



## DELIVERABLE D2.7

### DEMONSTRATION AND DOCUMENTATION OF THE FINAL VERSIONS OF CROSSGRID DEVELOPMENT TOOLS

#### WP2 Grid Application Programming Environment

---

Document Filename:	<b>CG2.0-D2.7-v1.1-FZK019-FinalReport.doc</b>
Work package:	<b>WP2 Grid Application Programming Environment</b>
Partner(s):	<b>CYFRONET, FZK, USTUTT, TUM, UCY, USC, CISC</b>
Lead Partner:	<b>FZK</b>
Config ID:	<b>CG2.0-D2.7-v1.1-FZK019-FinalReport</b>
Document classification:	<b>PUBLIC</b>

---

**Abstract:** This document (deliverable D2.7) presents an overview and a summary of the final results achieved in workpackage 2 “Grid Development Tools”. All tools together build a suite for Grid developers. This document describes briefly the tools and their functionalities. This document is complemented by the different guides for each tools listed in the appendix of this document.

### Delivery Slip

	Name	Partner	Date	Signature
<b>From</b>	Harald Kornmayer	FZK	22/12/2004	
<b>Verified by</b>	Harald Kornmayer	FZK	17/01/2005	
	Robert Pająk	CYFRONET	31/01/2005	
<b>Approved by</b>	Harald Kornmayer	FZK	31/01/2005	

### Document Log

Version	Date	Summary of changes	Author
0.1	21-12-2004	Skeleton	Harald Kornmayer
0.2	10-01-2005	Reviewed and contribution from g-PM	R. Wismueller, H. Kornmayer
0.3	11-01-2005	Contribution from MARMOT	B. Krammer, H. Kornmayer
0.4	11-01-2005	Contribution from PPC	F. Fernandez Rivera, H. Kornmayer
0.5	11-01-2005	Contribution from GridBench	G. Tsouloupas, H. Kornmayer
0.6	12-01-2005	Smaller changes by first review	H. Kornmayer
0.7	13-01-2005	Introduction, technology matrix	H.Kornmayer
0.8	13-01-2005	New contribution from PPC	F. Fernandez Rivera, H. Kornmayer
0.9	13-01-2005	Captions of figures and tables update	H.Kornmayer
0.95	14-01-2005	Summary of test and integration	R. Wismueller
0.96	16-01-2005	Some correction from MARMOT Update one GridBench picture	B. Krammer, G. Tsouloupas. H. Kornmayer
0.97	17-01-2005	Executive Summary, Update from PPC	H. Kornmayer, F. Fernandez Rivera
0.98	17-01-2005	Correction from Bettina, Fran, Roland, George, Hamza,	WP2
1.0	17-01-2005	Version submitted to internal reviewers	WP2
1.1	27-01-2005	Corrections according to internal review	Internal reviewers
	31-01-2005	Verified by the Quality Engineer	Robert Pajak

---

## CONTENT

<b>1. EXECUTIVE SUMMARY</b> .....	<b>4</b>
<b>DEFINITIONS, ACRONYMS AND ABBREVIATION</b> .....	<b>6</b>
<b>2. RESULTS OF GRID DEVELOPMENT TOOLS FROM CROSSGRID</b> .....	<b>7</b>
INTRODUCTION.....	7
TASK 2.2 – MPI CODE DEBUGGING AND VERIFICATION TOOL (MARMOT) .....	8
TASK 2.3 – GRID BENCHMARKING (GRIDBENCH) .....	13
TASK 2.4 – PERFORMANCE PREDICTION COMPONENT (PPC) .....	20
TASK 2.4 – GRID PERFORMANCE MEASUREMENT TOOL (G-PM).....	26
TASK 2.5 – INTEGRATION, TESTING AND REFINEMENT.....	34
TECHNOLOGY VERSUS APPLICATION MATRIX FOR WORKPACKAGE 2.....	36
<b>3. APPENDIX</b> .....	<b>38</b>
MARMOT .....	38
GRIDBENCH .....	38
PERFORMANCE PREDICTION COMPONENT .....	38
G-PM .....	38
<b>4. REFERENCES</b> .....	<b>39</b>

---

## 1. EXECUTIVE SUMMARY

This document reports on the final results of “Grid Development Tools” from CrossGrid workpackage 2. All tools are integrated in the CrossGrid environment. They are available as open source, and may therefore be used by other Grid projects.

Within workpackage 2 the following tools have been developed:

- MARMOT – MPI verification and debugging tool
- GridBench – A benchmarking suite for Grids
- PPC – Performance Prediction Component
- G-PM – Grid Performance Measurement tool

All tools from workpackage 2 build a suite for developers that support the migration of new MPI applications to a Grid environment. The support for MPI applications is of great importance for the evolution of e-science. And all tools from workpackage 2 help with the migration and development of new MPI applications on the Grid. The G-PM tool together with OCM-G has been chosen by the CrossGrid project as one of the key products of CrossGrid.

All tools from workpackage 2 are integrated in the CrossGrid environment and in the testbed. Task 2.5 (test and integration) worked closely with workpackage 4 to succeed with integration. All tools provide a set of guides (user guide, installation guide, developer guide) to support their exploitation. Tutorials for all “Grid development tools” are available too. These guides are an additional part of this deliverable. The products from workpackage 2 are integrated with tools from workpackage 3 like the Migrating Desktop (Task 3.1) or JIMS (Task 3.3). The source code and RPMs can be downloaded from the central CrossGrid software repository (<http://savannah.fzk.de/projects/cg-wp2>).

MARMOT is a tool to verify and debugging MPI code. The full MPI-1.2 standard is supported for both the C and FORTRAN language binding. The MARMOT tool is integrated in the CrossGrid environment. It has been tested with all the applications from workpackage 1. Applications using MARMOT can be submitted and monitored from the CrossGrid Migrating Desktop (WP 3.1). MARMOT is able to run on different platforms.

GridBench offers a wide suite of benchmarks for a Grid environment including low-level performance metrics as well as application specific performance measurements. With GridBench the Grid user can compare the capabilities of different Grid configurations and resources. For this purpose, the *GridBench Definition Language* (GBDL) was introduced. GridBench is extensible to new benchmark executables and to different Grid middleware. GridBench has developed a graphical user interface to access the benchmarking suite in a user friendly way. With this GUI benchmarks can be submitted just by drag’n’drop. Some application kernels from workpackage 1 are implemented. The application kernel from task 1.1 was studied in detail. The GridBench GUI is integrated in the Migrating Desktop (WP 3.1). GridBench uses infrastructure monitoring information from the JIMS tool (WP 3.3).

The Performance Prediction Component (PPC) task concentrated on the development of performance prediction models both for computation and communication. Inside this task a methodology was developed to obtain these models. With these models the performance of a given application/kernel can be predicted under the assumption of general Grid performance parameters (CPU power, bandwidth). These models are based on infrastructure measurements from the monitoring system JIMS (Task 3.3). Within PPC different kernels were studied in detail and models for these kernels were derived. The models can be visualized with the PPC graphical user interface. The tool can make predictions for the kernels under assumptions about the grid infrastructure. The PPC GUI is integrated in the Migrating Desktop (Task 3.1)

The Grid Performance Measurement tool (G-PM) is the online performance analysis tool from CrossGrid workpackage 2. This tool can display current performance data in the form of various

graphs. The available performance measures are standard performance metrics as well as user defined metrics. These last ones may be derived from the standard metrics as well as from application specific metrics based on specific marked events in the application code, called probes. These new metrics are based on the Performance Measurement Specification Language (PMSL) that was developed within this task. G-PM is made of three components: the Performance Measurement Component (PMC), the High-Level Analysis Component (HLAC) and the User Interface and Visualization Component (UIVC). The G-PM tool is OMIS compliant and therefore dependent on the OCM-G tools from workpackage 3.3. G-PM is also integrated in the Migrating Desktop (WP3.1)

---

## DEFINITIONS, ACRONYMS AND ABBREVIATION

API	Application Program Interface
Avispa	A Visualization tool for Paradise code
BLAS	Basic Linear Algebra Subprograms
CrossGrid	The EU Project CrossGrid IST-2001-32243
CSE	Common Subexpression Elimination
CVS	Concurrent Version System
DAG	Directed Acyclic Graph
F90	Fortran 90 language
FLOPS	Floating point Operations per Second
G-PM	The CrossGrid performance analysis tool
GridBench	The CrossGrid Benchmark Suite
GUI	Graphical User Interface
HEP	High Energy Physics
HLAC	High Level Analysis Component
HPF	High Performance Fortran
HPL	High Performance Linpack
IR	Intermediate Representation
LINPACK	Linear Algebra Package
MPI	Message Passing Interface ( <a href="http://www-unix.mcs.anl.gov/mpi/">http://www-unix.mcs.anl.gov/mpi/</a> )
NWS	Network Weather Service
OCM-G	OMIS Compliant Monitoring system for the Grid
OMIS	On-line Monitoring Interface Specification
PARAISO	Parallel Iterative Solvers Library
PEIR	Partially Evaluated Intermediate Representation
PETSc	Portable, Extensible Toolkit for Scientific Computation
PIPlot	PIPlot plotting library
PMC	Performance Measurement Component of G-PM
PMSL	Performance Metrics Specification Language
PPC	Performance Prediction Component
RPM	RedHat Package Management
SRS	Software Requirements Specifications
TCL/TK	Tool Command Language
UIVC	User Interface and Visualization Component of G-PM
UML	Unified Modelling Language
WP	Work Package
WP <sub>x</sub>	Work Package no. <i>x</i>
XML	eXtensible Markup Language

---

## 2. RESULTS OF GRID DEVELOPMENT TOOLS FROM CROSSGRID

### INTRODUCTION

The CrossGrid project started from the beginning with the development of tools for application developers to support the migration and development of applications on a Grid infrastructure. CrossGrid has ported four societal important applications from different domains (like medical or environmental) to a Grid environment. All CrossGrid applications required interactivity from the underlying middleware and the use of the MPI (Message Passing Interface) programming paradigm. CrossGrid addressed these middleware requirements with inside workpackage 3 and it has developed enhancements to the LCG middleware to submit interactive MPI jobs transparently to the Grid.

But developing a new application or migrating an existing application on a Grid needs to face new problems. The Grid is not under the control of one central administrator, which results in an always dynamically changing system without reproducible working conditions for the users and the developers. Therefore application debugging or performance studies are difficult tasks on a Grid.

Tools for application developers are now available from CrossGrid workpackage 2 (“Grid development tools and programming environment”) to support the migration and development of applications on a Grid infrastructure like the CrossGrid testbed. These tools address topics like

- benchmarking of a Grid configuration (GridBench)
- performance evaluation and prediction of Grid applications (PPC)
- verification and debugging of MPI programs (MARMOT)
- online performance measurements (G-PM)

These tools are described later in this document and more detailed information can be found in the dedicated chapters and the user’s, installation’s and developer’s guides for each tool.

With the results of workpackage 2, CrossGrid provides a suite of tools to support the development and optimisation of Grid applications. With GridBench one gets a better understanding of the capabilities of a given Grid infrastructure. By building computation and communication models with the CrossGrid PPC tool, developers get a better insight in their code. The results of both tools have the potential to be used in the future as input for a Grid resource broker to optimize the scheduling. MARMOT supports the developers of MPI Grid applications. After the usage of MARMOT, MPI code will be more portable to other systems. With G-PM application developers can study the performance of Grid applications at runtime. The quality of the code may be improved since performance bottlenecks may be eliminated.

All the CrossGrid development tools are integrated with the CrossGrid Migrating Desktop (MD). With the MD the access to the Grid infrastructure – and to the Grid development tools – is more user intuitive. The usage of CrossGrid development tools will finally produce better Grid-enabled applications.

As currently a lot of different projects start with the migration of MPI applications to the Grid (like the generic applications in EGEE [EGEE]), CrossGrid development tools can support these applications. They can benefit from the work done in the context of CrossGrid to enable MPI applications in a Grid infrastructure.

---

## TASK 2.2 – MPI CODE DEBUGGING AND VERIFICATION TOOL (MARMOT)

### Introduction

The Message Passing Interface (MPI) is a widely used standard for writing parallel code. However, the MPI-1.2 standard, with its 129 calls, has a size and complexity so that it is possible to incorrectly use its API. According to our experience there are several reasons for this:

- Developers do not only have to face all the problems that occur in serial programming. In addition, parallel applications get more and more complex and more error prone, especially with the introduction of optimisations like the use of non-blocking communication.
- MPI programs do not always behave deterministically. Deadlocks or race conditions may appear, depending on the platform environment or on the MPI implementation.
- The MPI standard leaves many decisions to the implementation, e.g. whether or not a standard communication is blocking. This implementation-dependent behaviour may cause problems when porting an application from one platform to another, for example, when porting an application from a local platform to the CrossGrid testbed.

### State of the Art

Debugging MPI programs has been addressed in various ways. The different solutions can be roughly grouped in four different approaches: classical debuggers, special MPI libraries and other tools that may perform a run-time or post-mortem analysis.

