug-/manual-mode.
-environment <*value*>	Define how to export environment. Default: **export** Legal: **'export'= all or 'mpi'= MPI_?? or 'none'**
-exact_match	Set exact-match-mode.
-execpath <*execpath*>	Set path to internal executables.
-help	Display available options.
-home <*directory*>	Set installation-directory.
-immediate_handling <*selection*>	Handling of immediates. Default: **lazy** Legal: **lazy, threaded, automatic**
-inherit_limits	Inherit userdefinable limits to processes.
-init_comm_world	Initialise MPI_COMM_WORLD at startup (all channels are created).
-inter_adapters <*adapters*>	Set list of sci adapters for inter-communication. Default: **all** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**

Table 3-5: Complete list of **mpimon** options

mpimon option	Description
-inter_channel_inline_threshold <*size*>	Set threshold for inlining (in bytes) per inter-channel. Default: **560**
-inter_channel_size <*size*>	Set buffer size (in bytes) per inter-channel. Default: **4K** Legal: **Powers of 2**
-inter_chunk_size <*size*>	Set chunk-size for inter-communication. Default: **512k** Legal: **Multiplum of pages**
-inter_eager_count <*count*>	Set number of buffers for eager inter-protocol. Default: **2**
-inter_eager_size <*size*>	Set buffer size (in bytes) for eager inter-protocol. Default: **128K** Legal: **Powers of 2**
-inter_pool_size <*size*>	Set buffer-pool-size for inter-communication. Default: **32M** Legal: **Multiplum of pages**
-inter_transporter_count <*count*>	Set number of buffers for transporter inter-protocol. Default: **4** Legal: **Powers of 2**
-inter_transporter_size <*size*>	Set buffer size (in bytes) for transporter inter-protocol. Default: **64K**
-intra_channel_inline_threshold <*size*>	Set threshold for inlining (in bytes) per intra-channel. Default: **560**
-intra_channel_size <*size*>	Set buffer size (in bytes) per intra-channel. Default: **4K** Legal: **Powers of 2**
-intra_chunk_size <*size*>	Set chunk-size for intra-communication. Default: **1M** Legal: **Multiplum of pages**
-intra_eager_count <*count*>	Set number of buffers for eager intra-protocol. Default: **2**
-intra_eager_size <*size*>	Set buffer size (in bytes) for eager intra-protocol. Default: **128K** Legal: **Powers of 2**
-intra_pool_size <*size*>	Set buffer-pool-size for intra-communication. Default: **4M** Legal: **Multiplum of pages**

Table 3-5: Complete list of **mpimon** options

mpimon option	Description
-intra_transporter_count <*count*>	Set number of buffers for transporter intra-protocol. Default: **4** Legal: **Powers of 2**
-intra_transporter_size <*size*>	Set buffer size (in bytes) for transporter intra-protocol. Default: **64K**
-manual <*selection*>	Set manual-mode for process(es). Default: **none** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**
-read <*file*>	Read parameters from the named file. Default: **none**
-separate_output <*selection*>	Enable separate output for process(es). Filename: ScaMPIoutput_host_pid_rank Default: **none** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**
-sm_debug <*selection*>	Set debug-mode for submonitor(s). Default: **none** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**
-sm_manual <*selection*>	Set manual-mode for submonitor(s). Default: **none** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**
-sm_trace <*selection*>	Enable trace for submonitor(s). Default: **none** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**
-statistics	Enable statistics.
-stdin <*selection*>	Distribute standard in to process(es). Default: **none** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**
-timeout <*timeout*>	Set timeout (elapsed time in seconds) for run. Legal: **Positive number**
-trace <*selection*>	Enable trace for process(es). Default: **none** Legal: **'n,m,o..'= (list) or 'n-m'= (range) or 'all'**
-verbose	Display values for user-options.
-Version	Display version of monitor.
-xterm <*xterm*>	Set xterm to use in debug-/manual-mode.

Table 3-5: Complete list of **mpimon** options

3.4.2 mpirun - wrapper script

mpirun is a wrapper script for **mpimon**, giving MPICH [21] style startup for ScaMPI applications. Instead of the **mpimon** syntax, where a list of pairs of host name and number of processes is used as startup specification, **mpirun** uses only the total number of processes.

Using **scaconftool** (see [5]), **mpirun** attempts to generate, by issuing *echo "list nodeid OK" | /opt/scali/bin/scaconftool -l d -q*, 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 processes among the nodes.

3.4.2.1 mpirun usage

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

Parameter	Description
<mpirunoptions>	**mpirun** options, see Table 3-7
<mpimonoptions>	Options passed on to **mpimon**, see Table 3-5.
<userprogram>	Name of application program to run.
<programoptions>	Program options passed on to the application program.

Table 3-6: mpirun format

mpirun option	Description
-np <*count*>	Total number of processes to be started, default 2.
-npn <*count*>	Maximum number of processes pr. node, default **np** <count>/nodes.
-dqs	Submit job to the DQS queue system. For details, see separate DQS documentation [15].
-dqsparams <*params*>	Specify DQS **qsub** parameters, see separate DQS documentation [15].
-v	Verbose.
-gdb	Debug all processes using the GNU debugger **gdb**.
-maxtime|-cpu <*time*>	Limit runtime to <time> minutes.
-noconftool	Do not use **scaconftool** for generating hostlist.
-noarchfile	Ignore the **/opt/scali/etc/ScaConf.nodearchmap** file (which describes each node).
-H <*frontend*>	Specify hostname of front-end running the **scaconf** server.
-mstdin <*proc*>	Distribute **stdin** to process(es). <*proc*>: all (default), none, or process number(s).
-q	Keep quiet, no **mpimon** printout.
<*params*>	Parameters not recognized are passed on to **mpimon**.

Table 3-7: mpirun options

3.5 Debugging ScaMPI applications

When ScaMPI applications need to be debugged, it is strongly recommended to use a parallel debugger, which can manage multiple processes simultaneously. On a Scali System running ScaMPI programs, **TotalView** is the recommended parallel debugger. For further information, please send an inquiry to **sales@scali.com**.

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

3.5.1 Debugging with a parallel debugger

This section contains a brief description of **TotalView**, describes how to set up the user environment, and outlines how to get started debugging ScaMPI programs using the multiprocess debugger.

By using a parallel debugger to start an ScaMPI program, the program can be stopped in the first MPI routine being called, i.e., in **MPI_Init()**. In this routine, the parallel debugger provides a single point of control for the MPI program(s) being debugged.

3.5.1.1 What is TotalView?

TotalView is a graphical multiprocess, multithread debugger that supports multiple parallel programming paradigms. It offers several debugging features, and has support for different platforms and languages. A product overview can be found in the *Products* section at
http://www.etnus.com.

3.5.1.2 TotalView user environment setup

The use of **TotalView** requires that some environment variables are defined, see table 3-8. For further information, see the TotalView Installation Guide [12]**.** Normallly, these environment variables are set by the System Administrator.

Name	Description
PATH	Path variable. Should include the directory where the **TotalView** binaries are installed, e.g., /opt/totalview/bin.

Table 3-8: Environment variables for **TotalView**

Name	Description
MANPATH	Path to man pages. Should point to the directory where the **TotalView** man pages are installed, e.g., /opt/totalview/man.
LD_LIBRARY_PATH	Path to libraries. Should point to the directory where the **TotalView** libraries are located,, e.g., /opt/totalview/lib.
LM_LICENSE_FILE	List of license files. A ":" separated list. One element should point to the **TotalView** license file, e.g., /opt/totalview/license.dat.

