# Scali MPI Connect™ Users Guide

Software release 4.4

*High Performance Clustering*

## *Acknowledgement*

The development of Scali 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 Scali MPI Connect.
The list of persons contributing to algorithmic Scali MPI Connect improvements is impossible to compile here. We apologize 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 Scali MPI Connect.

**SCALI "BRONZE" SOFTWARE CERTIFICATE**

**(hereinafter referred to as the "CERTIFICATE")**

**issued by**

**Scali AS, Olaf Helsets Vei 6, 0619 Oslo, Norway**

**(hereinafter referred to as "SCALI")**

**DEFINITIONS**

- "**SCALI SOFTWARE**" shall mean all contents of the software disc(s) or download(s) for the number of nodes the LICENSEE has purchased a license for (as specified in purchase order/invoice/order confirmation or similar) including modified versions, upgrades, updates, DOCUMENTATION, additions, and copies of software. The term SCALI SOFTWARE includes Software in its entirety, including RELEASES, REVISIONS and BUG FIXES, but not DISTRIBUTED SOFTWARE.
- "**DISTRIBUTED SOFTWARE**" shall mean any third-party software products, licensed directly to SCALI or to the LICENSEE by third party and identified as such.
- "**DOCUMENTATION**" shall mean manuals, maintenance libraries, explanatory materials and other publications delivered with the SCALI SOFTWARE or in connection with SCALI BRONZE SOFTWARE MAINTENANCE AND SUPPORT SERVICES . The term "DOCUMENTATION" (can be paper or on-line documentation) does not include specification of Hardware, SCALI SOFTWARE or DISTRIBUTED SOFTWARE.
- A "**RELEASE**" is defined as a completely new program with new functionality and new features over its predecessors identified as such by SCALI according to the ordinary SCALI identification procedures.
- A "**REVISION"** is defined as changes to a program with the aim to improve functionality and to remove deficiencies, identified as such by SCALI according to the ordinary SCALI identification procedures.
- A "**BUG FIX"** is defined as an immediate repair of dysfunctional software, identified as such by SCALI according to the ordinary SCALI identification procedures.
- **"INSTALLATION ADDRESS**" shall mean the physical location of the computer hardware and the location at which SCALI will have installed the SCALI SOFTWARE.
- "**INTELLECTUAL PROPERTY RIGHTS"** includes, but is not limited to all rights to inventions, patents, designs, trademarks, trade names, copyright, copyrighted material, programming, source code, object code, trade secrets and know how.
- "**SCALI REPRESENTATIVE"** shall mean any party authorized by SCALI to import, export, sell, resell or in any other way represent SCALI or SCALI's products.
- **"SHIPPING DATE"** shall mean the date the SCALI SOFTWARE was sent from SCALI or SCALI REPRESENTATIVE to the Licensee.
- **"INSTALLATION DATE"** shall mean the date the SCALI SOFTWARE is installed at the LICENSEE's premises.
- **"COMMENCEMENT DAY"** shall mean the day the SCALI SOFTWARE is made available to LICENSEE by SCALI for installation for permanent use on LICENSEE's computer system (permanent license granted by SCALI.
- **"LICENSEE"** shall mean the formal entity ordering and purchasing the license to use the SCALI SOFTWARE.

- **"CANCELLATION PERIOD"** shall mean the period between SHIPPING DATE AND INSTALLATION DATE, or if installation is not carried out, the period of 30 days after SHIPPING DATE, counted from the first NORWEGIAN WORKING DAYS after SHIPPING DATE.
- **"US WORKING DAYS"** shall mean Monday to Friday, except USA Public Holidays.
- **"US BUSINESS HOURS"** shall mean 9.00 AM to 5.00 PM Eastern Standard Time.
- **"NORWEGIAN WORKING DAYS"** shall mean Monday to Friday, except Norwegian Public Holidays.
- **"NORWEGIAN BUSINESS HOURS"** shall mean 9.00 AM to 5.00 PM Central European Time.
- **"SCALI BRONZE SOFTWARE MAINTENANCE AND SUPPORT SERVICES"** shall mean the Maintenance and Support Services as set out in this CERTIFICATE

## I ATTENTION

**USE OF THE "SCALI SOFTWARE" IS SUBJECT TO THE POSSESSION OF THIS SCALI BRONZE SOFTWARE CERTIFICATE AND THE ACCEPTANCE OF THE TERMS AND CONDITIONS SET OUT HEREIN. THE FOLLOWING TERMS AND CONDITIONS APPLIES TO ALL SCALI SOFTWARE. BY USING THE SCALI SOFTWARE THE LICENSEE EXPRESSLY CONFIRMS THE LICENSEE's ACCEPTANCE OF TERMS AND CONDITIONS SET OUT BELOW.**

**THE SCALI SOFTWARE MAY BE RETURNED TO THE SCALI'S REPRESENTATIVE WITHIN THE END OF CANCELLATION PERIOD IF THE LICENSEE DOES NOT ACCEPT THE TERMS AND CONDITIONS SET OUT IN THIS CERTIFICATE.**

**THE TERMS AND CONDITIONS IN THIS CERTIFICATE ARE DEEMED ACCEPTED UNLESS THE LICENSEE RETURNS THE SCALI SOFTWARE TO SCALI'S REPRESENTATIVE BEFORE THE END OF CANCELLATION PERIOD DEFINED ABOVE.**

**DISTRIBUTED SOFTWARE IS SPECIFIED ON THE URL ADDRESS http://www.scali.com/distributedsw . THE USE OF THE DISTRIBUTED SOFTWARE IS NOT GOVERNED UNDER THE SCOPE OF THIS CERTIFICATE, BUT SUBJECT TO ACCEPTANCE OF THE TERMS AND CONDITIONS SET OUT IN THE SEPARATE LICENSE AGREEMENTS APPLICABLE TO THE RESPECTIVE DISTRIBUTED SOFTWARE IN QUESTION. SUCH LICENSEE AGREEMENTS ARE MADE AVAILABLE AT THE URL ADDRESS http://www.scali.com/distributedsw .**

## II SOFTWARE LICENSE TERMS

### Commencement

This CERTIFICATE is effective from the end of CANCELLATION PERIOD as defined above, unless the SCALI SOFTWARE has been returned to SCALI REPRESENTATIVE or SCALI before the end of CANCELLATION PERIOD.

### Grant of License

Scali grants by this CERTIFICATE to LICENSEE a perpetual, non-exclusive, limited license to use the SCALI SOFTWARE during the term of this CERTIFICATE.

This grant of license shall not constitute any restriction for SCALI to grant a license to any other third party.

### Maintenance

SCALI may, from time to time, produce new REVISIONS and BUG FIXES of the RELEASE of the SCALI SOFTWARE with corrections of errors and defects and expanded or enhanced functionality. For 1 year after COMMENCEMENT DAY, SCALI will provide the LICENSEE with such REVISIONS and BUG FIXES for the purchased SCALI SOFTWARE at the URL address

www.scali.com/download free of charge. The Licensee may request such new REVISIONS and BUG FIXES of the RELEASE, and supplementary material thereof, made available on CD-ROM or paper upon payment of a media and handling fee in accordance with SCALI's pending price list at the time such order is placed.

The above maintenance services may, in certain cases, be excluded from the order placed by non-commercial customers, as defined by SCALI. In such case, the below provisions regarding maintenance does not apply for such non-commercial customers.

**Support**

For 1 year after COMMENCEMENT DAY, the LICENSEE may request technical assistance in accordance with the terms and conditions current from time to time for the SCALI BRONZE SOFTWARE MAINTENANCE AND SUPPORT SERVICES as set out below. Upon additional payment in accordance with the current price list from time to time, and acceptance of the specific terms and conditions related thereto, the LICENSEE may request prolonged or upgraded support services in accordance with the support policies made available from time to time by SCALI.

The above support services may, in certain cases, be excluded from the order placed by non-commercial customers, as defined by SCALI. In such case, the below provisions regarding support does not apply for such non-commercial customers.

***Restrictions in the use of the SCALI SOFTWARE***

LICENSEE may not modify or tamper the content of any of the files of the software or the online documentation or other deliverables made available by SCALI or SCALI REPRESENTATIVE, without the prior written authorization by SCALI.

The SCALI SOFTWARE contains proprietary algorithms and methods. LICENSEE may not attempt to;

- reverse engineer, decompile, disassemble or modify; or
- make any attempt to discover the source code of the SCALI SOFTWARE or create derivative works form such; or
- use a previous version or copy of the SCALI SOFTWARE after an updated version has been made available as a replacement of the prior version. Upon updating the SCALI SOFTWARE, all copies of prior versions shall be destroyed.
- translate, copy, duplicate or reproduce for any other purpose than for backup for archival purposes.

LICENSEE may only make copies or adaptations of the SCALI SOFTWARE for archival purposes or when copying or adaptation is an essential step in the authorized use of the SCALI SOFTWARE. LICENSEE must reproduce all copyright notices in the original SCALI SOFTWARE on all copies or adaptations. LICENSEE may not copy the SCALI SOFTWARE onto any public network.

**License Manager**

The SCALI SOFTWARE is operated under the control of a license manager, which is controlling the access and licensed usage of the SCALI SOFTWARE. LICENSEE may not attempt to modify or tamper with any function of this license manager.

**Sub-license and distribution**

LICENSEE may not sub-license, rent or lease the SCALI SOFTWARE partly or in whole, or use the SCALI SOFTWARE in the manner neither of a service bureau nor as an Application Service Provider unless specifically agreed to in writing by SCALI.

LICENSEE is permitted to print and distribute paper copies of the unmodified online documentation freely. In this case LICENSEE may not charge a fee for any such distribution.

**Export Requirements**

LICENSEE may not export or re-export the SCALI SOFTWARE or any copy or adaptation in violation of any applicable laws or regulations.

## III SCALI SERVICES TERMS

### SCALI BRONZE SOFTWARE MAINTENANCE AND SUPPORT SERVICES

Unless otherwise specified in the purchase order placed by the LICENSEE, SCALI shall provide SCALI BRONZE SOFTWARE MAINTENANCE AND SUPPORT SERVICES in accordance with its maintenance and support policy as referred to in this Clause and the Clause "SCALI's Obligations" hereunder, which includes error corrections, RELEASES, REVISIONS and BUG FIXES to the RELEASE of the SCALI SOFTWARE.

For customers in the Americas, SCALI shall provide technical assistance via E-mail on US WORKING DAYS from, 9.00 AM to 5.00 PM Eastern Standard Time.

For customers in the Americas, SCALI shall respond to the LICENSEE via e-mail and start technical assistance and error corrections within eight (8) US BUSINESS HOURS after the error or defect has been reported to SCALI via SCALI'S standard problem report procedures as defined from time to time on the url-address www.scali.com/support.

For customers outside the Americas, SCALI shall provide technical assistance via E-mail on NORWEGIAN WORKING DAYS from, 9.00 AM to 5.00 PM Central European Time.

For customers outside the Americas, SCALI shall respond to the LICENSEE via e-mail and start technical assistance and error corrections within eight (8) NORWEGIAN BUSINESS HOURS after the error or defect has been reported to SCALI via SCALI'S standard problem report procedures as defined from time to time on the url-address www.scali.com/support.

#### SCALI's Obligations

In the event that the LICENSEE detects any significant error or defect in the SCALI SOFTWARE, SCALI, in accordance with the standard warranty of the Scali Software License granted to the LICENSEE, undertakes to repair, replace or provide an adequate work-around of the SCALI SOFTWARE installed at the INSTALLATION ADDRESS within the response times listed in the Clause "SCALI BRONZE SOFTWARE MAINTENANCE AND SUPPORT SERVICES" above.

SCALI may provide a fix or update to the SCALI SOFTWARE in the normal course of business according to SCALI's scheduled or unscheduled new REVISIONS of the SCALI SOFTWARE. SCALI will provide, at the LICENSEE's request, a temporary fix for non-material errors or defects until the issuance of such NEW REVISION.

The services covered by this CERTIFICATE will be provided only for operation of the SCALI SOFTWARE. SCALI will provide services for the SCALI SOFTWARE only on the release level current at the time of service and the immediately preceding release level. SCALI may, within its sole discretion, provide support on previous release levels, for which the LICENSEE may be required to pay SCALI time and materials for the services rendered.

Should it become impossible to maintain the LICENSEE's RELEASE of the SCALI SOFTWARE during the currency of this CERTIFICATE, then SCALI may in its sole discretion, upon giving the LICENSEE 3 (three) months written notice to the effect, upgrade the SCALI SOFTWARE at the INSTALLATION ADDRESS to any later release of the SCALI SOFTWARE such that it can once again be maintained.

#### LICENSEE's Obligations

The LICENSEE shall notify SCALI in writing via the SCALI Standard Problem Report Procedure as defined from time to time on the url-address www.scali.com/support , following the discovery of any error or defect in the SCALI SOFTWARE or otherwise if support services by SCALI are requested.

The LICENSEE shall provide to SCALI a comprehensive listing of output and all such other data that SCALI may request in order to reproduce operating conditions similar to those present when the error or defect occurred or was discovered. In the event that it is determined that the problem was due to LICENSEE error in the use of the SCALI SOFTWARE, or otherwise not

related to, referring to or caused by SCALI SOFTWARE, then the LICENSEE shall pay SCALI's standard commercial time rates for all off-site and eventually any on-site services provided plus actual travel and per diem expenses relating to such services.

## IV GENERAL TERMS

### Fees for SCALI Software License and SCALI SOFTWARE MAINTENANCE AND SUPPORT SERVICES

Fees for the SCALI SOFTWARE, License and SCALI SOFTWARE MAINTENANCE AND SUPPORT SERVICES to be paid by the LICENSEE to SCALI under this CERTIFICATE are determined based on the current Scali Product Price List from time to time

Any requests for services or problems reported to SCALI, which in the opinion of SCALI are clearly defined as professional services not included in the payment made under the scope of this CERTIFICATE, including but not limited to;

- on-Site support;
- "tuning" and machine optimization;
- problems related to Hardware and Software not delivered by SCALI;
- backup Processing;
- installation of any software, including newer releases;
- consultancy and Training;
- DISTRIBUTED SOFTWARE;

shall be at SCALI's then prevailing prices, policies (several support levels, hereunder fees referring thereto, may be offered by SCALI), terms and conditions for such services. SCALI will, however, advise the LICENSEE of any such requests and obtain an official company order from the LICENSEE before executing the said request**.**

Any out of pocket expenses directly relating to the services rendered, and not included in the payment made under the scope of this CERTIFICATE, such as;

- travel and accommodation;
- per diem allowances as per Norwegian travel regulations;
- Internet connection fees; and
- cost of Internet access

shall be paid in addition to the total purchase price for this CERTIFICATE, and payable by the LICENSEE within their normal accepted terms with SCALI upon presentation of an invoice, which shall be at the value incurred and without any form of mark-up. The LICENSEE undertakes to arrange and cover all accommodation requirements that arise out of, or in conjunction with, this CERTIFICATE.

### Title to INTELLECTUAL PROPERTY RIGHTS

SCALI, or any part identified as such by SCALI, is the sole proprietor and holds all powers, hereunder but not limited to exploit, use, make any changes and amendments of all INTELLECTUAL PROPERTY RIGHTS, related to the SCALI SOFTWARE, or its parts, and any new version, hereunder but not limited to REVISION, BUG FIX or NEW RELEASES of the SCALI SOFTWARE or its parts, as well as to all other INTELLECTUAL PROPERTY RIGHTS resulting from the co-operation within the frame of this CERTIFICATE.

The LICENSEE hereby declares to respect title to INTELLECTUAL PROPERTY RIGHTS as set out above also after the expiration, termination or transfer of this CERTIFICATE, independent of the cause for such expiration, termination or transfer.

### Transfer

SCALI may transfer this CERTIFICATE to any third party. The LICENSEE may transfer this CERTIFICATE to a third party, upon the Transferee's written acceptance in advance of being

fully obliged by the terms and conditions set out in this CERTIFICATE and SCALI'S prior written approval of the transfer. SCALI's approval shall anyway be deemed granted unless contrary notice is sent from SCALI within 7 NORWEGIAN WORKING DAYS from receipt of notification of the transfer in question from the LICENSEE.

