Application Engineering Tools for MPI and PUL

Kesavan Shanmugam and Konstantinos Tourlas

EPCC–SS94–01

Application Engineering Tools for MPI and PUL

Kesavan Shanmugam and Konstantinos Tourlas

EPCC–SS94–01

September 1994

Abstract

This report describes the adaptation of VISPAD, a visualisation tool for performance analysis and debugging, from the CHIMP message passing system to the recently established MPI standard.

VISPAD is a post-mortem visualisation tool based on the concept of program execution phases. It consists of a number of displays, each of which presents a different aspect of the parallel program under consideration. Execution related information is collected at run-time in trace files by using calls to an instrumentation library. The processing of the trace files by VISPAD results in a graphical playback of all the recorded run-time events.

The process of adapting VISPAD to MPI included a restructuring of the instrumentation library, the implementation of an instrumented version of the MPI interface, changing the format of the trace files, the adaptation of existing displays, and the introduction of two new displays. The latter serve the purpose of visualising the rich set of communication operations supported by MPI.

Contents

1 Introduction

VISPAD stands for "Visualisation Tool for Performance Analysis and Debugging". It is a post-mortem visualisation tool that allows the programmer to see the behaviour of a parallel program in many different ways. It allows the programmer to visualise the execution behaviour of a program in terms of *phases* that can be categorised as belonging to several *classes*. In addition VISPAD animates the communication events that occur during the execution of the program. VISPAD also provides several other useful displays to meaningfully represent the behaviour of a parallel program.

VISPAD was originally developed for message passing programs that use the CHIMP message passing library. Because of the recent change in EPCC's policy, to move from CHIMP to MPI, VISPAD is being adapted to visualise the behaviour of MPI programs. This report gives the background information about VISPAD and the work accomplished this summer to adapt VISPAD to MPI based programs.

The following section explains the general structure of a post-mortem visualisation tool. Section 3 gives a brief description of the first prototype implementation of VISPAD called VISPAD-CHIMP. Then, section 4 explains the issues involved in porting VISPAD from CHIMP to MPI. The remaining sections explain in detail the design and implementation of VISPAD-MPI, the version of VISPAD for MPI programs. The report concludes with sections 9 and 10 that mention various ways to improve the functionality of VISPAD. The reader is assumed to have a reasonable familiarity with CHIMP and MPI. More information about CHIMP and MPI can be found in [Bruce *et al.*, 1993; Bruce *et al.*, 1994; Mes, 1994].

2 Post-mortem Visualisation Tools

Debugging and tuning parallel programs is, in general, a challenging task. In recent years, several tools have been developed to help the programmer in this task. Among them the most successful have been the visualisation tools that allow the programmer to visualise the behaviour of a parallel program.

There are mainly two types of visualisation tools: execution-time visualisation tools that allow the user to see what is happening in the program while the program is running and post-mortem visualisation tools that display what happened in the program after the program finishes running.

Almost all the post-mortem visualisation tools conform to the structure shown in figure 1. There are three main stages in post-mortem visualisation:

Instrumentation The program is first modified so that when it is run information about its behaviour can be collected. Often this is achieved by providing an *instrumented* library to the user. The user links his program with this library instead of the regular library. The instrumented library then generates *events* that contain important information about the program's behaviour during run-time. These events are collected in files called *trace files*.

Trace Analysis During this step, the events from the trace files are analysed to produce some agreed-upon data structures that contain information about the program's execution.

Visualisation This part of the tool uses the data structures produced by the trace analysis to display information about the program's behaviour in a meaningful way to the user.

There are several advantages to using a post-mortem visualisation tool. First, it incurs less overhead on the execution time of the parallel program compared to the execution-time visualisation. Second, for large parallel programs that run for several hours, post-mortem visualisation may be the only option. In addition, post-mortem visualisation tools make it possible to replay the program's execution several times to improve understanding of the program's behaviour.

VISPAD is one such post-mortem visualisation tool developed at EPCC. The following section describes the first prototype implementation of VISPAD.

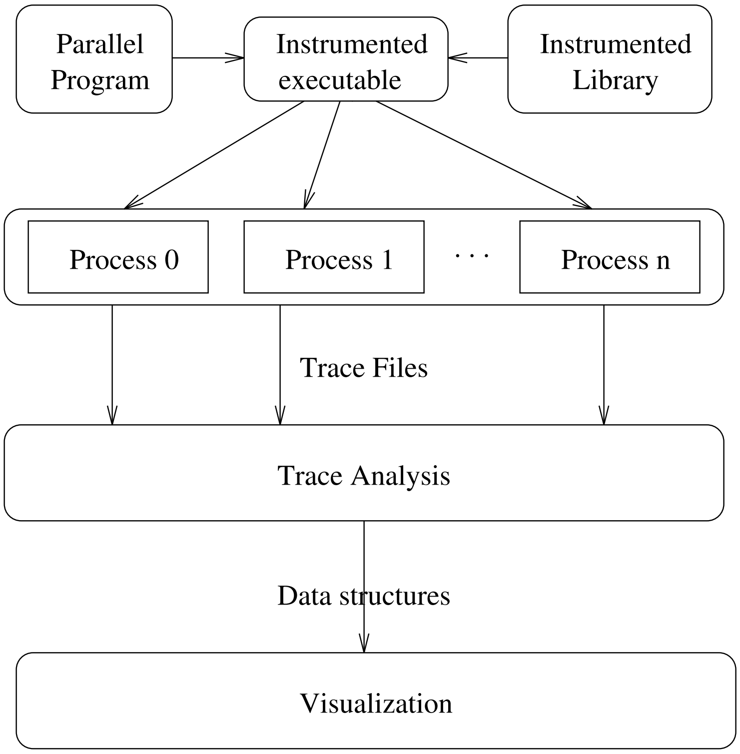


Figure 1: Stages in Post-Mortem Visualisation

3 VISPAD-CHIMP

VISPAD was originally developed for use with message passing programs that use the CHIMP message-passing library. The structure of VISPAD-CHIMP, shown in figure 2, is very similar to the general structure of a post-mortem visualisation tool described in the previous section.

The following sections briefly explain the components of VISPAD-CHIMP. For a more complete description of VISPAD-CHIMP, the user is referred to [Tomov and Wierenga, 1993].

3.1 The Instrumentation Library (INS)

The instrumentation library provides functions that can capture *events* that happen during the execution of a program. Events usually have a time-stamp and some data associated with them and are usually collected in binary *trace files*. The instrumentation library used by VISPAD-CHIMP, called INS, provides the following functionality:

- INS allows parts of an execution to be recorded as *phases*. Phases are just well defined portions of a program that are interesting to the programmer. For example, the entry and exit of a function call can make up a phase. Each phase has a beginning and an ending and it may be properly nested within another phase. Each phase also has a name and may have several *instances* during run-time.
- INS allows the phases to be categorised using *classes*. All the phases that happen between the beginning and ending of a class belong to that class. Classes have names and type numbers associated with them. INS provides three classes for use by VISPAD-CHIMP: APPLICATION, CHIMP and LIBRARY.
- Any communication done by a parallel program can be recorded using the *communication events* provided by INS. These events also have a beginning and an ending, but they need not have the nesting structure of phases.

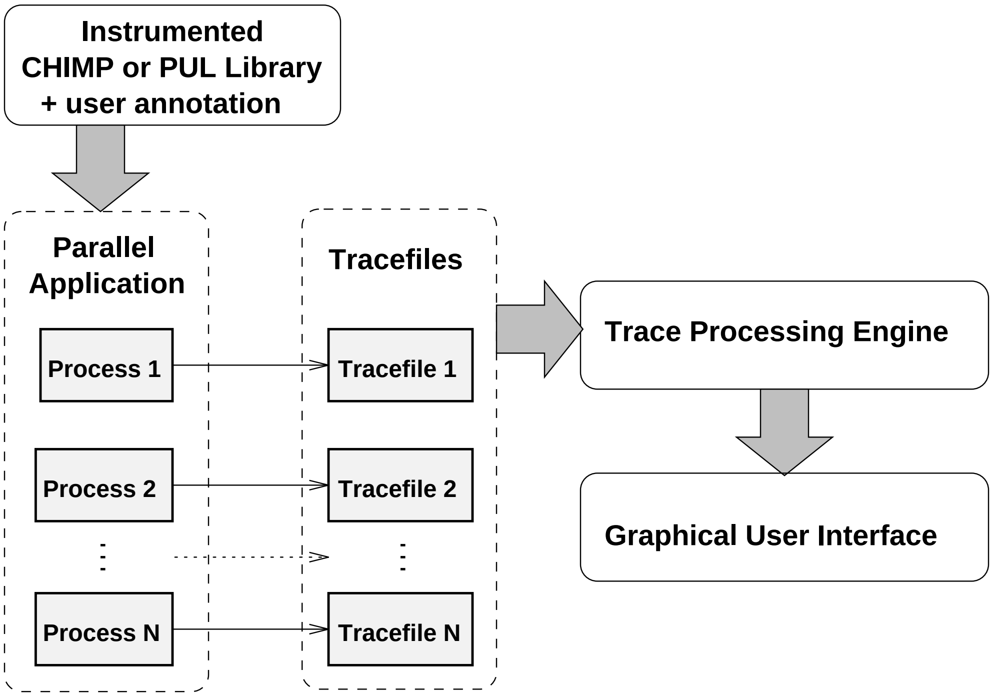


Figure 2: VISPAD-CHIMP

- INS provides facilities to classify communication events as *global* or *local* by the use of *contexts*. All the communication events that happen between the beginning and ending of a *global context* are classified as global events. Similarly for local events.

An instrumented version of the CHIMP library uses the above functionality of INS to output run-time information about CHIMP function calls. If a CHIMP communication function *f* is called by a program, the instrumented CHIMP library outputs a phase with the name *f* and outputs all the communication events that occur within that function. Similarly an instrumented version of the PUL library is also provided. Any programmer who wants to make use of VISPAD-CHIMP, then links his program against these instrumented CHIMP and PUL libraries instead of the regular ones.

In addition, the programmer can directly use INS to output information about the phases and other important events in his program. When an instrumented CHIMP program is run, every process in the parallel program produces a trace file containing all the phases and the events that occurred in that process.