Table 3-8: Environment variables for **TotalView**

3.5.1.3 Debugging using TotalView

First of all, to include debugging information in the executable program(s), the program must be compiled using the **-g** option. Depending on the compiler being used, it may be necessary to specify what kind of debugging information to be generated, please see the ScaMPI release notes.

To invoke the ScaMPI program in a single **TotalView** debugger window, use the **tvmpimon** script:

```
tvmpimon <mpimon options>
```

TotalView prompts for where to stop. A stop in **MPI_Init()**, provides a single point of control for the debugging session. The **<mpimon options>** are options passed on to **mpimon**, see Table 3-5.

Inside **TotalView**, all three mouse buttons have functions assigned to them.Use the *left* mouse button to select an object. The *middle* mouse button pops up a menu of possible commands. The *right* mouse button divides into many objects on the screen. For a complete description of how to use the **TotalView** debugger, see the TotalView Users's Guide [11].

3.5.2 Debugging with a sequential debugger

If no parallel debugger is available, an ScaMPI application 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 **-debugger <debugger>.**

To set debug-mode for one or more processes, specify the 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 ScaMPI processes being debugged.

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

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

Initially, for both process 0 and 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 ScaMPI 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 process is started. Then, when ready to run the process, paste the ***run <programoptions>*** into the debugger to get running.

3.6 Profiling ScaMPI applications

When developing MPI programs, one of the most useful tools is an analysis tool for performance analysis of the MPI programs being run. Before an ScaMPI application can be analysed, it must be linked and run with a profiling library to collect trace data for post-mortem performance analysis.
The appropriate profiling library is linked with ScaMPI through an MPI profiling interface defined in the MPI standard [1].

Two different set of profiling libraries are available with ScaMPI:

- The recommended **Vampirtrace** profiling library from Pallas GmbH. Currently, the commercially supported **Vampirtrace** library (see section 3.6.1) is available for ScaMPI on SPARC-Solaris and x86-Solaris platforms. The appropriate library in **Vampirtrace** generates a tracefile suitable for analysis with the **Vampir** tool (see section 3.6.1) from Pallas GmbH.

For information about **Vampirtrace** availability and supported platforms, please send an inquiry to **sales@scali.com**. Inquires about **Vampir** should be directed to **info@pallas.de**.
The product descriptions for **Vampirtrace** and **Vampir** can be found in the *Products* section at **http://www.pallas.de**.

- The adapted MPE profiling libraries from MPICH [21].
  The modified MPE profiling libraries (see section 3.6.2) are included for free in the ScaMPE library package.

3.6.1 Profiling and analysing using Vampirtrace and Vampir

This section contains a brief description of **Vampirtrace** and **Vampir**, explains how to set up the user environment, and outlines how to collect and analyse performance data for an ScaMPI application.

When the **Vampirtrace** distribution has been unpacked, a complete description of **Vampirtrace** is available in the *Vampirtrace Installation and User's Guide*, found in the **./doc** directory of the **Vampirtrace** root directory.

When the **Vampir** distribution has been unpacked, full documentation can be found in the **./doc** directory of the **Vampir** root directory. For help on installation, see the *Vampir Installation Guide,*
and for a complete description of Vampir, see the V*ampir User's Manual*.

3.6.1.1 What is Vampirtrace?

Vampirtrace is a profiling library for MPI programs. It works as an add-on for the ScaMPI implementation, and is the link between the ScaMPI application and **Vampir**. Simply relinking the ScaMPI application with the appropriate **Vampirtrace** library, enables tracing of all calls to MPI routines and tracing of all explicit message-passing as well**.** The trace information produced by **Vampirtrace** is saved to disk for post-processing when the application is about to finish. The generated trace file is suitable for analysis with **Vampir**.

Vampirtrace allows an application to define and record user defined events. If such events are to be traced, calls to the **Vampirtrace** API must be inserted in the application's source code. Thus, the affected code must be recompiled, and the application relinked. For information about how to use the API, see the *Vampirtrace Installation and User's Guide*.

3.6.1.2 What is Vampir?

Vampir is a tool for performance analysis of MPI programs. It allows (post-mortem) visualization and analysis of MPI programs, based on trace information created using the **Vampirtrace** library.

Vampir helps the user to organise the performance information, understand the application and communication behavior, evaluate load balancing, and identify communication hots pots.

Vampir converts the trace information into a variety of graphical views. For example, a timeline window display application and message passing activities. Communication statistics can be displayed for selected intervals of time and message length. Profiling statistics can display the execution times of routines.

3.6.1.3 Vampirtrace and Vampir user environment setup

In order to use **Vampirtrace** and/or run **Vampir**, valid license keys must exist. The use of **Vampirtrac**e and/or **Vampir** requires that at least one of two environment variables are defined. Normally, these environment variables are set by the System Administrator. The license keys are stored in a plain ASCII file, where each line normally contains a separate license key. The file's pathname must be defined by setting one of the two environment variables in table 3-9. If both are set, PAL_LICENSFILE takes precedence.

Name	Description
PAL_ROOT	Path variable. Points to the root of the **Vampirtrace/Vampir** installation. The pathname of the license key file is assumed to be **$PAL_ROOT/etc/license.dat.**
PAL_LICENSEFILE	Path variable. Specifies the complete pathname of the license key file. A relative pathname is interpreted starting from the user's home directory.

Table 3-9: Setup for **Vampirtrace** and **Vampir**

3.6.1.4 Linking an ScaMPI application with Vampirtrace

The **Vampirtrace** product contains one profiling library **libVT** that produces tracefiles suitable for the **Vampir** performance analysis tool, and another profiling library **libDT** that produces tracefiles suitable for the **Dimemas** performance prediction tool from Pallas GbmH. For information about **Dimemas**, contact **info@pallas.de**. The following description refers to the **libVT** library used for **Vampir**.

If $VAMPIRTRACE_ROOT is the root directory of the **Vampir** installation, the **Vampirtrace** libraries reside in the $VAMPIRTRACE_ROOT/lib directory. To link an application written in C with the **Vampirtrace** library, include the trace library **libVT** in the link command line *before* the ScaMPI library **libmpi**, e.g.,

```
-L$VAMPIR_LIB_DIR -lVT -lnsl -lmpi ...
```

To link a Fortran program, the Fortran wrapper library **libfmpi** must be linked in *before* libVT, e.g.,

```
-L$VAMPIR_LIB_DIR -lfmpi -lVT -lnsl -lmpi ...
```

In addition note that, to resolve some Internet symbol references used, the standard library **libnsl** must be included in the link command line.

To link and run an application that issues calls to the **Vampirtrace** API *without* generating traces, the application can be (re)linked with a dummy **libVTnull** version of the profiling library (and the ordinary profiling library **libVT**). The dummy library contains nothing but the **Vampirtrace** entry points, and can be linked *after* the ScaMPI library **libmpi**. When **libVTnull** is linked in, the executable program will not generate traces (and almost no extra profiling overhead will occur).