Upon transfer, LICENSEE must deliver the SCALI SOFTWARE, including any copies and related documentation, to the Transferee.

**Compliance with Licenses**

LICENSEE shall upon request from SCALI or its authorized representatives, within 30 days following the receipt of such request, fully document and certify that the use of the SCALI SOFTWARE is in accordance with this CERTIFICATE. If the LICENSEE fails to fully document that this CERTIFICATE is suitable and sufficient for the LICENSEE's use of the SCALI SOFTWARE, SCALI will use any legal measure to protect its ownership and rights in its SCALI SOFTWARE and to seek monetary damages from LICENSEE.

**Warranty of Title and Substantial Performance**

SCALI hereby represents and warrants that SCALI is the owner of the SCALI SOFTWARE.

SCALI hereby warrants that the SCALI SOFTWARE will perform substantially in accordance to the DOCUMENTATION for the ninety (90) day period following the LICENSEE's receipt of the SCALI SOFTWARE ("Limited Warranty"). To make a warranty claim, the LICENSEE must return the products to the location the SCALI SOFTWARE was purchased ("Back-to-Base") within such ninety- (90) day period. Any supplements or updates to the SCALI SOFTWARE, including without limitation, any service packs or hot fixes provided to the LICENSEE after the expiration of the ninety (90) day Limited Warranty period are not covered by any warranty or condition, express, implied or statutory.

If an implied warranty or condition is created by the LICENSEE's state/jurisdiction and federal or state/provincial law prohibits disclaimer of it, the LICENSEE also has an implied warranty or condition, but only as to defects discovered during the period of this Limited Warranty (ninety days). As to any defects discovered after the ninety (90) day period, there is no warranty or condition of any kind.

***Disclaimer of Warranty***

**Except for the limited warranty under the Clause "Warranty of Title and Substantial Performance" above, and to the maximum extent permitted by applicable law, SCALI and SCALI REPRESENTATIVES provide SCALI SOFTWARE and SCALI SOFTWARE MAINTENANCE AND SUPPORT SERVICES, if any, *as is* and with *all faults,* and hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness or responses, of results, of workmanlike effort, of lack of viruses and of lack of negligence, all with regard to the software or other deliverables by SCALI. Also, there is no warranty or condition of title, quiet enjoyment, quiet possession, correspondence to description or non-infringement with regard to the SCALI SOFTWARE or the provision of or failure to provide SCALI SOFTWARE MAINTENANCE AND SUPPORT SERVICES, SCALI does not warrant any title, performance, compatibility, co-operability or other functionality of the DISTRIBUTED SOFTWARE or other deliverables by SCALI.**

**Without limiting the generality of the foregoing, SCALI specifically disclaims any implied warranty, condition, or representation that the SCALI SOFTWARE;**

- **shall correspond with a particular description;**
- **are of merchantable quality;**
- **are fit for a particular purpose; or**
- **are durable for a reasonable period of time.**

**Nothing in this CERTIFICATE shall be construed as;**

- **a warranty or representation by SCALI as to that anything made, used, sold or otherwise disposed of under the license granted in the CERTIFICATE is or will be free from infringement of patents, copyrights, TRADEMARKS, industrial design or other INTELLECTUAL PROPERTY RIGHTS ; or**
- **an obligation by SCALI to bring or prosecute or defend actions or suits against third parties for infringement of patents, copyrights, trade-marks, industrial designs or other INTELLECTUAL PROPERTY or contractual rights.**

**Licensee's Exclusive Remedy**

In the event of any breach or threatened breach of this CERTIFICATE, hereunder the foregoing representation and warranty, the LICENSEE's sole remedy shall be to require SCALI and its SCALI REPRESENTATIVE's to either;

- procure, at SCALI's expense the right to use the SCALI SOFTWARE; or
- replace the SCALI SOFTWARE or any part thereof that is in breach and replace it with software of comparable functionality that does not cause any breach; or
- refund to the LICENSEE the full amount of the total purchase price paid by the LICENSEE for this CERTIFICATE upon the return of the SCALI SOFTWARE and all copies thereof to SCALI, deducted with the amount equivalent to the license and other services rendered until the matter causing the remedy in question occurred.

THE LICENSEE will receive the remedy elected by SCALI without charge, except that The LICENSEE is responsible for any expenses the LICENSEE may incur (e.g. cost of shipping the SCALI SOFTWARE to SCALI). Any commitment or obligation of SCALI to remedy LICENSEE in accordance with this CERTIFICATE is void if failure of the SCALI SOFTWARE or other breach of the CERTIFICATE has resulted from accident, abuse, misapplication, abnormal use or a virus. Any replacement SCALI SOFTWARE will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. Neither these remedies nor any product maintenance and support services offered by SCALI are available without proof of purchase directly from SCALI or through a SCALI REPRESENTATIVE. To exercise the LICENSEE's remedy, contact: SCALI as set out in ULR address WWW.scali.com or the SCALI REPRESENTATIVE serving the LICENSEE's district.

**Limitation on Remedies and Liabilities**

The LICENSEE's exclusive and maximum remedy for any breach of the CERTIFICATE is as set forth above. Except for any refund elected by SCALI, the LICENSEE is not entitled to any damages, including but not limited to consequential damages, if the SCALI SOFTWARE does not meet the DOCUMENTATION or SCALI otherwise does not meet the CERTIFICATE, and, to the maximum extent allowed by applicable law, even if any remedy fails of its essential purpose.

To the maximum event permitted by applicable law, in no event shall SCALI or SCALI REPRESENTATIVES be liable for any special, incidental, indirect or consequential damages whatsoever (including, but not limited to, damages for loss of profits or confidential or other information, for business interruption, for personal injury, for loss of privacy, for failure to meet any duty including of good faith or of reasonable care, for negligence, and for any other pecuniary or other loss whatsoever) arising out of or in any way related to the use of or inability to use the SCALI SOFTWARE, the provision of or failure to provide maintenance and support services, or otherwise under or in connection with any provision of this CERTIFICATE, even in the event of the fault, tort (including negligence), strict liability, breach of contract or breach of warranty of SCALI or a SCALI REPRESENTATIVE, and even if SCALI or a SCALI REPRESENTATIVE has been advised of the possibility of such damages.

No action, whether in contract or tort (including negligence), or otherwise arising out of or in connection this CERTIFICATE may be brought more than six months after the cause of action has occurred.

***Termination.***

SCALI has the right to terminate this CERTIFICATE with immediate effect if the LICENSEE breaches or is in default of any obligation hereunder which default is incapable of cure or which, being capable of cure, has not been cured within fifteen (15) days after receipt of notice of such default (or such additional cure period as the non-defaulting party may authorize).

SCALI may terminate this CERTIFICATE with immediate effect by written notice to the LICENSEE and may regard the LICENSEE as in default of this CERTIFICATE, if the LICENSEE substantially breaches the CERTIFICATE, becomes insolvent, makes a general assignment for the benefit of its creditors, files a voluntary petition of bankruptcy, suffers or permits the appointment of a receiver for its business or assets, or becomes subject to any proceeding under the bankruptcy or insolvency law, whether domestic or foreign, or has wound up or liquidated, voluntarily or otherwise. In the event that any of the above events occur, the LICENSEE shall immediately notify SCALI of its occurrence.

In the event that either party is unable to perform any of its obligations under this CERTIFICATE or to enjoy any of its benefits because of (or if loss of the Services is caused by) natural disaster, action or decreed or governmental bodies or communication line failure not the fault of the affected party (normally and hereinafter referred to as a "FORCE MAJEURE EVENT") the party who has been so affected shall immediately give notice to the other party and shall do everything possible to resume performance. Upon receipt of such notice, all obligations under this CERTIFICATE shall be immediately suspended. If the period of non-performance exceeds twenty-one (21) days from the receipt of notice of the FORCE MAJEURE EVENT, the party whose performance has not been so affected may, by giving written notice, terminate this CERTIFICATE with immediate effect.

In the event that this CERTIFICATE is terminated for any reason the LICENSEE shall destroy all data, materials, and other properties of SCALI then in its possession, provided as a consequence of this CERTIFICATE, hereunder but not limited to SCALI SOFTWARE, copies of the software, adaptations and merged portions in any form.

**Proprietary Information**

The LICENSEE acknowledges that all information concerning SCALI that is not generally known to the public is "CONFIDENTIAL AND PROPRIETARY INFORMATION". THE LICENSEE agrees that it will not permit the duplication, use or disclosure of any such CONFIDENTIAL AND PROPRIETARY INFORMATION to any person (other than its own employees who must have such information for the performance of their obligations under this CERTIFICATE), unless authorized in writing by SCALI.

These confidentiality obligations survive the expiration, termination or transfer of this Certificate, independent of the cause for such expiration, termination or transfer.

**Miscellaneous**

The Headings and Clauses of this CERTIFICATE are intended for convenience only and shall in no way affect their interpretation. Words importing natural persons shall include bodies corporate and other legal personae and vice versa. Any particular gender shall mean the other gender, and vice-versa. The singular shall include the plural and vice-versa.

All remedies available to either party for the breach of this CERTIFICATE are cumulative and may be exercised concurrently or separately, and the exercise of any one remedy shall not be deemed an election of such remedy to the exclusion of other remedies.

Any invalidity, in whole or in part, of any of the provisions of this CERTIFICATE shall not affect the validity of any other of its provisions.

Any notice or other communication hereunder shall be in writing.

No term or provision hereof shall be deemed waived and no breach excused unless such waiver or consent shall be in writing and signed by the party claimed to have waived or consented.

**Governing Law**

This CERTIFICATE shall be governed by and construed in accordance with the laws of Norway, with Oslo City Court (Oslo tingrett) as proper legal venue.

# Table of contents

# Chapter 1 Introduction

This manual describes Scali MPI Connect (SMC) in detail. SMC is sold as a separate stand-alone product, with an SMC distribution, and integrated with Scali Manage in the SSP distribution. Some integration issues and features of the MPI are also discussed in the Scali Manage Users Guide, the user's manual for Scali Manage.

This manual is written for users who have a basic programming knowledge of C or Fortran, as well as an understanding of MPI.

## 1.1 Scali MPI Connect product context

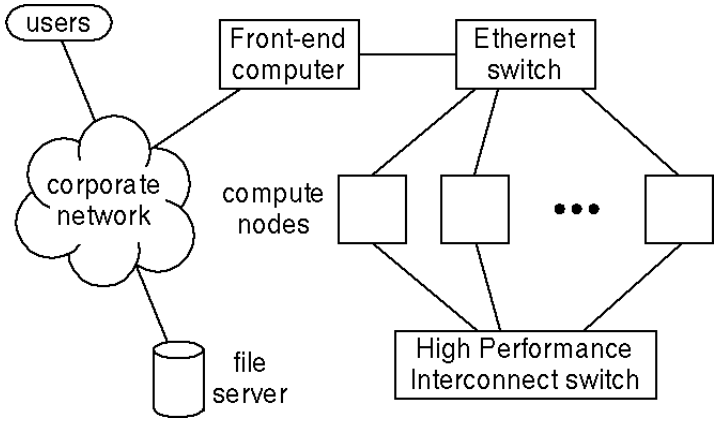



**Figure 1-1:** A cluster system

**Figure 1-1:** shows a simplified view of the underlying architecture of clusters using Scali MPI Connect: A number of compute nodes are connected together in a Ethernet network through which a front-end interfaces the cluster with the corporate network. A high performance interconnect can be attached to service communication requirements of key applications**.** The front-end imports services like file systems from the corporate network to allow users to run applications and access their data.
Scali MPI Connect implements the MPI standard for a number of popular high performance interconnects, like Gigiabit Ethenet, Infiniband, Myrinet and SCI.
While the high performance interconnect is optional, the networking infrastructure is mandatory. Without it the nodes in the cluster will have no way of sharing resources. TCP/IP functionality implemented by the Ethernet network enables the front-end to issue commands to the nodes, provide them with data and application images, and collect results from the processing the nodes perform.

The Scali Software Platform provides the necessary software components to combine a number of commodity computers running Linux into a single computer entity, henceforth called a cluster.

Scali is targeting its software at users involved in High Performance Computing, also known as supercomputing, which typically includes CPU-intensive parallel applications. Scali aims to produce software tools which assist its users in maximizing the power and ease of use of the computing hardware purchased.

CPU-intensive parallel applications are programmed using a programming library called MPI (Message Passing Interface), the state-of-the-art library for high performance computing. Note that the MPI library is NOT described within this manual; MPI is defined by a standards committee, and the API, along with guides for its use is available free of charge on the Internet. A link to the MPI Standard and other MPI resources can be found in chapter 7, "Related documentation", and on Scali's web site, **http://www.scali.com**.

Scali MPI Connect (SMC) consists of Scali's implementation of the MPI programming library and the necessary support programs to launch and run MPI applications. This manual often uses the term ScaMPI to refer to the specifics of the MPI itself, and not the support applications. Please note that in earlier releases of Scali Software Platform (SSP), the term ScaMPI was often used to refer to the parts of SSP which are now called SMC.

SSP is the complete cluster management solution, and includes a GUI, full remote management, power control, remote console and monitoring functionality, as well as a full OS+Scali Manage install/reinstall utility. While we strive to make SSP as simple and painless to use as possible, SMC as a stand-alone product is the bare minimum for MPI usage, and requires that the user installs another management solution. Please note that SMC continues to be included in SSP; at no time should they be installed together, and SSP and SMC distributions should never be mixed within a single cluster.

## 1.2 Support

### 1.2.1 Scali mailing lists

Scali provides two mailing lists for support and information distribution. For instructions on how to subscribe to a mailing list (i.e., **scali-announce** or **scali-user**), please see the *Mailing Lists* section of **http://www.scali.com/**.

### 1.2.2 SMC FAQ

An updated list of Frequently Asked Questions is posted on **http://www.scali.com**. In addition, for users who have installed SMC, the version of the FAQ that was current when SMC was installed is available as a text file in **/opt/scali/doc/ScaMPI/FAQ.**

### 1.2.3 SMC release documents

When SMC has been installed, a number of smaller documents such as the FAQ, RELEASE NOTES, README, SUPPORT, LICENSE_TERMS, INSTALL are available as text files in the **/opt/scali/doc/ScaMPI** directory.

### 1.2.4 Problem reports

Problem reports should, whenever possible, include both a description of the problem, the software version(s), the computer architecture, a code example, and a record of the sequence of events causing the problem. In particular, 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.**

### 1.2.5 Platforms supported

SMC is available for a number of platforms. For up-to-date information, please see the *SMC* section of **http://www.scali.com/**. For additional information, please contact Scali at **sales@scali.com**.

### 1.2.6 Licensing

SMC is licensed using Scali license manager system. In order to run SMC 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.**

### 1.2.7 Feedback

Scali appreciates any suggestions users may have for improving both this Scali MPI Connect User's Guide and the software described herein. Please send your comments by e-mail to **support@scali.com.**

Users of parallel tools software using SMC on a Scali System are also 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 also provides for communication between software authors and users.

## 1.3 How to read this guide

This guide is written for skilled computer users and professionals. It is assumed that the reader is familiar with the basic concepts and terminology of computer hardware and software since none of these will be explained in any detail. Depending on your user profile, some chapters are more relevant than others.