3.2 The Trace Processing Engine (TPE)

The main job of the trace processing engine is to read in all the trace files produced by an instrumented CHIMP program and create the data structures that are used by the Graphical User Interface (GUI). The GUI itself is explained in the next section. The trace processing engine of VISPAD-CHIMP accomplishes the following:

- Provides facilities to adjust the time stamps on all the events in a trace file. This is used by the GUI to do a post-mortem *alignment* of processes' clocks [Tomov and Wierenga, 1993].
- Produces a *Phase Tree* for each process. It is a data structure that contains information about all the phases that happened in that process. Since phases can be nested without any overlapping, a tree is used to represent all the phases.
- Produces a list of communication events, sorted in time order, for each process. This is used by the GUI to do an animation of the communications performed by the program.

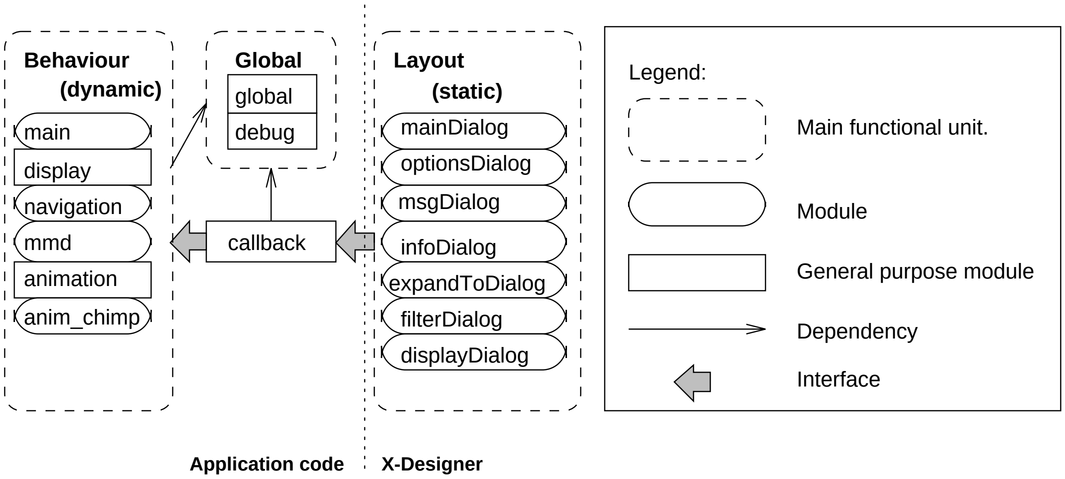


Figure 3: Dependencies between the modules in the GUI

- The instrumented CHIMP library outputs a *join* event to the trace file whenever a process joins a SAP group [Bruce *et al.*, 1993]. The TPE produces a list of join events, sorted in time order, for each process. This list is used by the GUI to create the Membership Matrix Display, explained later.
- For each data structure described above, the TPE provides functions to access and navigate the data structure conveniently.

After the TPE produces these data structures, the GUI uses them to create meaningful displays for the programmer. The GUI is explained in the next section.

3.3 Graphical User Interface

The front end of VISPAD, with which the user interacts, is its graphical user interface (GUI). The role of the GUI is dual: provides output to and handles input from the user. The information about the execution of a parallel program, which is initially recorded in trace files and subsequently processed by the TPE, is represented in a concise graphical way. The design of the user interface is based upon the video player metaphor, which in turn gives the user the ability to navigate through the graphical representation in a tape playback fashion.

The GUI comprises a number of displays, each of which shows a different aspect of parallel program execution, such as the succession of execution phases, inter-process communication, etc. The implementation of each of the displays consists of a static and a dynamic part. A static part (also called a layout part) determines the on-the-screen layout of interface components (called widgets), whereas the corresponding dynamic part performs the actual input and output functions (behaviour) associated with the particular display, together with the necessary interfacing to the TPE.

Dynamic parts were programmed in OSF-MOTIF, and as a consequence, the GUI program is an event-driven one. The XDesigner interactive interface builder was used for the design and generation of the layout parts. The structure of the GUI in general terms is illustrated in figure 3. For more detailed information about implementation details the reader is advised to consult [Tomov and Wierenga, 1993].

The following subsections introduce the three displays of VISPAD, namely the Navigation display, the Membership Matrix Display and the CHIMP Animation display.

3.3.1 The Navigation display

The navigation display provides an abstracted representation of program execution in terms of phases. Phases may be computation, communication or have any other user-defined meaning (as in the case of a program initialisation phase). A phase may be nested within another one, but obviously, phases at the same nesting depth cannot overlap in time. The execution of each process in the parallel program under consideration is divided into its own set of phases.

Animation of program execution is achieved by regularly updating the contents of the navigation display to show the currently active phases during successive run-time intervals. Animation is controlled by a set of buttons that are similar in appearance and role to those typically found on the front panel of a video player to control the playback. Animation also depends on a number of parameters the user is able to alter, such as the width of each time interval and the time step used to advance the current time. The animation of the navigation display is used to drive the animation of the rest of the displays. The latter are automatically updated to reflect program events relevant to the phases currently being shown in the navigation display.

Along with the animation facilities, various other mechanisms used for information refinement or information reduction are provided. These include facilities for expanding a set of phases to their corresponding sub-phases and filters used for excluding classes of phases that are of no interest to the kind of analysis the user performs. In addition, the user is also given the ability to enquire information about the state of a particular phase by clicking on it with the mouse.

This display uses the phase tree data structure maintained and manipulated by the TPE.

3.3.2 Membership Matrix display

The Membership Matrix display gives information about which processes join which SAP groups (a CHIMP concept explained in [Bruce *et al.*, 1993]). In CHIMP, processes may join SAP groups dynamically, so this display is animated to show join events appearing in the order in which they happen at program execution time. In addition to responding to membership changes, the display also updates when new SAP groups are formed.

The TPE data structure used by the Membership Matrix display is the join event list.

3.3.3 CHIMP Animation display

The purpose of the CHIMP Animation display is to graphically show the ordering in time of all the communication events in the program. SAPs are represented as blobs and a graphical encoding scheme is used to distinguish among the different types of CHIMP communication operations, such as sends versus receives and blocking versus non-blocking communication. Again, the animation of this display is closely related to the state of the animated navigation display, and only the communication events relevant to the currently active phases are shown. An important property of the display is that the user may interactively change the arrangement of the blobs to correspond a particular process topology (processes connected in a ring, for example) or pattern of communication.

It uses the communication event list data structure built by the TPE, in conjunction with a data structure maintained by the GUI which stores information about all the SAP groups formed in the program. In that sense, CHIMP animation also depends on the join event list data structure.

4 Issues involved in adapting VISPAD to MPI

The main goal of this project was to adapt VISPAD-CHIMP to MPI based programs. To do this, first we studied the main differences between CHIMP and MPI, summarised in the following section.

4.1 Differences between CHIMP and MPI

Profiling support MPI provides support for profiling via the name shifted MPI functions. This means that the programmer does not have to modify his program to use the instrumented MPI. CHIMP doesn't provide such support for instrumentation.

Collective communication MPI provides collective communication functions while CHIMP doesn't. The PUL-EM library built on top of CHIMP is responsible for providing collective communication.

SAP groups and Communicators SAP groups in CHIMP have a global name space. Communicators in MPI are local opaque objects. In addition, SAP groups have meaningful textual names associated with them, whereas communicators don't.

Non-blocking communications Some CHIMP implementations have a separate thread of control for handling the communications. This allows the instrumented CHIMP library to record the actual completion time of a non-blocking communication. However in MPI, the instrumentation is done using the name shifting support provided by MPI. The source code for the actual MPI communications is not modified. So the actual completion time of a non-blocking communication is not known in MPI.

Datatypes All the communications in CHIMP are byte–oriented. In MPI, messages are typed. MPI provides a rich set of functions to create new datatypes that can be used in MPI communications.

Inter-communicators Since communicators in MPI are local opaque objects, communication between processes in two different communicators must be done by explicitly creating an *inter–communicator*. This is not the case in CHIMP.

Communication modes CHIMP provides blocking and non-blocking synchronous sends and receives. MPI provides a variety of other communications modes as well: standard, buffered and ready.

These differences were used as the primary guide in porting VISPAD-CHIMP to MPI.

4.2 Main stages in the project

After studying the differences between CHIMP and MPI and the structure of VISPAD-CHIMP, four distinct stages were identified as the main stages in the project.

- Adapt and enhance the existing instrumentation library (INS).
- Use INS to write an instrumented version of the MPI library.
- Adapt and enhance the existing trace processing engine to analyse the trace files produced by a MPI program.
- Adapt the existing graphical user interface (GUI) and add MPI-specific displays to the GUI.

The following sections describe what has been accomplished in each of the above stages over the summer.

5 Changes to the instrumentation library

For more information about the instrumentation library used with VISPAD-CHIMP please refer to [Chapple, 1993] and [Tomov and Wierenga, 1993]. To know more about the instrumentation library used in VISPAD-MPI please refer to Appendix A.

5.1 Decoupling INS from CHIMP

The original instrumentation library used with VISPAD-CHIMP was not well-separated from CHIMP. It made calls to CHIMP functions and it depended on CHIMP in several ways. For example, it used

CHIMP for constructing the name of the trace file for each process. These dependencies are apparently due to the fact that it was written specifically for use with the CHIMP library.

So, to use the INS library for instrumenting MPI, it has to be separated from CHIMP. This is mostly a software management issue. This was done by restructuring the INS library so that it does not require any of the functionality of CHIMP or any other message passing library. All CHIMP specific functionality of the INS library were also moved out of INS. Such functionality can now be built on top of the functions provided by INS.

The current INS library can be used for instrumenting *any* program, not necessarily a message passing one. In fact, it can be used even with sequential programs.

5.2 Buffered instrumentation

The new INS library provides facilities to do buffered instrumentation. This means that the events to be recorded can be collected in a buffer before being dumped to the trace file. This is often more efficient than writing out all the events directly to the file. But unbuffered instrumentation is desirable in some cases where buffering may result in loss of some events (an interrupted program, for example). So the new INS library provides facilities to turn the buffering on and off. It also provides functions to forcefully flush the buffer. This might be useful in cases where a program wants to do something useful while it is waiting for something else. By default, the buffer is flushed only when it is full.

5.3 Flat events

As explained in earlier sections, the original INS library allowed a CHIMP program to record the *phases* and the *communication events* that occur during the execution of the program. These are structured events – they have a beginning and a matching ending. So the structure of the trace file was mostly determined by INS and the instrumented CHIMP library didn't require any additional structuring of the events.

