



DELIVERABLE D2.1

PART II: SOFTWARE REQUIREMENTS SPECIFICATION FOR MPI CODE DEBUGGING AND VERIFICATION

WP2.2 MPI Code debugging and verification

| | |
|--------------------------|--------------------------------------------------|
| Document Filename: | CG-2.2-DOC-USTUTT005-SRS |
| Work package: | WP2.2 MPI Code debugging and verification |
| Partner(s): | USTUTT,CSIC |
| Lead Partner: | USTUTT |
| Config ID: | CG-2.2-DOC-0003-1-0-FINAL-C |
| Document classification: | PUBLIC |

Abstract: This document specifies the software requirements for CrossGrid Task 2.2 'MPI code debugging and verification'.



Delivery Slip

| | Name | Partner | Date | Signature |
|--------------------|-----------------------------------------------|-----------------------|-------------|------------------|
| From | WP2, task 2 | USTUTT, CSIC | 4/4/2002 | |
| Verified by | N. Meyer, P. Wolniewicz, R. M. de Llano | PSNC, U. Cantabria | 21/5/2002 | |
| Approved by | | | | |

Document Log

| Version | Date | Summary of changes | Author |
|----------------|-------------|---------------------------------|---------------------------------|
| 1-0-DRAFT-A | 4 Apr 2002 | Draft version | Matthias Mueller |
| 1-0-DRAFT-B | 8 Apr 2002 | Almost final draft | Matthias Mueller |
| 1-0-Final | 7 May 2002 | Deliverable for Internal Review | Matthias Mueller, Doerthe Pesek |
| 1-0-Final C | 3 June 2002 | Deliverable | Matthias Mueller, Doerthe Pesek |

CONTENTS

| | |
|-----------------------------------------------------|-----------|
| 1. INTRODUCTION..... | 4 |
| 1.1. PURPOSE | 4 |
| 1.2. SCOPE | 4 |
| 1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS | 4 |
| 1.4. REFERENCES | 4 |
| 1.5. OVERVIEW | 4 |
| 2. OVERALL DESCRIPTION..... | 6 |
| 2.1. PRODUCT PERSPECTIVE | 6 |
| 2.1.1. <i>System interfaces</i> | 6 |
| 2.1.2. <i>User interfaces</i> | 7 |
| 2.1.3. <i>Hardware interfaces</i> | 7 |
| 2.1.4. <i>Software interfaces</i> | 7 |
| 2.1.5. <i>Communications interfaces</i> | 7 |
| 2.1.6. <i>Memory constraints</i> | 8 |
| 2.1.7. <i>Operations</i> | 8 |
| 2.1.8. <i>Site adaptation requirements</i> | 8 |
| 2.2. PRODUCT FUNCTIONS..... | 8 |
| 2.3. USER CHARACTERISTICS | 9 |
| 2.4. CONSTRAINTS | 9 |
| 2.5. ASSUMPTIONS AND DEPENDENCIES | 9 |
| 2.6. APPORTIONING OF REQUIREMENTS | 9 |
| 3. SPECIFIC REQUIREMENTS..... | 10 |
| 3.1. EXTERNAL INTERFACES | 11 |
| 3.2. FUNCTIONS | 11 |
| 3.3. PERFORMANCE REQUIREMENTS | 11 |
| 3.4. LOGICAL DATABASE REQUIREMENTS | 11 |
| 3.5. DESIGN CONSTRAINTS..... | 11 |
| 3.6. STANDARDS COMPLIANCE..... | 12 |
| 3.7. SOFTWARE SYSTEM ATTRIBUTES | 12 |
| 4. APPENDIXES..... | 13 |
| 5. INDEX..... | 14 |

1. INTRODUCTION

This document provides the software requirements for the MPI verification tool.

1.1. PURPOSE

This document specifies the software requirements for the MPI code verification tool, within Task 2.2 ‘MPI code debugging and verification’. The intended audience is both the Task itself and and dependent tasks.

1.2. SCOPE

The products of Task 2.2 are:

- (a) A tool for verifying the standard conformance of a MPI program
- (b) some simple programs to test the tool . Since they are used only internally in WP 2.2 the test programs are strictly speaking no product of this task. They are however crucial for first tests. Since they are required to test the features that will be implemented early a detailed description will be given with the design document of this tool.