## 1.4 Acronyms and abbreviations

| Abbreviation | Meaning |
|---|---|
| AMD64 | The 64 bit Instruction set arcitecture (ISA) that is the 64 bit extention to the Intel x86 ISA. Also known as x86-64. The Opteron and Athlon64 from AMD are the first implementations of this ISA. |
| DAPL | Direct Access Provider Library DAT Instantiation for a given interconnect |
| DAT | Direct Access Transport - Transport-independent, platform-independent Application Programming Interfaces that exploit RDMA |
| DET | Direct Ethernet Transport - Scali's DAT implementation for Ethernet-like devices, including channel aggregation |
| EM64T | The Intel implementation of the 64 bit extention to the x86 ISA. Also See AMD64. |
| GM | A software interface provided by Myricom for their Myrinet interconnect hardware. |
| HCA | Hardware Channel Adapter. Term used by Infiniband vendors referencing to the hardware adapter. |
| HPC | High Performance Computer |
| IA32 | Instruction set Architecture 32 Intel x86 architecture |

Table 1-1: Acronyms and abbreviations

| Abbreviation | Meaning |
|---|---|
| IA64 | Instruction set Architecture 64 Intel 64-bit architecture, Itanium, EPIC |
| Infiniband | A high speed interconnect standard available from a number of vendors |
| MPI | Message Passing Interface - De-facto standard for message passing |
| Myrinet™ | An interconnect developed by Myricom. Myrinet is the product name for the hardware. (See GM). |
| NIC | Network Interface Card |
| OEM | Original Equipment Manufacturer |
| Power | A generic term that cover the PowerPC and POWER processor families. These processors are both 32 and 64 bit capable. The common case is to have a 64 bit OS that support both 32 and 64 bit executables. See also PPC64 |
| PowerPC | The IBM/Motorola PowerPC processor family. See PPC64 |
| POWER | The IBM POWER processor family. Scali support the 4 and 5 versions. See PPC64 |
| PPC64 | Abbreviation for PowerPC 64, which is the common 64 bit instruction set architecture(ISA) name used in Linux for the PowerPC and POWER processor families. These processors have a common core ISA that allow one single Linux version to be made for all three processor families. |
| RDMA | Remote DMA Read or Write Data in a remote memory at a given address |
| ScaMPI | Scali's MPI - First generation MPI Connect product, replaced by SMC |
| SCI | Scalable Coherent Interface |
| SMC | Scali MPI Connect - Scali's second generation MPI |
| SMI | Scali Manage Install - OS installation part of Scali Manage |
| SSP | Scali Software Platform is the name of the bundling of all Scali software packages. |
| SSP 3.x.y | First generation SSP - WulfKit, Universe, Universe XE, ClusterEdge |
| SSP 4.x.y | Second generation SSP - Scali Manage + SMC (option) |
| VAR | Value Added Reseller |
| x86-64 | see AMD64 and EM64T |

Table 1-1: Acronyms and abbreviations

## 1.5 Terms and conventions

Unless explicitly specified otherwise, gcc (gnu c-compiler) and bash (gnu Bourne-Again-SHell) are used in all examples.

| Term | Description. |
|---|---|
| Node | A single computer in an interconnected system consisting of more than one computer |
| Cluster | A cluster is a set of interconnected nodes with the aim to act as one single unit |
| torus | greek word for ring, used in Scali documents in the context of 2- and 3-dimensional interconnect topologies |
| Scali system | A cluster consisting of Scali components |
| Front end | A computer outside the cluster nodes dedicated to run configuration, monitoring and licensing software |
| MPI process | Instance of application program with unique rank within MPI_COMM_WORLD |
| UNIX | Refers to all UNIX and lookalike OSes supported by the SSP, i.e. Solaris and Linux. |
| Windows | Refers to Microsoft Windows 98/Me/NT/2000/XP |

Table 1-2: Basic terms

## 1.6 Typographic conventions

| Term | Description. |
|---|---|
| **Bold** | Program names, options and default values |
| *Italics* | User input |
| `mono spaced` | Computer related: Shell commands, examples, environment variables, file locations (directories) and contents |
| **GUI style font** | Refers to Menu, Button, check box or other items of a GUI |
| **#** | Command prompt in shell with super user privileges |
| `%` | Command prompt in shell with normal user privileges |

Table 1-3: Typographic conventions

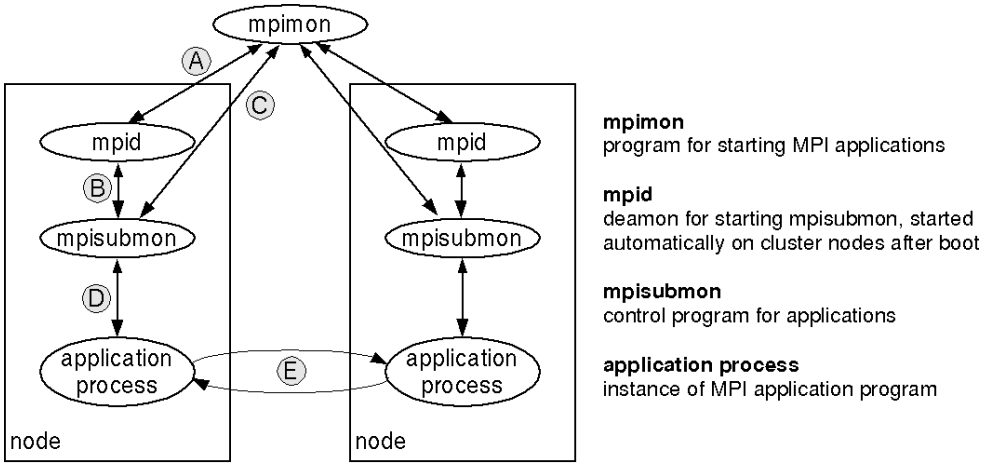
# Chapter 2 Description of Scali MPI Connect

This chapter gives the details of the operations of Scali MPI Connect (SMC). SMC consists of libraries to be linked and loaded with user application program(s), and a set of executables which control the start-up and execution of the user application program(s). The relationship between these components and their interfaces are described in this chapter. It is necessary to understand this chapter in order to control the execution of parallel processes and be able to tune Scali MPI Connect for optimal application performance.

## 2.1 Scali MPI Connect components

Scali MPI Connect consists of a number of programs, daemons, libraries, include and configuration files that together implements the MPI functionality needed by applications. Starting applications rely on the following daemons and launchers:

- **mpimon** is a monitor program which is the user's interface for running the application program.
- **mpisubmon** is a submonitor program which controls the execution of application programs. One submonitor program is started on each node per run.
- **mpiboot** is a bootstrap program used when running in manual-/debug-mode.
- **mpid** is a daemon program running on all nodes that are able to run SMC. **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




A: mpimon contacts mpid on each node to start the application
B: mpid starts a mpisubmon to control the application process
C: mpisubmon and mpimon communicate to establish application's context
D: mpisubmon starts the application process and feeds it information through MPI_Init()
E: instances of application processes communicate with each other using MPI

A and B are only used during startup.
C and D remain through the lifetime of the application run.

**Figure 2-1:** The way from application startup to execution

Figure 2-1: illustrates how applications started with mpimon have their communication system established by a system of daemons on the nodes. This process uses TCP/IP communication over the networking Ethernet, whereas optional high performance interconnects are used for communication between processes.

Parameter control is performed by **mpimon** to check as many of the specified options and parameters as possible. The user program names are checked for validity, and the nodes are contacted (using sockets) to ensure they are responding and that **mpid** is running.
Via mpid mpimon establishes contact with the nodes and transfers basic information to enable mpid to start the submonitor **mpisubmon** on each node. Each submonitor establishes a connection to mpimon for exchange of control information between each mpisubmon and mpimon to enable mpisubmon to start the specified userprograms (MPI-processes).
As **mpisubmon** starts all the MPI-processes to be executed they MPI_Init(). Once inside here the user processes wait for all the mpisubmons inovolved to coordinate via mpimon. Once all processes are ready mpimon will return a "start running" message to the processes. They will then return from MPI_Init() and start executing the user code.

Stopping MPI application programs is requested by the user processes as they enter the MPI_Finalize() call. The local mpisubmon will signal mpimon and wait for mpimon to return a "all stopped message". This comes when all processes are waiting in MPI_Finalize(). As the user processes return from the MPI_Finalize() they release their resources and terminates. Then the local mpisubmon terminates and eventuall mpimon terinates.

## 2.2 SMC network devices

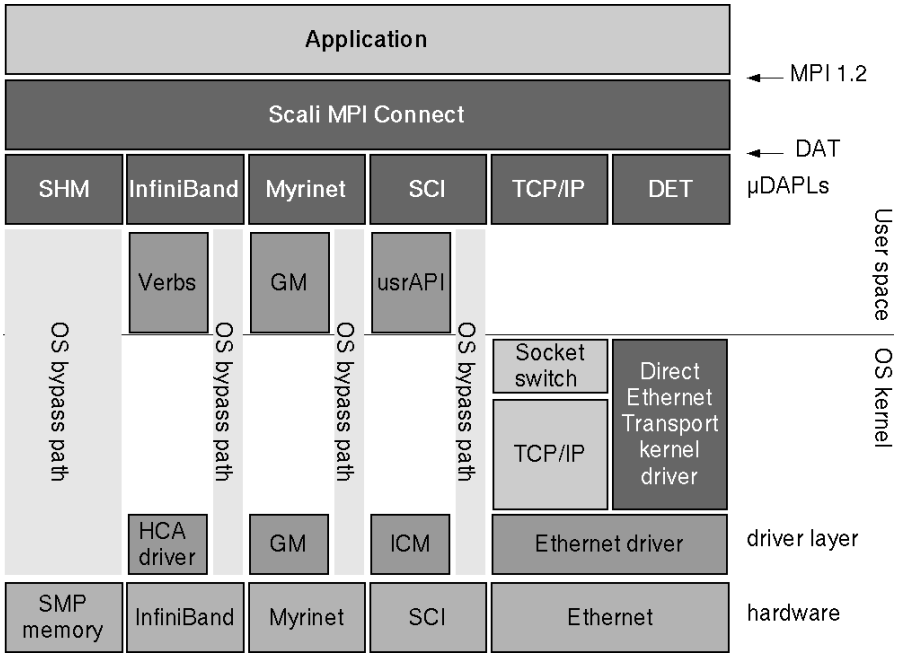



**Figure 2-2:** Scali MPI Connect relies on DAT to interface to a number of interconnects

Beginning with SSP 4.0.0 and SMC 4.0.0 the Scali MPI offers generic support for interconnects. This does not yet mean that every interconnect is supported out of the box, since SMC still requires a driver for each interconnect. But from SMC's point of view, a driver is just a

library, which in turn may (e.g. Myrinet or SCI) or may not require a kernel driver (e.g. TCP/IP). These *provider libraries* provide a *network device* to SMC.

## 2.2.1 Network devices

There are two basic types of network devices in SMC, native and DAT. The native devices are built-in and are neither replaceble nor upgradable without replacing the Scali MPI Connect package. There are currently five built in devices, SMP, TCP, IB, GM and SCI; the Release Notes included with the Scali MPI Connect package should have more details on this issue.

To find out what network device is used between two processes, set the environment variable SCAMPI_NETWORKS_VERBOSE=2. With value 2 the MPI library will print out during startup a table over every process and what device it's using to every other process.

### 2.2.1.1 Direct Access Transport (DAT)

The other type of devices use the DAT uDAPL API in order to have an open API for generic third party vendors. uDAPL is an abbrevation for User DAT Provider library. This is a shared library that SMC loads at runtime through the static DAT registry. These libraries are normally listed in **/etc/dat.conf**. For clusters using 'exotic' interconnects whose vendor provides a uDAPL shared object, these can be added to this file (if this isn't done automatically by the vendor). The device name is given by the uDAPL, and the interconnect vendor must provide it. Please note that Scali has a *certification program*, and may not provide support for unknown third party vendors.

The DAT header files and registry library conforming to the uDAPL v1.1 specification, is provided by the dat-registry package.

For more information on DAT, please refer to **http://www.datcollaborative.org.**

## 2.2.2 Shared Memory Device

The SMP device is a shared memory device that is used exclusively for intra-node communication and use SYS V IPC shared memory. Mulit CPU nodes are frequent in clusters, and SMP provide optimal communication between the CPUs. In cases where only one processor per node is used, SMP is not used.

## 2.2.3 Ethernet Devices

An Ethernet for networking is a basic requirement fpr a cluster. For some uses this also has enough performance for carrying application communication. To serve this Scali MPI Connect has a TCP device. In addition there are Direct Ethernet Transport (DET) devices which implement a protocol devised by Scali for aggregating multiple TCP-type interconnects.

### 2.2.3.1 TCP

The TCP device is really a generic device that works over any TCP/IP network, even WANs. This network device requires only that the node names given to mpimon map correctly to the nodes IP address. TCP/IP connectivity is required for SMC operation, and for this reason the TCP device is always perational.

**Note:** Users should always append the TCP device at the end of a devicelist as the device of last resort. This way communication will fall back to the management ethernet that anyway has to be present for the cluster to work.

#### 2.2.3.2 DET

Scali has developed a device called Direct Ethernet Transport (*DET*) to improve Ethernet performance. This device that bypasses the TCP/IP stack and uses raw Ethernet frames for sending messages. These devices are bondable over multiple Ethernets.

The **/opt/scali/sbin/detctl command** provides a means of creating and deleting DET devices. **/opt/scali/bin/detstat** can be used to obtain statistics on the devices.

#### 2.2.3.3 Using detctl

**detctl** has the following syntax :

```
detctl -a [-q] [-c] <hca index> <device1> <device2> ...
       -d [-q] <hca index>
       -l [-q]
```

Examples:

- Adding new DET devices temporarily with the detctl utility:
  `root# detctl -a 0 eth0` # creates a det0 device using eth0 as transport device.
  `root# detctl -a 1 eth1 eth2` # creates a det1 device using eth1 and eth2 as aggregated transport devices.
- Removing DET devices with detctl:
  root# detctl -d 0 # removes DET device 0 (det0) from the current configuration.
  root# detctl -d 1 # removes DET device 1 (det1) from the current configuration.
- Listing active DET devices
  root# detctl -l # lists all DET devices currently configured.

Please note that aggregating devices usually requires a special switch configuration.
Both devices have the same Ethernet address (MAC), and so there must either be one VLAN for the eth1's and another for the eth2's, or all the eth1's must be on one Ethernet switch, and all the eth2's on another switch.

Using **detctl** to add and remove devices is not permanent, as the contents of the **/opt/scali/kernel/scadet.conf** configuration file takes presedence. The contents of this file has the following format:

```
# hca     <hca index>   <ethernet devices>
```

Permanent changes must be done by editing **opt/scali/kernel/scadet.conf** , e.g., to add permanently the example above add the following lines:

```
hca       0             eth0
hca       1             eth1 eth2
```

#### 2.2.3.4 Using detstat

To gather transmission statistics (packets transmitted, received and lost) for a DET device use **detstat.** It can also be used to reset the statistics for DET devices.
detstat has the following syntax :

```
detstat [-r] [-a] <hca name>
```

Examples:

- root# detstat det0 #listing statistics for the det0 device, if it exists.
- root# detstat -a #listsing statistics for all existing DET devices.

- root# detstat -r det0 # reset statistics for the det0 device.
- root# detstat -r -a # resets statistics for all DET devices.

## 2.2.4 Myrinet

### 2.2.4.1 GM

This is a RDMA capable device that uses the Myricom GM driver and library. A GM release above 2.0 is required. This device is straight forward and requires no configuration other than the presence of the libgm.so library in the library path (see /etc/ld.so.conf).

**Note:**
*Myricom GM software is not provided by Scali. If you have purchased a Myrinet interconnect you have the right to use the GM source, and a source tar ball is available from Myricom. It is necessary to obtain the GM source since it must be compiled per kernel version. Scali provides tools for generating binary RPMs to ease installing and management. These tools are provided in the scagmbuilder package; see the Release Notes/Readme file for detailed instructions.*

*If you used Scali Manage to install your compute nodes, and supplied it with the GM source tar ball, the installation is already complete.*

## 2.2.5 Infiniband

### 2.2.5.1 IB

Infiniband is a relatively new interconnect that has been available since 2002, and became affordable in 2003. On PCI-X based systems you can expect latencies around 5υS and bandwidth up to 700-800Mb/s (please note that performance results may vary based on processors, memory sub system, and the PCI bridge in the chipsets).

There are various Infiniband vendors that provide slightly different hardware and software environments. Scali have established relationships with the following vendors: Mellanox, Silverstorm, Cisco, and Voltaire.
See release notes on the exact versions of software stack that is supported. Scali provide a utility known as ScaIBbuilder that does an automated install of some of these stacks. (See IBbuilders release notes).