- Classical debuggers have been extended to address MPI programs. This is done by attaching the debugger to all processes of the MPI program. There are many parallel debuggers, like the very well-known commercial Totalview [TV]. The freely available gdb debugger has currently no support for MPI. However, it may be used as a back-end debugger in conjunction with a front-end that supports MPI, e.g. mpigdb [MPIGDB]. Another example of such an approach is the commercial debugger DDT [DDT] by streamline computing, or the non-freely available p2d2 [P2d2].
- The second approach is to provide a debug version of the MPI library (e.g. mpich.) This version is not only used to catch internal errors in the MPI library, but it also detects some user incorrect usage of MPI, e.g. a type mismatch of sending and receiving messages.
- Another possibility is to develop tools dedicated to finding problems within MPI applications. At present, three different message-checking tools are under active development: MPI-CHECK, Umpire [UMPIRE] and MARMOT. MPI-CHECK [MPICHECK] is currently restricted to FORTRAN code and performs argument type checking or finds problems like deadlocks. Like MARMOT, Umpire uses the profiling interface. Unfortunately, Umpire is not freely available. These three tools all perform their analysis at runtime.
- The fourth approach is to collect all the information on MPI calls in a trace file, which can be analysed by a separate tool after program execution. A disadvantage with this approach is that such a trace file may be very large. However, the main problem is guaranteeing that the trace file is written in the presence of MPI errors, because the behaviour after an MPI error is implementation dependent.

### Design of Marmot

MARMOT does not change the MPI library nor the application. We use the profiling interface that is defined in the MPI standard 1.2, but probably not all users know this profiling interface or have ever

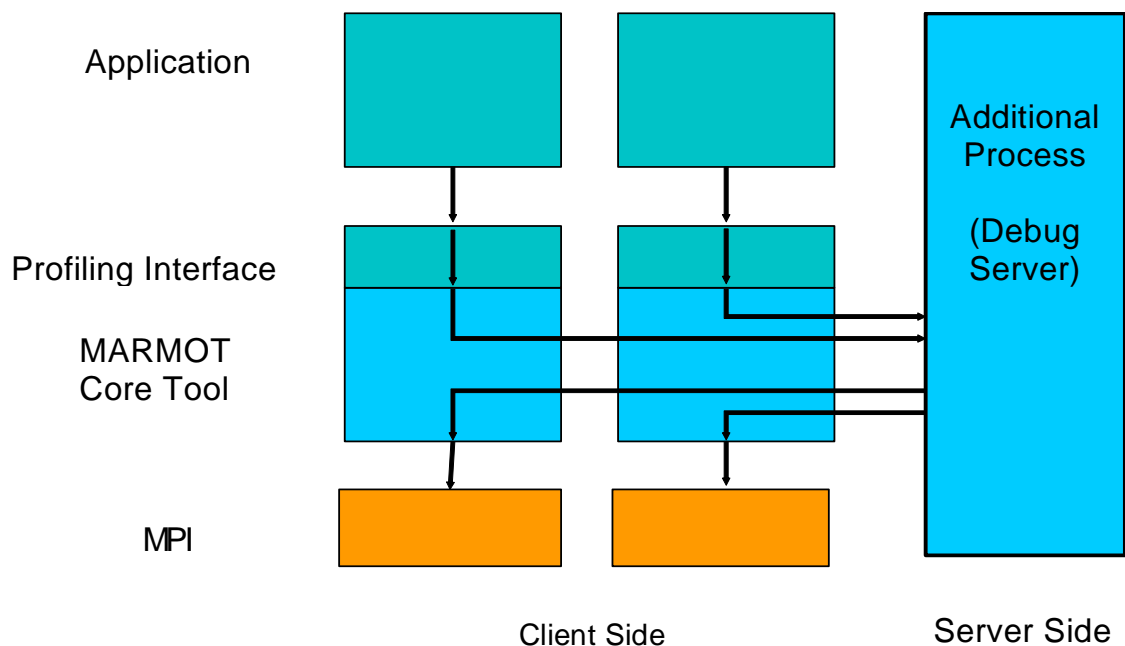
used it. The idea of the profiling interface is simple: any MPI call `MPI_Foo` can also be invoked by `PMPI_Foo` with exactly the same behaviour. Programmers may then define their own MPI routines. That is what MARMOT does. To put it in a simple way, we redefine `MPI_Foo` by

```
{
  do the MARMOT checks;
  PMPI_Foo;
}
```

Thus we check the MPI calls made by the application and pass them to the underlying MPI library. Marmot is just an addition to the MPI library, not a replacement.

MARMOT also has its own book-keeping of resources such as datatypes, groups, communicators, etc. This allows the tool to check if these resources are used in a correct way like for example if they are constructed, used and freed properly.

MARMOT requires an additional process (our debug server), which is responsible for tasks that require a global view, e.g. deadlock detection. Local tasks such as the verification of resources are performed on the client side. The additional process is caught automatically by the tool. The user only needs to run the application with one additional process. The figure below shows a graphical representation of the way MARMOT works.



**Figure 1: The basic architecture of MARMOT**

Details of the design can be found in previous deliverables and also in the developer's, installation and user's guides. MARMOT's functionality includes:

- No modification of the source code is necessary. One additional process working as MARMOT's debug server is requested.
- MARMOT is a library written in C++, which has to be linked to your application in addition to the existing MPI library.

- MARMOT will check if your application conforms to the MPI standard and will issue warnings if there are errors or non-portable constructs.
  - MARMOT is able to keep track of the proper construction, usage and freeing of all MPI resources, such as communicators, groups, datatypes, etc. It also checks if requests and other arguments (tags, ranks, etc.) are used correctly. The main functionality has been implemented for the C language binding, whereas the functionality for the FORTRAN language binding is obtained through a wrapper to the C interface. Special attention has to be paid to the verification of the datatypes because they are one of the major differences between the C and the FORTRAN language binding.
  - MARMOT issues warnings if the application relies on non-portable MPI constructs, and error messages if erroneous calls are made. Although high-quality MPI implementations detect some of these errors themselves, there are many cases where no warning is issued. For example, non-portable implementation-specific behaviour is not indicated by the implementation, nor are checks performed that would decrease the performance too much, such as consistency checks. MPI implementations tolerate some errors without warnings or crashing, simply giving wrong results. Without the help of a tool like MARMOT, users or developers may thus have a hard time finding these errors.
  - Possible race conditions can be identified by locating the calls that may be sources for them. One example is the use of a receive call with the wildcard `MPI_ANY_SOURCE` as source argument or the wildcard `MPI_ANY_TAG` as tag argument. One may argue that one does not need a tool to detect this sort of argument as a simple `grep` command on the source code would give the same result. However, a search command does neither show the execution flow nor will it be able to detect this argument if the application takes functions from some other library with hidden MPI calls. Another source of race conditions is the use of random numbers.
  - The tool also contains a mechanism to automatically detect deadlocks and notify the user where and why they have occurred. In general, deadlocks are caused by the non-occurrence of something else, for example mismatched send/receive operations or mismatched collective calls. Currently the deadlock detection is based on a timeout mechanism. MARMOT's debug server surveys the time each process is waiting in an MPI call. If this time exceeds a certain user-defined limit on all processes at the same time, the debug process issues a deadlock warning and allows the user to trace back the last few calls on each node.
  - MARMOT supports the complete MPI-1.2 standard, although not all possible tests (such as consistency checks) have been implemented so far. It can be used with any standard-conforming MPI implementation and may thus be deployed on any development platform available to the programmer
- MARMOT's output is a human-readable log file.
- The tool can be configured via environment variables.
- Currently MPI-1.2 standard is supported, both the C and FORTRAN language binding.

## Results

The development of MARMOT will still go on after the CrossGrid project has finished. The results of the final CrossGrid prototype are as follows:

- Integration in Grid environment:

- 
- The CrossGrid autobuild process works fine. MARMOT has been deployed with no problems on the CrossGrid testbed.
  - MARMOT has been tested with applications from WP1 and turned out to be useful, see previous deliverables.
  - Applications using MARMOT can be submitted and observed from the Migrating Desktop.
  - A simple example for the CrossGrid tutorial has been prepared. It contains some deliberate errors to demonstrate the tool behaviour.
  - Extended functionality and documentation:
    - MARMOT supports the full MPI-1.2 standard, i.e. it supports all the 129 MPI calls. The extension to parts of MPI-2, like parallel file I/O, is still an ongoing effort.
    - The C and FORTRAN language binding are available. The C++ language binding is defined in the MPI-2 standard and will have to be fully implemented in future.
    - The tool performs further checks of MPI calls. Some of these checks have to be implemented differently, depending on whether the C or the FORTRAN language binding is used.
    - The documentation has been improved: there are now installation, user and developer guides, as well as an improved version of the CrossGrid tutorial exercise and description. A precise and detailed description of the checks performed for all of the 129 MPI calls is under development.
  - Performance improvement:

We used the applications from WP1 and for example the NAS Parallel Benchmarks (see [NASPB-1], [NASPB-2]) to measure MARMOT's overhead. MARMOT's overhead seems reasonable for applications with a reasonable ratio of communication/computation. As a rule of thumb, the higher the ratio of communication/computation, the more the overhead of MARMOT. Indeed, the client/server architecture of the tool builds a bottleneck and is thus an obstacle to scalability.
  - Supported platforms:

MARMOT does not require any special hardware. MARMOT has been tested on the CrossGrid testbed with the CrossGrid applications and also on the following platforms (with CrossGrid applications, some other applications and benchmarks such as the NAS Parallel Benchmarks (see [NASPB-1], [NASPB-2]), using different compilers and MPI implementations):

    - NEC SX5, SX6
    - IBM Regatta
    - Linux IA32/64 clusters
    - Some old platforms, such as Cray T3e, Hitachi SR8000...
  - Dissemination activities:
    - Publications and presentations at various conferences and journals, see [MARMOT-1] – [MARMOT-5].
    - Brochures, poster presentations and live demos, e.g. at SuperComputing'04 like in previous years.
    - MARMOT's homepage contains a short overview and a detailed list of publications, see <http://www.hlr.de/organization/tsc/projects/marmot> .

- We also support users outside the CrossGrid project, for example
  - MARMOT is presented in Parallel Programming courses given by HLRS.
  - Currently two of the three National High Performance Computing Centers in Germany have MARMOT installed:
    - HLRS at Stuttgart (IA32, IA64, NEC SX)
    - NIC at Jülich (IBM Regatta just replaced their Cray T3E)
  - MARMOT has also been evaluated externally by David Cronk and Amanda Laake from Innovative Computing Laboratory, University of Tennessee, USA, within the U.S. DoD High Performance Computing Modernization Program (HPCMP) Programming Environment and Training (PET) project. The reports “Review of Marmot, an MPI debugging tool “ and “Evaluation of program correctness tools” are available (with anonymous login) at [http://pdb.cs.utk.edu/bt.pl?action=bug\\_view&bug\\_id=322](http://pdb.cs.utk.edu/bt.pl?action=bug_view&bug_id=322) and [http://pdb.cs.utk.edu/bt.pl?action=bug\\_view&bug\\_id=325](http://pdb.cs.utk.edu/bt.pl?action=bug_view&bug_id=325) . These reports look upon MARMOT favourably. MARMOT is mentioned as one of two freely available tools for the development of portable MPI programs (i.e. besides the freely available MPI-Check and the commercial Umpire. To our and their knowledge, these three tools are currently the only tools for checking MPI programs for correctness.)
- Quality Assurance:
  - MARMOT was improved according to some suggestions from the QA reports. Its quality is now at a reasonable level.
  - We gave detailed feedback to the CrossGrid Quality Engineer.

---

## TASK 2.3 – GRID BENCHMARKING (GRIDBENCH)

Grids and Grid Resources are often characterized by static information provided by Grid Information systems. Grid end users and central Grid Services (such as resource brokers) need a better source of information on which to base decisions. These decisions mainly refer to resource allocation or scheduling decisions. The use of results from micro and macro-benchmarks can improve the decision making process, but at this point there is no easy, automated way to obtain, manage and deliver these measurements. GridBench is aimed at fulfilling this purpose.

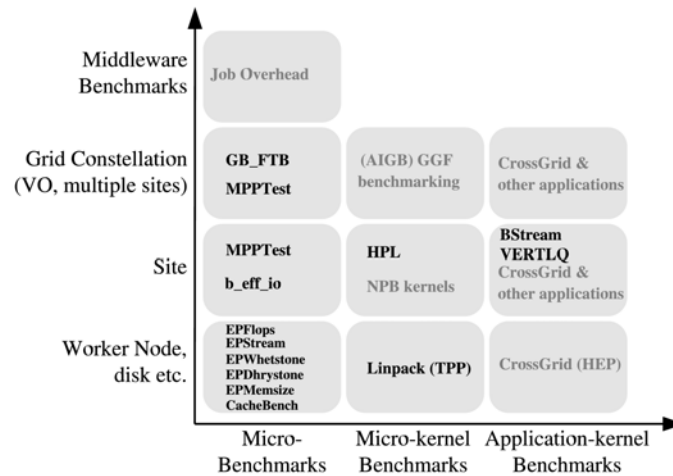
In this section, we describe GridBench, an extensible tool for benchmarking and testing Grid resources. We summarize the GridBench services and tools that provide easy invocation of benchmarks and management of results. In order to illustrate the usage of the tool, we describe scenarios to use the GridBench framework and the GridBench “virtual workbench” to perform benchmarking experiments.