1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

MPI: Message Passing Interface (<http://www-unix.mcs.anl.gov/mpi/>)

1.4. REFERENCES

MPI Standards: <http://www.mpi-forum.org/docs/docs.html>

Totalview: <http://www.etnus.com/Products/TotalView/index.html>

J.S. Vetter and B.R. de Supinski, “ Dynamic Software Testing of MPI Applications with Umpire.”, SC2000: High Performance Networking and Computing Conf., IEEE/ACM.

ISO/IEC 14882:1998(E) Programming languages – C++

GCC: <http://www.gnu.org/software/gcc/gcc.html>

1.5. OVERVIEW

This document provides the software requirements for the MPI verification tool.

Section 2 provides an overall description of the MPI verification tool, which specifies the product perspective, functions user characteristics, constraints, assumption and dependencies as well as apportioning of requirements.

Section 3 provides a description of specific requirements for the MPI verification tool, which comprise external interfaces, functions, performance requirements, logical database requirements, design constraints, standards compliance and software system attributes.

Throughout this document the terms “MPI code debugging and verification tool” , “MPI verification tool” and “MPI code verification tool” are used as synonyms,

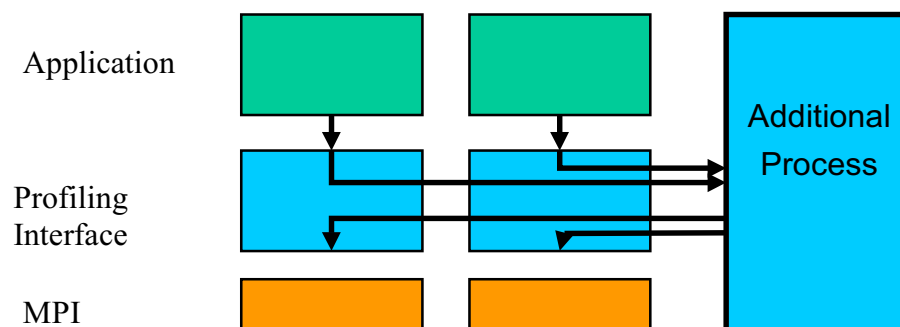
2. OVERALL DESCRIPTION

2.1. PRODUCT PERSPECTIVE

The objective is to develop a tool that verifies the correctness of parallel, distributed Grid applications using the MPI paradigm. The primary issues are how to make end-user applications portable, reproducible and reliable on any platform of the Grid. Another goal is to enable the debugging of applications executing on hundreds of processors. Simple (NxN) matrix algorithms, benchmarks developed in Task T2.3 as well as applications from Tasks T1.2, T1.3 and T1.4 will be used during the test and refinement phase.

The MPI standard allows certain implementation dependent behaviour, e.g. whether an MPI_Send is blocking or non-blocking. This tool will increase the portability by checking whether the application depends on this kind of differences between the different MPI implementations.

The technical basis of this development is the MPI profiling interface which allows a detailed analysis of the MPI application at runtime. This tool is therefore not a replacement of the MPI library, but an add on. It will be used in addition to the existing MPI library. The MPI standard contains a detailed description of the profiling interface in chapter 8 of MPI 1.1 and in section 4.18 of MPI 2.0.



From the user point of view this tool will be a library that is linked to his application. If this library is linked to the application a report will be generated at runtime which contains comments regarding the MPI standard conformance of the MPI program.

Timetable:

Month 3: Requirement specification, including MPI calls of application

Month 6: Design document

Month 12: First prototype: contains the subset of MPI calls required by the end user applications and is running on a local environment.

Month 24: Second prototype: is running on the Grid environment

Month 33: Final version: supports the full MPI 1.2 standard and is running on the Grid

2.1.1. System interfaces

No system interfaces required by the software itself, the requirements are imposed by the underlying MPI layer.

2.1.2. User interfaces

In order to meet the requirement of scalability to long runs on many processors, it is intended to produce an report regarding the MPI conformity in an automatic way. It will be written to standard output or to a file. Possible configuration will be done with environment variables. The format will be human readable.

2.1.3. Hardware interfaces

None required by the software itself. The requirements are imposed by the underlying MPI layer.