The new INS library is more flexible and provides facilities to record *flat* or unstructured events. The user of INS, the instrumented MPI library in our case, can build more structured event tracing based on these flat events. For example, the instrumented MPI library uses these flat events to mark the beginning and ending of a communication. Previously communication events were part of the INS itself.

5.4 Delayed events

There are cases when the user of the instrumentation library wants to record an event at a particular point in a program but does not have all the necessary data at that point. Take for example the tracing of a non-blocking send in CHIMP. Let us assume that the user wants to identify it using the handle returned by the send function. But this is not available before the function is called. But the user wants to record a *begin* event before the function was called.

INS provides *delayed* events for use in such situations. Events can be *registered* with INS at any point in the program. The data associated with the event can be changed before the event is *completed*, i.e, finally recorded in the trace file.

6 Instrumenting the MPI library

VISPAD-MPI supplies an instrumented version of the MPI library. MPI programs can be linked against this library instead of the regular MPI library. The resulting executable generates trace files that can be visualised using VISPAD. This section explains how the instrumented MPI library works and what information is collected during the execution of a MPI program. First we shall see how a MPI function can be instrumented.

```
int MPI_Send(void *buffer, const int count, MPI_Datatype type,
             int dest, int tag, MPI_Comm comm)
{
  int result;

  /* record some information here */

  /* call the actual MPI function */
  result = PMPI_Send(buffer, count, type, dest, tag, comm);

  /* record some more information here */

  /* return the original return value */
  return result;
}
```

Figure 4: Name-shifting in MPI

6.1 Profiling support provided by MPI

MPI provides facilities to profile or trace an MPI program via the *name-shifting* mechanism. Under this scheme, every MPI function MPI_xxx has a corresponding PMPI_xxx function that does the same work as MPI_xxx. A profiled or instrumented version of MPI_xxx can be provided by writing our own MPI_xxx function that calls PMPI_xxx and records additional information about the program in a trace file.

Figure 6.1 shows how MPI_Send can be instrumented using the name-shifting mechanism.

The instrumented MPI uses the INS library described earlier to record information about the program in trace files. Each process in the program creates one trace file containing all the information relevant to that process. The following sections explain what information is collected from a MPI program and how.

6.2 Phases in a MPI program

For every instrumented MPI function, the instrumented MPI library records a phase in the trace file. For example, in the case of MPI_Send, before calling PMPI_Send, it records that a phase called ”MPI_Send” has started and after PMPI_Send returns, it records that a phase called ”MPI_Send” has ended. In addition, the instrumented MPI library records a phase that begins when MPI_Init is called and ends when MPI_Finalize is called. Thus the life time of a process is recorded as a single phase that in turn contains various phases corresponding to MPI function calls.

This results in a two-level phase tree with a single root representing a process in the program and all the leaves in the next level representing calls to MPI functions.

6.3 Communication events in a MPI program

The instrumented MPI library records all the important communication events that happen in a MPI program. These include the beginnings and endings of all point-to-point communications and all collective communications. Notice that these beginnings and endings are properly nested without overlapping for all blocking communications but not for non-blocking communications. This is because a blocking communication begins when the corresponding MPI function is called and it ends when the function returns. So it retains the structure of function calls which is proper nesting without overlapping. On

Send	Ssend	Bsend	Rsend
Isend	Issend	Ibsend	Irsend
Recv	Irecv	Barrier	Bcast
Gather	Scatter	Allgather	Alltoall
Reduce	Allreduce	Scan	

Table 1: Types of communication

the other hand, a non-blocking communication begins when the MPI function is called, but it does not end when the function returns. Instead, a non-blocking communication in MPI ends when its test for completion using one of the many Wait/Test calls succeeds.

To match the beginnings and endings of non-blocking communications, the instrumented MPI library uses the request handles returned by the non- blocking MPI functions. Since such handles are available only after the function call returns, it uses the delayed event mechanism described earlier. Instrumented versions of MPI_Test and MPI_Wait record the endings of such communications by recording their corresponding request handles. Later during trace processing, these request handles are matched to match the beginnings and endings of non-blocking communication events. This scheme relies on the fact that the request handles can be converted to integers and that the same request handle cannot stand for two outstanding communications at the same time.

Blocking communications are handled similarly but are associated with an implicit request handle that cannot be the result of any non-blocking communication (that of MPI_REQUEST_NULL).

6.4 Communication event data

In addition to recording the beginning and ending of a communication event, the instrumented MPI library records data that may be specific to a particular type of communication event. Currently it handles 19 types of communication listed in table 1.

The information recorded for all these types of communication are summarised in table 2.

Notice that both Recv and Irecv record the actual tag used for communication and the actual source of the message even though the MPI functions may have been called with a wild card tag (MPI_ANY_TAG) or a wild card source process (MPI_ANY_SOURCE). This is done by enquiring the status object associated with the Recv/Irecv. In addition, the actual number of elements received is also recorded for Recv[1].

The datatypes of elements used in communication are recorded by converting their handles into integers and assuming that the same datatype handle cannot represent two different datatypes at the same time. Notice that both the datatype and request handles are local to a process. However, for each communicator, a globally unique integer is recorded. The communicator naming scheme described in the next section explains how a globally unique integer is associated with a communicator.

6.5 Communicator naming scheme

One of the differences between MPI and CHIMP is that while CHIMP SAP groups have globally unique names, communicators in MPI are represented by handles which are local to a process. But we need globally unique ids or names for communicators in VISPAD for the following reason: It is not enough to say that process 0 sent a message to process 1 using its communicator X and process 1 received a message from process 0 using its communicator Y. Instead we want to inform the programmer who is using VISPAD that process 0 and process 1 participated in a communication using a common communicator A.

[1] This is not possible for Irecv for which the communication completes in a separate Test/Wait function. In MPI, to get the actual count of elements received, one needs to know datatype of elements, which is not known within the Test/Wait functions.

Send, Ssend , Bsend, Rsend, Isend, Issend, Ibsend, Irsend	Number of elements sent Type of elements sent Tag used for communication Communicator used for send Rank of the receiving process
Recv	Actual number of elements recvd. Type of elements recvd. Actual tag used Communicator used for recv Rank of the sending process
Irecv	Max. number of elements recvd. Type of elements recvd. Actual tag used Communicator used for recv Rank of the sending process Request handle
Barrier	Communicator used
Bcast, Reduce	Number of elements Type of elements Communicator used Rank of the root process
Gather, Scatter	Number of elements sent Type of elements sent Max. number of elements recvd. Type of elements recvd. Rank of the root process Communicator used
Allgather, Alltoall	Number of elements sent Type of elements sent Max. number of elements recvd. Type of elements recvd. Communicator used
Allreduce, Scan	Number of elements Type of elements Communicator used

Table 2: Information recorded with communication events

	0	1	2	3	
MPI_COMM_WORLD	0	1	2	3	0.0
EVEN	0	-	1	-	0.1
ODD	-	0	-	1	1.0
RIGHT_HALF	-	-	0	1	2.0
MIDDLE	-	0	1	-	1.1

Figure 5: Communicator naming scheme

This globalisation of MPI name space is important because it helps the programmer understand his/her program better.

To achieve this, the instrumented MPI library uses a simple naming scheme. Whenever a communicator is created, it assigns a unique leader to that communicator and the leader assigns an integer to the communicator which is unique to that leader process. This two-part id – rank of the leader in MPI_COMM_WORLD and the integer assigned by the leader – gives a unique id to the communicator.

For intra-communicators, the process with rank 0 in that communicator is its leader. Inter-communicators already have two leader processes, one from each of the intra-communicators that make up the inter-communicator. Of the two, the one with a smaller rank in the peer communicator is made the leader. The rank of the leader in MPI_COMM_WORLD becomes the first part of the unique id of the communicator.

For the second part, each process keeps a count of the number of communicators for which it is the leader. Whenever it becomes the leader of a new communicator, it increments the count and uses it as the second part of the communicator id.

Figure 5 shows how the naming scheme is used. The top row shows all the processes in a MPI program. The left most column shows how the programmer views all the communicators in his/her program. The matrix in the middle shows the ranks of each process in each of the communicators. The right most column shows the names assigned to the communicators by the naming scheme described here.

The two-part id generated by the naming scheme is packed into a single integer by the leader process and then broadcast to all the members of the communicator. Then every member process in that communicator uses the caching facility in MPI [Mes, 1994, page 165] to associate that integer with the communicator. Whenever the information about a communicator has to be recorded, the instrumented MPI library records the corresponding integer id instead.

6.6 Implementation of MPI_Pcontrol

The MPI specification [Mes, 1994, page 199] gives guidelines for implementing a function called MPI_Pcontrol that gives the programmer flexible means of controlling the instrumentation. Our implementation of the instrumented MPI library conforms to those guidelines and provides ways of

- Switching off the instrumentation

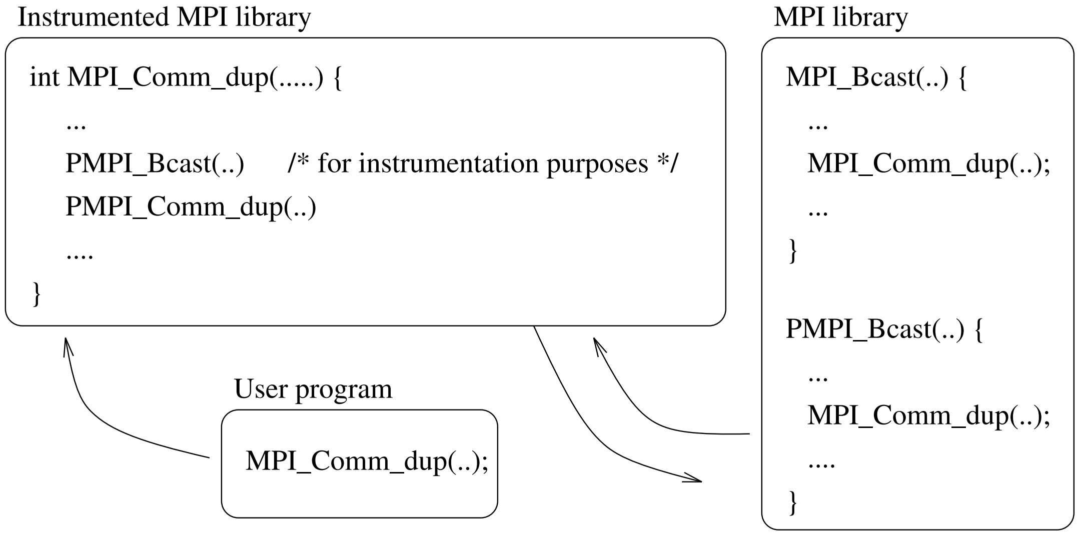


Figure 6: Nested instrumentation

- Switching on the instrumentation
- Flushing the instrumentation buffer
- Switching off the buffering and
- Switching on the buffering

6.7 Avoiding nested instrumentation

[Mes, 1994, pages 202-203] explains the problem of nested instrumentation that can result from the use of the name-shifting mechanism provided by MPI. The problem arises because some of the MPI functions may have been implemented using other MPI functions (MPI_Bcast in terms of MPI_Send, for example). The situation becomes more complicated if an instrumented MPI function itself tries to use other MPI functions. It is possible to have infinite mutual recursions because of this nesting. See figure 6.