GridBench, as a tool for benchmarking grids, has two main objectives:

1. Generate metrics that characterize the performance capacity of resources both belonging to a Virtual Organization and *spanning across multiple Grid nodes*, in terms of computational power, file-transfer speed, inter-process communication bandwidth, application-kernel performance, scalability etc.
2. Provide a *tool* for researchers that wish to investigate various aspects of Grid performance, using well-understood kernels that are representative of more complex applications deployed on the Grid. Having access to a corpus of such kernels and being able to easily specify and dispatch parameterized runs of them on Grids facilitates the characterization of factors that affect application and infrastructure performance, the quantitative comparison of different middleware solutions, algorithms for scheduling, resource allocation, etc.

To address the two main objectives mentioned, GridBench has two constituents: the *GridBench Benchmark Suite* and the *GridBench Benchmarking Framework*. The GridBench Benchmark suite is a collection of new and existing micro-benchmarks, micro-kernel benchmarks and application benchmarks; its purpose is to generate the metrics that will characterize resources and virtual organizations. The GridBench suite takes a layered approach as shown in Figure 2. The multi-layered structure of the Grid calls for performance measurements at the different layers of the Grid. GridBench seeks to investigate performance properties of the following “layers” of the Grid architecture:

- The Resource, for example a cluster node or a Storage Element;
- The Site, which is a collection of resources interconnected through a local- or system-area network, and belonging to one administrative domain (e.g. a cluster of PCs or a symmetric multiprocessor system);
- The Grid Constellation, which includes multiple sites constituting the computing platform of a Virtual Organization.
- The Middleware, that is the software layer providing access to shared resources of a Grid constellation and which gives the programmer the Grid as a shared resource.



**Figure 2: A layered approach to benchmarking, with micro-benchmarks, micro-kernel benchmarks and application benchmarks on the x-axis, and resource, site and grid constellation on the y-axis.**

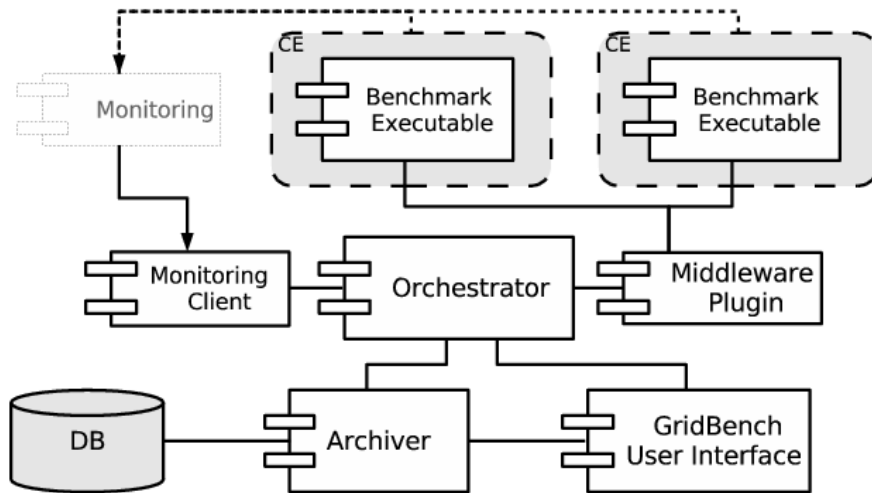
The suite includes benchmarks for CPU (Floating Point and Integer operations), memory bandwidth, cache performance, detection of available physical memory size, interconnection performance (MPI), synthetic benchmarks and application kernels. A detailed description of the GridBench suite is beyond the scope of this article, more details on the GridBench suite can be found in [GTMD03, GTMDTR04].

### The GridBench backend

The GridBench Benchmarking Framework provides facilities for defining and running benchmarks as well as archiving, retrieving and analyzing the results of the GridBench benchmark suite.

GridBench was designed to be as independent of specific middleware as possible. The design is open enough to allow easy replacement of the underlying middleware by the use of *Middleware plug-ins*. Currently implemented are plug-ins for Globus and the LCG2-compatible EU CrossGrid middleware. The user can use the Globus MDS [MDS97] for information retrieval, and either the EU CrossGrid Resource Broker or the Globus GRAM for job execution.

Figure 3 outlines the software architecture of GridBench and (at a very high level) indicates which components interact with each other. This is indicated by a line connecting the two interacting components. The main components of this architecture are:



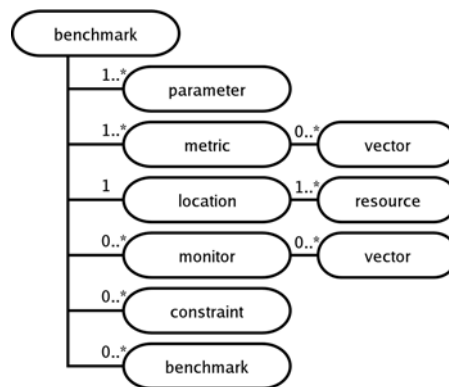
**Figure 3: The GridBench architecture overview. An outline of system's major components and their interaction.**

1. The **GridBench Suite** is made up of the benchmark executables (e.g. Linpack).
2. The **GridBench UI** allows the user to define and submit benchmarks to the *Orchestrator*. It interacts with the *Archiver* to retrieve benchmark *models* and results. These results can be analyzed by the creation of charts, etc.
3. The **Orchestrator** web-service accepts benchmark definitions generated by the *GridBench UI* (or any other source) and manages their execution by the use of the appropriate *Middleware Plugin*. It monitors the job status for each benchmark job and on completion retrieves and archives the resulting metrics.
4. The **Archiver** web-service maintains a repository of benchmark results and model definitions. Provides an interface to a relational database back-end.
5. The **Middleware Plugin** provides the implementations to use different Middleware for benchmark execution and output retrieval. The XML description of benchmarks is transformed to a specific job description language. There are currently two implementations of the Middleware Plugin interface: one for Globus and one for the EU-CrossGrid middleware.
6. The **Monitoring Client** collects infrastructure monitoring data (as specified in each GBDL file) by using different *Monitoring Clients*. Infrastructure monitoring data can be used to interpret benchmark results based on the state of the infrastructure during benchmark execution.

### The GridBench Definition Language

The GridBench Definition Language was introduced to the system for several reasons:

1. To allow easy definition of benchmarks, including work-flow benchmarks;
2. To introduce a middleware-independent definition of benchmarks;
3. To serve as a container for associating a definition to the resulting metrics as well as the collected monitoring data.



**Figure 4: Schematic overview of GBDL**

Figure 4 provides a high-level schematic view of the GridBench Definition Language. The benchmark definition includes all necessary information needed to run a benchmark. It includes a set of *parameters*, which specify details for the benchmark execution (such as the path to the executable and benchmark-specific parameters). It also contains a *location* which specifies the resources on which it should run. A *benchmark* can be hierarchical in nature, meaning that it can be made up of other *benchmarks*. This, in conjunction with the use of the execution *constraint* elements, can be used to specify simple workflows. A benchmark *metric* may be in the form of a single value or in the form of a *vector* of values (such as bandwidth at different packet-sizes).

### GridBench Webservices

The *Archiver* webservice allows the storage and retrieval of results generated by executions of the GridBench Suite Benchmarks through the GridBench Framework.

The *Archiver* was introduced in order to serve the following purposes:

1. To manage a potentially large number of results depending on the size of the Grid under study, the number of benchmarks and the frequency of their execution.
2. To provide a central repository for the results allowing access to measurements for users or Grid services.
3. To hold a set of *model* definitions serving as customizable benchmark definitions.

The *Archiver* is an *interface* implemented as a webservice. The *Archiver* interface may have several implementations depending on the back-end in use. There are already implementations for using the *Apache Xindice* native XML database as a back-end and the (newer) *MySQLArchiver* implementation using the MySQL relational database as a back-end.

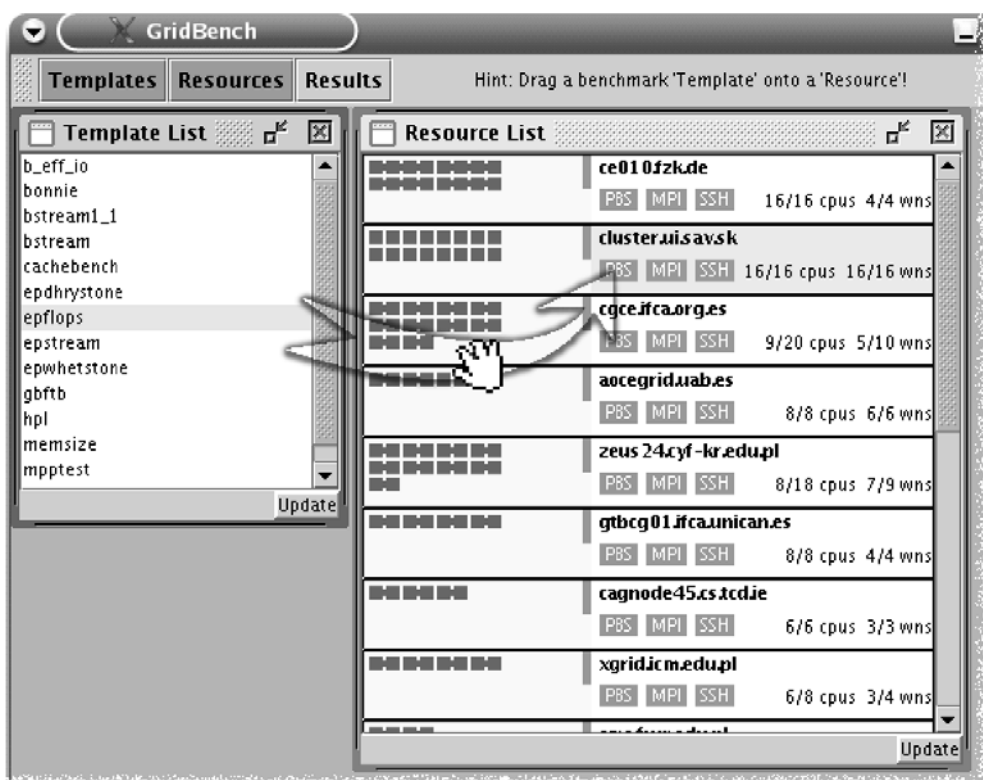
The *Orchestrator* webservice is central to the system as it is responsible for the execution of benchmarks. When a new benchmark description (in the form of GBDL) is delivered to the *Orchestrator* web service for execution the GBDL is translated to the Job Description Language required by the underlying middleware. All specified monitoring data collection is initiated and the job is submitted. When the job finishes, its output (the metrics) are incorporated into the benchmark, as well as all the collected monitoring data. The final GBDL is then archived using the Archiver service.

### The GridBench User Interface: the “Virtual Workbench”

GridBench provides a user-friendly graphical interface for defining and executing benchmarks, as well as browsing results. Additionally it provides tools for result analysis through the easy construction of custom graphs from archives results. Figure 5 shows the main graphical use interface for the definition of benchmarks.

In Figure 5 we can observe the list of available benchmarks (the list on the left window) and the available resources (the list on the right window). The “Resource List” window shows resources retrieved from one or more Grid Information Systems (MDS), with details about each resource's composition such as free/busy CPU's and Worker nodes, dual/single CPU machines etc. Additionally a set of tests can be performed on each resource. In Figure 5 we can see tests such as the “PBS” and the “MPI” tests. These tests will test each resource for correct configuration of PBS and MPI respectively. Tests involving multiple sites (e.g. using MPICH-G2) can also be performed. Such tests are useful for detecting configuration problems as well as connectivity/firewall issues. More tests (e.g. targeting other local queuing systems) can be easily added by implementing simple Java interfaces.

Defining and executing a benchmark is as easy as dragging a benchmark onto one of the resources (shown in Figure 5). The user has the opportunity to tune the benchmark parameters prior to execution via a benchmark configuration panel.



**Figure 5: Screenshot of the GridBench Graphical User Interface. The list of available GridBench benchmark is shown on the left window. The available resources of a virtual organisation and their current status in term of free/busy CPUS are listed on the right panel. Invoking a benchmark is as easy as dragging one item from the benchmark list onto the resource list.**

### Use-case scenarios

We present 2 simple use-case scenarios for GridBench in order to illustrate the functionality visible to the end-user and the overall simplicity in using the tool to get performance metrics for Grid resources. First we describe the scenario where a user would like to get a “picture” of the current status of a set of resources in terms of low-level performance metrics. In the second case, the user has a specific application in mind and would like to select a resource onto which to execute the application. Many other use-case scenarios are possible; in fact some do not even involve an end-user. For example,

metrics obtained through GridBench mechanisms can be used by a scheduler that performs resource ranking on an application basis in a way that is completely transparent to the user. These Use-case scenarios are described in more detail in the “GridBench developers Guide”.