3.6.1.5 Running an ScaMPI application with Vampirtrace

The program linked with **Vampirtrace** is started in the same way as an ordinary ScaMPI application using the monitor program **mpimon**, or by using the wrapper script **mpirun**. However, to ensure that all the participating processes have the correct license file setup, launch the program using the **mpimon** option **-environment export**.
When the application is about to finish, the trace data is written from local memory to a trace file for post-processing, and, within **MPI_Finalize**(), an information message is printed to **stdout** by **Vampirtrace**. The message provides the actual trace file name. A binary **Vampirtrace** file has the suffix *.bpv*. For a description of how to force the name for the trace file, see the *Vampirtrace Installation and User's Guid*e.

3.6.1.6 Analyzing an ScaMPI application using Vampir

When the performance data has been collected using **Vampirtrace**, the event traces produced by the ScaMPI application can be analysed using **Vampir**. If $VAMPIR_ROOT is the root directory of the **Vampir** installation, the **Vampir** executable image resides in the $VAMPIR_ROOT/bin directory.

There are two ways to open a trace file. One way is to specify its name directly on the command line, and the other way is to open it from within **Vampir**.

Invoke **Vampir**:

```
$VAMPIR_ROOT/bin/    vampir   [<file name>]
```

After **Vampir** has completed startup, and its main window is displayed, select the menu option
File/Open Tracefile to browse and select an appropriate (*.*bpv*) trace file. The graphical user interface of **Vampir** provides an easy to use interface.

Inside **Vampir**, all three mouse buttons have functions assigned to them. Generally, a single click with the *left* mouse button is used to select a single process. A single click with the *middle* mouse button is used to deselect all prior selected objects, or to close the view. A single click with the *right* mouse button pops up a context menu with view specific functions.

3.6.2 Profiling with ScaMPE

The ScaMPE libraries are modified versions of the MPE libraries from MPICH [21]. 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.

Linking an ScaMPI application
Profiling using one of the ScaMPE libraries is achieved by linking with the appropriate ScaMPE library *before* the standard ScaMPI 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 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, ScaMPI is modifying argv[0], as described in section 3.4. 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 [21].

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**.

Examine the generated log file - upshot
To examine a log file generated using **liblmpi**, the parallell 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/ScaMPE**.

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 [21].

Chapter 4 Description of ScaMPI

4.1 General description

ScaMPI consists both of libraries to be linked and loaded with the user application program, and a set of executables which control the startup and execution of the user application program(s). For a more in-depth description of the ScaMPI design and implementation, please see [8].

4.1.1 ScaMPI libraries

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

Table 4-1: Libraries

4.1.2 ScaMPI executables

A number of executable programs are included in ScaMPI.

4.1.2.1 mpimon - monitor program

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

4.1.2.2 mpisubmon - submonitor program

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

4.1.2.3 mpiboot - bootstrap program

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

4.1.2.4 mpid - daemon program

mpid is a daemon program running on all hosts 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 host boots, and must run at all times.

4.2 Starting ScaMPI application programs

ScaMPI uses socket communication for control purposes. Schematically, startup of application programs in a Scali System are performed as described in the following sections.

4.2.1 Application startup - phase 1

Step	Description
mpimon: parameter control.	**mpimon** does as much control of the specified options and parameters as possible. The userprogram names are checked for validity, and the hosts are, using sockets, contacted to ensure they are responding and that **mpid** is running.
mpimon: connecting to **mpid.**	**mpimon** establishes a connection to the **mpid** daemon on each host specified, and transfers basic information to enable the daemon to start the submonitor **mpisubmon**.

Table 4-2: Application startup - phase 1

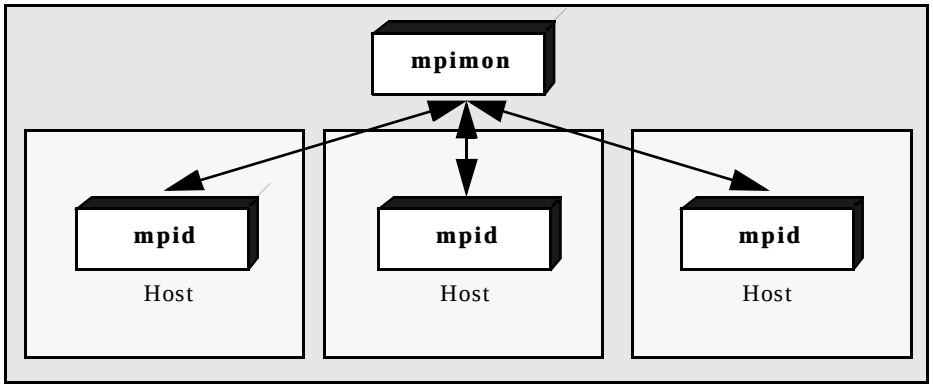

Figure 4-1: Application startup - phase 1

4.2.2 Application startup - phase 2

Step	Description
mpid: starting submonitors.	On each host, **mpid** starts the submonitor **mpisubmon**.
mpisubmon: connecting to the monitor program **mpimon.**	Each submonitor establishes a connection to **mpimon.** Control information are exchanged between each **mpisubmon** and **mpimon** to enable **mpisubmon** to start the specified userprograms (processes).
mpisubmon: creating shared memory	On each host, **mpisubmon** creates memory segments to be shared between the interconnected hosts.

Table 4-3: Application startup - phase 2

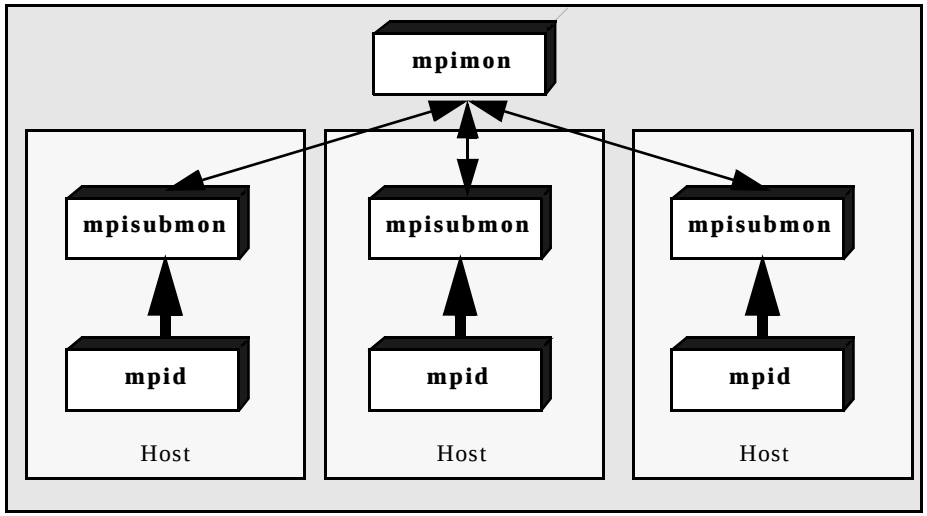

Figure 4-2: Application startup - phase 2

4.2.3 Application startup - phase 3

Step	Description
mpisubmon: invoking processes. Processes start and enter **MPI_Init**().	On each host, **mpisubmon** starts all the processes to be executed.
Processes synchronize.	Upon receipt of all control information, the processes will, via there 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 appropriate processes.
Processes return from **MPI_Init**() and start to run.	The user program(s) takes control.