The different vendors' InfiniBand switches vary in feature sets, but the most important difference is whether they have a built in subnet manager or not. An InfiniBand network must have a subnet manager (SM) and if the switches don't come with a builtin SM, one has to be started on a node attached to the IB network. The SMs of choice for software SMs are **OpenSM** or **minism**. If you have SM-less switches your vendor will provide one as part of their software bundle.

SMC uses either the uDAPL (User DAT Provider Library) supplied by the IB vendor, or the low level VAPI/IBA layer. DAT is an established standard and is guaranteed to work with SMC. However better performance is usually achieved with the VAPI/IBT interfaces. However, VAPI is an API that is in flux and SMC is not guaranteed to work with all (current nor future) versions of VAPI.

### 2.2.6 SCI

This is a built-in device that uses the Scali SCI driver and library (ScaSCI). This driver is for the Dolphin SCI network cards. Please see the ScaSCI Release Notes for specific requirements. This device is straight forward and requires no configuration itself, but for multi-dimensional toruses (2D and 3D) the Scali SCI Management system (ScaConf) needs to be running somewhere in your system. Refer to Appendix C for installation and configuration of the Scali SCI Management software.

## 2.3 Communication protocols on DAT-devices

In SMC, the communication protocol used to transfer data between a sender and a receiver depends on the size of the message to transmit, as illustrated in Figure 2-3:.

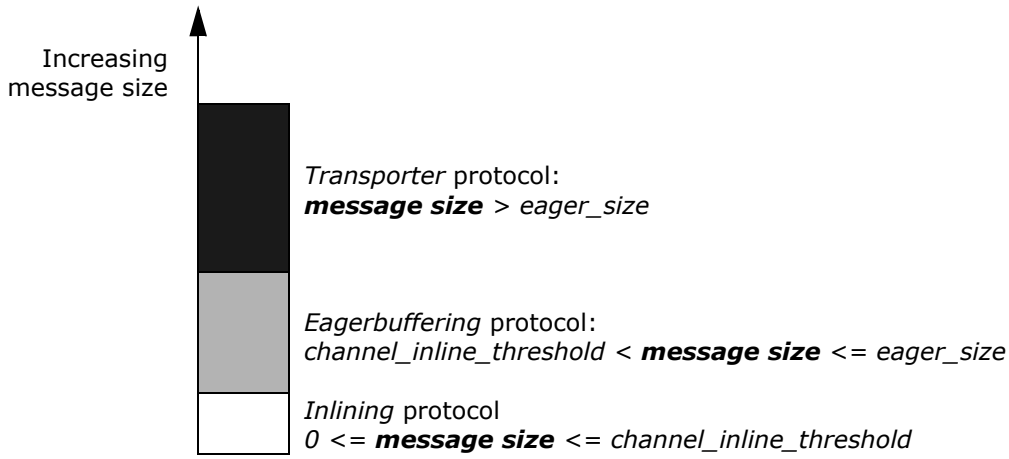


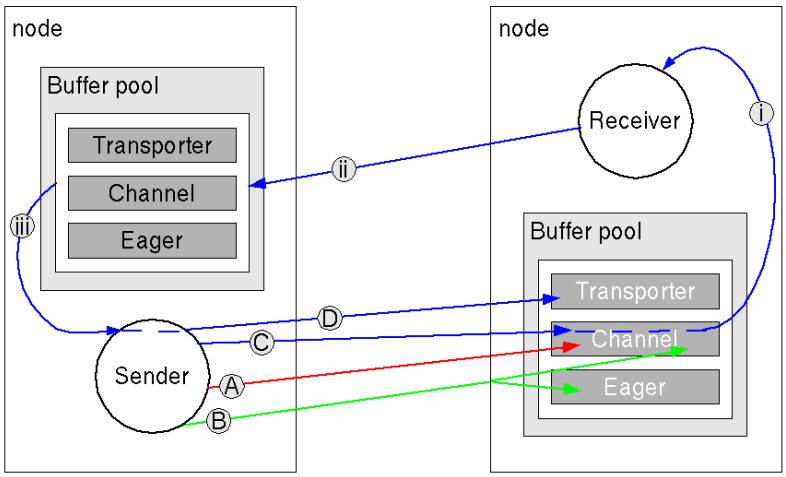

**Figure 2-3:** Thresholds for different communication protocol

The default thresholds that control whether a message belongs to the inlining, eagerbuffering or transporter protocols can be controlled from the application launch program (`mpimon`) described in chapter 3.

Figure 2-4: illustrates the node resources associated with communication and mechanisms implemented in Scali MPI Connect for handling messages of different sizes. The three communication protocols from Figure 2-3: rely on buffers located in the main memory of the nodes. This memory is allocated as shared, i.e., it is not private to a particular process in the node. Each process has one set of receiving buffers for of the processes it communicates with. As the figure shows all communication relies on the sending process depositing messages directly into the communication buffers of the receiver. For Inline and Eagerbuffering the management of the buffer resources does not require participation from the receiving process, because of their designs as ring buffers.

### 2.3.1 Channel buffer

The Channel ringbuffer is divided into equally sized entries. The size varies differs for different architectures and networks; see "Scali MPI Connect 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 used by the *inline* protocol, the *eagerbuffering* protocol, and the *transporter* protocol. In addition, one ore more entries are used by the *inline* protocol for application data being transmitted.




Ⓐ Inlining protocol: the sender writes a message into the receiver's channel ringbuffer

Ⓑ Eagerbuffering protocol: the sender deposits data in the eager buffer and writes a message into the receiver's channel ringbuffer identifying the data's position in the eager buffer

Ⓒ Transporter protocol: the sender communicates with receiver to negotiate which resources to use for data transport, the receiver's runtime system processes the message ⓘ and returns a response ⓘⓘ. The response is handled by the sender's runtime system ⓘⓘⓘ and the data is written to the agreed position in the transporter ring buffer Ⓓ

**Figure 2-4:** Resources and communication concepts in Scali MPI Connect

### 2.3.2 Inlining protocol

With the in-lining protocol the application's data is included in the message header. The in-lining protocol utilizes one or more channel ringbuffer entries.

### 2.3.3 Eagerbuffering protocol

The *eagerbuffering* protocol is used when medium-sized messages are to be transferred. The protocol uses a scheme where the buffer resources which are allocated by the sender are released by the receiver, without any explicit communication between the two communicating partners.

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

### 2.3.4 Transporter protocol

The *transporter* protocol is used when large messages are to be transferred. The transporter protocol utilizes one channel ringbuffer entry for the message header, and transporter buffers for the application data being sent. The protocol takes care of fragmentation and reassembly of large messages, such as those whose size is larger than the size of the transporter ringbuffer-entry (transporter_size).

### 2.3.5 Zerocopy protocol

The *zerocopy* protocol is special case of the transporter protocol t. It includes 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 if the underlying hardware can support it. To disable it, set the zerocopy_count or the zerocopy_size parameters to 0

## 2.4 Support for other interconnects

A uDAPL 1.1 module must be developed to interface other interconnects to Scali MPI Connect. The listing below identifies the particular functions that must be implemented in order for SMC 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_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_set_consumer_context
```

## 2.5 MPI-2 Features

At the time being SMC does not implement the full MPI-2 functionality. At the same time some users are asking for parts of the MPI-2 functionality, in particular the MPI-I/O functions. To fill the users needs Scali are now using the Open Source ROMIO software to offer this functionality.

ROMIO is a high-performance, portable implementation of MPI-IO, the I/O chapter in MPI-2 and has become a de-facto standard for MPI-I/O (in terms of interface and semantics). ROMIO is a library parallel to the MPI library for the application, but depend on an MPI to set up the environment and do communication. See chapter 3.2.6 for more information on how to compile and link applications with MPI-IO needs.

# Chapter 3 Using Scali MPI Connect

This chapter describes how to setup, compile, link and run a program using Scali MPI Connect, and briefly discusses some useful tools for debugging and profiling.

Please note that the "Scali MPI Connect Release Notes" are also available as a file in the `/opt/scali/doc/ScaMPI` directory.

## 3.1 Setting up a Scali MPI Connect environment

### 3.1.1 Scali MPI Connect environment variables

The use of Scali MPI Connect requires that some environment variables be defined. These are usually set in the standard startup scripts (e.g..**bashrc** when using **bash**), but 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 Scali MPI Connect library's header files `mpi.h` and `mpif.h` reside in the `$MPI_HOME/include` directory.

## 3.2 Compiling and linking

MPI is an "Application Programming Interface" (API) and not an "Application Binary Interface" (ABI). This means that in general applications should be recompiled and linked when used with Scali MPI Connect. Since the MPICH-implementation is widely used Scali has made SMC ABI-compatible, depending on the versions of MPICH and SMC used. Please check the "Scali MPI Connect Release Notes" for details. For applications that are dynamically linked with MPICH it should only be necessary to change the library-path `(LD_LIBRARY_PATH)`**.** For applications with the necessary object files, only a relinking is needed.

### 3.2.1 Running

Start the **hello-world** program on the three nodes called 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
```

### 3.2.2 Compiler support

Scali MPI Connect is a C library built using the GNU compiler. Applications can however be compiled with most compilers, as long as they are linked with the GNU runtime library. The details of the process of linking with the Scali MPI Connect libraries vary depending on which compiler is used. Check the "Scali MPI Connect Release Notes" for information on supported compilers and how linking is done.

When compiling the following string *must* be included as compiler flags (**bash** syntax):
`"-I$MPI_HOME/include"`

The pattern for compiling is:

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

### 3.2.3 Linker flags

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

```
"-L/opt/scali/lib -lmpi"
```

The following versions of MPI libraries are available:
- **libmpi** - Standard library containing the C API.
- **libfmpi** - Library containing the Fortran API wrappers.

The pattern for linking is:

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

### 3.2.4 Notes on Compiling and linking on AMD64 and EM64T

AMDs AMD64 and Intels EM64T (also known as x86-64) are instruction set architectures (ISA) that add 64 bit extensions to Intel x86 (ia32) ISA.

These processors are capable of running 32 bit programs at full speed while running a 64 bit OS. For this reason Scali supports running both 32 bit and 64 bit MPI programs while running 64 bit OS.

Having both 32 bit and 64 bit libraries installed at the same time require some tweaks to the compiler and linker flags.

All compilers for x86_64 generate 64 bit code by default, but have flags for 32 bit code generation. For gcc/g77 these are **-m32** and **-m64** for making 32 and 64 bit code respectively. For Portland Group Compilers these are **-tp k8-32** and **-tp k8-64**. For other compilers please check the compiler documentation.

It is not possible to link 32 and 64 bit object code into one executable, (no cross dynamic linking either) so there must be double set of libraries. It is common convention on x86_64 systems that all 32 bit libraries are placed in `lib` directories (for compatibility with x86 OS'es) and all 64 bit libraries in `lib64`. This means that when linking a 64 bit application with Scali MPI, you must use the `-L$MPI_HOME/lib64` argument instead of the normal `-L$MPI_HOME/lib`.

### 3.2.5 Notes on Compiling and linking on Power series

The Power series processors (PowerPC, POWER4 and POWER5) are both 32 and 64 bit capable. There are only 64 bit versions of Linux provided by SUSE and RedHat, and only a 64 bit OS is supported by Scali. However the Power families are capable of running 32 bit programs at full speed while running a 64 bit OS. For this reason Scali supports running both 32 bit and 64 bit MPI programs.

Note that gcc default compiles 32 bit on Power, use the the gcc/g77 flags -m32 and -m64 to explicitly select code generation.

The PowerPC and POWER4/5 have a common core instruction set but different extensions, be sure to read the specifics in the documentation on the compilers code generations flags for optimal performance.

It is not possible to link 32 and 64 bit object code into one executable, (no cross dynamic linking either) so there must be double set of libraries. It is common convention on ppc64 systems that all 32 bit libraries are placed in `lib` directories and all 64 bit libraries in `lib64`. This means that when linking a 64 bit application with Scali MPI, you must use the `-L$MPI_HOME/lib64` argument instead of the normal `-L$MPI_HOME/lib`.

### 3.2.6 Notes on compiling with MPI-2 features

To compile and link with the Scali MPI-IO features you need to do the following depending on whether it is a C or a Fortran program:

For C programs `mpio.h` must be included in your program and you must link with the `libmpio` shared library in addition to the Scali MPI 1.2 C shared library (`libmpi`):

```
<CC> <program>.o -I/opt/scali/inlude -L/opt/scali/lib -lmpio -lmpi -o <program>
```

For Fortran programs you will need to include `mpiof.h` in your program and link with the `libmpio shared` library in addition to the Scali MPI 1.2 C and Fortran shared libraries:

```
<F77> <program>.o -I/opt/scali/include -L/opt/scali/lib -lmpio -lmpif -lmpi -o
<program>
```

## 3.3 Running Scali MPI Connect programs

Note that executables issuing SMC calls cannot be started directly from a shell prompt. SMC programs can either be started using the MPI monitor program **mpimon**, the wrapper script **mpirun**, or from the Scali Manage GUI [See Scali Manage User Guide for details].

### 3.3.1 Naming conventions

When an application program is started, Scali MPI Connect is modifying the program name **argv[0]** to help in identifying the instances. 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**.
**<nodename>** is the name of the node where **mpimon** is running.

**Note:** SMC requires a homogenous file system image, i.e. a file system providing the same path and program names on all nodes of the cluster on which SMC is installed.

## 3.3.2 mpimon - monitor program

The control and start-up of an **Scali MPI Connect** application are monitored by **mpimon**. A complete listing of `mpimon` options can be found in "Mpimon options" on page 34.

### 3.3.2.1 Basic usage

Normally **mpimon** is invoked as:

```
mpimon <userprogram> <program options> -- <node name> [<count>][<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 defaults to 1

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
```

Bracket expansion and grouping (if configured) can also be used :
```
mpimon /opt/scali/examples/bin/hello -- node[1-16] 2 node[17-32] 1
```

for more information regarding bracket expansion and grouping, refer to Appendix D.

### 3.3.2.2 Identity of parallel processes

The identification of nodes and the number of processes to run on each particular node translates directly into the rank of the MPI processes. For example, specifying `n1 2 n2 2` will place process 0 and 1 on node n1 and process 2 and 3 on node n2. On the other hand, specifying `n1 1 n2 1 n1 1 n2 1` will place process 0 and 2 on node n1 while process 1 and 3 are placed on node n2.

This control over placement of processes can be very valuable when application performance depends on all the nodes having the same amount of work to do.

#### 3.3.2.3 Controlling options to mpimon

The program **mpimon** has a multitude of options which can be used for optimising SMC performance. Normally it should not be necessary to use any of these options. However, unsafe MPI programs might need buffer adjustments to solve deadlocks. Running multiple applications in one run may also be facilitated with some of the "advanced options".

Mpimon also has an advanced tracing option to enable pinpointing of communication bottlenecks.

The complete syntax for the program is as follows:
**mpimon [<mpimon option>]... <program & node-spec> [-- <program & node-spec>]...**

where

| | |
|---|---|
| **<mpimon options>** | are options to mpimon (see "See Mpimon options" and the SMC Release Notes for a complete list of options), |
| **<program & node-spec>** | is an application and node specification consisting of <program spec> -- <node spec> [<node spec>]..., |
| **<program spec>** | is an application and application-options specification (<userprogram>[<programoptions>]) |
| **--** | is the separator that signals end of user program options. |
| **<node spec>** | specifies which node and how many instances (<nodename> [<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/bandwidth" with 2 processes on "munin":

```
mpimon /opt/scali/examples/bin/hello -- hugin --
               /opt/scali/examples/bin/bandwidth -- munin 2
```

Changing one of the mpimon-parameters:

```
mpimon -channel_entry_count 32 /opt/scali/examples/bin/hello -- hugin 2
```

#### 3.3.2.4 Standard input

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**.

#### 3.3.2.5 Standard output

By default the processes' output to **stdout** all appear in the **stdout** of `mpimon`, where they are merged in some random order. It is however possible to keep the outputs apart by directing them to files that have unique names for each process. This is accomplished by giving `mpimon` the option `-separate_output <seletion>`, e.g., `-separate_output all` to have each process deposit its stdout in a file. The files are named according to the folowing template: `ScaMPIoutput_<host>_<pid>_<rank>`, where `<host>` and `<pid>` identify the particular invokation of `mpimon` on the host, and `<rank>` identifies the process.