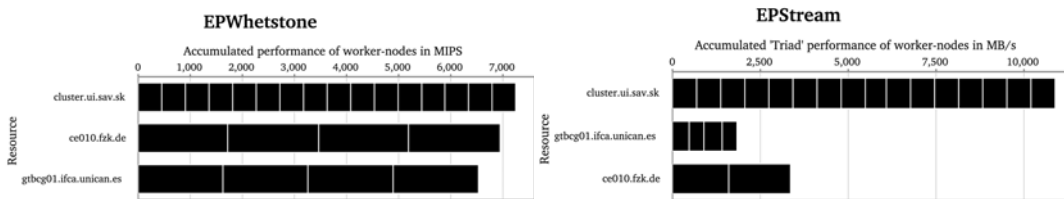
Scenario 1: Comparing resources

As a first use-case, we consider a user who wants to compare a set of resources in terms of 2 “basic” performance factors: CPU FLOP/s and memory bandwidth. The user would like to use “fresh” data so she opts to invoke new benchmark executions instead of fetching historical data. The user can perform the following steps:

1. Determine which metrics will tell you what you want to know about the resources. In this case, the metrics for these factors can be delivered by a set of benchmarks as summarized below:

Factor	Metric	micro-benchmark
CPU	OP/s	EPWhetstone
Memory	bandwidth	EPStream

2. Using the GridBench GUI simply drag each of the benchmarks onto each resource and submit the benchmark (Figure 5). When the benchmark execution finishes, the result will be automatically archived.
3. Using the GridBench GUI put together comparative charts for the resources for each benchmark (Figure 6).



**Figure 6: CPU and Memory micro-benchmark results (first use-case scenario).**

From the results in Figure 6 (the charts were generated using the GridBench GUI) we observe that in terms of aggregate CPU performance they vary only slightly, but in terms of memory bandwidth the performance varies greatly as shown on the right in Figure 6. A memory-intensive application where the main requirement is memory bandwidth would (other things being equal) be better off running at the site with the better memory performance.

Scenario 2: Application Performance

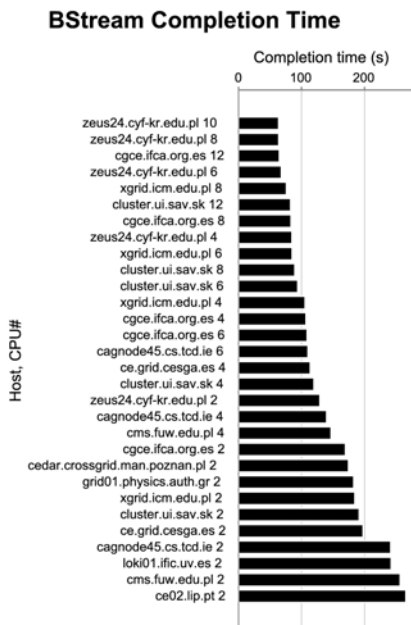
As a second use-case we consider a user that wants to compare resources based on performance of a given application or kernel (In this case the “Bstream” blood flow simulation kernel from WP1.1). The user, in this case a surgeon, needs to find the best resources to run a set of simulations. The user has a given application that is used *frequently*, it is therefore justifiable to perform some trivial instrumentation/timings on the application's computational kernel (e.g. to measure iteration times or simply measure completion time on a given dataset) and make it part of the benchmarks available in GridBench.

One of the primary design goals of the GridBench framework is the easy inclusion of new benchmarks/kernels. In this use-case scenario the user wishes to include a frequently used kernel. The following steps need to be taken:

1. Create a new GBDL description (model) and add it to the Archiver database;

2. Write a simple implementation of the ParameterHandler Java interface (see the developer guide for more information);
3. Instrument the code of the kernel to generate metrics.

Figure 7 shows the results obtained from running the kernel benchmark:



**Figure 7: Results obtained using the kernel benchmark described in the second use-case.**

---

## TASK 2.4 – PERFORMANCE PREDICTION COMPONENT (PPC)

Performance evaluation, instrumentation, prediction and visualization of parallel code is a complex multidimensional problem in parallel and distributed systems. In grid environments these topics are even more critical. In general, tuning the performance of code on distributed memory systems has been a very time-consuming task for users. The reasons for poor performance of parallel message-passing and data parallel code can be varied and complex. The user needs to be able to understand and correct performance problems in order to achieve good results. This is especially relevant when high level libraries and programming languages are used to implement parallel code because the lost of direct control over the program. Performance data collection, analysis, prediction and visualization environments are needed to detect the effects of architectural and system variations.

The Performance Prediction Component (PPC) established a methodology to obtain analytical models for programs behaviour on the Grid. These models are based on exhaustive measurements of execution times obtained in a monitored environment. The general idea is to correlate execution times with monitored information. In particular, the monitoring information that we considered is the computational power of each node as well as the network communication bandwidth and latency. With this information both computational and communicational models can be derived. PPC consists of two main components:

1. A methodology to obtain models to characterize the behaviour of selected kernels.
2. An interactive GUI to show different views of specific aspects about kernels performance when they are executed on a grid.

According to our approach, the models are application-dependent. Therefore, we developed computation and communication models for the applications of the CrossGrid project as well as for a few well known general purpose kernels. Note that only kernels for which computations and/or communications are the main factor in performance were considered. A specific model for each MPI-enabled kernel on the grid was established.

Once the models are obtained, the performance predictions can be produced very fast, since these models are just simple analytical expressions. That is, no simulations or large trace files are required. In addition, the information that is shown in the GUI focuses on aspects that are important for the developers of each application. For example, for the air pollution application, the balance in terms of FLOPs; and for the HEP application, the behaviour of some point-to-point communications.

In the next sections we introduce the two components of PPC.

### 1. The methodology to obtain the models:

To obtain a precise model for the performance of the computational kernels, a large number of executions of them under different scenarios were performed. In particular, the kernels were executed on different sites and on the whole grid.

Initially, the kernels were characterized statically, in terms of the precise number of relevant events. One of these events is the number of floating point operations (FLOPs) required for the target kernel. This number was counted manually or using tools like PAPI when it was necessary. After that, this value is summarized in simple algebraic expressions involving parameters of the kernel, such as the size of some matrices, or the number of iterations of some loop, or the number of non-zero entries in a sparse matrix, or the grade of some polynomial preconditioners, etc. All this parameters are statically established.

The study of the communication patterns generated by the MPICH-G2 routines used in the kernels is essential for predicting their overheads. In the same way as for the number of FLOPs, information about the number and size of the communications performed by each processor can be statically stated. In order to model MPI collective communications, the behaviour of many collective routines in

terms of individual point to point communications were extracted. In this way, with the information about latency and bandwidth, the cost of these routines can be estimated.

For the communications kernels, MPICH-G2 distributes the processors in groups in different levels according to the communication behaviour. For example, level 0 is for WANs, level 1 is for LANs, and so on. Therefore, characterizing collective communications in MPICH-G2 in a set of point to point communications is based on the hierarchy of protocol levels: WAN, LAN, Local, ... In particular:

- The MPI\_Bcast is implemented sequentially in level 0, and as a binary tree in other levels.
- MPI\_Scatter is sequential in level 0, and a binary tree in other levels.
- MPI\_Gather is also sequential in level 0, and a binary tree in other levels.
- The associative MPI\_Reduce is implemented as an MPI\_Bcast but in reverse order, and it includes the arithmetic or logic operation in each processor.
- The no associative MPI\_Reduce is implemented as a MPI\_Gather, and after that, the operation is performed sequentially in the root processor
- The implementation of the MPI\_Barrier is based on a hypercube communication in each level followed by an all-to-all communication in level 0.

Figure 8 shows an example of this hierarchical structure for a broadcast in a system with 12 nodes distributed in different levels.

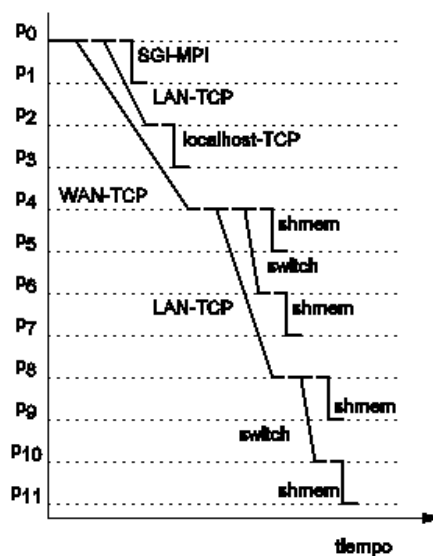
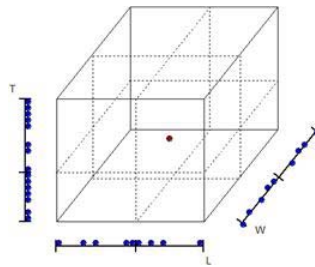


Figure 8: A broadcast on 12 nodes.

As soon as all this static information about computations and communications was modelled, we dealt with the development of the performance models. In this stage, JIMS, a monitoring tool developed in the CrossGrid project, was used. The cost for each kernel is measured through a large number of executions under different network situations. Even though JIMS offers a broad amount of information, just a reduced set of its functionalities is needed for our purposes. In particular:

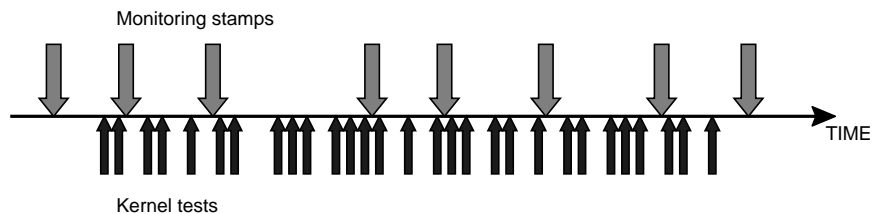
- The online workload per CPU is used to establish the performance models for computations.
- The latency and bandwidth between each pair of processors to define the models for communications.

The main idea of our methodology is to obtain the correlation between monitoring information and features of the kernel with measured execution times. This method is based on the concept of “cube of tests”. Consider, as an example, a point to point communication of a certain size. This kernel is executed  $K$  times in a short period of time, producing  $K$  measures of runtime named  $T_i$ . Consider that  $M$  monitoring tests were performed in this short period of time, producing  $M$  measures of latency and bandwidth named  $L_j$  and  $W_j$  respectively. A cube of tests is defined as the cube in a, in this particular case, three dimensional space  $\{L, W, T\}$  limited by the minimum and maximum values achieved for these three parameters. Figure 9 illustrates a cube of tests. Note that this cube is defined for a certain size of the message. Therefore a fourth dimension has to be taken into account to obtain the model, that is, the size of the message.



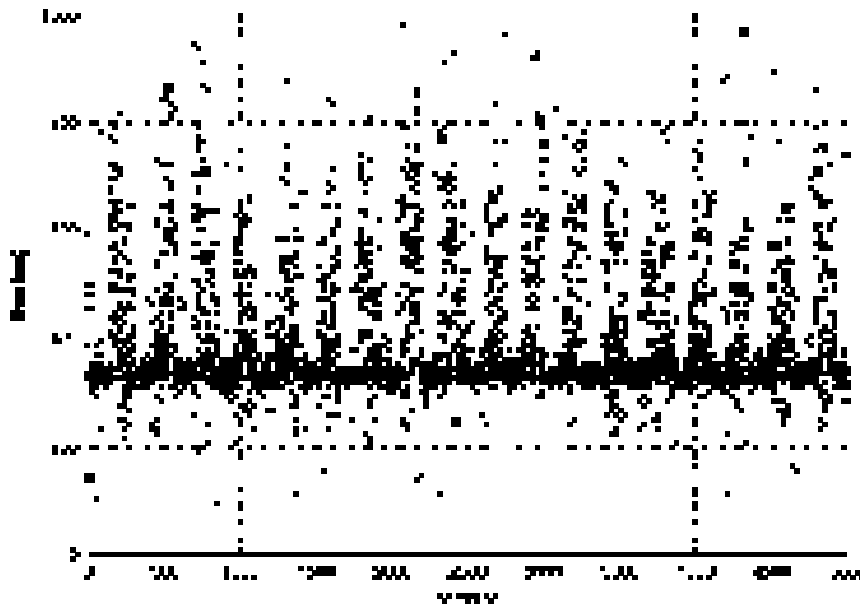
**Figure 9: A cube of tests**

Figure 10 illustrates how the monitored information is extracted when the kernels are executed. In order to minimize the number of executions that are influenced by the monitoring process, their number has to be higher than the number of monitoring tests. According to our experience, about 10 executions of the kernel between two consecutive monitoring stamps are enough.