This is avoided by having a static flag that indicates whether one is already inside an instrumented MPI function. In which case, the corresponding PMPI function is called directly without any instrumentation.

7 Changes to the Trace Processing Engine (TPE)

The Trace Processing Engine (TPE) is responsible for reading in all the trace files, analysing all the data and producing the data structures required by the Graphical User Interface (GUI). VISPAD-CHIMP's TPE is explained in detail in [Tomov and Wierenga, 1993]. This section describes the modifications and extensions done to the original TPE for VISPAD-MPI.

7.1 Inter-process trace analysis

The original TPE handled the trace files one at a time. This means that it read a trace file containing all the events from one process and analysed it in isolation. The new TPE reads in all the trace files and analyses them all. This enables the TPE to do more sophisticated trace analysis. Almost all the enhancements to the TPE, described in the following section, depend on inter-process trace analysis.

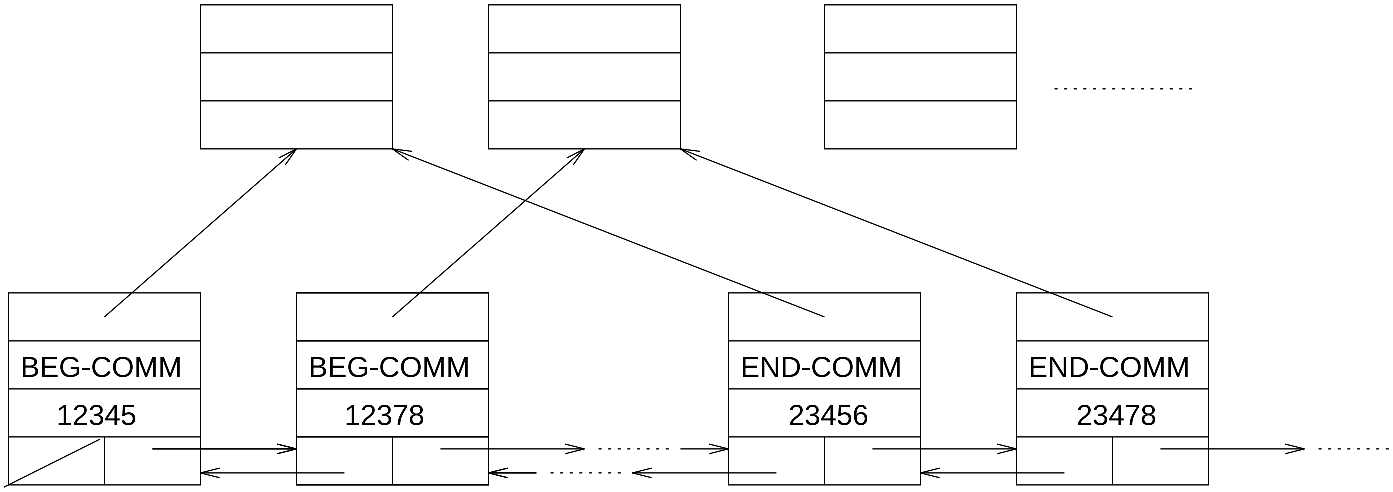


Figure 7: The original communication event list

7.2 Phase Trees

For each process, the TPE generates a tree of all the phases that occurred in that process. At the end of inter-process trace analysis, it returns an array containing the phase trees from all the processes. The structure of the phase tree itself hasn't changed from that of VISPAD-CHIMP. The interface to access the phase tree hasn't changed either.

7.3 List of communication events

The new TPE generates a list of all communication events that happen in a process, just like the original TPE did. But the new TPE also merges the communication event lists from all the processes to produce a single merged list of communication events sorted in time order. This makes the job of the animation display in the GUI a lot easier. It does not have to compare the next outstanding event from all the processes to decide the next earliest communication event to animate. With the new merged communication event list, the GUI just picks up the next event to animate from the list.

There is another slight modification to the structure of the communication events list. Originally it looked like figure 7. The events are connected in a doubly linked list with a BEGIN COMM event sharing all the communication information with its matching END COMM event. This is quite inefficient because every time the animation module wants to find a matching END COMM event for a BEGIN COMM event or vice versa, the TPE has to search through the communication event list.

The new communication event list has reverse pointers from nodes that contain communication information to the events in the linked list. With this, matching events just becomes a two-pointer traversal. The new data structure is shown in figure 8.

The new TPE provides functions to

- To check whether an event is a BEGIN event or an END event,
- To check whether an event is a point-to-point communication event or a collective communication event
- To check if it is a blocking communication event or a non-blocking communication event
- To get the type of communication event (one of the 19 types described earlier)
- To find out the communicator in which a communication event occurred

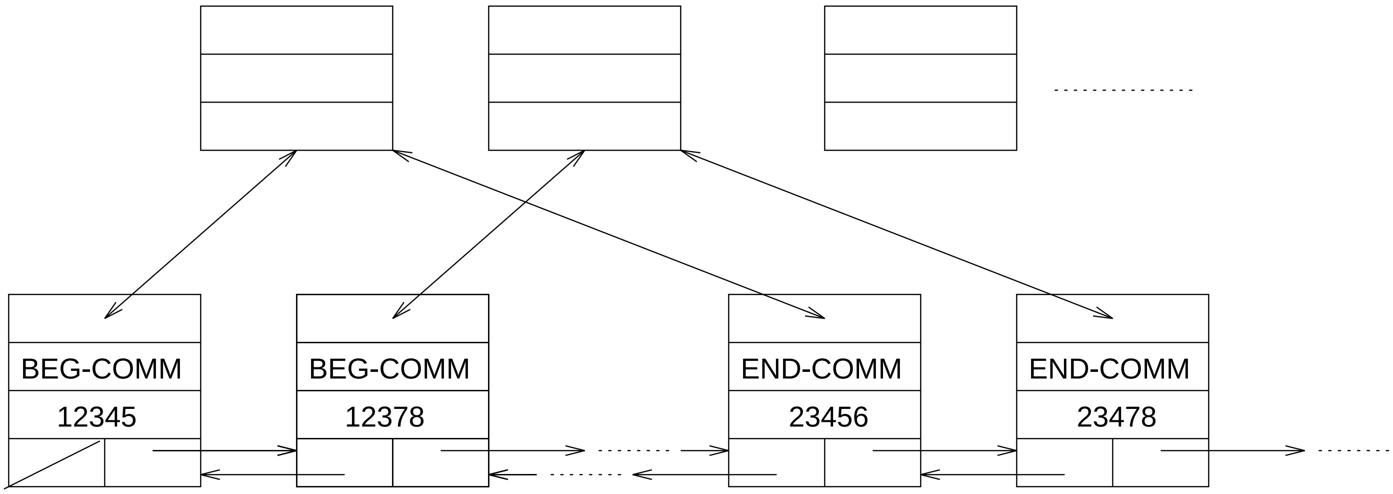


Figure 8: The new communication event list

The next section describes how the TPE allows only some of the communication events to be selected so that the level of information displayed during animation can be reduced.

7.4 Selecting events in the communication list

When there is a lot of communication events happening in a program, the user may want to filter out some of them and see only the remaining. For example, the user may want to see only the communication events happening in a particular communicator. Or only those happening between two processes. The new TPE provides a selection mechanism to provide such filtering capabilities.

It provides functions to select all the events belonging to a particular process and all the events that use a particular communicator. These selections are additive in nature. The TPE provides functions to move to the next or previous selected event in the merged communication list. An event is said to be selected if the process which it belongs to has been selected *and* the communicator it uses has been selected.

7.5 Translating ranks to pids

The new TPE treats the processes in a MPI program differently from the old TPE. Each process is said to have a *pid* which is the same as its rank in MPI_COMM_WORLD. In addition, each process can have a *rank* in each communicator. While pids are global, ranks are local to a communicator.

One of the main jobs of the TPE is to figure out which processes participated in a communication event. But in several cases, the recorded information in the trace file only provides the *rank* of a process in a communicator rather than its global *pid*. For example, the trace file for process 0 may contain an event that is a send to the process with rank 2 in the communicator with id 0.1. For animation, we need to know which process has rank 2 in the communicator 0.1.

The new TPE provides functions that translate the ranks of processes in communicators to their global pids. The TPE accomplishes this by creating a list of all communicators in the program and information about their member processes. This is explained in the following sections.

7.6 Join event list

The instrumented MPI library records all the communicator creation events. Whenever a process creates a communicator with other processes, it is said to *join* that communicator. The instrumented MPI library records every join event along with the rank of the process that joins a communicator. During trace processing, the TPE creates a list of all the join events that happen in a process. It then merges such lists from all the processes to produce a list of all join events that occur in the MPI program. This merged join event list is used by the GUI to create an animated display of processes joining the communicators.