#### 3.3.2.6 How to provide options to mpimon

There are three different ways to provide options to mpimon. The most common way is to specify options on the command line invoking mpimon. Another way is to define environment variables, and the third way is to define options in configuration file(s).

- **Command line options:** Options for mpimon must be placed after mpimon, but before the program name

- **Environment-variable options:** Setting an mpimon-option with environment variables requires that variables are defined as `SCAMPI_<uppercase-option>` where SCAMPI_ is a fixed prefix followed by the option converted to uppercase. For example SCAMPI_CHANNEL_SIZE=64Kmeans setting -channel_size to 64K

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

The options described either on the command line, as environment variables or in configuration files are prioritized the following way (ranked from lowest to highest):

1. System-wide configuration-file(/opt/scali/etc/ScaMPI.conf)
2. Configuration-file on home-directory(~/ScaMPI.conf)
3. Configuration-file on current directory(./ScaMPI.conf)
4. Environment-variables
5. Command line-options

#### 3.3.2.7 Network options

Scali 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 devices are selected by giving the option "**-networks <net-list>**" to mpimon. <net-list> is a comma-separated list of device names. Scali MPI Connect uses the list when setting up connections to other MPI-processes. It starts off with the first device in the list and sets up all possible connections with that device. If this fails the next on list is tried and so on until all connections are live or all adapters in <net-list>have beentried. A list of possible devices can be obtained with the **scanet** command.

For systems installed with the Scali Manage installer, a list of preferred devices is provided in **ScaMPI.conf**. An explicit list of devices may be set either in a private **ScaMPI.conf**, through the SCAMPI_NETWORKS environment variable, or by the -networks parameter to mpimon. The values should be provided in a comma-separated list of device names.

**Example:** `mpimon -networks smp,gm0,tcp ...`

For each MPI process SMC will try to establish contact with each other MPI process, *in the order listed*. This enables mixed interconnect systems, and provides a means for working around failed hardware.

In a system interconnect where the primary interconnect is Myrinet, if one node has a faulty card, using the device list in the example, all communication to and from the faulty node will happen over TCP/IP while the remaining nodes will use Myrinet. This offers the unique ability to continue running applications over the full set of nodes even when there are interconnect faults.

## 3.3.3 mpirun - wrapper script

**mpirun** is a wrapper script for **mpimon**, providing legacy **MPICH** style startup for SMC 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.

### 3.3.3.1 mpirun usage

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

where

**<mpirunoptions>** **mpirun** options
**<mpimonoptions>** options passed on to **mpimon**
**<userprogram>** name of application program to run.

and

**<programoptions>** program options passed on to the application program.

The following mpirunoptions exist:

| | |
|---|---|
| **-cpu** <time> | Limit runtime to <time> minutes. |
| **-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> The 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**. |

## 3.4 Suspending and resuming jobs

From time to time it is convenient to be able to suspend regular jobs running on a cluster in order to allow a critical, maybe real-time job to be use the cluster. When using Scali MPI Connect to run parallel applications suspending jobs to yield the cluster to other jobs can be achieved by sending a `SIGUSR1 or SIGTSTP` signal to the `mpimon` representing the job.

Assuming that the process identifier for this mpimon is `<PID>`, the user interface for this is:

```
user% kill -USR1 <PID>
or
user% kill -TSTP <PID>
```

Similarly the suspended job can be resumed by sending it a `SIGUSR2 or SIGCONT` signal, i.e.,

```
user% kill -USR2 <PID>
or
user% kill -CONT <PID>
```

## 3.5 Running with dynamic interconnect failover capabilities

If a runtime failure on a high speed interconnect occurs, ScaMPI has the ability to do an interconnect failover and continue running on a secondary network device. This high availability feature is part of the Scali MPI Connect/HA product, which requires a separately priced license. Once this license is installed, you may enable the failover functionality by setting the environment variable SCAMPI_FAILOVER_MODE to 1, or by using the mpimon command line argument **-failover_mode**.

Currently the Scali MPI Infiniband (ib0), Myrinet (gm0) and all DAT-based drivers are supported. SCI is not supported. Note also that the combination of failover and tfdr is not supported in this version of Scali MPI Connect.

Some failures will not result in a explicit error value propagating to Scali MPI. Scali MPI handles this by treating a lack of progress within a specified time as a failure. You may alter this time by setting the environment variable SCAMPI_FAILOVER_TIMEOUT to the desired number of seconds.

Failures will be logged using the standard syslog mechanism.

## 3.6 Running with tcp error detection - TFDR

Errors may occur when transferring data from the network card to memory. When offloading the tcp stack in hardware, this may result in actual data errors. Using the wrapper script **tfdrmpimon** (Transmission Failure Detection and Retransmit), Scali MPI will handle such errors by adding an extra checksum and retransmit the data if an error should occur. This high availability feature is part of the Scali MPI Connect/HA product which requires a separate license.

As this feature is limited to tcp communication only, it will not have any effect when using native RDMA drivers such as Infiniband or Myrinet. Note that the combination of tfdr and failover mode is not supported in this version of Scali MPI Connect.

Data errors will be logged using the standard syslog mechanism.

## 3.7 Debugging and profiling

The complexity of debugging programs can grow dramatically when going from serial to parallel programs. So to assist in debugging MPI programs Scali MPI Connect has a nnumber of features built-in, like starting processes directly in a debugger and tracing proceses' traffic patterns.

### 3.7.1 Debugging with a sequential debugger

SMC 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 **-debugger <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 an application using the default **gdb** debugger do:

**user% mpimon -debug all <application +parameters> -- <node specification>**

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 SMC application being debugged. Finally, one debugger is started for each session. In each debugger's `xterm` window, first input the appropriate debugging action before the MPI-process is started. Then, when ready to run the MPI-process, paste ***<programoptions>*** into the debugger to start running.

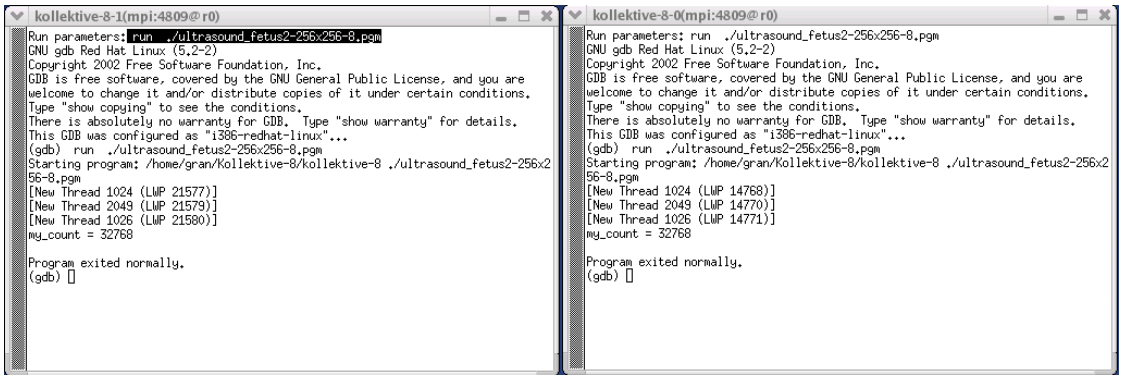



**Figure 3-1:** /opt/scali/bin/mpirun -debug all ./kollektive-8 ./ultrasound_fetus-256x256-8.pgm

## 3.7.2 Built-in-tools for debugging

Built-in tools for debugging in Scali MPI Connect covers discovery of the MPI calls used through tracing and timing, and an attachment point to processes that fault with segmentation violation. The tracing and timing is covered in Chapter 4.

#### 3.7.2.1 Using built-in segment protect violation handler

When running applications that terminate with a **SIGSEGV**-signal it is often useful to be able to freeze the situation instead of exiting, the default behavior. The built-in SIGSEGV-handler can be made to do this by defining the environment-variable **SCAMPI_INSTALL_SIGSEGV_HANDLER**.

**Legal options are:**

6. The handler dumps all registers and starts looping. Attaching with a debugger will then make it possible to examine the situation which resulted in the segment protect violation.

7. The handler dumps all registers but all processes will exit afterwards.

All other values will disable the installation of the handler.

To attach to process `<pid>` on a machine with the GNU debugger (`gdb`) do;

```
user% gdb /proc/<pid>/exe <pid>
```

In general, this will allow `gdb` to inspect the stack trace and identify the functions active when the `sigsegv` occurred, and disssasemble the functions. If the application is compiled with debug info (`-g`) and the source code is available, then source level debugging can be carried out.

## 3.7.3 Assistance for external profiling

Profiling parallel applications is complicated by having multiple processes at the same time. But Scali MPI Connect comes to assistance; through the `SCAMPI_PROFILE_APPLICATION` environment variable together with the `-separate_output` option (`SCAMPI_SEPARATE_OUTPUT`) the output from the application runs is directed at one file per process for easier use.

The environment variables `SCAMPI_PROFILE_APPLICATION_START and SCAMPI_PROFILE_APPLICATION_END` are also available for steering the range of memory addresses applicable to profiling.

## 3.7.4 Debugging with Etnus Totalview

SMC applications can be debugged using the Etnus Totalview, see **http://www.etnus.com** for more information about this product.

To start the Etnus Totalview debugger with a Scali MPI application, use the **tvmpimon** wrapper script. This wrapper script accepts the regular options that **mpimon** accepts, and sets up the environment for Totalview. The **totalview** binary must be in the search path when launching **tvmpimon**. If the **mpirun** script is the preferred way of starting jobs, it accepts the "standard" -tv option, however the same rules applies with regards to the search path.

## 3.8 Controlling communication resources

Even though it is normally **not** necessary to set buffer parameters when running applications, it can be done, e.g., for performance reasons. Scali MPI Connect automatically adjusts communication resources based on the number of processes in each node and based on pool_size and chunk_size.

The built-in devices SMP and TCP/IP use 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 end. 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 - Sets the size of the transmit buffer.
- SCAFUN_TCP_RXBUFSZ - Sets the size of the receive buffer.
- SCAFUN_SMP_BUFSZ - Sets the size of the buffer for intranode-communication.

The ringbuffers are divided into equally sized entries. The size varies differs for different architectures and networks; see "Scali MPI Connect 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 used by the *inline* protocol, the *eagerbuffering* protocol, and the *transporter* protocol. In addition, one ore more entries are used by the *inline* protocol for application data being transmitted.

**mpimon** has the following interface for the eagerbuffer and channel thresholds:

- *Channel* threshold definitions
  **-channel_inline_threshold <size>** to set threshold for inlining

- *Eager* threshold definitions
  **-eager_threshold <size>** to set threshold for eager buffering

### 3.8.1 Communication resources on DAT-devices

All resources (buffers) used by SMC reside in shared memory in the nodes. This way multiple processes (typically when a node has multiple CPUs) can share the communication resources.

SMC operates on a buffer pool. The pool is divided into equally sized parts called chunks. SMC 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 |

# 3.9 Good programming practice with SMC

## 3.9.1 Matching MPI_Recv() with MPI_Probe()

During development and testing of SMC, Scali has come across 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 as desired. However, the code will **not** work as expected with SMC. SMC uses 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:

```
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();
}
```

## 3.9.2 Using MPI_Isend(), MPI_Irecv()

If communication and calculations do not overlap, using immediate calls, e.g., `MPI_Isend()` and `MPI_Irecv()`, is usually performance ineffective.

## 3.9.3 Using MPI_Bsend()

Using buffered send, e.g., `MPI_Bsend()`, usually degrade performance significantly in comparison with their unbuffered relatives.

## 3.9.4 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 examples 3.15 and 3.16 in the MPI 1.1 standard [1]. Fairness can also be enforced, e.g. through the use of several tags or separate communicators.

### 3.9.5 Unsafe MPI programs

Because of different buffering behavior, some programs may run with MPICH, but **not** with SMC. Unsafe MPI programs may require resources that are not always guaranteed by SMC, and deadlock might occur (since SMC uses spin locks, these may appear to be live locks). If you want to know more about how to write portable MPI programs, see for example *MPI: The complete reference: vol. 1, the MPI core* [2].

A typical example that will **not** work with SMC (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();
}
```

Thsi code tries to se the same buffer for both sending and receiving. Such logic can be found, e.g., where processes from a ring where they communicate with their neigbours. Unfortunately writing the code this way leads to deadlock, and to make it work the `MPI_Send()` must be replaced with `MPI_Isend()` and `MPI_Wait()`, or the whole construction should be replaced with `MPI_Sendrecv()` or `MPI_Sendrecv_replace()`.

### 3.9.6 Name space pollution

The SMC library is written in C and all of its C names are prefixed with **scampi_**. Depending on the compiler used, the user may run into problems if he/she has C code using the same **scampi_** prefix. In addition, there are a few global variables that may cause problems. All of these functions and variables are listed in the include files **mpi.h** and **mpif.h**. Normally, these files are installed in /**opt/scali/include**.

Given that SMC has not fixed its OS routines to specific libraries, it is good programming practice to avoid using OS functions or standard C-lib 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 behavior.

## 3.10 Error and warning messages

### 3.10.1 User interface errors and warnings

User interface errors usually result from problems where the setup of the environment causes difficulties for **mpimon** when starting an MPI 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 the necessary 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 detailed warnings.

### 3.10.2 Fatal errors

When a fatal error occurs, SMC prints an error message before calling **MPI_Abort()** to shut down all MPI-processes.

## 3.11 Mpimon options

The full list of optiona accepted by `mpimon` is listed below. To obtain the actual values used for a particular run include the `-verbose` option when starting the application.

```
-automatic <selection>          Set automatic-mode for process(es)
-backoff_enable <selection>     Set backoff-mode for process(es)
-channel_entry_count <count>    Set number of entries per channel
-channel_entry_size <size>      Set entry_size (in bytes) per channel
-channel_inline_threshold <size> Set threshold for inlining (in bytes)
                                per inter-channel
-channel_size <size>            Set buffer size (in bytes) per inter-channel
-chunk_size <size>              Set chunk-size for inter-communication
-debug <selection>              Set debug-mode for process(es)
-debugger <debugger>            Set debugger to start in debug-mode
-disable-timeout               Disable process timeout
-display <display>              Set display to use in debug-/manual-mode
-dryrun <mode>
-eager_count <count>            Set number of buffers for eager protocol
-eager_factor <factor>          Set factor for subdivision of eagerbuffers
-eager_size <size>              Set buffer size (in bytes) for eager protocol
-eager_threshold <size>         Set threshold (in bytes) for eager protocol
-environment <value>
-exact_match
-execpath <execpath>            Set path to internal executables
-help                           Display available options
-home <directory>               Set installation-directory
-inherit_limits                 Inherit user definable limits to processes
-init_comm_world
-manual <selection>             Set manual-mode for process(es)
--                              Separator, marks end of user program options
-networks <networklist>         Define prioriy order when seaching network
-pool_size <size>               Set buffer-pool-size for communication
-pr_trace <selection>
-separate_output <selection>    Enable separate output for process(es)
                                Filename: ScaMPIoutput_host_pid_rank
-sm_debug <selection>
-sm_manual <selection>
-sm_trace <selection>
-statistics                     Enable statistics
-stdin <selection>              Distribute standard in to process(es)
-timeout <timeout>              Set timeout (elapsed time in seconds) for run
-timing <timing-spec>           Enable builtin-timing trace
-transporter_count <count>      Set number of buffers for transporter-protocol
-transporter_size <size>        Set buffer size (in bytes) for transporter-
                                protocol
-trace <trace-spec>             Enable builtin trace
-verbose                        Display values for user-options
-Version                        Display version of monitor
-working_directory <directory>  Set working directory
-xterm <xterm>                  Set xterm to use in debug-/manual-mode
-zerocopy_count <count>         Set number of buffers for zerpcopy-protocol
-zerocopy_size <size>           Set buffer size (in bytes) for zerocopy-protocol
```

### 3.11.1 Giving numeric values to mpimon

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

**[<prefix>]<numeric value>[<postfix>]**

where

| | |
|---|---|
| **<prefix>** | selects numeric base when interpreting the value<br>"0x" indicates hex-number (base = 16)<br>"0" indicates octal-number (base = 8)<br>if <prefix> is omitted, decimal-number (base = 10) is assumed |
| and | |
| **<postfix>** | selects a multiplication factor<br>**"K":** means a multiplication with 1024<br>**"M":** means a multiplication with 1024 * 1024 |

Examples:

| Input: | Value as interpreted by mpimon (in decimal): |
|---|---|
| 123 | 123 |
| 0x10 | 16 |
| 0200 | 128 |
| 1K | 1024 |
| 2M | 2 097 152 |

# Chapter 4 Profiling with Scali MPI Connect

The Scali MPI communication library has a number of built-in timing and trace facilities. These features are built into the run time version of the library, so no extra recompiling or linking of libraries is needed. All MPI calls can be timed and/or traced. A number of different environment variables control this functionality. In addition an implied *barrier* call can be automatically inserted before all collective MPI calls. All of this can give detailed insights into application performance.

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

There are different tools available that can be useful to detect and analyze the cause of performance bottlenecks:

- Built-in proprietary trace and profiling tools provided with SMC
- Commercial tools that collect information during run and postprocesses and presents results afterwards such as Vampir from Pallas GmbH. See **http://www.pallas.de** for more information.

The main difference between these tools is that the SMC built-in tools can be used with an existing binary while the other tools require reloading with extra libraries.

The powerful run time facilities Scali MPI Connect trace and Scali MPI Connect 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 an application's usage of MPI. Common to most of these logs are the massive amount of data which can sometimes be overwhelming, especially when run with many processes and using both trace and timing concurrently.

The second 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 unrealistically high. However, it reflects the total time spent in all MPI calls. For situations in which benchmarking focuses primarily on timing rather than tracing MPI calls, the timing functionality is more appropriate. 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.

## 4.1 Example

To illustrate the potential of tracing and timing with Scali MPI Connect consider the code fragment below (full source reproduced in A-2).