Table 4-4: Application startup - phase 3

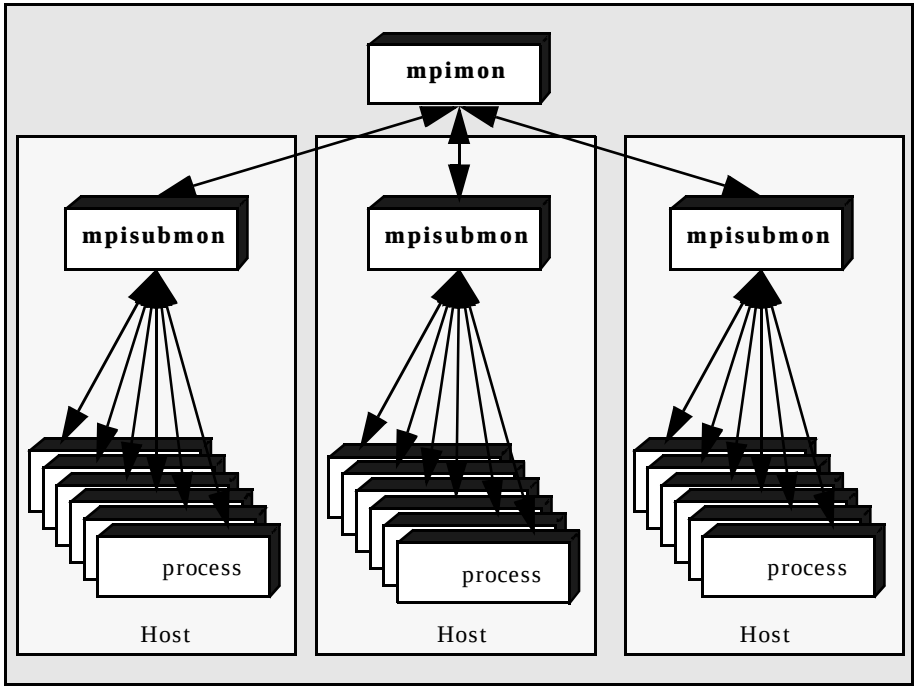

Figure 4-3: Application startup - phase 3

4.3 Stopping ScaMPI application programs

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

Step	Description
Processes enters **MPI_Finalize**().	Each process signals, via its local **mbisubmon**, to **mpimon** that it has entered **MPI_Finalize**(), and it is now waiting.
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**().
Processes leave **MPI_Finalize**().	Processes terminate, each **mpisubmon** releases shared memory segments and exits, and finally **mpimon** terminates.

Table 4-5: Application termination

below.

4.4 Communication resources

All resources (buffers) used by ScaMPI reside in shared memory, and are allocated by **mpisubmon** on demand from the sender process. ScaMPI uses a *on demand* scheme for allocating resources. *On demand* means that buffers are not allocated until needed. To get a list of the resource settings, pass the **-verbose** option to **mpimon**.

mpisubmon operates on two separate buffer pools suitable for sharing - both pools in shared memory. One pool (local shared memory) provides resources for *intra-node* communication, and the other pool (SCI shared memory) provides resources for *inter-node* communication. The size of each buffer pool, and the size of each chunk may be set using options to **mpimon.** The *pool size* limits the total amount of shared memory, and the *chunk size* limits the maximum block of memory that can be allocated as a single buffer. By default, the *pool size* is set to 4M for intra-node and 32M for inter-node, and the *chunk size* is set to 1M for intra-node and 512Kfor inter-node.

To set the *pool size* and the *chunk size* limits, use **mpimon** and specify:

-intra_pool_size <size> to set the buffer pool size for intra-node communication

-intra_chunk_size <size> to set the chunk size for intra-node communication

-inter_pool_size <size> to set the buffer pool size for inter-node communication

-inter_chunk_size <size> to set the chunk size for inter-node communication

Sections 4.4.1 - 4.4.3 outlines the various types of resources (*channel*, *eagerbuffer*, *transporter*) being used, and lists the **mpimon** options used to enforce specific buffering. However, it is normally **not** necessary to set the buffer parameters. Automatic buffer management is performed by ScaMPI, as described in the *'automatic buffer management'* section of Table 5-1 on page 50.

4.4.1 Channel buffer

For a sender-receiver pair, one *channel* ringbuffer is used for each communicator.

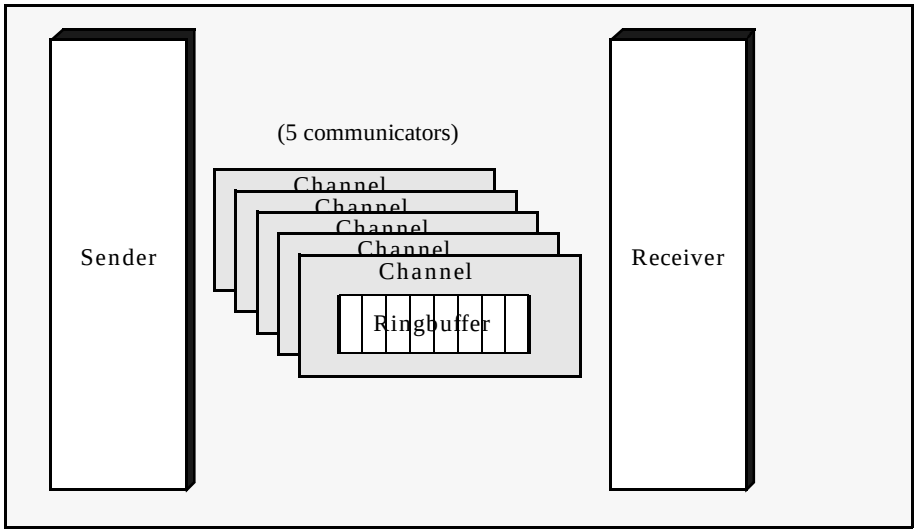

Figure 4-4: Channel resource

Each entry in the ringbuffer is 64 bytes long. 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 (see section 4.5). In addition, one ore more entries are utilized by the *inline* protocol for application data being transmitted.

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

-intra_channel_size <size> to set the ringbuffer size (in bytes) per intra-channel

-inter_channel_size <size> to set the ringbuffer size (in bytes) per inter-channel

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

-intra_channel_inline_threshold <size> to set threshold for inlining per intra-channel
-inter_channel_inline_threshold <size> to set threshold for inlining per inter-channel

4.4.2 Eagerbuffer buffer

For a sender-receiver pair, one, and only one, *eagerbuffer* buffer is used.

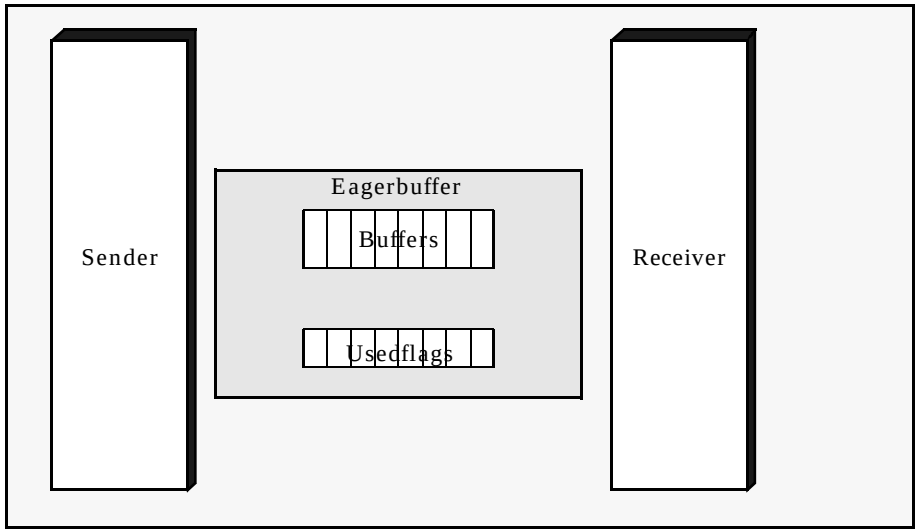

Figure 4-5: Eagerbuffer buffer

An *eagerbuffer* buffer is allocated when medium size messages (see figure 4-7) are to be transferred, and is utilized by the *eagerbuffering* protocol (see section 4.5.2).