7.7 List of communicators

In addition to the merged join list described above, the TPE creates a list of all communicators in the program. For each communicator, the TPE also collects the member processes and their ranks. For an inter-communicator, the rank of a process is same as its rank in the intra-communicator which forms part of the inter-communicator. In addition, the TPE collects the ids of the two intra-communicators that form an inter-communicator. With all this information, the TPE provides facilities to

- Locate a communicator with a particular id
- Check if a communicator is an inter-communicator or an intra-communicator
- Check if a process is a member of a particular communicator
- Find out the global pid of a process which is a member of a communicator (translating the rank to pid)
- Find out the rank of a process in a communicator (translating the pid to rank)
- Find out the ids of the two intra-communicators that form an inter-communicator

7.8 Collecting statistics

The TPE collects several useful metrics from the phase trees and the communication event list. For each process, it collects the following metrics:

- Number of messages sent
- Number of messages received
- Number of integers sent
- Number of integers received
- Number of floats sent
- Number of floats received
- Maximum blocking time
- Minimum blocking time
- Average blocking time
- Total blocking time
- Number of blocking communication calls
- Maximum waiting time
- Minimum waiting time
- Average waiting time
- Total waiting time
- Number of MPI_Wait calls
- Maximum testing time

- Minimum testing time
- Average testing time
- Total testing time
- Number of MPI_Test calls
- Total time spent in MPI function calls

8 Changes to the User Interface

The adaptation of VISPAD-CHIMP to MPI required a number of changes to the user interface of the tool. Of the three displays implemented last year, two (the membership matrix and the CHIMP animation displays) were dependent on CHIMP concepts. The changes made affected both the graphical representation of events and the underlying animation control code. In addition to adapting the existing displays, two new displays were introduced: the statistics and profile displays. Finally, there were some changes made to improve the usability of the interface as a whole and enhance the quality and efficiency of the underlying code.

8.1 Some usability issues

This section introduces some human-computer interaction concepts that are considered useful in the design of good user interfaces. It does not intend to provide a complete or scientifically precise list of usability issues. It merely describes (in an informal way) some of the factors taken into consideration when parts of the interface were redesigned.

Important to any user interface is the provision of adequate and prompt *feedback*. Informative, clearly visible and noticeable messages, progress indicators and sounds are some of the most common means for providing feedback in good interactive programs. The role of feedback is to communicate the current state of the program and information about the availability and completion of various operations to the user.

Another important consideration in tools like VISPAD is that of dialog initiative. In *system pre-emptive* dialogues, the system initiates all dialogue and the user simply responds to requests. On the other hand, in user pre-emptive dialogues, the user is free to initiate any action toward the system. The letter type of dialogue is obviously the most preferable for VISPAD, and care was taken to make interaction with VISPAD as user pre-emptive as possible. However, in certain situations (such the occurence of errors), the program should be able to temporarily take over the control of the interaction from the user.

In many application programs, interaction can enter one of several *modes*. The interpretation of actions changes, as the mode changes. In order to avoid problems, the user should always be aware of the current mode. Better still, modes should be avoided altogether.

8.2 Adapting the existing displays to MPI visualisation

8.2.1 Navigation display

Visualisation and animation of program execution phases are, at least conceptually, independent of the message passing system used in the parallel program under consideration. For this reason, the implementation of the navigation display in VISPAD-CHIMP did not depend on any CHIMP concepts. Most of the work on this display concentrated on improving the user interface itself and adding some features that were not implemented last year. The navigation display of VISPAD-MPI is shown in figure 9.

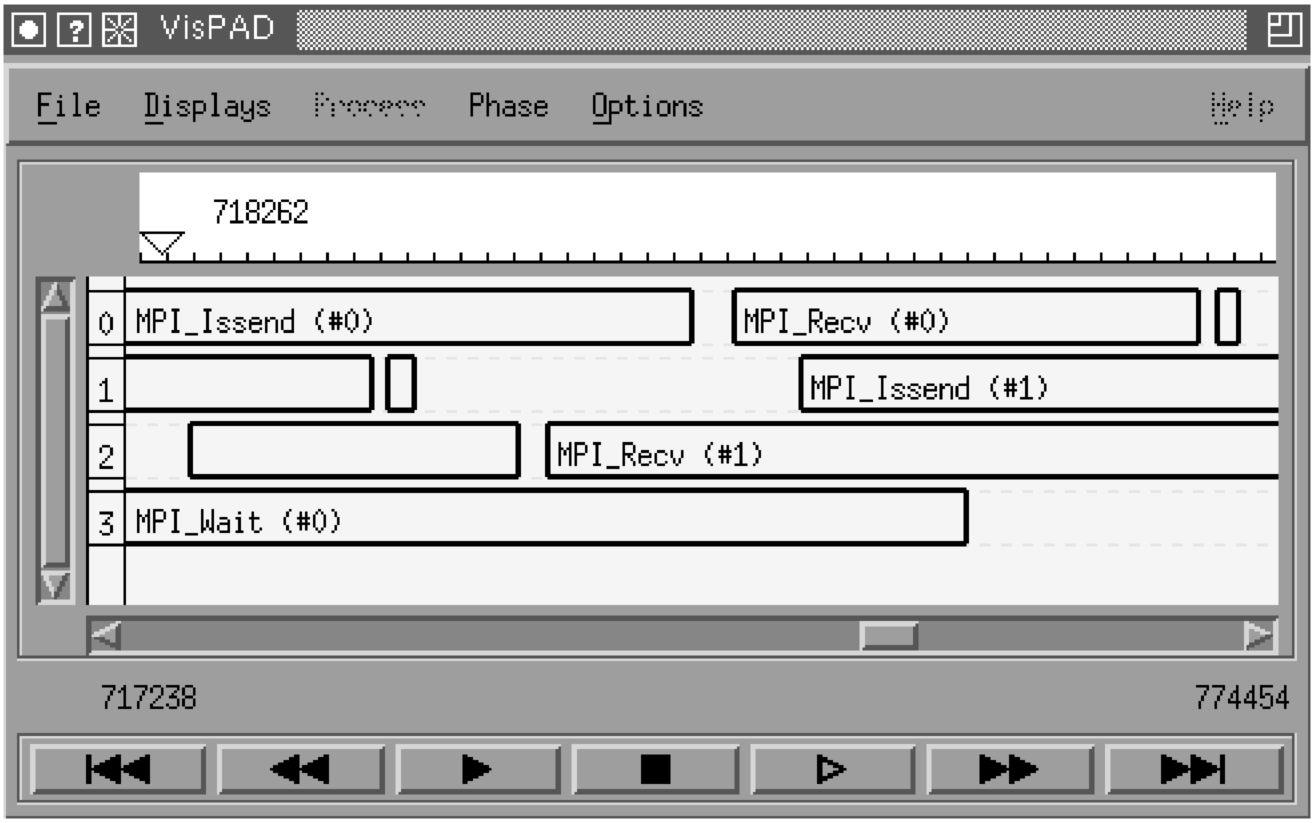


Figure 9: Navigation display

In the original version, the arrangement of phases was such that the phases associated with the i-th process in the program occupied the i-th row on the display. In addition to that, a process numbering facility was introduced to identify which process a particular row of phases belongs to. Each process in the parallel program is numbered according to its unique rank in MPI_COMM_WORLD. The same numbering scheme is also employed in all the other displays.

The navigation display interface allowed certain operations, such as selection, filtering and expansion, to be performed on phases. The introduction of processes in the representation gave rise to a new range of possible operations. Such operations may be the selection and elimination of processes, and cut, copy and paste operations to alter the ordering in which processes appear on the display. Although only the process selection mechanism was implemented this year, the interface was changed to reflect the classification of all available operations in process-oriented and phase-oriented. Such a classification is desirable, as far as the user interface was concerned, because it directly corresponds to the two main objects of interest: processes and phases. The above two sets of operations were grouped under two distinct menu items which appear on the menu bar at the top of the display. As a result, the buttons which were originally used for the selection, filtering and expansion mechanisms were removed.

In the original navigation display, the phase selection and process elimination mechanisms had the effect of putting the dialogue into modes, namely the selection and elimination modes. In addition to the difference in the behaviour of the Filter and Expand mechanisms according to the particular mode in which the tool was in, exit from either of these modes was not always clearly signposted. In VISPAD-MPI both of the above modes were eliminated.

The original messaging mechanism used a special message area which occupied the bottom part of the display. Messages displayed in this area were collected in a message history list. A problem with messages that always appear in a fixed place is that they are not likely to attract the user's attention. In our version the code was modified to pop up a message window in a central location on the display. Apart from being instantly noticeable, this general-purpose message window blocks any interaction with the rest of the tool, until the user responds to the message.

Apart from changes to the user interface alone, the animation control code was also modified to speedup the animation of the navigation display. In VISPAD-CHIMP, a drawing area was used as a slide window on a (usually) larger pixmap. The latter was where all the drawing was taking place. For every frame in the animation sequence, all the current phases for all processes were found and drawn on the pixmap. In the most common case of a drawing area not large enough to accommodate the total number of processes, much of the processing and drawing time was wasted on phases that, eventually, were to be mapped outside the boundaries of the drawing area. In the current version, the dimensions of the pixmap equal those of the drawing area. The vertical scroll bar no longer logically slides the drawing area along the height of the pixmap, but it is used to determine the range of currently visible processes. The latter are the only ones for which phases are drawn during animation. The extra work required to display processes and phases that were previously not visible was moved outside the main animation loop to the functions that handle window resizing and scrollbar movements.

8.2.2 Membership Matrix display

The role of the membership matrix display (MMD) in VISPAD-MPI is to visualise which communicators each process in the parallel program belongs to. In VISPAD-CHIMP the various memberships of processes in SAP groups were animated. Although animation is still possible in the new version of the MMD, a static representation of memberships was also introduced. The advantage of such a representation is that all memberships are shown on the display, regardless of what point in the trace file the animation is currently at.

The graphical representation of the membership matrix remained essentially the same as that of the original version. Process numbers are displayed along the top of the matrix, whereas communicator names are listed vertically on the left hand side of the matrix. If process j is a member of communicator i, then a circle is drawn in position (i,j) of the matrix. Within each circle, the rank of the corresponding process in that communicator is shown.

When the MMD is initialised, the list of all communicators in the program is scanned in order to obtain all communicators each process is a member of. These memberships are initially drawn as described above. This constitutes the static representation of memberships. Animation is accomplished by progressively filling in the circles with colour as the animation clock goes past each of the join events. Figure 10 shows an example of the membership matrix display.

The only local data structure maintained in the GUI code for MMD is a look-up table used to return the row of the matrix occupied by a communicator given its unique ID. The two-dimensional SAP table of the original version was discarded.