```
int main( int argc, char** argv )
{
  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  /* read image from file */

  /* broadcast to all nodes */
  MPI_Bcast( &my_count, 1, MPI_INT, 0, MPI_COMM_WORLD );
  /* scatter the image */
  MPI_Scatter( pixels, my_count, MPI_UNSIGNED_CHAR, recvbuf,
           my_count, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD );
  /* sum the squares of the pixels in the sub-image */
```

```
  /* find the global sum of the squares */
  MPI_Reduce( &my_sum, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD );
  /* let rank 0 compute the root mean square */

  /* rank 0 broadcasts the RMS to the other nodes */
  MPI_Bcast( &rms, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD );
  /* perform filtering operation (contrast enhancement) */

  /* gather back to rank 0 */
  MPI_Gather( recvbuf, my_count, MPI_UNSIGNED_CHAR, pixels,
          my_count, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD );
  /* write image to file */

  MPI_Finalize();
}
```

This code uses collective operations (broadcast, scatter, reduce and gather) to employ multiple processes to perform operations on a image. For example, the figure below (Figure 4-1:) shows the result of processing an ultrasonic image of a fetus.

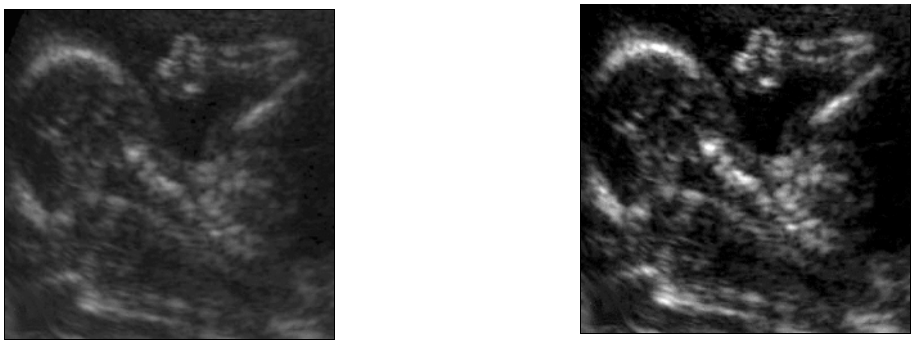



**Figure 4-1:** A ultrasound image before and after contrast enhancement

## 4.2 Tracing

Tracing enables features in Scali MPI Connect's implementataion of MPI that report detail about the MPI calls. By capturing the printout of the tracing information the user can monitor the development of the application run, perform analysis of the application run, figure out how the application uses the communication mechanisms, and discover details that can be used to improve performance.

#### 4.2.1 Using Scali 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<br><mode> = "sync": Synchronize with MPI_Barrier before starting the 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>.<br>MPI_call = 'MPI_call' \| 'call' |
| **-x** <call-list> | Disable for MPI_calls in <call-list>.<br>MPI_call = 'MPI_call' \| 'call' |
| **-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.

Calls may be specified with or without the "MPI_"-prefix, and in upper- or lower-case. The default format of the output has the following parts:
**<absRank>: <MPIcall><commName>_<rank><call-dependant-parameters>** where

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

This format can be extended by using the **"-f"-option**. Adding **"-f arguments"** will provide some additional information concerning message length. If **"-f timing"** is given some timing information between the **<absRank>**and **<MPIcall>**-fields is provided.

The extra field has the following format:
**+<relSecs> S <eTime>**
where

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

"**-f rate**" will add ***some*** rate-related 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.

Normally no error messages are provided concerning the options which have been selected. But if -verbose is added as a command-line option to mpimon, errors will be printed.
Trace provides information about which MPI routines were called and possibly information about parameters and timing.

Example using the test application:

```
%SCAMPI_TRACE="-p all" mpimon ./kollektive-8 ./uf256-8.pgm -- r1 r2
```

Prints a trace of all MPI calls for this run that is relatively simple:

```
0:  MPI_Init
0:  MPI_Comm_rank Rank: 0
0:  MPI_Comm_size Size: 2
```

```
   0:  MPI_Bcast root: 0 Id: 0
my_count = 32768
   0:  MPI_Scatter Id: 1
   1:  MPI_Init
   1:  MPI_Comm_rank Rank: 1
   1:  MPI_Comm_size Size: 2
   1:  MPI_Bcast root: 0 Id: 0
my_count = 32768
   1:  MPI_Scatter Id: 1
   1:  MPI_Reduce Sum root: 0 Id: 2
   1:  MPI_Bcast root: 0 Id: 3
   0:  MPI_Reduce Sum root: 0 Id: 2
   0:  MPI_Bcast root: 0 Id: 3
   1:  MPI_Gather Id: 4
   1:  MPI_Keyval_free
   0:  MPI_Gather Id: 4
   0:  MPI_Keyval_free
```

If more information is needed the arguments to SCAMPI_TRACE can be enhanced to request more information. The option "-f arg;timing" requests a list of the arguments given to each MPI call, including message size ( useful information when evaluating interconnect performance).

Example:

```
%SCAMPI_TRACE="-f arg;timing" mpimon ./kollektive-8 ./uf256-8.pgm -- n1 2

   0:  +-0.951585 s  951.6ms MPI_Init
   0:  +0.000104 s     3.2us MPI_Comm_rank Rank: 0
   0:  +0.000130 s     1.7us MPI_Comm_size Size: 2
   0:  +0.038491 s    66.3us MPI_Bcast root: 0 sz: 1 x 4 = 4 Id: 0
my_count = 32768
   0:  +0.038634 s   390.0us MPI_Scatter Id: 1
   1:  +-1.011783 s    1.0s  MPI_Init
   1:  +0.000100 s     3.8us MPI_Comm_rank Rank: 1
   1:  +0.000129 s     1.7us MPI_Comm_size Size: 2
   1:  +0.000157 s    69.6us MPI_Bcast root: 0 sz: 1 x 4 = 4 Id: 0
my_count = 32768
   1:  +0.000300 s   118.7us MPI_Scatter Id: 1
   0:  +0.039267 s    38.8ms MPI_Reduce Sum root: 0 sz: 1 x 4 = 4 Id: 2
   0:  +0.078089 s    18.8us MPI_Bcast root: 0 sz: 1 x 8 = 8 Id: 3
   1:  +0.000641 s    56.2us MPI_Reduce Sum root: 0 sz: 1 x 4 = 4 Id: 2
   1:  +0.000726 s   113.6us MPI_Bcast root: 0 sz: 1 x 8 = 8 Id: 3
   1:  +0.002547 s    99.0us MPI_Gather Id: 4
   0:  +0.079834 s   579.5us MPI_Gather Id: 4
   1:  +0.002758 s    26.4us MPI_Keyval_free
   0:  +0.113244 s     1.4us MPI_Keyval_free
```

There are a number of parameters for selecting only a subset, either by limiting the number of calls and intervals as described above under 'Timing' , or selecting or excluding just some MPI calls.

#### 4.2.2 Features

The **"-b" option** is useful when trying to pinpoint which MPI-call has been started but not completed (i.e. are deadlocked). The **"-s/-S/-c/-C" -options** also offer useful support for an application that runs well for a longer period and then stop, or for examining some part of the execution of the application.

From time to time it may be desirable or feasible to trace only one or a few of the processes. Specifying the "-p" options offers the ability to pick the processes to be traced.

All MPI-calls are enabled for tracing by default. To view only a few calls, specify a **"-t <call-list>" option**; to exclude some calls, add a **"-x <call-list>" option**. The "-t" will disable all tracing and then enable those calls that match the <call-list>. The matching is done using "regular-posix-expression"-syntax. **"-x"** will lead 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

## 4.3 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.

#### 4.3.1 Using Scali MPI Connect built-in timing

To use the built-in timing functionality in SMC, the mpimon-option **-timing "<options>"** must be set, specifying which options are to be applied.

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<br><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' |
| **-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 "-s <seconds>" is specified, or for a fixed number-of-calls-interval, if "-c <calls>" ia used. It is also possible to obtain output after specific MPI-calls by using "-f <call-list>"; see above for details on how to write <call-list>.

The output has two parts: a timing-part and a buffer-statistics-part.

The first part has the following layout:
All lines start with **<rank>:** where <rank>: is rank within MPI_COMM_WORLD. This part is included to facilitate separation of output (grep).
Example:

```
user% SCAMPI_TIMING="-s 1" mpimon ./kollektive-8 ./uf256-8.pgm -- r1 r2
```

where '<seconds>' is the number of seconds per printout from Scali MPI Connect produces:

```
 1: 13.26.10       -------------Delta----------  ---------Total----------
 1: Init+0.002659 s    #calls    time   tim/cal  #calls    time   tim/cal
 1: MPI_Bcast               2  169.0us   84.5us       2  169.0us   84.5us
```

```
1: MPI_Comm_rank           1    3.1us    3.1us       1    3.1us    3.1us
1: MPI_Comm_size           1    1.5us    1.5us       1    1.5us    1.5us
1: MPI_Gather              1  109.9us  109.9us       1  109.9us  109.9us
1: MPI_Init                1    1.0s     1.0s        1    1.0s     1.0s
1: MPI_Keyval_free         1    1.2us    1.2us       1    1.2us    1.2us
1: MPI_Reduce              1   51.5us   51.5us       1   51.5us   51.5us
1: MPI_Scatter             1  138.7us  138.7us       1  138.7us  138.7us
1: Sum                     9    1.0s   112.8ms       9    1.0s   112.8ms
1: Overhead                0    0.0ns                9   27.2us    3.0us
1: ===============================================================================
0: 13.26.40       -------------Delta----------  ---------Total----------
0: Init+0.111598 s    #calls    time   tim/cal  #calls    time   tim/cal
0: MPI_Bcast               2   79.6us   39.8us       2   79.6us   39.8us
0: MPI_Comm_rank           1    3.3us    3.3us       1    3.3us    3.3us
0: MPI_Comm_size           1    1.4us    1.4us       1    1.4us    1.4us
0: MPI_Gather              1  648.8us  648.8us       1  648.8us  648.8us
0: MPI_Init                1  965.9ms  965.9ms       1  965.9ms  965.9ms
0: MPI_Keyval_free         1    1.1us    1.1us       1    1.1us    1.1us
0: MPI_Reduce              1   37.6ms   37.6ms       1   37.6ms   37.6ms
0: MPI_Scatter             1  258.1us  258.1us       1  258.1us  258.1us
0: Sum                     9    1.0s   111.6ms       9    1.0s   111.6ms
0: Overhead                0    0.0ns                9   35.6us    4.0us
0: ===============================================================================
```

The <seconds> field can be set to a large number in order to collect only final statistics.

We see that the output gives statistics about which MPI calls 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 cumulative statistics at the end are printed. The timings are presented for each process, and with many processes this can yield a huge amount of output. There are many options for modifying SCAMPI_TIMING to reduce this output. The selection parameter can time only those MPI processes to be monitored. There are also other ways to minimize the output, by screening away selected MPI calls either before or after a certain number of calls or between an interval of calls. Some examples are:

The rest of the format has the following fields:
**<MPIcall><Dcalls><Dtime><Dfreq> <Tcalls><Ttime><Tfreq>**
where

| | |
|---|---|
| <MPIcall> | is the name of the MPI-call |
| <Dcalls> | is the number of calls to <MPIcall> since the last printout |
| <Dtime> | is the sum of the execution-time for calls to <MPIcall> since the last printout |
| <Dfreq> | is the average time-per-call for calls to <MPIcall> since the last printout |
| <Tcalls> | is the number of calls to <MPIcall> |
| <Ttime> | is the sum of the execution-time for calls to <MPIcall> |
| <Tfreq> | is the average time-per-call for calls to <MPIcall> |

After all detail-lines (one per MPI-call which has been called since last printout), there will be a line with the sum of 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:
**<Comm><rank> recv from <from>(<worldFrom>):<commonFields>**
where

| | |
|---|---|
| <Comm> | is the communicator being used |
| <rank> | is the rank within <Comm> |
| <from> | is the rank within <Comm> |
| <worldFrom> | is the rank within MPI_COMM_WORLD |

"Send-lines" has the following fields:
**<Comm><rank> send to <to>(<worldTo>):<commonFields>**
where

| | |
|---|---|
| <Comm> | is the communicator being used |
| <rank> | is the rank within <Comm> |
| <to> | is the rank within <Comm> |
| <worldTo> | is the rank within MPI_COMM_WORLD |

The **<commonFields>** are as follows:
**!<count>!<avrLen>!<zroLen>!<inline>!<eager>!<transporter>!**
where

| | |
|---|---|
| <count> | is the number of sends/receives |
| <avrLen> | is the average length of messages in bytes |
| <zroLen> | is the number of messages sent/received using the zero-bytes mechanism |
| <inline> | is the number of messages sent/received using the inline mechanism |
| <eager> | is the number of messages sent/received using the eagerbuffer mechanism |
| <transporter> | is the number of messages sent/received using the transporter mechanism |

More details on the different mechanisms can be found in "Description of Scali MPI Connect" on page 11.

## 4.4 Using the scanalyze

Tracing and timing the image processing example above produced little data, and interpreting the data posed little problem. However, most applications run for longer time, with correspondingly larger logs as a result; output from tracing and timing with Scali MPI Connect easily amounts to megabytes of data.

In order to extract information from the huge amount of data, Scali has developed a simple analysis tool called `scanalyze`. This analysis tool accept output from SMC applications run with certain predefined trace and timing variables set.

### 4.4.1 Analysing `all2all`

The all2all program in /opt/scali/examples/bin is a simple communication benchmark, but tracing and timing it produces massive log files. For example running

```
user% SCAMPI_TRACE="-f arg;timing" mpimon ./all2all -- r1 r2
```

on a particular system produced a 2354159 byte log file, while running

```
user% SCAMPI_TIMING=”-s 10” mpimon ./all2all -- r1 r2
```

produced a 158642 byte file

Digesting the massive information in these files is a challenge, but `scanalyze` produces the following summaries for tracing:

```
                Count    Total    < 128    < 1k     < 8k     < 256k   < 1M
--------------- -------- -------- -------- -------- -------- -------- --------
   MPI_Alltoall    24795     5127     3078     4104    10260     2226        0
    MPI_Barrier       52        0        0        0        0        0        0
  MPI_Comm_rank        2        0        0        0        0        0        0
  MPI_Comm_size        2        0        0        0        0        0        0
       MPI_Init        2        0        0        0        0        0        0
MPI_Keyval_free        2        0        0        0        0        0        0
      MPI_Wtime      102        0        0        0        0        0        0

                Timing   Total    < 128    < 1k     < 8k     < 256k   < 1M
--------------- -------- ------- -------- -------- -------- -------- --------
   MPI_Alltoall    21.20    0.21     0.15     0.41     9.35    11.08     0.00
    MPI_Barrier     0.01    0.00     0.00     0.00     0.00     0.00     0.00
  MPI_Comm_rank     0.00    0.00     0.00     0.00     0.00     0.00     0.00
  MPI_Comm_size     0.00    0.00     0.00     0.00     0.00     0.00     0.00
       MPI_Init     2.00    0.00     0.00     0.00     0.00     0.00     0.00
MPI_Keyval_free     0.00    0.00     0.00     0.00     0.00     0.00     0.00
      MPI_Wtime     0.00    0.00     0.00     0.00     0.00     0.00     0.00
```

and for timing;

```
================================================================================
                    #calls    time   tim/cal      #calls    time   tim/cal
0: MPI_Alltoall          0    0.0ns                12399   10.6s   855.1us
0: MPI_Barrier           0    0.0ns                   26    1.2ms   45.8us
0: MPI_Comm_rank         0    0.0ns                    1    3.2us    3.2us
0: MPI_Comm_size         0    0.0ns                    1    1.4us    1.4us
0: MPI_Init              0    0.0ns                    1    1.0s     1.0s
0: MPI_Keyval_free       1   27.9us   27.9us          1   27.9us   27.9us
0: MPI_Wtime             1    1.1us    1.1us         52   33.9us  652.7ns
0: Sum                   2   29.0us   14.5us       12481   11.7s   933.5us
0: Overhead              0    0.0ns                12481   12.6ms    1.0us
================================================================================
                    #calls    time   tim/cal      #calls    time   tim/cal
1: MPI_Alltoall          0    0.0ns                12399   10.6s   854.9us
1: MPI_Barrier           0    0.0ns                   26    2.9ms  109.6us
1: MPI_Comm_rank         0    0.0ns                    1    3.5us    3.5us
1: MPI_Comm_size         0    0.0ns                    1    1.5us    1.5us
1: MPI_Init              0    0.0ns                    1    1.0s     1.0s
1: MPI_Keyval_free       1   10.8us   10.8us          1   10.8us   10.8us
1: MPI_Wtime             1    1.5us    1.5us         50   36.5us  730.2ns
1: Sum                   2   12.3us    6.1us       12479   11.6s   931.1us
1: Overhead              0    0.0ns                12479   12.7ms    1.0us
```

## 4.5 Using SMC's built-in CPU-usage functionality

Scali MPI Connect has the capability to report wall clock time, and user and system CPU time on all processes with a built-in CPU timing facility. To use SMC's built-in CPU-usage-timing it is necessary first to set the environment variable SCAMPI_CPU_USAGE.

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

The output has two different blocks. The first block contains CPU-usage by the sub monitors on the different nodes. One line is printed for each sub monitor 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.

For example, to get the CPU usage when running the image enhancement program do:

```
user% SCAMPI_CPU_USAGE=1 mpirun -np 4 ./kollektive-8 ./uf256-8.pgm
```

This produces the following report:

```
                                  ---------- Own ----------- ------ Own+Children ------
Submonitor timing stat. in secs    Elapsed    User   System     Sum    User   System     Sum
Submonitor-1@r9                    2.970    0.000    0.000    0.000    0.090    0.030    0.120
Submonitor-2@r8                    3.250   -0.000    0.000   -0.000    0.060    0.040    0.100
Submonitor-3@r7                    3.180   -0.000   -0.000   -0.000    0.050    0.030    0.080
Submonitor-4@r6                    3.190    0.010    0.000    0.010    0.090    0.020    0.110
Total for submonitors             12.590    0.010   -0.000    0.010    0.290    0.120    0.410
Average per submonitor             3.147    0.003   -0.000    0.003    0.073    0.030    0.103

                                           ---------- Own -----------
Process timing stat. in secs         Elapsed     User   System      Sum
kollektive-8-0@r9                      0.080    0.070    0.030    0.100
kollektive-8-1@r8                      0.050    0.020    0.040    0.060
kollektive-8-2@r7                      0.050    0.020    0.030    0.050
kollektive-8-3@r6                      0.010    0.020    0.020    0.040
Sum for processes                      0.190    0.130    0.120    0.250
Average per process                    0.048    0.033    0.030    0.062

Elapsed is walltime used by user-process/submonitor
User is cpu-time used in user-process/submonitor
System is cpu-time used in system-calls
Sum is total cpu-time used by user-process/submonitor
```

# Chapter 5 Tuning SMC to your application

Scali MPI Connect allows the user to exercise control over the communication mechanisms through adjustment of the thresholds that steer which mechanism to use for a particular message. This is one technique that can be used to improve performance of parallel applications on a cluster.

Forcing size parameters to **mpimon** is usually **not** necessary. This is only a means of optimising SMC to a particular application, based on knowledge of communication patterns. For unsafe MPI programs it may be necessary to adjust buffering to allow the program to complete.

## 5.1 Tuning communication resources

The communication resources allocated by Scali MPI Connect are shared among the MPI processes in the node.

- Communication buffer adaption: If the communication behaviour of the application is known, explicitly providing buffersize settings to mpimon, to match the requirement of the application, will in most cases improve performance.

  **Example**: Application sending only 900 bytes messages.
  Set **channel_inline_threshold 964** (64 added for alignment) and increase the channel-size significantly (32-128 k).

  Setting **eager_size 1k** and **eager_count** high (16 or more).
  If all messages can be buffered, the transporter-{size, count} can be set to low values to reduce shared memory consumption.

- How do I control shared memory usage?
  Adjusting SMC buffer sizes

- 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
  ```

#### 5.1.1 Automatic buffer management

The *pool-size* is a limit for the total amount of shared memory. The automatic buffer size computations is based on 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.

## 5.2 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't's for SMC.

#### 5.2.1 Performance analysis

Learn about the performance behaviour of your particular MPI applications on a Scali System by using a performance analysis tool.

#### 5.2.2 Using processor-power to poll

To maximize performance, ScaMPI is using poll when waiting for communication to terminate, instead of using interrupts. Polling means that the CPU is performing busy-wait (looping) when waiting for data over the interconnect. All exotic interconnects require polling.

Some applications create treads which may end up having more active threads than you have CPUs. This will have huge impact on MPI performance. In threaded application with irregular communication patterns you probably have other threads that could make use of the processor. To increase performance in this case, Scali has provided a "backoff" feature in ScaMPI. The backoff feature will still poll when waiting for data, but will start to enter sleep states on intervals when no data is coming. The algorithm is as follows: ScaMPI polls for a short time (idle time), then stops for a periode, and polls again.

The sleep periode starts a parameter controlled minimum and is doubled every time until it reaches the maximum value. The following environment variables set the parameters:

SCAMPI_BACKOFF_ENABLE (turns the mechanism on)
SCAMPI_BACKOFF_IDLE=n (defines idle-period as n ms [Default = 20 ms])
SCAMPI_BACKOFF_MIN=n (defines minimum backoff-time in ms [Default = 10 ms])
SCAMPI_BACKOFF_MAX=n (defines maximum backoff-time in ms [Default = 100 ms])

#### 5.2.3 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.

## 5.3 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 making comparisons with similar results.

#### 5.3.1 How to get expected performance

- **Caching the application program on the nodes.**
  For benchmarks with short execution time, total execution time may be reduced when running the process repetitively. For large configurations, copying the application to the local file system on each node will reduce startup latency and improve disk I/O bandwidth.
- **The first iteration is (very) slow.**
  This may happen because the MPI-processes in an application are not started simultaneously. Inserting an **MPI_Barrier()** before the timing loop will eliminate this.

#### 5.3.2 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, decreasing `chunck_size` may help.

## 5.4 Collective operations

A collective communication is a communication operation in which a group of processes works together to distribute or gather together a set of one or more values. Scali MPI Connect uses a number of different approaches to implement collective operations. Through environment variables the user can control which algorithm the application uses.

Consider the Integer Sort (IS) benchmark in NPB (NAS Parallel Benchmarks). When running on ten processes on 5 nodes over Gigabit Ethernet (`mpimon -net smp,tcp bin/is.A.16.scampi -- r1 2 r2 2 r3 2 r4 2 r5 2)` the resulting performance is:

```
Mop/s total     = 34.05
Mop/s/process   = 2.13
```

Extracting the MPI profile of the run can be done as follows:

```
user% export SCAMPI_TRACE="-f arg;timing"
user% mpimon bin/is.A.16.scampi  -- $ALL2  > trace.out
```

And running the output through scanalyze yields the following:

| MPI Call | <128 | 128-1k | 1-8k | 8-32k | 32-256k | 256k-1M | >1M |
|---|---|---|---|---|---|---|---|
| MPI_Send | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| MPI_Irecv | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| MPI_Wait | 0.69 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| MPI_Alltoall | 0.14 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| MPI_Alltoallv | 11.20 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| MPI_Reduce | 1.04 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| MPI_Allreduce | 0.00 | 0.00 | 15.63 | 0.00 | 0.00 | 0.00 | 0.00 |
| MPI_Comm_size | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| MPI_Comm_rank | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| MPI_Keyval_free | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

The MPI_Alltoallv uses a high fraction of the total execution time. The communication time is the sum of all used algorithms and the total timing may depend on more than one type of communication. If one type or a few operations dominate the time consumption, they are promising candidates for tuning/optimization.

**Note:** Please note that the run time selectable algorithms and their values may vary on different Scali MPI Connect release versions. For information on which algorithms that are selectable at run time and their valid values, set the environment variable **SCAMPI_ALGORITHM** and run an example application:

```
# SCAMPI_ALGORITHM=1 mpimon /opt/scali/examples/bin/hello -- localhost
```

This will produce a listing of the different implementations of particular collective MPI calls. For each collective operation a listing consisting of a number and a short description of the algoritmn is produced, e.g., for `MPI_Alltoallv()` the following:

```
SCAMPI_ALLTOALLV_ALGORITHM alternatives
       0 pair0
       1 pair1
       2 pair2
       3 pair3
```

```
       4 pair4
       5 pipe0
       6 pipe1
       7 safe
def    8 smp
```

By looping through these alternatives the performance of IS varies:

```
  algorithm 0: Mop/s total = 95.60
  algorithm 1: Mop/s total = 78.37
  algorithm 2: Mop/s total = 34.44
  algorithm 3: Mop/s total = 61.77
  algorithm 4: Mop/s total = 41.00
  algorithm 5: Mop/s total = 49.14
  algorithm 6: Mop/s total = 85.17
  algorithm 7: Mop/s total = 60.22
  algorithm 8: Mop/s total = 48.61
```

For this particular combination of Alltoallv-algorithm and application (IS) the performance varies significantly, with algorithm 0 close to doubling the performance over the default.

#### 5.4.1 Finding the best algorithm

Consider the image processing example from Chapter 4 which containes four collective operations. All of these can be tuned with respect to algorithm according to the following pattern:

```
user% for a in <range>; do
\>; SCAMPI_<MPI-function>_ALGORITHM=$a \
\>;mpimon <application> -- <nodes> ><application>.out.$a; \
\>; done
```

For example, trying out the alternative algorithms for MPI_Reduce with two processes can be done as follows (assuming Bourne Again Shell [bash]:

```
user% for a in 0 1 2 3 4 5 6 7 8; do
\>; SCAMPI_REDUCE_ALGORITHM=$a
\>; mpimon ./kollektive-8 ./uf256-8.pgm -- r1 r2;
\>; done
```

Given that the application then reports the timing of the relevant parts of the code a best choice can be made. Note however that with multiple collective operations working in the same program there may be interference between the algorithms. Also, the performance of the implementations is interconnect dependent.

# Appendix A Example MPI code

## A-1 Programs in the ScaMPItst package

The ScaMPItst package is installed together with installation of Scali MPI Connect. The package contains a number of programs in `/opt/scali/examples` with executable code in `bin/` and source code in `src/`. A description of the programs can be found in the README file, located in the `/opt/scali/doc/ScaMPItst` directory. These programs can be used to experiement with the features of Scali MPI Connect.

## A-2 Image contrast enhancement

```
*
 * Adopted from "MPI Tutorial", by Puri Bangalore, Anthony Skjellum and
 * Shane Herbert, High Performance Computing Lab, Dept. of Computer Science
 * and NSF Engineering Research Center, Mississippi State University,
 * Feb 2000
*/

#include <mpi.h>
#include <stdio.h>
#include <math.h>
int main( int argc, char** argv )
{
   int width, height, rank, size, sum, my_sum;
   int numpixels, my_count, i, val;
   unsigned char pixels[65536], recvbuf[65536];
   unsigned int buffer;
   double rms;
   FILE *infile;
   FILE *outfile;
   char line[80];

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &rank );
   MPI_Comm_size( MPI_COMM_WORLD, &size );
   if ( rank == 0 )
   {
     /* assume valid file name in argv[1] */
     infile = fopen( argv[1], "r" );
     if ( ! infile )
     {
       printf("%s :: can't open file\n", argv[1]);
       MPI_Finalize();
       exit(-1);
     }
     /* valid file available */
     fscanf( infile, "%s", line );
     fscanf( infile, "%d", &height );
     fscanf( infile, "%d", &width );
     fscanf( infile, "%u", &buffer );
     numpixels = width * height;
```

```
  /* read the image */
  for ( i = 0; i < numpixels; i ++ )
    {
              fscanf( infile, "%u", &buffer );
              pixels[i] = (unsigned char)buffer;
    }
  fclose( infile );
   /* calculate number of pixels for each node */
   my_count = numpixels / size;
}
/* broadcast to all nodes */
MPI_Bcast( &my_count, 1, MPI_INT, 0, MPI_COMM_WORLD );
/* scatter the image */
MPI_Scatter( pixels, my_count, MPI_UNSIGNED_CHAR, recvbuf,
             my_count, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD );
/* sum the squares of the pixels in the sub-image */
my_sum = 0;
for ( i = 0; i < my_count; i ++ )
   my_sum += recvbuf[i] * recvbuf[i];
/* find the global sum of the squares */
MPI_Reduce( &my_sum, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD );
/* let rank 0 compute the root mean square */
if ( rank == 0 )
{
   rms = sqrt( (double)sum / (double)numpixels );
}
/* rank 0 broadcasts the RMS to the other nodes */
MPI_Bcast( &rms, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD );
/* perform filtering operation (contrast enhancement) */
for ( i = 0; i < my_count; i ++ )
{
   val = 2 * recvbuf[i] - rms;
   if ( val < 0 ) recvbuf[i] = 0;
   else if ( val > 255 ) recvbuf[i] = 255;
   else recvbuf[i] = val;
}
/* gather back to rank 0 */
MPI_Gather( recvbuf, my_count, MPI_UNSIGNED_CHAR, pixels,
            my_count, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD );
/* dump the image from rank 0 */
if ( rank == 0 )
  {
    outfile = fopen( "try.pgm", "w" );
    if ( ! outfile )
              {
                printf("unable to open try.ppm\n");
              }
    else
              {
                fprintf( outfile, "%s\n", (unsigned char*)line );
                fprintf( outfile, "%d %d\n", height, width );
                fprintf( outfile, "255\n");
                /*              numpixels = height * width; */
                for ( i = 0; i < numpixels; i ++ )
                  {
                    fprintf( outfile, "%d\n", (int)pixels[i] );
```

```
                    }
                  fflush( outfile );
                  fclose ( outfile );
                }
    }
  MPI_Finalize();
  return 0;
}
```

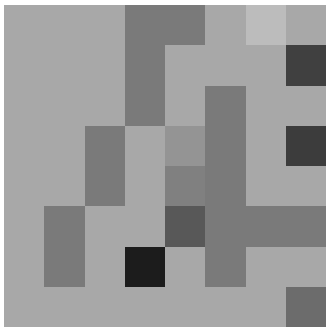
### A-2.1 File format

The code contains the logic to read and write images in .pgm format. This "Portable Gray Map" format uses ASCII characters for encoding pixel intensities, as illustrated by the example below:

```
P2
8 8
255
160 160 160 137 137 160 170 160
160 160 160 137 160 160 160 108
160 160 160 137 160 137 160 160
160 160 137 160 150 137 160 106
160 160 137 160 140 137 160 160
160 137 160 160 120 137 137 137
160 137 160  90 160 137 160 160
160 160 160 160 160 160 160 130
```

**original**

```
P2
8 8
255
168 168 168 122 122 168 188 168
168 168 168 122 168 168 168  64
168 168 168 122 168 122 168 168
168 168 122 168 148 122 168  60
168 168 122 168 128 122 168 168
168 122 168 168  88 122 122 122
168 122 168  28 168 122 168 168
168 168 168 168 168 168 168 108
```



**enhanced contrast**

# Appendix B Troubleshooting

This appendix offers initial suggestions for what to do when something goes wrong with applications running together with SMC. When problems occur, first check the list of common errors and their solutions; an updated list of **SMC-related Frequently Asked Questions** (FAQ) is posted in the Support section of the Scali website (**http://www.scali.com**). If you are unable to find a solution to the problem(s) there, please read this chapter before contacting **support@scali.com**.