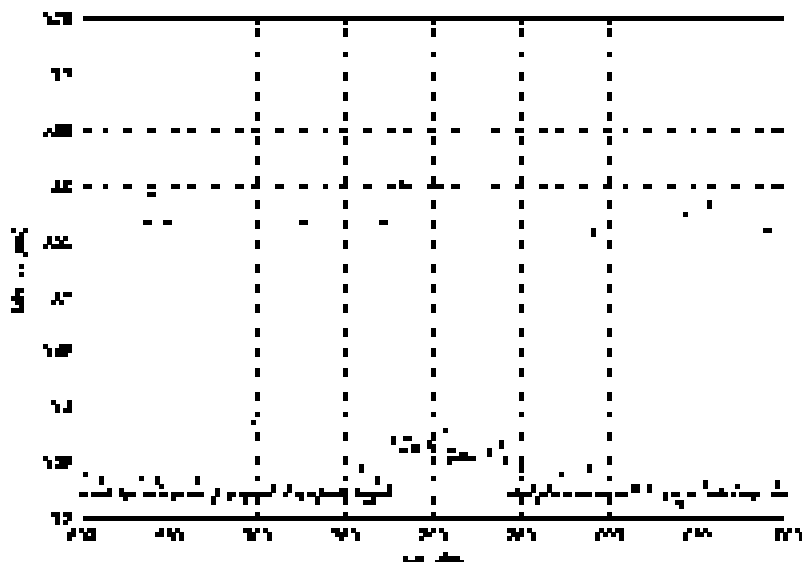
**Figure 10: Samples of the execution of the kernels in a monitored environment.**

The monitoring information obtained in the period of time that defines the measurements must be homogeneous. If this is not the case, we assume that the status of the system is not stable, and it can not produce a consistent cube of tests. In fact, some threshold to guaranty this homogeneity must be established to define the cubes. Figure 11 shows, as an example, the measured execution times for a ping-pong kernel executed 5000 times. The message is 32kB long. Note that most of the measures (in this case 3150) are in a thin interval of execution times, the others are not considered because they are influenced by the monitoring process, or they are considered as spurious measurements.



**Figure 11: 5000 samples of the execution of the ping-pong kernel.**

The monitoring process is also affected by the execution of the kernels. As an example, Figure 12 shows the values of the latency obtained from JIMS when the experiment is being performed. Note that these values change when the kernel is, in fact, executed (this corresponds to the band in the middle of the figure, around sample number 800 in the figure). Therefore the values obtained just before and after the execution of the kernel are considered to define the cube of tests. Note that some spurious values were obtained, they have to be discarded.



**Figure 12: Samples of the monitored latency when the ping-pong kernel is executed.**

The kernels considered in the tool are the following:

- Paraiso is a MPI library of iterative methods for solving large sparse matrix systems. This library includes the implementation of analytical models to characterize a number of communication routines that can also be considered as kernels. These models for the communication routines are not only needed to implement the models of the kernels, but also for other users who are not interested in these specific kernels. The dense matrix vector product was also included.
- From the air pollution application of the CrossGrid project, we focused on the routine that consumes most of the runtime of this application. It is called `vertlq`. It was analyzed, and an analytical model for characterizing its performance was developed. This routine basically consists of an intensive computational part that is executed in parallel, i.e. locally, and just one reduction operation that involves communications. Both parts are uncoupled, so the model adds both contributions. In this application, the performance models were used by the developers to include a new routine to dynamically balance the workload among the nodes involved in the execution. [MOU04]
- From the flooding application of the CrossGrid project, we developed some models for the kernels of this task. For this application the standard `PETSC` library for solving large sparse systems is the main kernel. In particular we focussed in the routine called `slessolve` that implements a Krylov method with the `asm` preconditioner. Both communications and computations were considered.
- Concerning the HEP application of the CrossGrid project, this application was coded using the master-slave paradigm. Their main kernel is a learning process of a neural network that includes two parts: a parallel part that is computational intensive, and two collective communications: a gather and a scatter from the master to the nodes and some point to point communications.

## 2. The graphical User Interface

An interactive graphical user interface (GUI) was developed in this task. It shows three types of information:

- Information based on the analytical models, like predicted execution times, or load balance based on the predicted execution times on each node.
- Information about features of the kernel, like the number and size of some collective communication.
- Information about the status of the grid, like latency between a certain pair of nodes.

Therefore, this tool can include detailed information about specific parameters of the code, as well as the predicted information about the execution of the code. The user can act on this information, changing some parameters, like, for example, the latency between a pair of nodes. In this way, the user can analyze their effect over the overall performance. The interface provides interactivity in the analysis of the behaviour of the code under different conditions (number of processors, distributions, input data, network parameters, ...).

This GUI is designed to be used by applications developers and users interested in analyzing the performance of selected kernels under different scenarios. The results could be used to modify the parameters of the parallel execution of the application, like the number of nodes, the size of the problem, the distribution of data, etc. To allow a user to quickly find his way in the multidimensional design space, the GUI needs to be used interactively. In the long term, it will be also used by resource brokers and schedulers to select the best platform according to the predicted figures offered by the tool.

Some of the main characteristics of this tool are:

- It is an application dependent tool. It is specifically applied to selected kernels for which analytical models are available. But the usage of the tool is not restricted to CrossGrid application developers. For example, it may be used by people interested in studying the performance of communication routines.
- It is an interactive tool in the sense that the user can easily change the parameters that define the system or the code, and then analyze the influence of these changes in the overall performance.
- It uses analytical models, and therefore the predictions are obtained very fast. This feature is very important for the interactivity.
- It is specifically developed for heterogeneous systems.

The GUI is a Java applet. A Java 2 compiler and runtime environment are necessary to build and run the GUI of PPC. The package has been tested with Sun J2SDK1.4.1 and J2SDK1.4.2 (<http://java.sun.com>). This package uses the SGT Graphics Package [SGT].

In summary, the GUI of the PPC tool is devoted to be used by applications developers and users that are interested in analyzing the performance of the selected kernels under different scenarios. The results could be used to modify the parameters of the parallel execution of the application, like the number of nodes, the size of the problem, the distribution of data, etc. To enable a user to quickly find his way in the multidimensional design space, PPC needs to be used interactively. In long term, it can be also used by resource brokers and schedulers to select the best platform according to the predicted figures offered by the tool.

The description of the main features of the GUI of the tool is included in the user's manual [PPCUG].

---

## TASK 2.4 – GRID PERFORMANCE MEASUREMENT TOOL (G-PM)

The orientation of the CrossGrid project towards interactive applications, as well as the nature of the grid itself, poses new requirements and constraints on the performance analysis process. First, the user must be allowed to control the application during its runtime. To facilitate such an application steering process, the necessary performance data must be provided in an on-line fashion. Moreover, the performance analysis infrastructure must be adapted to users that are not specialists. Therefore, the performance information should not be limited to raw, low-level data, but rather be more abstract and application specific.

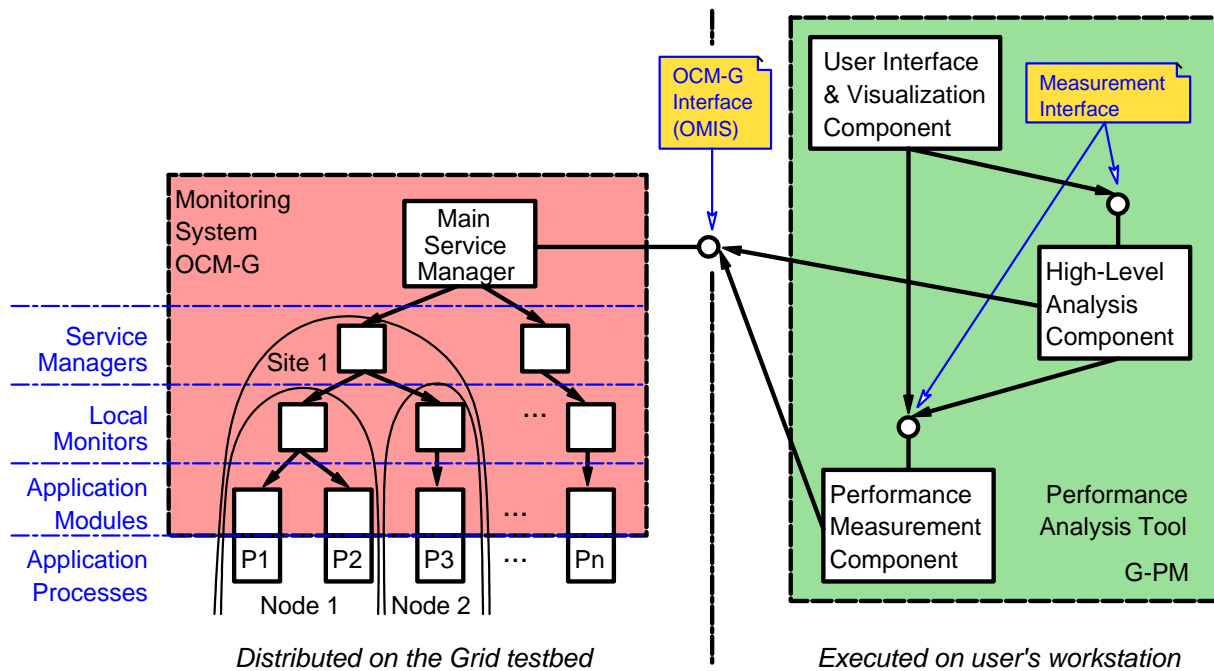
The G-PM tool - as the performance analysis component of the CrossGrid infrastructure - has been designed to satisfy the outlined requirements and constraints. It operates in an on-line manner, displaying current performance data in a form of various visualization graphs. A set of standard performance metrics has been built into the tool. They allow monitoring basic properties of application behaviour, e.g. the volume of communication or I/O operations, synchronization delays, CPU usage, etc. These performance metrics can be measured for the application as a whole, or can be restricted to a subset of the computing nodes, application processes, or specific locations in the application's code.

While providing very useful insight into computing performance, the built-in metrics may omit some information vital in the application context, or may not summarize the information in the most suitable form. Therefore, the application developer is given the opportunity to specify new performance metrics. This is done by combining two basic concepts. First, the developer can mark specific events in the application code. Such events, called probes, can carry application-specific information to the G-PM, which can be used in the computation of the metrics value. Second, a dedicated Performance Metrics Specification Language PMSL is provided by G-PM. With this language, the developer can create a higher-level metrics on top of both standard metrics and previously defined application-specific metrics. The PMSL not only allows combining data from already-existing metrics, but also allows taking into account the occurrences of probe events in the computation of new metrics.

It should be noted that besides probes that provide application specific events, all other instrumentation is done automatically using the OCM-G, which is part of the CrossGrid middleware [OCMGUG]. The application just needs to be linked with appropriate (i.e., pre-instrumented) versions of programming libraries (e.g., the MPI libraries), which are part of the OCM-G installation.

### G-PM Tool Architecture

The G-PM is composed of three main modules: the Performance Measurement Component (PMC), the High-Level Analysis Component (HLAC) and the User Interface and Visualization Component (UIVC). The relationships between this components, as well as links to external CrossGrid services are presented in Figure 13



**Figure 13: The component structure of the G-PM and relationship to other CrossGrid services**

The **PMC** (Performance Measurement Component) is responsible for handling of the built-in performance metrics. It communicates with the OCM-G service of the CrossGrid middleware, and programs it in accordance to the currently conducted performance measurement. This programming is done using the OMIS protocol [LUD04].

The **HLAC** (High Level Analysis Component) component handles the measurement of all performance metrics created by the user. It is responsible for parsing the metrics specifications and translating them into requests to PMC for the underlying built-in metrics. Furthermore, HLAC handles the events generated by probes and processes the application-specific information they carry. While the HLAC reuses the PMC for performing non-probe-based measurements, it communicates directly with the OCM-G to provide efficient implementation of probe-based metrics. Similarly to PMC, this communication is based on the OMIS protocol. To optimise the evaluation of high-level performance metrics, HLAC aims at programming most of the computation into the OCM-G. This approach allows a distributed evaluation, as the OCM-G maintains a local monitor component on every computing node [WIS04].

The **UIVC** (User Interface and Visualization Component) encapsulates all the graphical interface of the tool. Most importantly, it provides a set of visualization windows for performance data, ranging from bar graphs and pie charts to value-versus-time function plots. In addition, the UIVC implements several helper windows. One of them supports the creation of performance measurements by allowing the user to choose a set of metrics and to specify the nodes, processes and code regions in which the measurement should be performed. In a second one, the user can specify the type and parameters of a measurement's visualization diagram. Finally, the UIVC also provides a simple editor for the PMSL metrics specifications.

### Basic Performance Measurements

As said before, the G-PM supports a wide set of built-in performance metrics. Generally, these metrics fall into two categories: function-based metrics and sampled metrics. The function-based metrics are

---

related to instrumented versions of specific functions and are suitable for monitoring the behaviour of applications with respect to various libraries they use. For example, the metrics associated with the instrumented versions of the MPI library enable the assessment of application communication performance and parallelisation-associated overhead. The G-PM provides the following MPI related function metrics:

- Delay, communication volume, and invocation count of the `MPI_Send`, `MPI_BSend`, `MPI_SSend` and `MPI_RSend` functions; both for individual functions and as a total delay in all this functions,
- Delay, communication volume, and invocation count in the `MPI_Recv` function,
- Delay and invocation count of virtually all other communication-related MPI functions.

See the G-PM user's manual [GPMUG] for a complete list.

All of the above metrics can be measured for the whole application or some part of it. In particular, the user can select a set of sites, computing nodes, and/or application processes, as well as code regions of the application and request the G-PM to measure the given metrics (or set of metrics) only within this selection. Of course, it is not necessary to always select single code regions, rather it can end at any level of the presented hierarchy of computing units. Furthermore, in case of metrics associated with point-to-point communication, one can specify not only the origin of the message, but also can narrow the measurement to communication that occurs between a particular origin and destination, or to whole communication directed to a given destination (i.e. regardless of the message origin). Finally it should be noted, that any function of the application can serve as a code region. Using probes and PMSL, metrics can also be measured between two arbitrary points in the application's code.