2.1.4. Software interfaces

Required software:

- MPI library: since the program that is to be verified is written using MPI, this library is needed to run the program. The verification is done at runtime. There are several reasons for an analysis at runtime:
 1. The MPI standard itself allows to set various variables like MPI_COMM_WORLD at runtime, this may conflict with compile time analysis
 2. The programming style supported by MPI is dynamic in nature. Major decisions taken by the program are done at runtime. A static compile time analysis will therefore not be able to cover the complete behaviour of the program.
 3. A compile time analysis would require language parsers for the different versions of Fortran (F77,F90, F95) as well as for C and C++. A compile time analysis only requires two versions of the library with the bindings supported by MPI 1.2 (C and Fortran).

The disadvantage of a runtime verification is, that all MPI calls have to be supported. Otherwise a complete runtime analysis would not be possible. Since the functionality and semantics of the called MPI functions have to be done, the task of writing such a tool is comparable to writing a complete MPI library.

- The tool verifies the calls made by the program with the use of the so called profiling interface. This allows to replace the native MPI calls with routines provided by this tool. The calls are then checked for possible errors and afterwards passed on to the real MPI library. The profiling interface is part of the MPI standard. Any MPI implementation that conforms to the MPI standard needs to provide this interface. Therefore, this requirement should not limit the selection of possible MPI libraries.
- C++ compiler: it is intended to implement the tool in C++ as far as possible. The compiler should implement the ISO/IEC 14882 language specification of C++. At least gcc 3.0.4 is required for a sufficient support of the C++ language standard. Intel Compilers are an alternative, they are available for no cost for non-commercial use on linux platforms.
- To support the Fortran binding of the MPI standard a Fortran compiler is required. The same Fortran compiler should be used to compile the application.

2.1.5. Communications interfaces

All communication is done using MPI.

2.1.6. Memory constraints

The memory constraints are dominated by the running application. The tool will add some requirements to the memory footprint, but it is expected to be negligible.

2.1.7. Operations

As with all tools using the MPI profiling interface a relink of the application is required. The library containing the tool replaces the MPI library and the library containing shifted names of the MPI profiling interface is also linked. No source code modification is required. In the current envisioned implementation of the tool an additional MPI process needs to be added at startup. E.g. the user will specify N+1 processes to the mpirun command, whereas the application will use only N MPI processes. This will be transparent to the application. The additional process may run on any machine in any place, for the test-bed it is just an additional MPI process. This additional process will monitor the progress of the application and signal conditions like dead-locks. The communication with this process and the other processes is done with MPI.

2.1.8. Site adaptation requirements

If the application is started in an automatic mode the number of processes provided to the application should be different than the number provided to the mpirun command. It is expected that this imposes no conflict, since the number of processes is a redundant information than can be queried at runtime using `MPI_Comm_size`.

The library will be a static library, therefore it is not required that it is installed on all grid nodes. Just on the node where the program is compiled/linked.

2.2. PRODUCT FUNCTIONS

Although the tool will focus on the application support with the involved MPI calls (16, 9, 10, and 14 different calls depending on the application) it will finally support the complete MPI 1.2 standard with a total of 128 calls.

The tool will verify the correctness of the MPI program in several ways:

- Verify that redundant information is self-consistent
- Verify that the arguments adhere to the requirements specified in the MPI standard
- Verify that resources handled by the user are handled in a correct way. Examples for resources are:
 1. `MPI_Requests`
 2. `MPI_Datatypes`
 3. `MPI_Communicators`
 4. `MPI_Groups`
 5. User defined MPI Operators.

Each of this resources have to be traced and managed by the verification tool. Normally the user has the responsibility for the correct management of this resources, e.g. there is no state associated with MPI calls other than in the `MPI_Request`. Therefore, there is no support for the correct management within the MPI library.

Examples for incorrect usage of MPI calls and its arguments are:

- non-valid ranks in group constructor calls
- non-distinct ranks in `MPI_Group_incl`, `MPI_Group_excl`, `MPI_Group_ranks_excl`
- group is no subset of communicator in `MPI_Comm_create`
- `color < 0` in `MPI_Comm_split`
- grid size > group size in `MPI_Cart_create`
- wrong communicator structure in topology calls

2.3. USER CHARACTERISTICS

The intended user is an application programmer writing parallel applications using MPI. He is experienced in writing and running MPI applications on one platform, e.g. a linux cluster. Experiences on different platform are not required and the user will be supported to achieve portability.