Problems and fixes reported to Scali will eventually be included in the appropriate sections of this manual. Please send relevant remarks by e-mail to **support@scali.com**.

Many problems find their origin in not using the right application code, daemons that Scali MPI Connect rely on are stopped, and incomplete specification of network drivers. Below some typical problems and their solutions are described. Troubleshooting the DAT functionality is described in C-11.

## B-1 When things do not work - troubleshooting

This section is intended to serve as a starting point to help with software and hardware 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 Manage GUI, see the Scali System Guide.

### B-1.1 Why does not 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. To insure that these are correct, try the following:

1. Set the environment variable **MPI_HOME** correctly
2. Restart mpid, because a new version of ScaMPI has been installed without restarting **mpid**
3. Reinstall SMC, because a new version of SMC was not cleanly installed on all nodes.

**? Set working directory failed**

◆ SMC assumes that there is a 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 is 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**

◆ SMC uses a hardware-supported high precision timer for MPI_Wtime. This timer can be disabled by using SCAMPI_DISABLE_HPT=1

### B-1.2 Why can I not start mpid?

mpid opens a socket and assigns a predefined mpid port number (see **/etc/services** for more information), to the end point. If mpid is terminated abnormally, the mpid port number cannot be re-used until a system defined timer has expired. To resolve:

- Use **netstat -a | grep mpid** to observe when the socket is released. When the socket is released, restart **mpid** again.

#### B-1.2.1 Bad clean up

**? A previous SMC 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).

**Note**: core dumping takes time.

#### B-1.2.2 Space overflow

**? The application has required too much SCI or shared memory resources.**

- The **mpimon** *pool-size* specifications are too large, and must be reduced.

### B-1.3 Why does my program terminate abnormally?

#### B-1.3.1 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 the debugger.

### B-1.4 General problems

**? Are you reasonably 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 "Communication buffer adaption: If the communication behaviour of the application is known, explicitly providing buffersize settings to mpimon, to match the requirement of the application, will in most cases improve performance. Example: Application sending only 900 bytes messages. Set channel_inline_threshold 964 (64 added for alignment) and increase the channel-size significantly (32-128 k). Setting eager_size 1k and eager_count high (16 or more). If all messages can be buffered, the transporter-{size, count} can be set to low values to reduce shared memory consumption." on page 47.

**? The program terminates without an error message**

- Investigate the core file, or rerun the program in a debugger.

# Appendix C Install Scali MPI Connect

Scali MPI Connect can be installed on clusters in one of two ways, either as part of installing clusters from scratch with Scali Manage, or by installing it on each particular node in systems that do not use Scali Manage. In the first case the default when building clusters is to include Scali MPI Connect as well, whereas in the second case the cluster is probably managed with some other suite of tools that do not integrate with Scali MPI Connect. In the following the steps needed to manually install Scali MPI Connect are detailed.

## C-1 Per node installation of Scali MPI Connect

Scali MPI Connect must be installed on every node in the cluster. When running **smcinstall** you should give arguments to specify your interconnects.

The -h option gives you details on the installation command and shows you which options you need to specify in order to install the software components you want :

```
root# ./smcinstall -h

This is the Scali MPI Connect (SMC) installation program. The script will install
and configure Scali MPI Connect at the current node.
Usage: smcinstall [-atemszulixVh?]
    -a                Automatically accept license terms.
    -t                Install Scali MPI Connect for TCP/IP.
    -e <eth devs>     Install Scali MPI Connect for Direct Ethernet.
                      Use comma separated list for channel aggregation, and
                      additional -e options for multiple providers.
    -m <filename|path>Install Scali MPI Connect for Myrinet.
                      <filename> is gm-2.x source file package (.tar.gz).
                      <path> path to pre-installed GM-2 software.
    -b <filename|path>Install Scali MPI Connect for Infiniband.
                      <filename> is Mellanox SDK-3.x or IBGD source
                      file package (.tar.gz)
                      Please make sure to use the correctdriver-kernel version!
                      Consult the Mellanox Release Notes if in doubt.
                      <path> path to pre-installed Mellanox compatible software
                      (ex. software from InfiniCon, TopSpin or Voltaire)
    -s                Install Scali MPI Connect for SCI.
    -z                Install and configure SCI management software.
    -u <licensefile>  Install/upgrade license file and software.
    -n <hostname>     Specify hostname of scali license server.
    -l                Create license request (only necessary on license server).
    -i                Ignore SSP check.
    -x                Ignore errors.
    -V                Print version.
    -h/-?             Show this help message.
```

**Note:** You must have root privileges to install SMC.

One or more of the product selection options (-t, -e, -m, -b and -s) must be specified to install a working MPI environment. The -u option can be used to install the license manager software on a license sever, and the -z option can be used to install the SCI management software.

## C-2 Install Scali MPI Connect for TCP/IP

To install Scali MPI Connect for TCP/IP, please specify the -t option to smcinstall. No further configuration is needed.

## C-3 Install Scali MPI Connect for Direct Ethernet

To install Scali MPI Connect for Direct Ethernet, please specify the -e option to smcinstall. This option has the following additional syntax :

**-e** <eth devs> : configures DET provider(s). Use comma separated list
for channel aggregation. Use multiple -e options for additional DET providers.

**Example:**

```
root# ./smcinstall -e eth0 -e eth1,eth2
```

The command in the example will create a DET device (det0) using Ethernet interface eth0 and then a DET device (det1) using eth1 and eth2 aggregated. Please not that aggregated devices usually require special switch configurations, for example separate switches for each interface channel or in some cases two different VLANs one for each channel.

## C-4 Install Scali MPI Connect for Myrinet

To install Scali MPI Connect for Myrinet, please specify the -m option to smcinstall. This option has the following additional syntax :

**-m** <filename|path>Install Scali MPI Connect for Myrinet.
<filename> is gm-2.x source file package (.tar.gz)
<path> is path to an exisitng gm installaton.

**Examples:**

```
root# ./smcinstall -m /home/download/gm-2.0.8_Linux.tar.gz
```

uses the GM source package `/home/download/gm-2.0.8_Linux.tar.gz.`

```
root# ./smcinstall -m /usr/local/gm
```

uses the GM installation in /usr/local/gm

When this option is selected SMC will default to Myrinet as the default transport device.
If this is not desired, modify the networks line in the global `/opt/scali/etc/ScaMPI.conf` configuration file. See chapter2.2 “SMC network devices” on page 12 for more information regarding network selection.

When SMC has finished installing all the required packages and an existing installation isn’t found, the Myrinet GM drivers will start to build. If the build process finishes successfully, it will install a package containing the relevant GM libraries, driver and binaries. The location of the libraries and binaries is `/opt/gm`, and the kernel driver is installed in the appropriate kernel module directory.

## C-5 Install Scali MPI Connect for Infiniband

When installing for InfiniBand you must obtain a software stack from your vendor. The different vendors provide stacks that differs. If you got a binary release, install it before SMC and give the path to the infiniband software to the -b option to smcinstall.

**Example:**
```
root# ./smcinstall -b /opt/Infinicon
```

It is no problem if you install the InfiniBand software after SMC, you only need to modify `/opt/scali/etc/ScaMPI.conf` to have the line:

```
networks = smp,ib0,tcp
```

and ensure that the VAPI library (libvapi.so) is in a directory listed in `/etc/ld.so.conf`.

If you're using the Mellanox source distribution you can give the path to the tarball directly and smcinstall will compile, make a rpm and install it for you.

**Example:**
```
root# ./smcinstall -b /tmp/mellanox_sdk.tar.gz
```

## C-6 Install Scali MPI Connect for SCI

To install Scali MPI Connect for SCI, please specify the -s option to smcinstall. When this option is selected, SMC will default to SCI as the default transport device .If this is not desired, modify the networks line in the global `/opt/scali/etc/ScaMPI.conf` configuration file. See “SMC network devices” on page 12 for more information regarding network selection.

## C-7 Install and configure SCI management software

This option must be used separately, and is needed when you are installing Scali MPI Connect for SCI. It must be installed on *only* one node in your system, and it doesn’t have to be one of the nodes you’re installing the other MPI software on, i.e it can be an management only node, the only requirement is that this node must be connected and on the same TCP/IP subnet as the others.

When using this option you are asked for the names of the other nodes in your cluster, and also the topology of your SCI network (ring, 2D torus or 3D torus).

## C-8 License options

**-u <licensefile>** - Install/upgrade license file and software.

If not specified during install, only the license manager software is installed. Without a license file (license.dat), the software expects a centralized license scheme and looks for a license server (specified in `/opt/scali/etc/scalm.conf`)

If **smcinstall** is run as:

```
root# /opt/scali/sbin/smcinstall -u <licfile>
```

the specified license file is installed and the Scali license manager software (scalm) is installed.

**-n <hostname>** - Specify hostname of Scali license server

This option tells the software which host to contact to check out a license. This can also be manually edited by modifying the scalm_net_server parameter in `/opt/scali/etc/scalm.conf`.

**-l** - Creates a license request to be sent to **license@scali.com**. Host information from the license server must be included in the license request.

Scali MPI Connect is licensed software. You need a license from Scali to be able to run an MPI application using the **mpirun** or **mpimon** program launcher. Usually Scali will provide a time-limited demo license to be used for installation and system test. Then a permanent license request is sent to license@scali.com by the user. Scali will process the license request and reply with a permanent license file. This file must be installed as `/opt/scali/etc/license.dat` on the license server using the following command (as described above):

```
root# /opt/scali/sbin/smcinstall -u <licfile>
```

## C-9 Scali kernel drivers

Scali MPI Connect contains proprietary kernel mode drivers which are loaded into the kernel. The drivers (ScaKal, ScaDET and ScaSCI) will automatically build to fit the running kernel, provided that a fully configured kernel source tree is installed. This is provided by the kernel-source RPM on SUSE and RedHat distributions, however SUSE might require some manual configuration.

If the automatic build process fails, the drivers must be built manually using the script `/opt/scali/libexec/rebuild_module.sh` in the following way :

```
root# /opt/scali/libexec/rebuild_module.sh scakal <path to your linux kernel source>
```

optionally :

```
root# /opt/scali/libexec/rebuild_module.sh scadet <path to your linux kernel source>
```

if Scali MPI Connect for Direct Ethernet is installed, and :

```
root# /opt/scali/libexec/rebuild_module.sh ssci <path to your linux kernel source>
```

if Scali MPI Connect for SCI is installed. To complete the process, re-run the `smcinstall` script with the same options as previously used.

## C-10 Uninstalling SMC

To remove Scali MPI Connect, use the script :

```
root# /opt/scali/sbin/smcunistall
```

## C-11 Troubleshooting Network providers

The Scali MPI Connect now uses DAT as its API to connect to drivers for different interconnects. In DAT terminology the drivers are called *provider libraries*, or *dapl*'s.

### C-11.1 Troubleshooting 3rdparty DAT providers

The only requirements are that the libraries have the proper permissions for shared objects, and that the **/etc/dat.conf** is formatted according to the standard.

All available devices are listed with the `scanet` command.

### C-11.2 Troubleshooting the GM provider

The GM provider provides a network device for each Myrinet card installed on the node, named gm0, gm1... etc.

To verify that the gm0 device is operational, run an MPI test job on two, or more, nodes in question:

```
user% mpimon -networks gm0 /opt/scali/examples/bin/bandwidth -- node1 node2
```

If the gm0 devices fails, the MPI job should fail with a "[ 1] No valid network connection from 1 to 0" message.

First of all, keep in mind that the GM source must be obtained from Myricom, and compiled on your nodes. Scali provides the ScaGMbuilder package to do the job for you, the README, and RELEASE_NOTES (under /opt/scali/doc/ScaGMbuilder) describes the procedure.

If you have just in installed your cluster, upgraded the GM source or just replaced the kernel, the compilation of GM is in progress (takes about 10 min) is run. Verify that the GM binary is installed with:

```
root# rpm -q gm
```

This should report whether the package is installed or not.

The (re)build process require that compiler tools, and kernel source is installed on all nodes.

Verify that the "gm" kernel module is loaded by running `lsmod(8)` on the compute node in question.

Verify that GM is operational, a

```
root# /opt/gm/bin/gm_board_info
```

is enough to check, you should see all the nodes on your GM network listed. (This command must be run on a node with a Myrinet card installed!)

A simple cause of failure is that `/opt/gm/lib` is not in `/etc/ld.so.conf` and/or `ldconfig` is not run, you will get a unable to find libgm.so error message is this is the case.

# Appendix D Bracket expansion and grouping

To ease usage of Scali software on large cluster configuration, many of the command line utilities have bracket expansion and grouping functionality.

## D-1 Bracket expansion

The following syntax applies:

```
<bracket>                 == "["<number_or_range>[,<number_or_range>]*"]"
<number_or_range>         == <number> | <from>-<to>[:<stride>]
<number>                  == <digit>+
<from>                    == <digit>+
<to>                      == <digit>+
<stride>                  == <digit>+
<digit>                   == 0|1|2|3|4|5|6|7|8|9
```

This is typically used to expand nodenames from a range, using from 1 to multi-dimensional numbering, or an explicit list.

If <to> or <from> contains leading zeros, then the expansion will contain leading zeros such that the width is constant and equal to the larger of the widths of <to> and <from>.

The syntax does not allow for negative numbers. <from> does not have to be less that <to>.

**Examples:**

n[0-2]
is equivalent to
n0 n1 n2

n[00-10:3]
is equivalent to
n00 n03 n06 n09

## D-2 Grouping

Utilities that use scagroup will accept a group alias wherever a host name of hostlist i expected. The group alias will be resolved to a list of hostnames as specified in the file scagroup config file. If there exists a file .scagroup.conf in the users home directory, this will be used. Otherwise, the system default file **/opt/scali/etc/scagroup.conf** will be used.

#### D-2.1 File format

Each group has the keyword group at the beginning of a line followed by a group alias and a list of hostnames included in the group. The list may itself contain previously defined group aliases which will be recursivly resolved. The host list may use bracket expressions which will be resolved as specified above.

The file may contain comments which is a line starting with # .

**Examples:**

```
group master n00
group slaves n[01-32]
group all master slaves
```

# Appendix E Related documentation


**[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] Dat Collaborative: User-level API-spesification(uDAPL)**
http://www.datcollaborative.org

**[5] Scali Manage Users Guide**
Scali AS, http://www.scali.com/

**[6] Scali MPI Connect Product Description**
Scali AS, http://www.scali.com/

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

**[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 - High Performance Debugger Forum, http://www.ptools.org/hpdf/

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

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

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

**[13] ROMIO - A high-performance, portable implemetation of MPI-IO**
The I/O chapter in MPI-2. Homepage: http://www-unix.mcs.anl.gov/romio/

# List of figures

# Index

**T**