The second category of metrics, i.e., the sampled metrics, are not related to any functions or events. Rather, they represent quantities whose value can be accessed at any point in time. Examples of such metrics are the CPU time of an application process, or the memory used by it.

### User defined Metrics

The built-in metrics are quite useful to application developers, allowing them to inspect the details of various parts of the application. However, the measured quantities are low-level and do not provide a clear insight into the performance of complex applications executing in the Grid. For example, the developer may want to relate the application's performance to the performance of the Grid environment it is running on. Or, in case of interactive applications or services, he may be interested in the inspection of the performance of individual interactions or service requests. Furthermore, the application may use higher-level libraries, and it should be possible to measure performance data specific to these libraries. Finally, the developer may be interested in performance metrics tailored to the particular application, e.g. in measuring the number of algorithm iterations per second or in inspecting the change of the residuum value of an iterative numerical solver.

For this reasons means of adjusting the performance tool to the needs of the users are needed. In G-PM, this is achieved by supporting user-defined metrics, which allow both to combine data from several existing metrics, as well as combining this data with application or library specific events. User-defined metrics thus support a higher, application specific level of abstraction. Once such a metrics, which is specified using a simple definition language, has been entered through the tool's user interface, it can be used in the same way as its built-in counterparts. In particular, it can be visualized using any of the available visualization windows or it can be used as a basis for other user-defined metrics. The essential difference between the G-PM approach and other related work is the fact that the user-defined metrics are evaluated on-line and in a distributed, scalable way.

A user-defined metrics in the G-PM tool is based on already existing metrics, plus (optionally) some minimal, specific information from the user's application i.e.:

- occurrences of important events in the application's execution,

- associations between related events,
- performance data computed by the application itself (if required).

This information is provided via *probes*, i.e. special function calls, which receive a virtual time and, optionally, additional performance data as parameters. The virtual time is an arbitrary, but monotonically increasing integer value, which is used to specify associations between different events, i.e. probe executions.

Using this concept, it is possible to derive new metrics from existing ones (e.g., by relating two metrics to each other) or to measure a metrics only for specific program phases. For example, in an interactive, parallel application, the programmer may provide two probes which mark the beginning and end of the processing of a certain class of user interactions. Probe calls belonging to the same interaction are identified by receiving the same value for the virtual time. With this preparation, G-PM can be used, e.g., to measure the amount of data transferred to other processes in each interaction.

Since at the time of G-PM development no specification mechanism existed, which would be suitable to express performance metrics similar to the one presented above, a new language had to be designed. The goal was to create a language which is simple to use and does not force the user to think about the implementation of the metrics. Therefore, the resulting language PMSL is a declarative, purely functional language. It only provides single assignment variables and neither includes control flow constructs nor alterable state. To support metrics based on events, PMSL includes a special operator (*AT*), which takes the value of an expression at the time of an event occurrence. The language also provides a range of set operations used for data aggregation.

In the following, the specification of the example metrics introduced above is discussed. The PMSL specification is shown in Figure 14. It should be noted that this example demonstrates most of the language's features. For a full discussion, please refer to the G-PM user's guide [GPMUG].

In general, each metrics has some parameters that define:

1. the object(s) to be measured (`Process[] processes`: a list of processes),
2. restrictions to the measurement, like partner objects or code regions (`Process[] partners`),
3. time specification. This can be a point in real time at which the measurement is done, a measurement interval in real time (`TimeInterval time`), or a point in virtual time (`VirtualTime vt`).

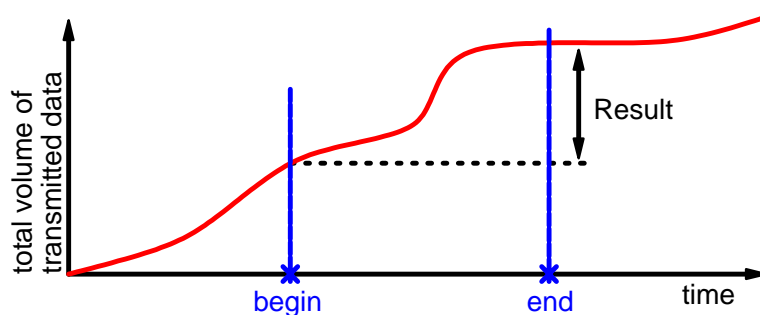
```

1  Comm_volume_for_interaction_vt(Process[] processes,
                                Process[] partners, VirtualTime vt)
2  {
3      PROBE end(Process p, VirtualTime vt);
4      PROBE begin(Process p, VirtualTime vt);
5      Process p;
6      Value[] volume;
7      volume[p] = Send_volume(p, partners, [START, NOW]) AT end()
8      - Send_volume(p, partners, [START, NOW]) AT begin();
9      return SUM(volume[p] WHERE p IN processes);
10 }
11 Comm_volume_for_interaction(Process[] processes,
                              Process[] partners, TimeInterval time)
12 {
13     VirtualTime vt;
14     Value[] volume;
15     volume[vt] = Comm_volume_for_interaction_vt(processes, partners, vt);
16     return SUM(volume[vt] WHERE volume[vt].time IN time);

```

Figure 14: PMSL specification for the example metrics

In Figure 14 two metrics are defined. *Comm\_volume\_for\_interaction\_vt* specifies a metrics for the amount of sent data for a point in virtual time, in our example for a single user interaction. Line 7 tells how to compute the contribution of one process: subtract total communication volume at the *begin* event from the one at the *end* event, as is visualized in Figure 15. The term *[START, NOW]* defines the measurement time interval for the *Send\_volume* metrics (which is a built-in metrics): the lower bound is the start time of the whole measurement, the upper bound is the current time, i.e. the time of the event. Finally, line 8 states that the return value is the sum of the contributions of all measured processes. *Comm\_volume\_for\_interaction* is a metrics that refers to a given interval of (real) time. Its result is the communication volume caused by all user interactions in the specified measurement time interval. Since all measurement values in G-PM are time-stamped, this metrics can be computed by considering the results of *Comm\_volume\_for\_interaction\_vt* for all virtual times and by summing up all results whose time-stamps lie within the measurement interval.



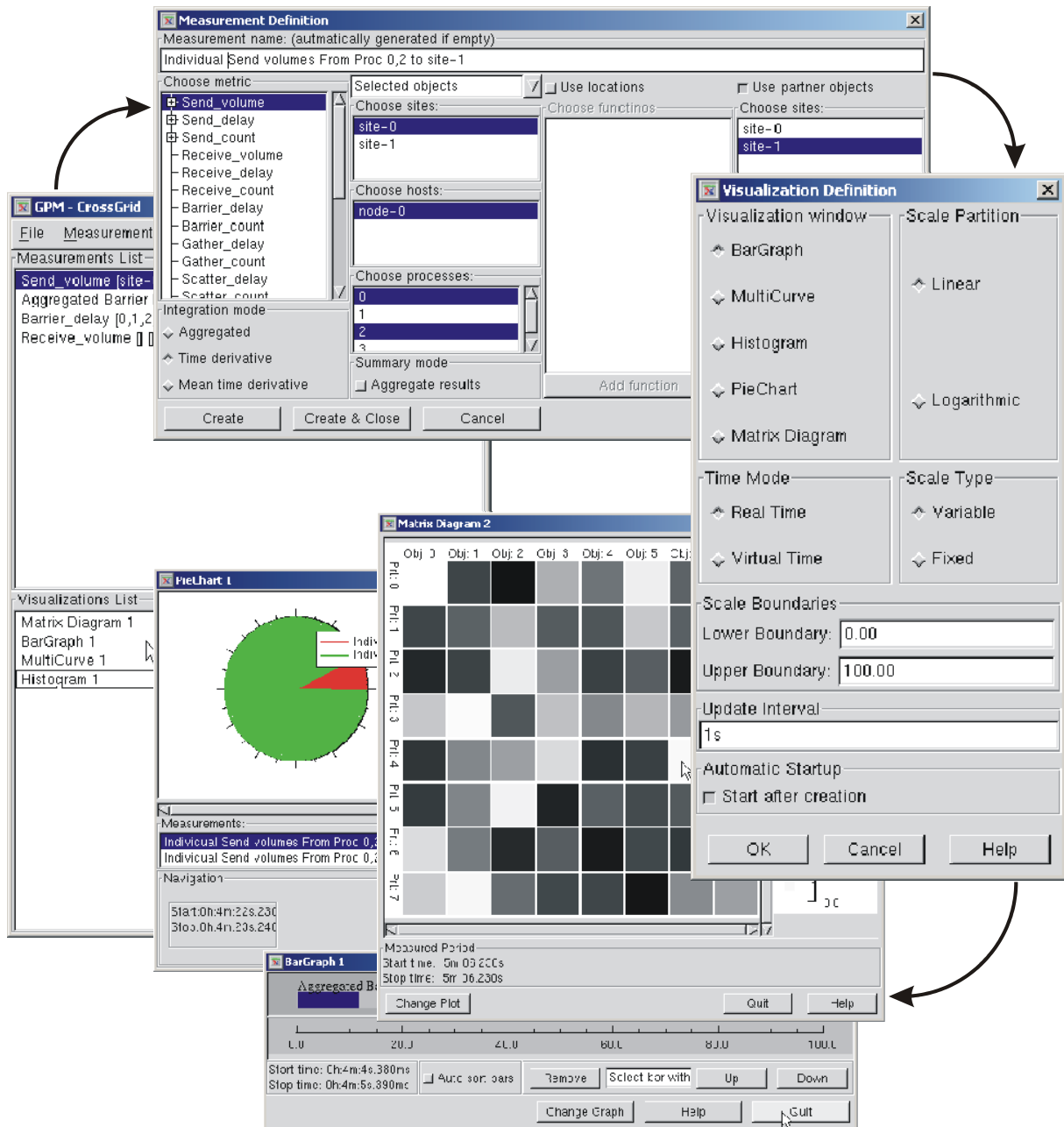
**Figure 15: The principal idea for measuring a metrics between two events in an application**

The example shows that the metrics are specified at a rather high level of abstraction, thus, users of G-PM should be able to define new metrics according to their needs. Nevertheless, the specifications still can be converted to efficient, distributed on-line measurements. It should be noted that G-PM can also read PMSL metrics from files, which allows compiling libraries of useful metrics and also allows accompanying programming libraries with appropriate, library-specific performance metrics.

### User Interface

The user interface of the G-PM tool is designed around the Main Window, which groups, in two lists, all the defined measurements and active visualizations. The Main Window menu contains all the commands responsible for defining and managing measurements, including the user-defined ones, and visualizations.

The typical path throughout the G-PM interface involves, first, defining the measurement in the Measurement Definition window. Then, the user selects the visualisation type in the Visualisation Definition window. Afterwards, the user can start the observation and analysis of the obtained data in the visualisation windows. The whole process of defining the measurement and visualization, from the Main Window, through Measurement and Visualization Definition windows, to a running visualization, is depicted in Figure 16.



**Figure 16: The G-PM User Interface. Performance analysis cycle comprises three steps: (1) main menu with the Main Window, (2) definitions with Measurement Definition Window and Visualization Definition Window, and (3) visualization in relevant display windows**

The Measurement Definition window allows the user to specify the quantities to be measured. It supports the choice of the type of the metrics, as well as the selection of individual objects (e.g. sites or processes) for which the metrics should be evaluated. The tool also allows for restriction of the measurement to events taking place within specific functions. The Measurement Definition window is a necessary initial step in any monitoring or analysis. However, measurements, once defined, can be saved to an external file, and loaded later. Thus, the user does not need to specify the same measurement several times, provided that he is monitoring the same application in the same environment.

The next step in the process leading to observing the performance quantities is the Visualisation Definition window. It enables the user to choose the form in which a measurement defined in the Measurement Definition window will be presented. The user can also specify the parameters of the visualization, including the scale types and boundaries or measurement update interval.

The G-PM offers five diverse visualization types. These include BarGraph, MultiCurve, PieChart, Histogram and Matrix diagrams. These display types are described below in brief.

The BarGraph display shows one or several independent values in a form of horizontal bars of variable length. The length of a bar specifies the value of the measurement while the name of the measurement and the current value are displayed above the corresponding bar. This type of measurement is suitable for comparing the actual values of a large set of measured quantities.

The MultiCurve display presents multiple values simultaneously as scrollable curves. Each curve represents an evolution in time of a single measured quantity. This type of measurement is a general-purpose visualisation method. It is especially useful for comparing the evolution in time of several measured quantities

The PieChart displays values of measured quantities as a percentage of the sum of all measured values. This information is presented in a form of a pie, divided between single measurement values. This type of display is useful for comparing the relation between each of the measured values, as well as their relation to the sum of values for all measurements.

The Histogram display shows a statistical distribution, for a given point in time, of values of individual measurements for specific objects or partner objects. The visualisation has a form of vertical bars of variable length. The length of a bar specifies the number of objects for which the values of the measurements fall into the range specified below that bar.