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

-intra_eager_size <size> to set the buffer size (in bytes) for intra-node communication

-intra_eager_count <count> to set number of buffers for intra-node communication

-inter_eager_size <size> to set the buffer size (in bytes) for inter-node communication

-inter_eager_count <count> to set number of buffers for inter-node communication

4.4.3 Transporter buffer

For a sender-receiver pair, one, and only one, *transporter* buffer is used.

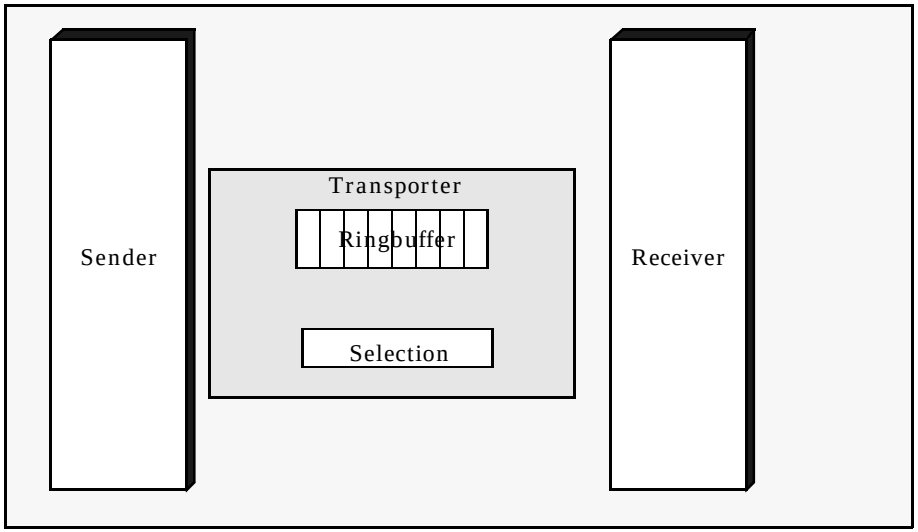

Figure 4-6: Transporter buffer

A *transporter* buffer is allocated when large messages (see figure 4-7) are to be transferred, and is utilized by the *transporter* protocol (see section 4.5.3).

To force the *transporter* resource definitions, use **mpimon** and specify:

-intra_transporter_size <size> to set the buffer size (in bytes) for intra-node communication
-intra_transporter_count <count> to set number of buffers for intra-node communication

-inter_transporter_size <size> to set the buffer size (in bytes) for inter-node communication
-inter_transporter_count <count> to set number of buffers for inter-node communication

4.5 Communication protocols

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.

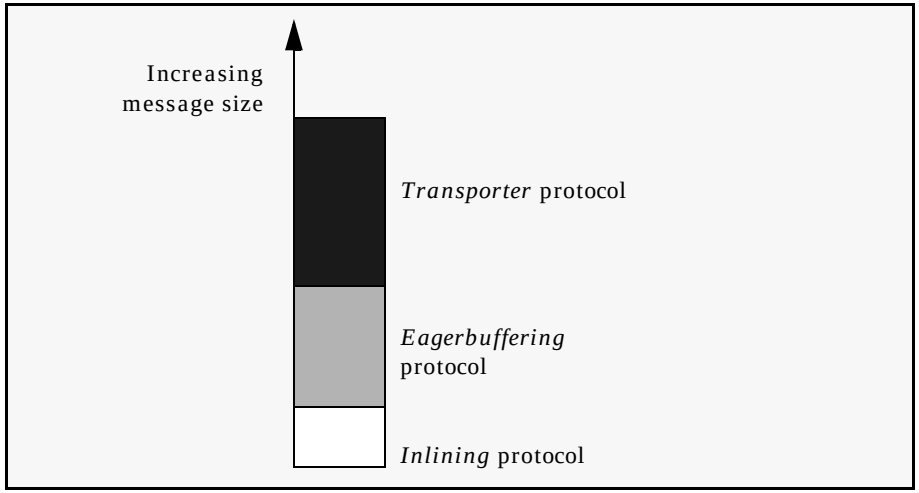

Figure 4-7: Thresholds for different communication protocols

The various communication protocols used, are briefly outlined in the following sections.

4.5.1 Inlining protocol

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

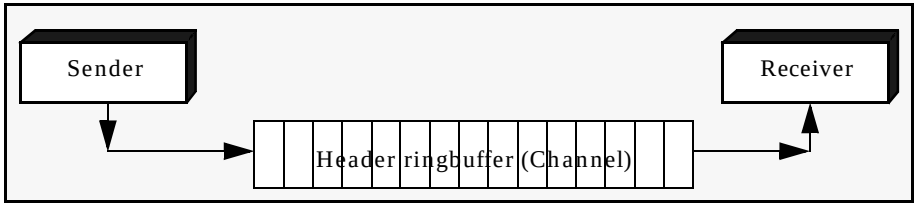

Figure 4-8: Inlining protocol

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 4.4.1.

The inlining protocol is selected when:

0 <= message size <= intra/inter_channel_inline_threshold bytes.

4.5.2 Eagerbuffering protocol

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

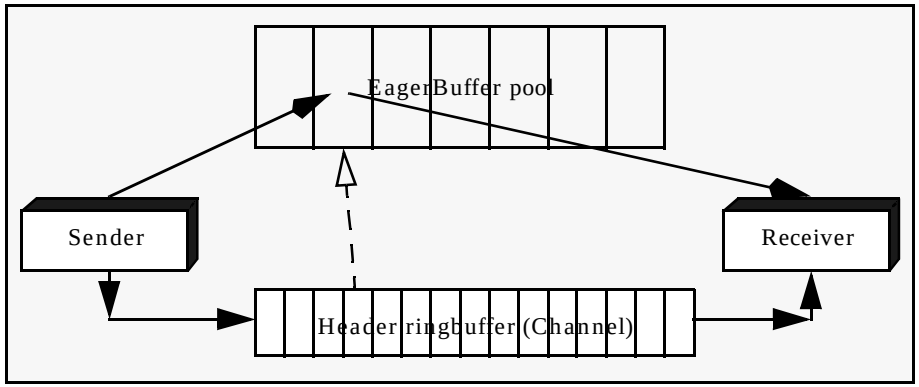

Figure 4-9: Eagerbuffering protocol

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:
intra/inter_channel_inline_threshold bytes < message size <= intra/inter_eager_size bytes.