8.2.3 MPI Communication display

The MPI communication display implements a graphical representation of and mechanisms for animating the communication events in a parallel program which uses the MPI message passing system. In that sense, its role is similar to that of the CHIMP animation display of VISPAD-CHIMP. However, this new display is more than just the CHIMP animation display adapted to MPI. The numerous differences between CHIMP and MPI suggested that it should better be redesigned from scratch. This allowed the introduction of new features which extend the functionality of the original display and which suggest new ways in which the display may be used. The most important of these features are:

- Multiple instances of the display may be open simultaneously. Each instance shows a (possibly) different set of communication events.
- For each instance, a textual description of events supplements the graphical representation.
- A user controlled mechanism for step-by-step animation of communication events only is provided.

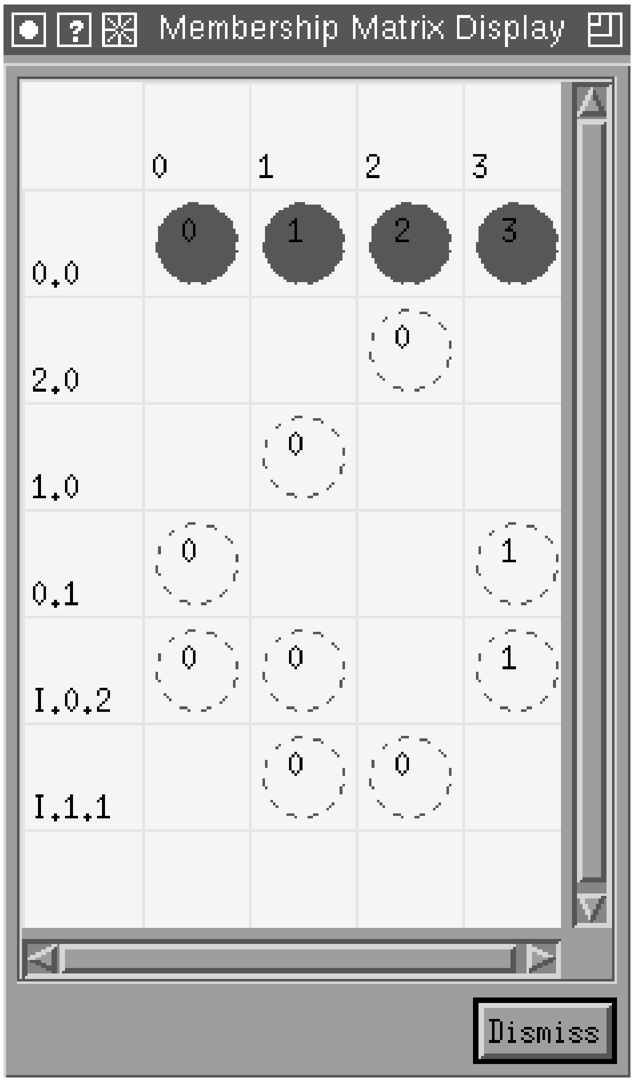


Figure 10: Membership matrix display

- Visualisation of both point-to-point and collective communication.

The various components of the MPI communication display are shown in figure 12.

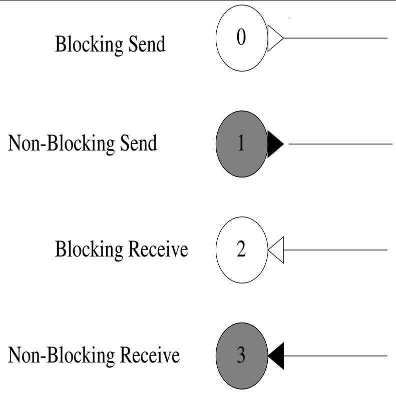


Figure 11: Graphical encoding of MPI point-to-point communication operations

The idea of having more than one instance of the display open at the same time stems from the concept of MPI communicators. A communicator in MPI represents a self-contained "communication universe", and all communication that takes place in a particular communicator does not interfere with any other communication elsewhere in the program. The idea of associating a particular instance of the display with a single communicator was extended to allow for any user-defined set of processes and communicators. Each time a new instance of the display is opened, the user is presented with a list of all the processes and a list of all the communicators in the parallel program. From these lists, the user is then able to select any two sets of processes and communicators which define a (possibly empty) subset of communication events to be visualised in the newly created instance.

The graphical representation of events used is a slightly modified version of the one used in the CHIMP animation display, extended to include MPI collective operations. The graphical encoding scheme used for point-to-point operations is illustrated in figure 11. Processes are represented as blobs and communication operations (sends and receives) as arrow heads attached to the peripheries of the blobs. For the purpose of visualisation, each point-to-point operation (receives included) is assumed to have a source and a destination, which map onto blobs on the display. As long as any of the two processes corresponding to the source and destination blobs is engaged in communication with the other, a line is drawn between the blobs, to clearly identify them on the display. This is different from the original version in that the line was drawn only when the send and receive operations happened to overlap in time. It is believed that the new scheme enhances the clarity of representation. Additionally, blobs are coloured as either empty or filled to indicate whether the corresponding processes are currently being blocked by a blocking communication event or not. Collective operations are represented as symbolic labels drawn within blobs. For those collective operations which distinguish a particular process as being the root of the operation, the label in the blob corresponding to the root process is coloured differently. Table 3 lists all the instrumented collective functions and their corresponding symbolic labels.

In the CHIMP animation display, a grid was used for the on-the-display arrangements of blobs. VISPAD-MPI favours a ring-shaped arrangement of blobs, as it provides a neat, uncluttered representation of simultaneous communication events and minimises the potential overlap between the various graphical elements on the display (i.e. blobs, arrow heads and lines). However, such an arrangement is quite space consuming and becomes impractical for large numbers of processes. The MPI communication display selects either one of the above two patterns according to the number of processes in the parallel program.

In addition to the graphical representation, a list of single-line textual descriptions of each event is also provided. The textual description aims primarily at:

- resolving any ambiguities that might be present in the graphical representation
- providing timing information about the begining and completion of events

Barrier	X
Broadcast	B
Gather	G
GatherV	GV
AllGather	AG
AllGatherV	AGV
Scatter	S
ScatterV	SV
AllToAll	AA
AllToAllV	AAV
Reduce	R
AllReduce	AR
ReduceScatter	RS
Scan	SC

Table 3: MPI collective operations and corresponding labels

- supplying the user with a history list of communication events, so that the current state of the display can be related to previous ones.

VISPAD-CHIMP featured a single animation mechanism for all displays. In addition to the this mechanism, which was extended to include backward animation, a step-by-step animation mechanism for use with the communication display alone was implemented. Its purpose is to allow the user to manually browse through the sequence of communication events and it is shared by all the instances of the communication display. Animation proceeds in discrete steps which are explicitly triggered by the user. The necessary control interface is provided in a separate window called the communication browser in the program. Step-by-step browsing is suitable for the close examination of the ordering of communication events in time, as it does not need to go through long non-communication phases before the next communication event is displayed. The browsing mechanism also updates the rest of the displays.

For animation, each instance maintains information about the various blobs and the set of communication events associated with it. At initialisation time, each instance obtains a copy of the single communication event list supplied by the TPE and which contains all the communication events recorded in the trace files. When the user finishes the selection of processes and communicators for this instance, TPE functions are called to mark the appropriate events in the list as selected. Animation skips unselected events and updates all the instances in such a way that all appear to be synchronised by a common clock whose value at each step equals the time stamp of the most recently displayed communication event.

8.3 New displays

8.3.1 Statistics display

The statistics display provides a list of metrics for each process as depicted in figure 13. These metrics, which are collected during trace processing, are communication specific quantities. The list of metrics supported in the current version include

- the number of messages sent and received by a each process
- the minimum, average and maximum times spent by each process on blocking communication and calls to wait and test functions.

The above metrics may be used in performance analysis to reveal communication bottlenecks. Future versions of the tool may convert the representation of metrics to a more graphical one, which would indicate at a glance which processes are engaged in heavy communication.