The Matrix display shows the value of the measurements in a form of a matrix, with objects in columns and partner objects in rows. The intensity of the colour of the field in a given row and column represents the value of the measurement for an object and partner object defined by the given row and column. This display is mainly useful for communication measurements.

### Availability and Integration in the CrossGrid Testbed

The G-PM performance analysis tool is available in the CrossGrid production testbed since April 2004 (Version 0.4.0). The current version 0.8.0 is presently in the test and validation process. All test and validation requests except the very first one have resulted in the recommendation to install the software, with only minor issues found.

The current version of G-PM is available in the WWW:

- As a binary RPM: <https://savannah.fzk.de/distribution/crossgrid/autobuilt/i386-rh7.3-gcc3.2.2/wp2/RPMS/cg-wp2.4-gpm-0.8.0-1.i386.rpm>
- As a source RPM: <https://savannah.fzk.de/distribution/crossgrid/autobuilt/i386-rh7.3-gcc3.2.2/wp2/SRPMS/cg-wp2.4-gpm-0.8.0-1.src.rpm>
- As a source tar-ball: <https://savannah.fzk.de/distribution/crossgrid/autobuilt/i386-rh7.3-gcc3.2.2/wp2/SRPMS/cg-wp2.4-gpm-0.8.0-1.src.rpm>

In addition, the latest development version can be downloaded from the CVS repository:

[http://savannah.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp2/wp2\\_4-perf/wp2\\_4\\_1-perfmon/](http://savannah.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp2/wp2_4-perf/wp2_4_1-perfmon/)

The current version of G-PM is fully integrated with the CrossGrid middleware. It works with all supported kinds of Grid jobs (sequential, Mpich-P4, Mpich-G2, ...) and all submission mechanisms supported in the CrossGrid testbed. In addition, special support for G-PM has been integrated in the Migrating Desktop (MD) via its plugin-mechanism. This allows starting a Grid job together with OCM-G and G-PM via the MD. Since G-PM is a Linux-based X-Windows application, the only restriction is that it can only be started from a Linux host.

Since its availability in the testbed, the G-PM tool has been successfully used to monitor applications from three WP1 tasks. As an example, see [BAL04] for a description of one particular application of G-PM, where a prototype of the blood flow solver from Task 1.1 has been analysed.

---

## TASK 2.5 – INTEGRATION, TESTING AND REFINEMENT

### Testing

The tools developed in WP2 were tested at two different levels. In the first level there are the tests performed by the development teams. These tests are performed on the CrossGrid testbed and local computers, using the applications from WP1, as well as other applications and test programs. Most tests had to be performed manually due to the following characteristics of the tools:

1. GridBench, PPC, and G-PM are interactive tools with graphical user interfaces. For this class of programs it is hard to automatically provide the input and compare the output against some specification, as it would be necessary for automatic regression tests.
2. MARMOT, GridBench, and G-PM monitor the behaviour of parallel, distributed program executions, which are non-deterministic and non-reproducible by nature. Thus, verification of the tool's output is complex and needs human intelligence.

This does, however, not preclude that parts of the tools can be and have been tested automatically. E.g. G-PM provides a simple test driver which allows regression tests of the PMC and most parts of the HLAC component. More details on the testing procedures can be found in the installation guides and developer guides, which are listed in the appendix.

The second level of testing is the test and validation procedure performed by WP4 before deploying the software in the CrossGrid testbed (See [D4.1] for a description of this procedure). In total, 10 test and validation requests for different versions of the WP2 tools have been submitted. Out of these, only three resulted in major issues being found, in five cases, no or only minor issues were found. At the time of writing this document, two requests are still ongoing (for a complete list of the test and validation results, see <http://www.lip.pt/computing/projects/crossgrid/wp4task4/test-results.htm>). Most of the issues found in previous test and validation procedures concerned the lack of documentation. These issues have been solved with the compilation of the current installation, user, and developer manuals.

### Integration

The tools developed by WP2 are now fully integrated into the CrossGrid environment. This integration covers three main aspects:

1. Integration into the autobuild process. The tools are configured to conform with this build process, which regularly compiles all the code and creates RPM packages for installation in the testbed (see <http://gridportal.fzk.de/autobuild/i386-rh7.3-gcc3.2.2-night/BuildResults.html> and <http://gridportal.fzk.de/autobuild/i386-rh7.3-gcc3.2.2-night/BuildResultsHEAD.html>).
2. Integration into the testbed. The tools are available in the CrossGrid development testbed.
3. Integration with the Migrating Desktop (MD). All tools are integrated with the CrossGrid graphical user interface provided by the MD as much as possible and useful:
  - It is possible to submit applications using MARMOT from the MD and watch MARMOT's output at runtime (see Figure 17).
  - The GridBench user interface has been fully integrated into the MD using the plug-in mechanism.
  - The PPC viewer has been fully integrated into the MD using the plug-in mechanism.
  - A plugin in MD's job submission wizard allows easy submitting jobs under control of the OCM-G monitoring system, and to automatically start-up G-PM in a proper way to monitor the submitted job.

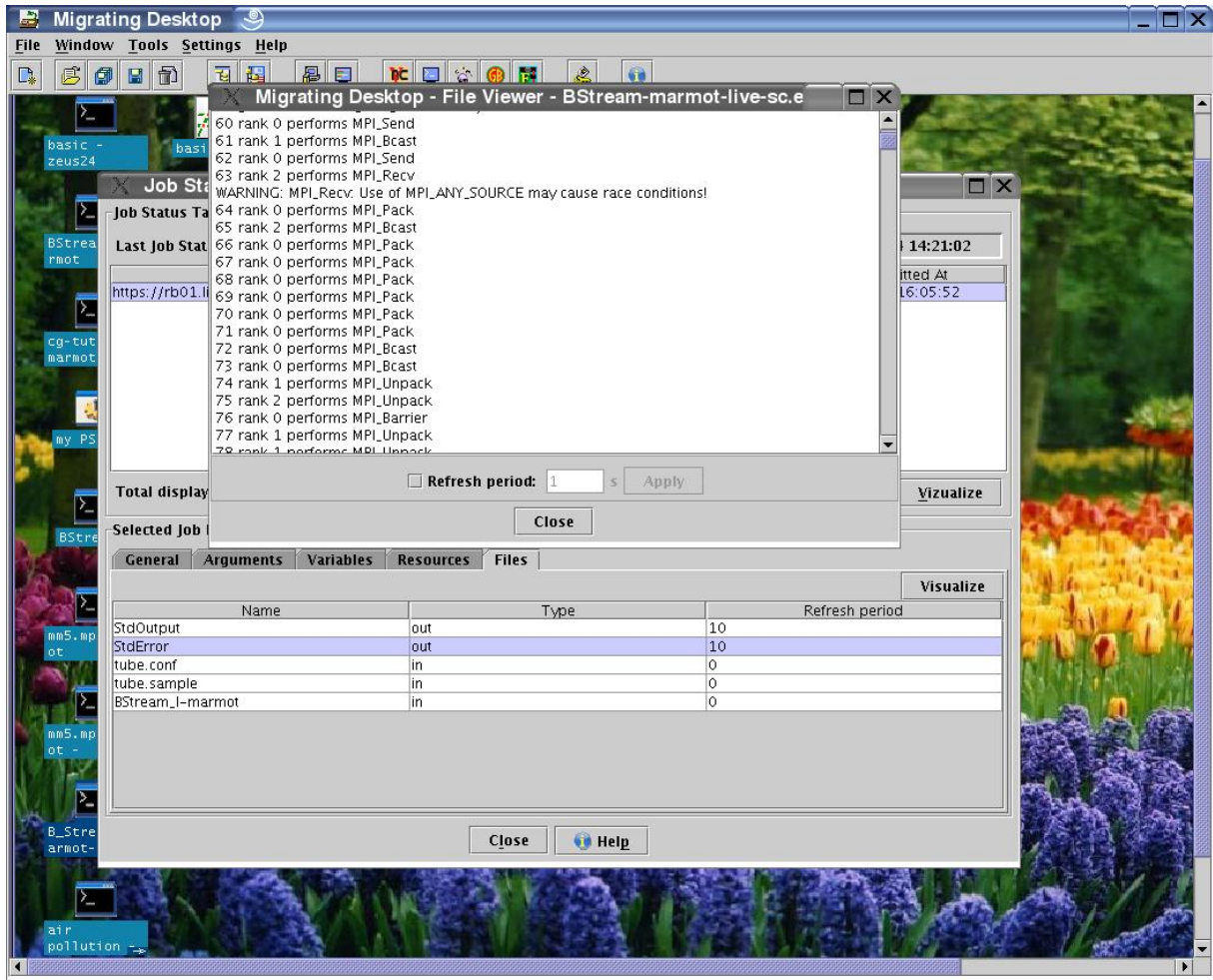


Figure 17: The integration of MARMOT on the CrossGrid Migrating Desktop

## TECHNOLOGY VERSUS APPLICATION MATRIX FOR WORKPACKAGE 2

The CrossGrid workpackage 2 provided Grid development tools for Grid applications. These tools were used by the CrossGrid applications with the support of the members of workpackage 2. The final distribution of workpackage 2 tools is presented in Table 1. The table shows the final integration level.

	WP 1.1 BioMed	WP 1.2 Flooding	WP1.3 HEP	WP1.4 Meteo
<b>MARMOT</b>	Used	Used	Used	Used
<b>GridBench</b>	Kernel Integrated	--	--	Vertlq kernel integrated
<b>PPC</b>	--	Slessolve kernel	kernel under study	Vertlq integrated
<b>G-PM</b>	Used	Used	Used	--

**Table 1: Integration of CrossGrid applications in Grid development tools**

Not all tools are used by all CrossGrid applications. This can be explained by dependencies from other CrossGrid tasks and by the internal details of the applications.

- GridBench and PPC were dependent on monitoring information from task 3.3 (JIMS). Their development was delayed at the beginning of the project. In the last year GridBench and PPC worked together with the JIMS developers to integrate the JIMS monitoring in both components. After this integration the PPC and GridBench team decided to concentrate on a few application kernels. Therefore not all components were integrated.
- Since the Lisbon meeting the whole CrossGrid consortium concentrated on the quality of the tools. Therefore some manpower was shifted to provide tested and validated packages with an efficient documentation including installation, user and developer guides.
- The first prototype of GridBench did not fulfil the requirement of a user friendly GUI. Therefore a redesign was necessary after the review in Dagstuhl (June2003). The improvements were substantial and the GridBench tool is now in a good shape.
- Not all applications were suitable to develop computation and communication models with the PPC tool. Therefore this task concentrated on two application kernel. As the application kernel from workpackage 1.1 consists mainly of Input/Output operations, PPC decided to concentrate on the other kernels.

But all tools are non application dependent and can be used by other applications easily. The developers take care on the extensibility of the tools.

All Grid development tools of workpackage 2 are running on the CrossGrid testbed. All tools were tested and validated with the standard CrossGrid procedure.

The CrossGrid workpackage 2 uses also tools developed in the CrossGrid workpackage 3 and the testbed of workpackage 4. Table 2 shows the usage of them by the tasks from WP2. One component is already based on webservices.

	MARMOT	GridBench	PPC	G-PM
<b>CrossGrid testbed</b>	integrated	integrated	integrated	integrated
<b>Migrating desktop</b>	integrated	integrated	integrated	integrated

---

<b>JIMS</b>		used	used	
<b>OCM-G</b>				Used
<b>Webservices</b>		implemented		

**Table 2: The integration of CrossGrid middleware developments in the Grid development tools of workpackage 2**

### 3. APPENDIX

Detailed information about the usage and installation of the CrossGrid development tools can be found in the user and installation guide for each tool. The developer guide describes the internal architecture of the tool and is foreseen to be used by developers and system programmers. In this chapter the links to the user, installation and developer guides for each tool are presented. The guides were not included in this document to ease the exploitation of the CrossGrid development tools. But the listed guides are a substantial part of the final deliverable of workpackage 2 “Grid development tools”.

#### MARMOT

The download area ([http://savannah.fzk.de/distribution/crossgrid/crossgrid/wp2/wp2\\_2-verify/](http://savannah.fzk.de/distribution/crossgrid/crossgrid/wp2/wp2_2-verify/)) contains all information about the MARMOT tool.