4.5.3 Transporter protocol

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

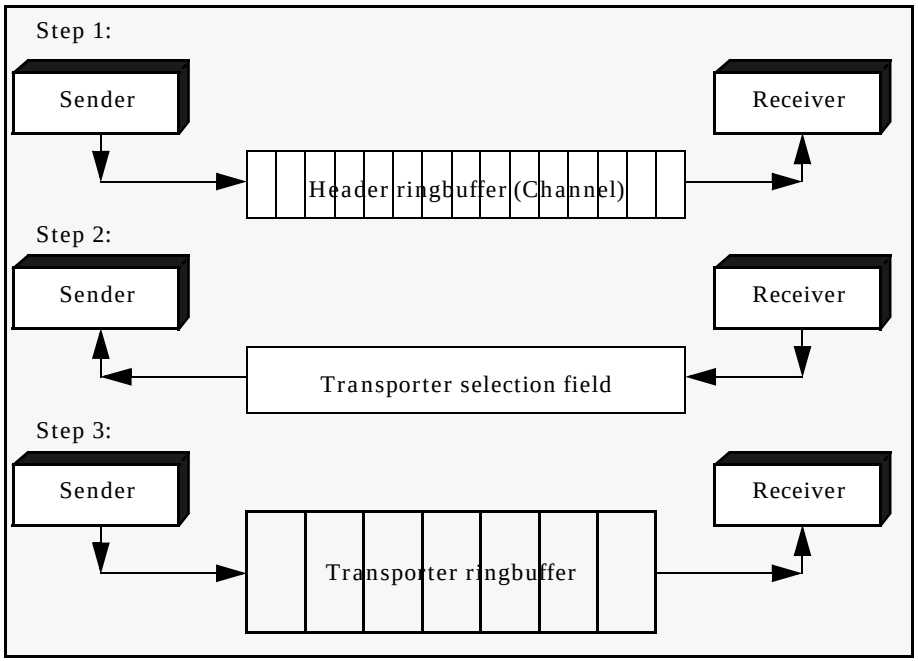

Figure 4-10: Transporter protocol

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 one transporter buffer 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 (intra/ inter_transporter_size bytes).

The transporter protocol is selected when:

message size > intra/inter_eager_size bytes.

Chapter 5 Tips &Tricks

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, as well as chapter 6, before contacting **support@scali.com**.

Topics covered:;

- Section 5.1 - notes about why some programs may run with MPICH, and not with ScaMPI.
- Section 5.2 - gives a list of names to avoid when programming with ScaMPI.
- Section 5.3 - explains error messages that may occur when using ScaMPI.
- Section 5.4 - provides the solution to common problems.
- Section 5.5 - gives some hints on how to improve performance of an ScaMPI application.
- Section 5.6 - deals with benchmarking on Scali Systems.

Currently, these sections are by no means complete. Problems reported to Scali will eventually be included in section 5.4. Thus, please send your relevant remarks by e-mail to **support@scali.com**.

5.1 Application program notes

5.1.1 MPI_Probe() and MPI_Recv()

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

```
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 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 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:

```
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();
}
```

5.1.2 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 occure (since ScaMPI use spinnlocks, 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):

```
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()**.

5.2 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**.

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.

5.3 Error and warning messages

5.3.1 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 a lot more warnings than default. **mpimon** assume that all environment variables starting with **MPI_** are meant for starting the application. If, for example, **MPI_LIBS** is defined as an environment variable, an unknown environment variable will be introduced to **mpimon**. When detected, **mpimon** will print the following warning message:

--- *mpimon* --- *Environment: MPI_LIBS, Not recognized* ---

The message is a warning only, and handling it does not affect the application.

5.3.2 Fatal errors

Upon a fatal error, ScaMPI prints an error message before starting **MPI_Abort()** to shut down all MPI processes.

5.4 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 Desktop GUI, see the Scali System Guide [5].

Problem	*Description* / Solution
How do I start my program? *use mpimon from shell prompt...* *use mpimon from Scali Desktop GUI...*	For practical examples, see Chapter 2 Getting Started. The **-stdin** option specifies which process rank should receive the input. You can in fact send **stdin** to all the processes with the **all** argument, but this requires that all 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 -- host1 host2 ... < input_file* **Note** that default direction of **stdin** is **-stdin none**. It is also possible to start applications using the Scali Desktop GU: **/opt/scali/bin/scadesktop &**
Why doesn’t my program start to run? *incorrect setting of environment variables...*	*mpimon: command not found.* **Todo:** Include **/opt/scali/bin** in the **PATH** environment variable. *mpimon can’t find mpisubmon.* **Todo:** Set **MPI_HOME=/opt/scali** or use the **-execpath** option. *The application has problems loading libraries (libsca*).* **Todo-1:** Update the **LD_LIBRARY_PATH** to include **/opt/scali/lib**. **Todo-2:** Link your application with path to the Scali libraries (gnu: **-Wl, -R/opt/scali/lib**).
Why doesn’t my program start to run? *Include of non-ScaMPI mpi.h detected!*	*An MPICH headerfile or an application compiled with a pre-SSP 2.0 library is detected.* **Todo:** Recompile your application with the latest ScaMPI libraries.

Table 5-1: When things don’t work

Problem	*Description* / Solution
Why doesn't my program start to run? *How to resolve incompatible mpi versions.*	*mpid, mpimon, mpisubmon and the libraries all have version variables that are checked at start-up.* **Todo:** An incompatibility have normally one of three reasons: * your environment variable **MPI_HOME** is set incorrect, * a new version of ScaMPI is installed without restart of **mpid** or * a new version of ScaMPI was not cleanly installed on all nodes.
What monitor-options are available?	**Todo:** Run **mpimon -help** to get the list of all **mpimon** options with a short explanation, or check the **mpimon** description in chapter 4.
Why doesn't my program start to run? *daemons in the night...*	*To run your application, several daemons need to be running:* * **scid**: Handles initialisation of and faults on the SCI card. * **mpid**: Enables communication between the processes in the program. * **scacomd**: Information scatter & gather between **scaconf** and node. * **scaconfnd**: Administration, i.e., setting of routing, for the node. * **scaconfsd**: **scaconfnd** master running on the frontend. * **scasnmpd**: Status gather and state setting from the Scali Desktop GUI If a required daemon is not running (check with **/opt/scali/bin/scaps**) it needs to be restarted. For more details on daemons, see the Scali System Guide [5]. **Todo-1:** Restart the missing daemon manually by **/opt/scali/bin/scash** \ **-pa -n "nodelist" /opt/scali/init.d/<daemon> restart**. **Todo-2:** Use the fix or the daemon command in **/opt/scali/sbin/scaconftool**. **Todo-3:** Use the Scali Desktop GUI.
Why does mpid not 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.* **Todo:** Use **netstat -a | grep mpid** to observe when the socket is released. When the socket is released, restart **mpid** again.