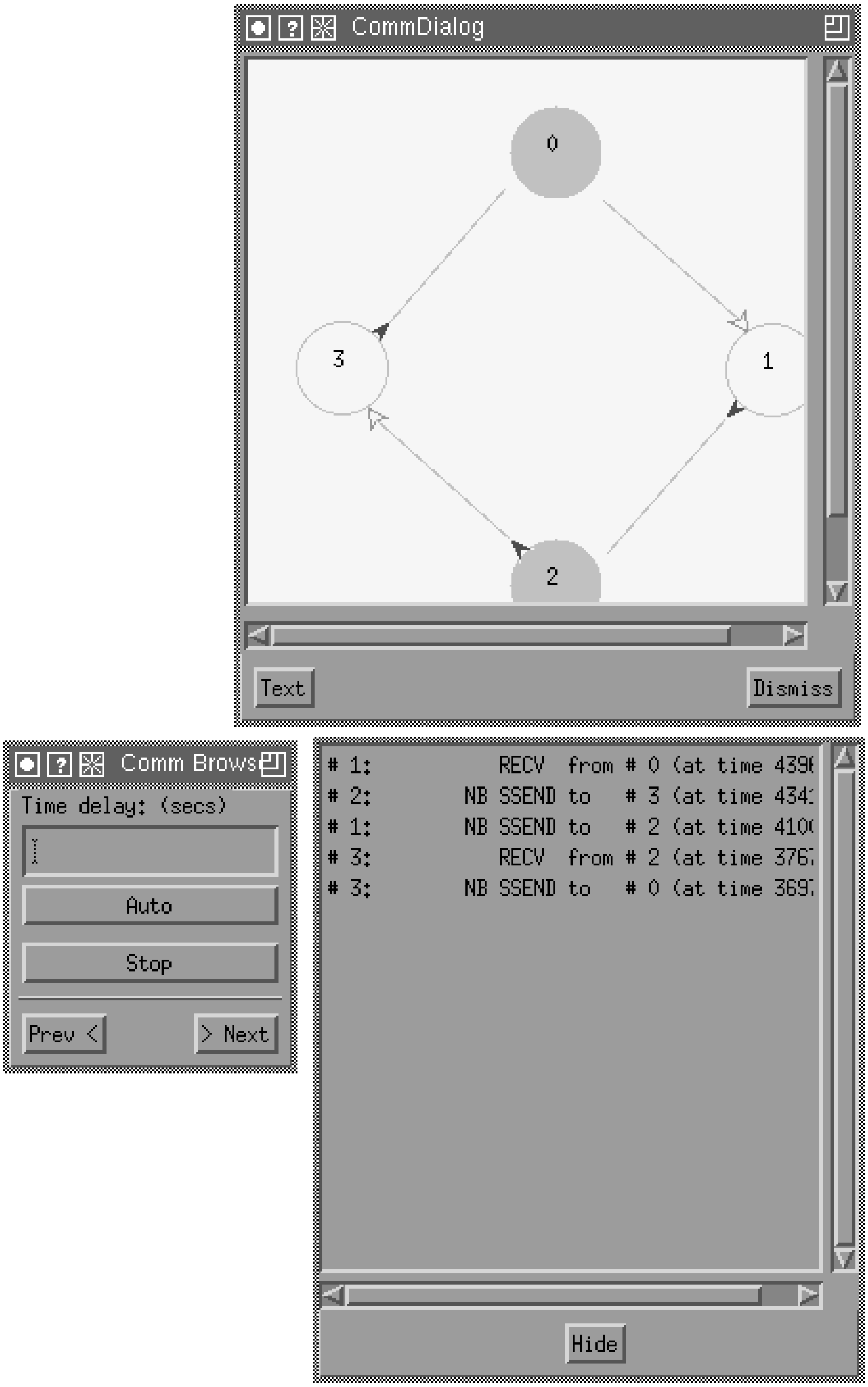


Figure 12: MPI communication display

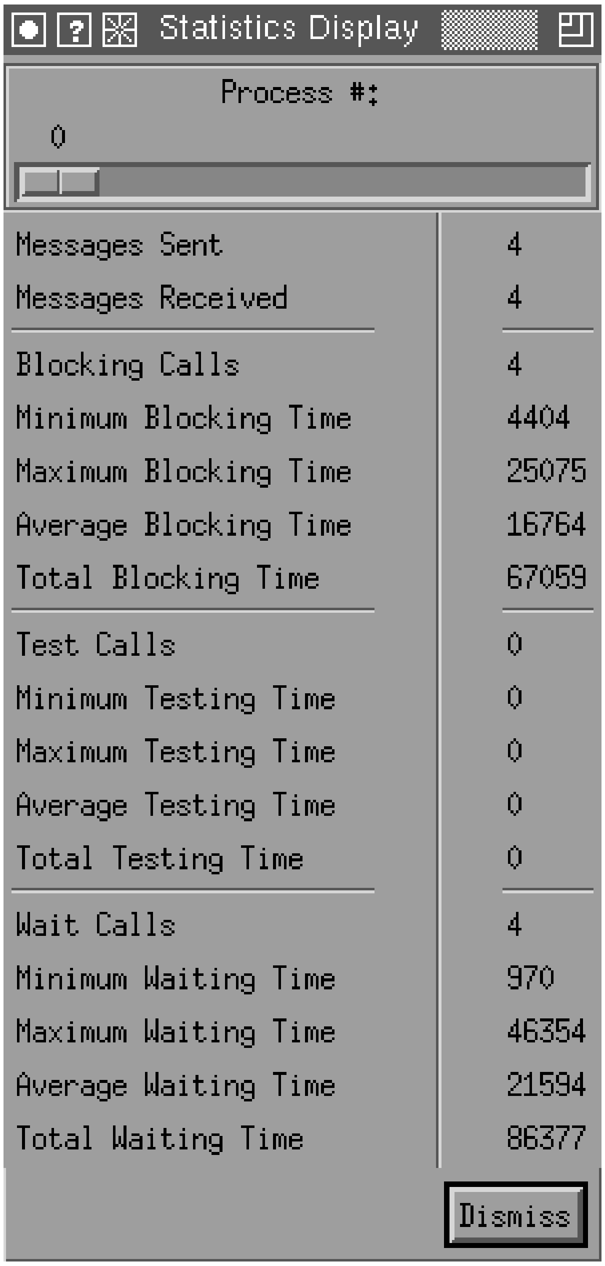


Figure 13: Statistics display

8.3.2 Profile display

The profile display presents quantitative information about the various execution phases in a parallel program. This is different from the representation of phases in the navigation display, in which emphasis is given on the ordering of phases in time. The information provided is mainly concerned with the execution time of a particular instance of a phase in the program (e.g. a single MPI_SSend phase) or the total time spent on a particular phase (e.g. total time spent on MPI_SSends) for each process. In addition to absolute numeric values, a bar-graph scheme is also employed to clearly indicate the most time consuming phases and facilitate comparisons between groups of different phases.

An experimental version of this display was first implemented by Anna Hondroudakis and was incorporated in VISPAD-MPI without changes. The display itself is shown in figure 14

8.4 The display module

In the original version of VISPAD, the display module contained the definition of a data structure (displayDataStruct) and a set of functions for use in all displays based on the so called display dialogue. The latter consisted of a widget hierarchy which implemented a simple scrolled window (a drawing area with scrollbars) and a push button. This structure was used as the backbone of both the MMD and the CHIMP animation display. Although the functions provided were concerned only with setting the various parameters of X drawing functions and with clearing pixmaps, the basic idea was to establish a uniform way of performing some operations across different displays.

Both the data structure and the set of functions provided were extended to include facilities for general purpose high level display management. The module is now completely independent of the display dialog and all convenience functions may be used with any display layout, even with ones that do not contain drawing areas. Nevertheless, special attention was given to displays which have a drawing area (such as the MMD, for example). There are new functions to create pixmaps and graphics contexts, handle resize and exposure events (if necessary), and automatically update the contents of the drawing area whenever the user moves any of the scrollbars. The only assumption made is that the dimensions of the pixmap remain constant for as long as the display is open. (As this is not the case with the navigation display, the latter makes use of its own unique callback functions for resizing and scrollbar management.)

The main benefits of using the functions in this module are:

- Lengthy calls to complex X library functions are replaced by single-line calls to simple functions and need not be manually duplicated each time a common operation is required.
- No code needs to be written for processing difficult to handle events, s uch as resize, exposure and scrollbar generated ones (unless the behaviour of a particular display needs to be different from the supported one).
- Displays are manipulated in a uniform way.

Future developers are advised to exploit (and possibly extend) the functionality provided in this module when implementing new displays for VISPAD.

9 Future work

9.1 Future work on the instrumented MPI library

- More MPI functions can be instrumented thereby enhancing the functionality of VISPAD.
- Experiments can be performed to measure the instrumentation overhead incurred by MPI programs which use VISPAD's instrumented MPI library. The instrumentation can be optimised to reduce this overhead.

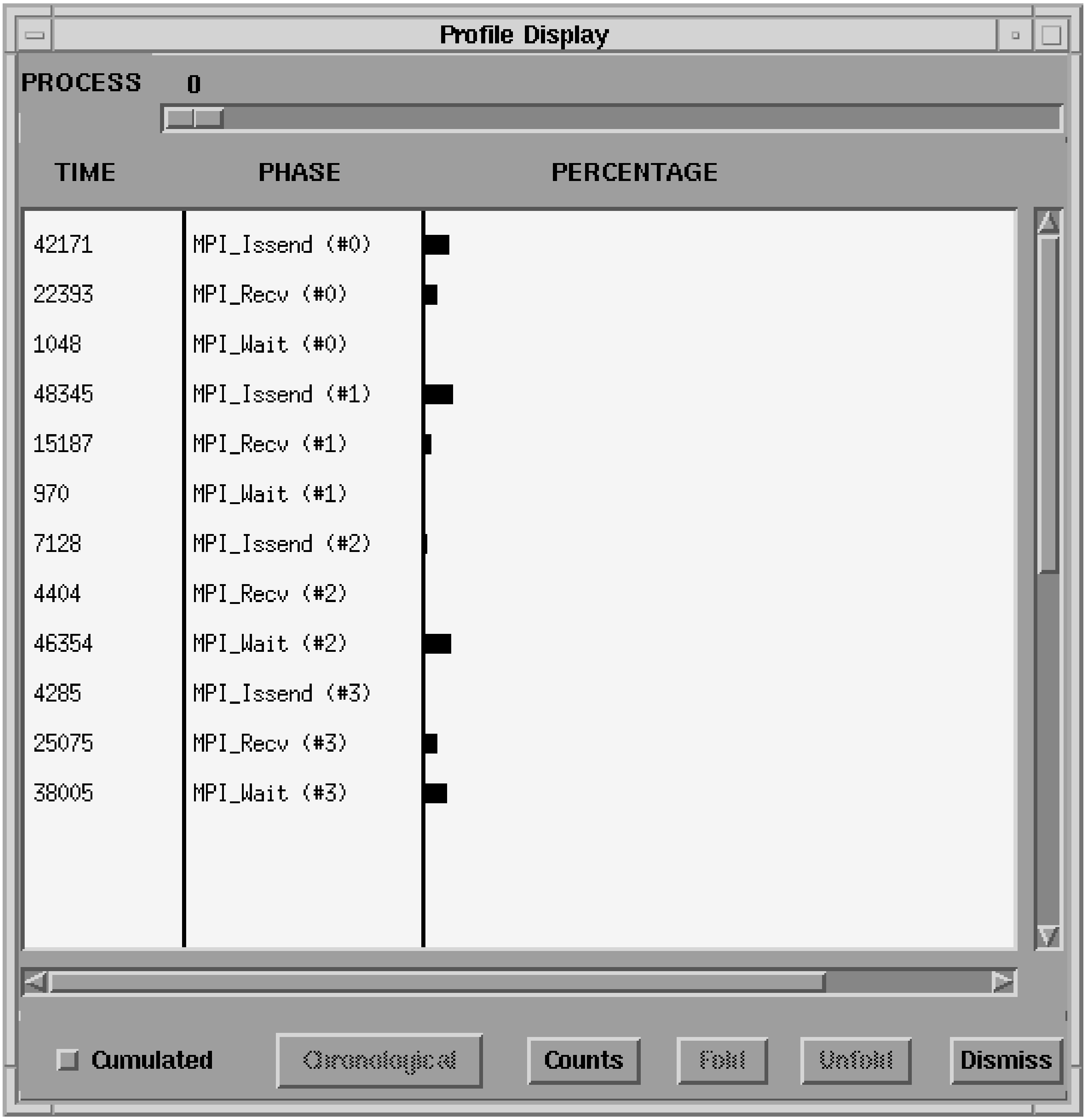


Figure 14: Profile display

- More experiments can be done to tune the parameters for buffered instrumentation. Buffered instrumentation is not more efficient always. So experiments can be performed to find out the best buffer size, ideal points to flush the buffer and other interesting parameters.