The manuals can be found at the following URLs too:

MARMOT user guide	<a href="http://www.eu-crossgrid.org/user_manuals.htm">http://www.eu-crossgrid.org/user_manuals.htm</a>
MARMOT installation guide	<a href="http://www.eu-crossgrid.org/installation_guides.htm">http://www.eu-crossgrid.org/installation_guides.htm</a>
MARMOT developer guide	<a href="http://www.eu-crossgrid.org/developer_manuals.htm">http://www.eu-crossgrid.org/developer_manuals.htm</a>
MARMOT tutorial (password protected)	<a href="http://www.eu-crossgrid.org/wp5-1-login/CGTutorial.htm">http://www.eu-crossgrid.org/wp5-1-login/CGTutorial.htm</a> (password protected)

#### GRIDBENCH

GRIDBENCH user guide	<a href="http://www.eu-crossgrid.org/user_manuals.htm">http://www.eu-crossgrid.org/user_manuals.htm</a>
GRIDBENCH installation guide	<a href="http://www.eu-crossgrid.org/installation_guides.htm">http://www.eu-crossgrid.org/installation_guides.htm</a>
GRIDBENCH developer guide	<a href="http://www.eu-crossgrid.org/developer_manuals.htm">http://www.eu-crossgrid.org/developer_manuals.htm</a>
GRIDBENCH tutorial (password protected)	<a href="http://www.eu-crossgrid.org/wp5-1-login/CGTutorial.htm">http://www.eu-crossgrid.org/wp5-1-login/CGTutorial.htm</a>

#### PERFORMANCE PREDICTION COMPONENT

PPC user guide	<a href="http://www.eu-crossgrid.org/user_manuals.htm">http://www.eu-crossgrid.org/user_manuals.htm</a>
PPC installation guide	<a href="http://www.eu-crossgrid.org/installation_guides.htm">http://www.eu-crossgrid.org/installation_guides.htm</a>
PPC developer guide	<a href="http://www.eu-crossgrid.org/developer_manuals.htm">http://www.eu-crossgrid.org/developer_manuals.htm</a>
PPC tutorial (password protected)	<a href="http://www.eu-crossgrid.org/wp5-1-login/CGTutorial.htm">http://www.eu-crossgrid.org/wp5-1-login/CGTutorial.htm</a> (password protected)

#### G-PM

G-PM user guide	<a href="http://www.eu-crossgrid.org/user_manuals.htm">http://www.eu-crossgrid.org/user_manuals.htm</a>
G-PM installation guide	<a href="http://www.eu-crossgrid.org/installation_guides.htm">http://www.eu-crossgrid.org/installation_guides.htm</a>
G-PM developer guide	<a href="http://www.eu-crossgrid.org/developer_manuals.htm">http://www.eu-crossgrid.org/developer_manuals.htm</a>
G-PM tutorial (password protected)	<a href="http://www.eu-crossgrid.org/wp5-1-login/CGTutorial.htm">http://www.eu-crossgrid.org/wp5-1-login/CGTutorial.htm</a> (password protected)

---

## 4. REFERENCES

- [BAL04] Balis, B., Bubak, M., Funika, W., Wismüller, R., Radecki, M., Szepieniec, T., Arodz, T., Kurdziel, M. *Grid environment for on-line application monitoring and performance analysis*. In: Scientific Programming, 10 (2004), pp. 1-13, IOS Press.
- [CrossGrid] [http://www.eu-crossgrid.org/CrossGridAnnex1\\_v31.pdf](http://www.eu-crossgrid.org/CrossGridAnnex1_v31.pdf)
- [DDT] The Distributed Debugging Tool  
[http://www.streamline-computing.com/softwaredivision\\_1.shtml](http://www.streamline-computing.com/softwaredivision_1.shtml)
- [EGEE] <http://www.eu-egee.org>
- [GPMUG] CrossGrid User Manual Guide: G-PM, Dec. 2004,  
[http://www.eu-crossgrid.org/user\\_manuals/usermanual\\_G-PM.zip](http://www.eu-crossgrid.org/user_manuals/usermanual_G-PM.zip)
- [GTMD03] George Tsouloupas and Marios D. Dikaiakos.  
Gridbench: A tool for benchmarking grids.  
In *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, pages 60-67, Phoenix, AZ, November 2003. IEEE.
- [GTMDTR04] George Tsouloupas and Marios D. Dikaiakos.  
Characterization of computational grid resources using low-level benchmarks.  
Technical Report TR-2004-5, Dept. of Computer Science, University of Cyprus, 2004.
- [HPL] <http://www.netlib.org/benchmarks/hpl/documentation.html>
- [LUD04] T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode. *OMIS - On-line Monitoring Interface Specification (Version 2.0)*, volume 9 of LRR-TUM Research Report Series. Shaker-Verlag, Aachen, Germany, 1997. ISBN 3-8265-3035-
- [MARMOT] <http://www.hlr.de/organization/tsc/projects/marmot/>
- [MARMOT-1] Bettina Krammer, Matthias S. Müller, Michael M. Resch. "MPI Application Development Using the Analysis Tool MARMOT". In Technical Session on Tools for Program Development and Analysis in Computational Science at ICCS 2004, Krakow, Poland, June 7-9, 2004. Published in Lecture Notes in Computer Science Vol. 3038, pp. 464 - 471, Springer, 2004.
- [MARMOT-2] Bettina Krammer, Katrin Bidmon, Matthias S. Müller, Michael M. Resch. "MARMOT: An MPI Analysis and Checking Tool", in Proceedings of Parallel Computing 2003, PARALLEL COMPUTING: Software Technology, Algorithms, Architectures & Applications, Ed. Joubert, Nagel, Peters, Walter, pp. 493-500, Elsevier, 2004.
- [MARMOT-3] Bettina Krammer, Matthias S. Müller, Michael M. Resch. "MPI I/O Analysis and Error Detection with MARMOT". Proceedings of EuroPVM/MPI 2004, Budapest, Hungary, September 19-22, 2004. Published in Lecture Notes in Computer Science Vol. 3241, pp. 242 - 250, Springer, 2004.
- [MARMOT-4] Bettina Krammer, Matthias S. Müller. "MARMOT - an MPI Analysis and Checking Tool". inSiDE Vol.2 No. 2, Autumn 2004.

- 
- [MARMOT-5] Rainer Keller, Edgar Gabriel, Bettina Krammer, Matthias S. Müller, Michael M. Resch, Towards efficient execution of MPI applications on the Grid: Porting and Optimization issues. *Journal of Grid Computing*, Vol. 1 (2), pp. 133-149, 2003.
- [MDS97] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symp. On High-Performance Distributed Computing*, pages 365-375. IEEE Computer Society, 1997.
- [MOU04] J.C. Mourino, M. Martin, P. Gonzalez, J.C. Cabaleiro. T.F. Pena, F.F. Rivera, R. Doallo, "A grid-enabled air pollution simulation", *Lecture notes in computer science*, Vol 2970, pp 155-162, 2004
- [MPI] <http://www-unix.mcs.anl.gov/mpi/>
- [MPICHECK] <http://www.hpc.iastate.edu/MPI-CHECK.htm>  
Glenn Luecke, Hua Chen, James Coyle, Jim Hoekstra, Marina Kraeva, Yan Zou. MPI-CHECK: a Tool for Checking Fortran 90 MPI Programs. *Concurrency and Computation: Practice and Experience*. 2003, vol. 15, pp 93-100.
- [MPIGDB]: <http://www-unix.mcs.anl.gov/mpi/mpich/docs/userguide/node26.htm#Node29>
- [NASPB-1] NAS Parallel Benchmarks. <http://www.nas.nasa.gov/Software/NPB/>
- [NASPB-2] R.F.van der Wijngaart. NAS Parallel Benchmarks Version 2.4.NAS Technical Report NAS-02-007. NASA Ames Research Center, MoffettField, CA, 2002.
- [OCMGUG] CrossGrid User Manual Guide: OCM-G, Nov. 2004,  
[http://www.eu-crossgrid.org/user\\_manuals/usermanual\\_OCM-G.pdf](http://www.eu-crossgrid.org/user_manuals/usermanual_OCM-G.pdf)
- [P2d2] <http://www.nas.nasa.gov/Groups/Tools/Projects/P2D2/>  
D. Cheng and R. Hood. A Portable Debugger for Parallel and Distributed Programs. In *Proc. Supercomputing'94*, pp 723-732, Washington D.C., Nov. 1994.
- [PARAISO] <http://www-gpaa.dec.usc.es/Paraiso/paraiso.html>
- [PPCUG] CrossGrid User Guide:Performance Prediction Component, Dec. 2004,  
[http://www.eu-crossgrid.org/user\\_manuals/CG-2.3.2-PPC-UserManual-v1.0-USC.pdf](http://www.eu-crossgrid.org/user_manuals/CG-2.3.2-PPC-UserManual-v1.0-USC.pdf)
- [SGT] The SGT Graphics Package, <http://www.epic.noaa.gov/java/sgt/index.html>
- [TV] Totalview: <http://www.etnus.com/Products/TotalView/>
- [UMPIRE] Jeffrey S. Vetter and Bronis R. de Supinski. Dynamic Software Testing of MPI Applications with Umpire. In *SC2000: High Performance Networking and Computing Conf. ACM/IEEE*, 2000.
- [WIS04] R. Wismüller, M. Bubak, W. Funika, T. Arodz, and M. Kurdziel. Support for User-Defined Metrics in the On-line Performance Analysis Tool G-PM. In Marios D. Dikaiakos, editor, *Grid Computing -- Second European AcrossGrids Conference AxGrids2004*, volume 3165 of *Lecture Notes in Computer Science*, pages 159-168, Nicosia, Cyprus, January 2004. Springer-Verlag.
- [D2.1, part x] **CrossGrid Deliverable D2.1** <http://www.eu-crossgrid.org/M3deliverables.htm>  
**part I:** General Requirements and detailed Planning for Programming Environment, CrossGrid Document CG-2-D2.1-0005-SRS.pdf  
**part II:** Software Requirements Specification for MPI Code Debugging and Verification, CrossGrid Document CG-2.2-DOC-USTUTT005-SRS.pdf

**part III:** Software Requirements Specification for GridBench (The CrossGrid Benchmark Suite), CrossGrid Document CG-2.3-D2.1-v1.1-UCY001-SRS.pdf

**part IV:** Software Requirements Specification for Grid-enabled Performance Measurement and Performance Prediction, CrossGrid Document CG-2.4-DOC-CYFONET001-SRS.pdf

[D2.2, part x] **CrossGrid Deliverable D2.2** <http://www.eu-crossgrid.org/M6deliverables.htm>

**part I:** Design Documents and Interfaces of the CrossGrid Application Programming Environment, CrossGrid Document CG2.0-D2.2-v2.1-FZK001-DesignDocumentsSummary

**part II:** Design Document for MPI Code Debugging and Verification, CrossGrid Document CG2.2-D2.2-v1.1-UST001-VerifDesign

**part III:** Design Document for GridBench, CrossGrid Document CG2.3-D2.2-v1.0-UCY001-GridBenchDesign

**part IV:** Design Document for Interactive and Semiautomatic Performance Evaluation Tools, CrossGrid Document CG-2.4.1-D2.2-v1.0-CYF003-PerfToolsDesign

**part V:** Analysis of WP1 Application Kernels for the Performance Prediction Component, CrossGrid Document CG-2.4.2-D2.2-v1.3-USC001-ApplicationKernelAnalysis

[D2.3, part x] **CrossGrid Deliverable D2.3** <http://www.eu-crossgrid.org/M12deliverables.htm>

**part I:** Prototype Documentation of the CrossGrid Application Programming Environment, CrossGrid Document CG2.0-D2.3-v1.0-FZK006-PrototypeDoc

**part II:** Prototype Documentation for MPI Code Debugging and Verification, CrossGrid Document CG2.2-D2.3-v1.0-UST004-PrototypeDoc

**part III:** Prototype Documentation for GridBench, CrossGrid Document CG2.3-D2.3-v1.0-UCY006-PrototypeDoc

**part IV:** Prototype Documentation for Interactive and Semiautomatic Performance Evaluation Tools, CrossGrid Document CG-2.4.1-D2.3-v0.2-CYF001-PrototypeDoc

**part V:** Prototype Documentation for the Performance Prediction Component, CrossGrid Document CG-2.4.2-D2.3-v1.1-USC002-PrototypeDoc

[D2.4, part x] **CrossGrid Deliverable D2.4** <http://www.eu-crossgrid.org/M15deliverables.htm>

**part I:** Internal Progress Report on Software Evaluation and Testing, CrossGrid Document CG2.0-D2.4-v1.1-FZK015-ReportDoc

**part II:** Internal Progress Report on Software Evaluation and Testing of MPI Code Debugging and Verification Tool, CrossGrid Document CG2.2-D2.4-v1.6-UST011-ReportDoc

**part III:** Internal Progress Report on Software Evaluation and Testing for Benchmarks and Metrics, CrossGrid Document CG2.3-D2.4-v1.2-UCY010-ReportDoc

**part IV:** Internal Progress Report on Software Evaluation and Testing for Interactive and Semiautomatic Performance Evaluation Tools, CrossGrid Document CG-2.4.1-D2.4-v1.2-CYF001-ReportDoc

**part V:** Internal Progress Report on Software Evaluation and Testing for the Performance Prediction Component, CrossGrid Document CG-2.4.2-D2.4-v1.4-USC007-ReportDoc

**part VI:** Internal Progress Report on Software Evaluation and Testing: Detailed Integration Plan, CrossGrid Document CG-2.5D2.4v1.1TUM003-ReportDoc

[D3.5] **CrossGrid Deliverable D3.5** <http://www.eu-crossgrid.org/M24deliverables.htm>

Report on the Results of the WP3 2nd and 3rd Prototypes.

[D4.1] **CrossGrid Deliverable D4.1, Appendix D**

<http://www.eu-crossgrid.org/Deliverables/M3pdf/CG-4-D4.1-004-TEST.pdf>

Middleware Test Procedure.