Table 5-1: When things don't work

Problem	*Description* / Solution
Why doesn't my program start to run? *interconnect problems: routing...*	*The program terminates with an ICMS_NO_RESPONSE error message* This happens when one or more processes cannot create a remote memory mapping to another node within a (long) period of time. **Todo-1:** Check if all relevant nodes are alive by issuing any command with **scash**, e.g., **/opt/scali/bin/scash -p hostname**. **Todo-2:** Check if SCI network routing is properly set with **/opt/scali/sbin/scaconftool** (command: **sciping OK**), or use the Scali Desktop GUI.
Why doesn't my program start to run? *interconnect problems: bad clean up...*	*A previous ScaMPI run has not terminated properly.* **Todo-1:** Check for processes on the nodes using **/opt/scali/bin/scaps**. *A leftover process holds SCI or shared memory resources.* Note: Core dumping takes time... **Todo-1:** Use **/opt/scali/sbin/scidle** **Todo-2:** Use **/opt/scali/bin/scash** to check for leftover shared memory segments on all nodes (**ipcs** for Solaris and Linux).
Why doesn't my program start to run or why does it terminate abnormally? *interconnect problems: space overflow...*	*Your application have required too much SCI or shared memory resources.* **Todo-1:** Your **mpimon** *pool-size* specifications are too large. **Todo-2**: Number of communicators in the program is higher than expected when doing automatic buffer calculations. Since memory by default is allocated in large chunks, try to reduce the *chunk-size* parameter to **mpimon** (use **mpimon -verbose** to get current buffer settings).
Why doesn't my program start to run or why does it terminate abnormally? *core dump...*	*The application core dumps.* **Todo:** Use a parallel debugger e.g., **TotalView** to locate the point of violation. The application needs to be recompiled to include symbolic debug information (**-g** for most compilers). A sequential (per process) debug session is possible using **gdb**, **dbx**, **pgdbg**, or another similar sequential debugger.
Why doesn't my program start to run or why does it terminate abnormally? *SCI interconnect failures...*	*The program terminates with an ICMS_* message.* **Todo:** 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 Desktop 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**.

Table 5-1: When things don't work

Problem	*Description* / Solution
Why does my program terminate abnormally?	*Are you reasonable certain that your algorithms are MPI safe?* **Todo:** Check if every send has a matching receive. *The program just hangs:* **Todo-1:**Try starting the program with **-init_comm_world** specified; if it doesn't start, there is a buffer allocation problem. Further information is available in the *'How do I control SCI and local shared memory usage?'*section. **Todo-2:** 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:* **Todo:** Investigate the core file, or rerun the program in a debugger.
How do I control SCI and local 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 control SCI and local shared memory usage? *one communication channel...*	The **eager buffer**s are used for small messages, while the **transporter buffer**s are used for handling large messages (larger than eager size). The **channel buffer**s is a send queue where each entry is 64 bytes, i.e., in a 8k buffer there is room for 128 outstanding requests. The function of the various buffers is outlined in section 4.4. All buffers are created when needed (i.e., when tried used for the first time), or at start up when **-init_comm_world** is specified. The buffer space required by a communication channel is approximately: channel = (2 * channel-size * communicators) + (transporter-size * transporter-count) + (eager-size * eager-count) + 512 (give-or-take-a-few-bytes) **Note:** Messages up to 560 bytes (the upper limit can be set using the option **channel_inline_threshold <size>** to **mpimon**) get inlined in the channel buffer. If you frequently use short messages, increasing the *channel-size* beyond 4k bytes might be a good idea.

Table 5-1: When things don't work

Problem	*Description* / Solution
How do I control SCI and local shared memory usage? *pool size...*	The communicators parameter depends on the application (assumed to be two in the automatic approach). If more communicators than expected by the buffer size calculations are used, the application may run out of shared memory. To overcome this, reduce the *chunk-size*. The *pool-size* is a limit for the total amount of shared memory. Default *pool-size* is set to 32M inter node and 4M intra node.

Table 5-1: When things don't work

Problem	*Description* / Solution
How do I control SCI and local shared memory usage? *automatic buffer management...*	The automatic buffer size computations is based on a full connectivity, i.e., all communicating with all others. If all process *P* in a program communicate with all the other processes, each process will communicate with *P_intra* processes intra node (it-self inclusive) and *(P - P_intra)* processes inter node. Given a total *pool* of memory dedicated to communication, each communication channel will be restricted to use a partition of only: inter_partition = inter_pool_size / (P_intra*(P-P_intra)) intra_partition = intra_pool_size / (P_intra * P_intra) The automatic approach is to downsize all buffers associated with a communication channel until it fits in its part of the pool. The *chunk size* sets the size of each individual allocated memory segment. The automatic chunk size is calculated to wrap a complete communication channel.
How do I control SCI and local shared memory usage? *barrier buffer...*	The barrier buffer is one page, and up to a maximum of *barrier_fanout+1* buffer mappings are created. By default, the **-barrier_fanout <count>** parameter is set to 8.

Table 5-1: When things don't work

Problem	*Description* / Solution
How do I control SCI and local shared memory usage? *an example... with ScaMPI 1.9.1:*	*Running two processes on one node with channel-size 256k* mpimon -intra_channel_size 256k -intra_pool_size 4m \ /opt/scali/examples/bin/bandwidth -- <nodename> 2 *Would terminate without starting with the message:* --- mpimon --- intra_pool_size = 4194304 must be at least 4227072 bytes (2113536 bytes * 2 processes) for given set of parameters --- (Calculation is left out as an exercise...). *Using the minimum pool size:* mpimon -intra_channel_size 256k -intra_pool_size 4227072 \ /opt/scali/examples/bin/bandwidth -- <nodename> 2 *Would start with the following parameters:* -intra_channel_size 256K -intra_chunk_size 1M -intra_eager_count 2 -intra_eager_size 1K -intra_pool_size 4227072 -intra_transporter_count 4 -intra_transporter_size 256 *A more natural choice of parameters may be:* mpimon -intra_channel_size 256k -intra_pool_size 6m \ /opt/scali/examples/bin/bandwidth -- <nodename> 2 *Would start with the following parameters:* -intra_channel_size 256K -intra_chunk_size 1M -intra_eager_count 4 -intra_eager_size 64K -intra_pool_size 6M -intra_transporter_count 4 -intra_transporter_size 32K **Note**: channel-size 256k is an unusual high value.

Table 5-1: When things don't work

5.5 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.

Problem	*Description* / Solution
Performance analysis.	*Learn about the performance behaviour of your MPI application on a Scali System by using a performance analysis tool.* The recommended profiling software tools to use with ScaMPi are the Vampirtrace MPI profiling library and the Vampir visualization and analysis tool. The freely available ScaMPE profiling library may also be used with ScaMPI. For more information, please see section 3.6.
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.
Using MPI_Bsend().	Using buffered send, e.g., **MPI_Bsend()**, usually degrade performance significantly compared to their unbuffered relatives.
Avoid starving processes - fairness.	MPI programs may, if not special care is taken, be unfair and may starve 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.
Creating more threads than available processors.	When immediate send & receive are used, ScaMPI creates an additional thread for handling this. Having more than one thread on a multi-processor usually improve performance, while increasing the number of threads beyond the number of processors may reduce performance (due to frequent context switching).

Table 5-2: How to get good performance using ScaMPI

Problem	*Description* / Solution
Using MPI_Sendrecv() transforms to MPI_Isend() and MPI_Recv()...	*In ScaMPI,* ***MPI_Sendrecv()*** *transforms into* ***MPI_Isend()*** *&* ***MPI_Recv()****, i.e., two threads. Using* ***MPI_Sendrecv()*** *to communicate on two processes on one node will therefore transform into 4 threads for long messages (assuming single threaded application). Running this on a dual processor node will therefore be very slow (frequent context swithing).* **Todo-1:** For small messages (<= eager-size) **MPI_Isend()** is treated as **MPI_Send()** and hence acceptable performance. To shift the performance degradation to out of observation field, set the **-intra_eager_size 1M** (or a size larger than the maximum message size). **Todo-2**: Use the **-immediate_handling lazy** parameter to **mpimon**. This is the default setting for ScaMPI 1.9.1 and later.
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. **Todo-1**: Set **channel-inline-threshold 964** (64 added for alignment) and increase the channel-size significantly (32-128 k). **Note**: the **channel-inline-threshold** can not be increased beyond 1023. **Todo-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.
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 process wanting to send data.