9.2 Future work on the TPE

- Utilities can be written to translate the trace files obtained on one platform to another. Currently, VISPAD has to be run on a machine of the same architecture as the one on which the original MPI program was run on.
- Trace processing can be optimised further. Using more advanced data structures can reduce the time for trace analysis.
- Statistics collection can be done more elegantly. The datatypes should be handled more generally during trace processing. Additional metrics can be collected and a method of filtering the statistical information can be incorporated into VISPAD.

9.3 Future work on the GUI

- A graphical representation of process metrics could be implemented. Such a representation could use a colour scale to indicate differences in the magnitute of a particular metric across processes.
- The profile display, as it is currently implemented, is neither scalable nor resizable. Attachments between the various widgets in the definition of its layout were not set up correctly. The code for the display should be restructured in a neat way and the display module should be used to simplify window management.
- A dragging facility that would allow the user to reposition blobs in the MPI communication display would be useful.
- A facility to automaticaly advance the step-by-step animation of communication events at fixed, user-defined time intervals (every two seconds, for example) was left unimplemented.
- Currently, the visualisation of point-to-point communication does not handle the case of receive operations with unspecified sources.

10 Conclusions

VISPAD is a very useful post-mortem visualisation tool. During this summer, the original implementation of VISPAD for CHIMP was adapted for use with MPI programs. In addition to reusing the existing INS library, the trace processing engine and the GUI, this required the implementation of an instrumented version of the MPI library. A lot of enhancements were also made to the INS library, the TPE and the GUI. A few more displays were also added to the tool. The existing displays were adapted to display MPI-specific information.

The tool itself is available for public use. It currently runs on sun4 workstations and can work with trace files produced by MPI programs running on sun4 workstation clusters. VISPAD has a lot of functionality that can be very useful to the programmer. We hope that MPI programmers in EPCC and elsewhere will be able to use the tool to analyse and tune their MPI programs.

A A guide to INS

To use the INS library for collecting events from a program, follow the steps given below.

- Include event.h and ins.h, in that order, in your program. Both are located in the directory /home/vispad/vispad-MPI/include.
- Use ins_init before calling any other INS function. It initialises a trace file with the given name, which can be any character string.
- Use ins_event to record an event in the trace file. The arguments are a character code identifying the event type, a short integer, a signed integer, a pointer to a buffer and the number of bytes in that buffer. Look in event.h to see what character codes are already used by INS. Do not use them.
- To record the beginning of a phase, use ins_beginPhase. The argument to ins_beginPhase is the name of the phase. To record the end of a phase, use ins_endPhase, which also takes the name of the phase as an argument.
- If you want to record a phase with some associated data, use ins_beginPhaseData and ins_endPhaseData, both of which take as arguments, a pointer to a buffer containing the data and the number of bytes in that buffer, in addition to the name of the phase.
- To associate data with a phase that has already been recorded, or that is going to be recorded in the future, use ins_addData. It takes a pointer to a buffer and the number of bytes in the buffer as arguments. The phase with which the data is to be associated can be specified as one of INS_PREVPHASE, INS_CURRPHASE or INS_NEXTPHASE.
- To classify the phases to be recorded as belonging to a particular class, use ins_beginClass. The arguments are the name of the class and an integer type of the class. All the phases recorded after calling ins_beginClass are considered to be of the specified class until a corresponding ins_endClass is called. ins_endClass also takes the name and the type of a class as arguments.
- Comments can be added to the trace file by using ins_annotate. The string given as the argument is inserted as a comment in the trace file.
- Delayed instrumentation can be achieved using ins_registerEvent and ins_completeEvent. Both are similar to ins_event and take all the arguments that ins_event takes. But ins_registerEvent only registers an event when it is called. The event is not written to the tracefile. ins_completeEvent can be used to change the contents of a registered event and to write the new event to the tracefile. The fields that are to be changed are specified using the flags CH_CODE, CH_HDL, CH_PIN, CH_STAMP, CH_DLEN and CH_DATA. These flags can be *or*ed so that any arbitrary combination of fields in the event can be changed.
- The instrumentation can be switched off by ins_off and switched back on by ins_on.
- By default, the INS library uses unbuffered instrumentation. Buffering can be switched on using ins_buffOn and switched off using ins_buffOff. The event buffers can be flushed explicitly using ins_buffFlush.
- Finally, use ins_exit to close the trace file.
- Then compile the program and link with libins.a located in the directory /home/vispad/vispad-MPI/lib.
- When the executable is run, it produces a trace file with the same name as the argument given to ins_init.

To write a program that reads the trace files produced by INS, follow the steps given below.

- Include event.h and ins.h, in that order, in your program. Both are located in the directory /home/vispad/vispad-MPI/include.

- Call ins_readInit in your program giving the name of the trace file as the argument.
- Use ins_readEvent to read the next event in the trace file. It requires a pointer to a valid *Event* data structure. Look in event.h to see the contents of *Event* structure. Don't forget to allocate a large enough buffer to the *data* member of the structure.
- Once an event has been read, its contents can be accessed directly. Look in event.h to find out the contents of an *Event* structure. The interpretation of the contents of an event depends on how the original program generated the trace file.
- Then compile the program and link with libins.a located in the directory /home/vispad/vispad-MPI/lib.
- A template program that reads in the trace files produced by INS can be found in /home/vispad/vispad-MPI/src/ins/src/ptrace.c. In addition to interpreting the events generated by INS, it handles events generated by the instrumented MPI library that uses INS.

B A guide to the instrumented MPI library

The instrumented MPI library uses the INS library to record important events in a MPI program. To use the instrumented INS library with your MPI program, follow the steps given below.

- No change to the source code is necessary.
- If your program is written in C, then link it with libinsmpi.a located in /home/vispad/vispad-MPI/lib. If you are using the EPCC's mpicc script, this can be done by specifying -pmpi, -L/home/vispad/vispad-MPI/lib and -linsmpi as additional flags. For example,

```
mpicc -mpiarch sun4 -pmpi -L/home/vispad/vispad-MPI/lib
                  -o xx xx.c -linsmpi
```

- If your program is in Fortran, then use the -pmpil option with /home/vispad/vispad-MPI/lib/libinsmpi.a specified directly.

```
mpif77 -mpiarch sun4 -pmpil /home/vispad/vispad-MPI/lib/libinsmpi.a
                 -o xx xx.F
```

- The instrumented MPI library is currently available only for sun4.
- The executable can then be run using the mpirun script available with EPCC's MPI distribution. If the executable is named xx and is run with 4 processes, it produces 4 trace files named xx_0.ins, xx_1.ins, xx_2.ins and xx_3.ins. This applies only for programs that are written in C. A Fortran program with 4 processes produces trace_0.ins, trace_1.ins, trace_2.ins and trace_3.ins instead.

C A guide to VISPAD-MPI

To produce trace files that can be used with VISPAD-MPI, refer to Appendix B. To use VISPAD-MPI with these trace files follow the steps given below.

- Make sure that /home/vispad/vispad-MPI/bin is your PATH. If not, add it to your path by doing

  ```
  export PATH=$PATH:/home/vispad/vispad-MPI/bin
  ```

  This can be done more permanently, by adding the above line to your ~/.brc file and then doing source ~/.brc. For those who are not using bash, contact the system support to know how this can be done for the shell you are using.

- Make sure that /home/vispad/vispad-MPI/bin/%B is your XBMLANGPATH. If not, add it to XBMLANGPATH by doing

  ```
  export XBMLANGPATH=$XBMLANGPATH:/home/vispad/vispad-MPI/bin/%B
  ```

  As described earlier, this can also be added to your .brc file.

- If the trace files are named xx_0.ins, xx_1.ins, xx_2.ins and xx_3.ins, then run

  ```
  vispad xx &
  ```

- Remember that currently vispad is available only for sun4.
- There is no user guide for vispad available yet. So contact simonc@epcc.ed.ac.uk or smith@epcc.ed.ac.uk or ah@dcs.ed.ac.uk or kesavans@cs.uoregon.edu or kxt@dcs.ed.ac.uk for more help.

References

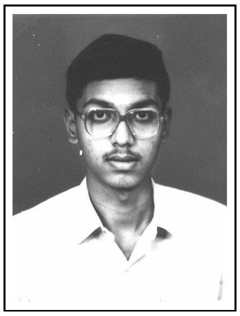
[Bruce *et al.*, 1993] R. Alasdair A. Bruce, Simon R. Chapple, Neil B. MacDonald, and Arthur S. Trew. CHIMP and PUL: Support for Portable Parallel Computing. Technical Report EPCC-TR93-07, Edinburgh Parallel Computing Centre, The University of Edinburgh, 1993.

[Bruce *et al.*, 1994] R. Alasdair A. Bruce, James G. Mills, and A. Gordon Smith. *CHIMP/MPI User Guide*. Edinburgh Parallel Computing Centre, The University of Edinburgh, 1994. EPCC-KTP-CHIMP-V2-USER 1.2.

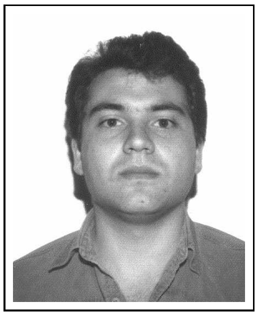
[Chapple, 1993] Simon R. Chapple. *Instrumentation Library Interface*. Edinburgh Parallel Computing Centre, The University of Edinburgh, 1993. EPCC-SSP-VISPAD-INS-FS 0.4.

[Mes, 1994] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, May 1994.

[Tomov and Wierenga, 1993] Neven Tomov and Klass-Jan Wierenga. Application Engineering Tools for CHIMP and PUL. Technical Report EPCC-SS93-15, Edinburgh Parallel Computing Centre, The University of Edinburgh, 1993.



Kesavan Shanmugam is a graduate student at the Department of Computer and Information Science, University of Oregon, USA. His interests include parallel programming, performance analysis of parallel programs and compilers for parallel machines. He is currently a member of the SAGE/pC++ Development Team at the University of Oregon and is responsible for the development of a performance extrapolation tool called XtraP for the parallel C++ language pC++. *e-mail: kesavans@cs.uoregon.edu*.



Konstantinos Tourlas is an ERASMUS undergraduate student at the Department of Computer Science, the University of Edinburgh, UK. His interests include parallel architectures, computer graphics and digital signal processing.

This project was supervised by Simon R. Chapple (EPCC), A. Gordon Smith (EPCC) and Anna Hondroudakis (Department of Computer Science, University of Edinburgh).