2.4. CONSTRAINTS

The tool will be implemented in C++, a compiler adhering to ISO 14882 is required. For the Fortran binding a fortran compiler is necessary. It should be the same compiler the application is using, there are no special requirements from WP 2.2 to the fortran compiler.

The program that is to be verified will be executed in parallel. The start-up will be handled by the normal MPI library, or by the procedure used by this library or the Crossgrid testbed.

The expectation is that the running application that is verified at runtime will behave to the test-bed like a “normal” application. However, the behaviour will be different from the unverified application (additional MPI messages, different runtime, slightly increased memory size).

2.5. ASSUMPTIONS AND DEPENDENCIES

It is assumed that the parallel programs can be executed on the CrossGrid testbed in a parallel way. One additional process needs to be added to allow the verification process.

2.6. APPORTIONING OF REQUIREMENTS

3. SPECIFIC REQUIREMENTS

In order to provide optimal user support for WP1 the tool should focus on the MPI calls made by the application of WP1. That means that this calls should be implemented before other calls in order to provide the user with a useful tool as soon as possible.

These are:

Task 1.1:

- MPI_Barrier
- MPI_Bcast
- MPI_Cart_create
- MPI_Cart_rank
- MPI_Cart_shift
- MPI_Comm_rank
- MPI_Comm_size
- MPI_Finalize
- MPI_Init
- MPI_Pack
- MPI_Recv
- MPI_Reduce
- MPI_Send
- MPI_Sendrecv
- MPI_Unpack
- MPI_Wtime

Task 1.2:

- MPI_INIT
- MPI_COMM_SIZE
- MPI_COMM_RANK
- MPI_GET_PROCESSOR_NAME
- MPI_BCAST
- MPI_SEND
- MPI_RECV
- MPI_BARRIER
- MPI_FINALIZE

WP 1.3:

```
MPI_INIT(&argc, &argv);
MPI_COMM_SIZE(MPI_COMM_WORLD, &nproc);
MPI_COMM_RANK(MPI_COMM_WORLD, &rank);
MPI_GET_PROCESSOR_NAME(processor_name, &namelen);
MPI_BARRIER(MPI_COMM_WORLD);
MPI_BCAST(buffer_int, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
MPI_FINALIZE();  
MPI_RECV(buffer_dbl, num_var, MPI_DOUBLE, i, 30, MPI_COMM_WORLD, &status);  
MPI_SEND(buffer_dbl, num_var, MPI_DOUBLE, 0, 30, MPI_COMM_WORLD);  
MPI_REDUCE(buffer_dbl_slave, buffer_dbl_master, num_datos, MPI_DOUBLE, MPI_SUM, 0,  
           MPI_COMM_WORLD);
```

WP 1.4:

```
MPI_TYPE_HVECTOR  
MPI_SCATTERV  
MPI_BCAST  
MPI_GET_PROCESSOR_NAME  
MPI_GATHERV  
MPI_FINALIZE  
MPI_INIT  
MPI_TYPE_STRUCT  
MPI_WTIME  
MPI_COMM_SIZE  
MPI_BARRIER  
MPI_COMM_RANK  
MPI_TYPE_EXTENT  
MPI_TYPE_COMMIT
```

3.1. EXTERNAL INTERFACES

The input to the tool consists of a sequence of MPI calls together with the provided arguments. The possible calls and the arguments are described in the MPI standard. The output contains of a list of problems that have been identified during the execution of the program.

3.2. FUNCTIONS

The system shall provide a list of detected problems at the end of program execution. Since the program may possibly crash during program execution this output should optional be produced during the runtime of the program, in order to get most of the output before the program fails. Since there is no support for error recovery within the MPI standard the system is not expected to recover from failures of the underlying MPI system.

3.3. PERFORMANCE REQUIREMENTS

The system should be able to verify programs running on the CrossGrid testbed. The number of supported processors should be limited by the testbed, not by the tool.

3.4. LOGICAL DATABASE REQUIREMENTS

None.

3.5. DESIGN CONSTRAINTS

3.6. STANDARDS COMPLIANCE

Violation of the MPI standard will be reported as errors. Unusual behaviour or possible problems will be reported as warning. E.g. it is unusual but allowed to have zero length messages.

3.7. SOFTWARE SYSTEM ATTRIBUTES

The tool should be portable to any system that supports ISO/IEC 14882 and the MPI standard.

4. APPENDIXES

5. INDEX