Table 5-2: How to get good performance using ScaMPI

5.6 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.

.

Problem	*Description* / Solution
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 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 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**. *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 5.4. (See *How do I control SCI and local shared memory usage?* in table 5-1.)

Table 5-3: Benchmarking with ScaMPI

Chapter 6 Support

6.1 Feedback

Scali appreciates any suggestions to improve both this ScaMPI 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 [19]. 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.

6.2 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/support**.

6.3 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/support**. In addition, the FAQ is, when ScaMPI has been installed, available as a text file in **/opt/scali/doc/ScaMPI/FAQ.**

6.4 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.

6.5 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 helpfull. The report should be sent by e-mail to **support@scali.com.**

6.6 Platforms supported

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

6.7 Licensing

ScaMPI is licensed, see appendix A-1.3, using Scali license manager system. In order to run ScaMPI, the license server software must be installed, and a valid demo or a permanent license must be obtained.

To obtain the appropriate license, please send an inquiry to **license@scali.com**. Any technical issues should be addressed to **support@scali.com.**

Chapter 7 Related documentation

7.1 References

[1] **MPI: A Message-Passing Interface Standard**
The Message Passing Interface Forum, Version 1.1, June 12, 1995,
Message Passing Interface Forum, *http://www.mpi-forum.org*.

[2] **MPI: The complete Reference: Volume 1, The MPI Core**
Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, Jack Dongarra. 2e, 1998,
The MIT Press, *http://www.mitpress.com*.

[3] **MPI: The complete Reference: Volume 2, The MPI Extension**
William Grop, Steven Huss-Lederman, Ewing Lusk, Bill Nitzberg, W. Saphir, Marc Snir, 1998,
The MIT Press, *http://www.mitpress.com*.

[4] **The Message Passing Interface (MPI) standard**
Argone National Laboratory, *http://www-unix.mcs.anl.gov/mpi*

[5] **Scali System Guide**
Scali AS, *http://www.scali.com/download*.

[6] **ScaMPI User's Guide**
Scali AS, *http://www.scali.com/download*

[7] **ScaMPI Datasheet**
Scali AS, *http://www.scali.com/download*

[8] **ScaMPI - Design and Implementation**
Scali AS, *http://www.scali.com./download*

[9] **Scali Free Tools**
Scali AS, *http://www.scali.com./download*

[10] **ScaMPI Product Description**
Scali AS, *http://www.scali.com/products*

[11] **TotalView User's Guide - Multiprocess Debugger**
Etnus Inc, *http://www.etnus.com*

[12] **TotalView Installation Guide - Multiprocess Debugger**
Etnus Inc, *http://www.etnus.com*

[13] **Vampir - Visualization and Analysis of MPI Programs**
Pallas GmbH, *http://www.pallas.de.*

[14] **Vampirtrace - MPI Profiling Library**
Pallas GmbH, *http://www.pallas.de.*

[15] **DQS - Distributed Queueing System**
The DQS home page, *http://www.scri.fsu.edu/~pasko/dqs.html*

[16] **CCS: Computing Center Software, a resource management software for HPC systems**
Paderborn Center of Parallel Computing, *http://www.uni-paderborn.de/pc2/projects/ccs.*

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

[18] **Debugging Tools and Standards**
HPDF - High Performance Debugger Forum, *http://www.ptools.org/hpdf/.*

[19] **Parallel Systems Software and Tools**
NHSE - National HPCC Software Exchange, *http://www.nhse.org/ptlib.*

[20] **Cluster Computing Papers**
IEEE Task Force on Clustered Computing, *http://www.dgs.monash.edu.au/~rajkumar/tfcc.*

[21] **MPICH - A Portable Implementation of MPI**
The MPICH home page, *http://www.mcs.anl.gov/mpi/mpich/index.html.*

[22] **MPI Test Suites freely available**
Argone National Laboratory, *http://www-unix.mcs.anl.gov/mpi/mpi-test/tsuite.html*

Appendix A ScaMPI installation

This appendix briefly describes som aspects of the ScaMPI installation process. The required software and instructions are available on the Scali CD-ROM. In addition, software and instructions can be downloaded from the *Download* area at **http:// www.scali.com**.

Installing Scali software onto a Scali System with many nodes is most easily done using the Scali Desktop GUI, see the Scali System Guide [5].

A-1 Installing

The installation guidelines provided are appropriate when installing on a single node at a time.

A-1.1 Requirements

Before you can run an MPI program with ScaMPI, the following requirements must be full-filled:

- Access to a set of SCI interconnected nodes with supported SCI drivers installed, or, alternatively, access to a single node.
- The network file system must provide a similar file system image from all the nodes. This applies only for the path to programs and files used when executing an MPI program.
- The supported GNU **gcc** compiler (or a similar version), found in the ScaFegcs package, must have been installed.
- The license server daemon **scald** is installed, preferably on a front-end of the cluster. To obtain the proper license file, please send an inquiry to **license@scali.com**.
- The appropriate licenses have been succesfully installed on each node. ScaMPI should not be installed until the licenses are installed; otherwise, after installing the license software, the daemon program **mpid** must be manually started on each node of the Scali System.

A-1.2 Distribution file

ScaMPI is distributed as a single package file, named **ScaMPI.os.arch-x.y.z.package**, where:

x.y.z	is the release number, e.g. 2.0.1,
os	is the operating system, e.g. Solaris2, Linux2,

arch	is the architecture, e.g. sparc-u or i86pc,
package	is the package type, e.g. pkg or rpm.

A-1.3 Licensing

ScaMPI is licensed using the Scali license software(ScaLM). In order to run ScaMPI, the license server software must be installed, and a valid demo or a permanent license must be obtained.

To obtain the appropriate license, please send an inquiry to **license@scali.com**. Any technical issues should be addressed to **support@scali.com.**
For more information on ScaLM, refer to Scali System Guide[5].

A-2 The Scali System directory tree

Almost all Scali software package files are installed under the directory **/opt/scal**i.

The **/opt/scali** directory looks like:

- **bin** : All executables to be run by regular users.
- **sbin** : All daemons and executables to be run by administrators.
- **libexec** : Executables which are hidden from normal invokation, e.g., executables used by applications under **bin.**
- **lib** : Libraries used by Scali applications and by end users.
- **include** : Include files for libraries under **lib**.
- **doc** : All documentation.
- **etc** : Configuration files for Scali software.
- **license** : Files belonging to a 3rd party licensing system (FLEXlm).
- **examples** : Source and documentation for the example program and the test programs.
- **contrib** : 3rd party software adapted to Scali software.
- **init.d** : Boot and startup scripts are installed here.

Under **/etc/rc.d/init.d**, soft links are made to system startup catalogues.

A-3 Useful 3rd party parallel software

It should be possible to integrate any kind of parallel MPI software with ScaMPI. For the most recent information concerning available parallel software, please send an inquiry to **sales@scali.com.** In addition, useful information about 3rd party parallel software are found at a number of web sites, e.g., [9], [17], [19], [20], [21].

Chapter 8 List of figures

List of tables