Paraver

Version 3.1

Parallel Program Visualization and Analysis tool

REFERENCE MANUAL

October 2001

CEPBA

UPC

Contents

List of Figures

Chapter 1

Introduction

Paraver is a flexible parallel program visualization and analysis tool based on an easy-to-use Motif GUI. Paraver was developed responding to the basic need of having a qualitative global perception of the application behaviour by visual inspection and then to be able to focus on the detailed quantitative analysis of the problems. Paraver provides a large amount of information on the behaviour of an application. This information directly improves the decisions on whether and where to invert the programming effort to optimize an application. The result is a reduction of the development time as well as the minimization of the hardware resources required for it.

Some Paraver features are the support for :

- Detailed quantitative analysis of program performance
- Concurrent comparative analysis of multiple traces
- Fast analysis of very large traces
- Mixed support for message passing and shared memory (networks of SMPs)
- Easy personalization of the semantics of the visualized information

One of the main features of Paraver is the flexibility to represent traces coming from different environments. Traces are composed of state transitions, events and communications with an associated timestamp. These three elements can be used to build traces that capture the behaviour along time of very different kinds of systems. The Paraver distribution includes, either in its own distribution or as an additional package, the following instrumentation tools:

1. Sequential application tracing: it is included in the Paraver distribution. It can be used to trace the value of certain variables, procedure invocations, ... in a sequential program.
2. Parallel application tracing: a set of modules are optionally available to capture the activity of parallel applications using shared–memory (OpenMP directives), message–passing (MPI library) paradigms, or a combination of them.
3. System activity tracing in a multiprogrammed environment: an application to trace processor allocations and migrations is optionally available in the Paraver distribution.
4. System activity tracing in a multiprogrammed environment: an application to trace processor allocations and migrations is optionally available in the Paraver distribution.

This document includes a detailed description of the Paraver programming model and trace format. This allows Paraver users to develop their own tracing facilities according to their own interests and requirements. The possibilities offered by the visualization, semantic and quantitative analyzer modules are powerful enough allowing users to analyze and understand the behaviour

of the traced system. Paraver also allows the customization of some of its parts as well as the plug–in of new functionalities.

So expressive power, flexibility and the capability of efficiently handling large traces are key features addressed in the design of Paraver. The clear and modular structure of Paraver plays a significant role towards achieving these targets. Let us briefly describe the key design philosophy behind these points.

Views

Paraver offers a minimal set of views on a trace. The philosophy behind the design was that different types of views should be supported if they provide qualitatively different analysis types of information. Frequently, visualization tools tend to offer many different views of the parallel program behaviour. Nevertheless, it is often the case that only a few of them are actually used by users. The other views are too complex, too specific or not adapted to the user needs.

Paraver differentiates three types of views :

- Graphic view : to represent the behaviour of the application along time in a way that easily conveys to the user a general understanding of the application behaviour. It should also support detailed analysis by the user such as pattern identification, causality relationships, ...
- Textual view : to provide the utmost detail of the information displayed.
- Analysis view: to provide quantitative data.

The **Graphic View** is flexible enough to visually represent a large amount of information and to be the reference for the quantitative analysis. The Paraver view consists of a time diagram with one line for each represented object. The types of objects displayed by Paraver are closely related to the parallel programming model concepts and to the execution resources (processors). In the first group, the objects considered are : *application* (Ptask in Paraver terminology), *task* and *thread*. Although Paraver is normally used to visualize a single application, it can display the concurrent execution of several applications, each of them consisting of several tasks with multiple threads.

The information in the graphics view consists of three elements : a time dependent value for each represented object, flags that correspond to puntual events within a represented object, and communication lines that relate the displayed objects. The visualization control module determines how each of these elements are displayed. The essential choices are :

- Time dependent value : displayed as a function of time or encoded in color. Furthermore, time and magnitude scale can be changed to zoom the visualization.
- Flags : displayed or not and color.
- Communication : displayed or not.

The visualization module blindly represents the values and events passed to it, without assigning to them any pre-conceived semantics. This plays a key role in the flexibility of the tool. The semantics of the displayed information (activity of a thread, cache misses, sequence of functions called,...) lies in the mind of the user. Paraver specifies a trace format but no actual semantics for the encoded values. What it offers is a set of building blocks (semantic module) to process the trace before the visualization process. Depending on how you generate the trace and combine the building blocks, you can get a huge number of different semantic magnitudes.

Expressive power

The separation between the visualization module which controls how to display data and the semantic module which determines the value visualized offers a flexibility and expressive above than frequently encountered in other tools.

Paraver semantic module is structured as a hierarchy of functions which are composed to compute the value passed to the visualization module. Each level of function corresponds to the hierarchical structure of the process model on which Paraver relies. For example: when displaying threads, a thread function computes from the records that describe the thread activity, the value to be passed for visualization; when displaying tasks, the thread function is applied to all the threads of the task and a task function is used to reduce those values to the one which represents the task; when displaying processors, a processor function is applied to all the threads that correspond to tasks allocated to that processor.

Many visualization tools include a filtering module to reduce the amount of displayed information. In Paraver, the filtering module is in front of the semantic one. The result is added flexibility in the generation of the value returned by the semantic module.

Combining very simple semantic functions (sum, sign, trace_value_as_is,....), at each level, a tremendous expressive power results. Besides the typical processor time diagram, it is for example possible to display:

- The global parallelism profile of the application.
- The total per CPU consumption when several tasks share a node.
- Average ready queue length of ready tasks when several tasks share a node.
- The instantaneous communication load geometrically averaged over a given time.
- The evolution of the value of a selected variable, ...

The default filtering and semantic function setting of the tool results in the same type of functionality of many other visualization tools. A much higher semantic potential can be obtained with limited training.

Direct and Derived metrics

Paraver offers a very flexible mechanism to compute and display a large number of performance indices and derived metrics from the trace.

There is a first level of semantic functions which obtain the values in function of time directly from the records in the trace. A second level of semantic functions can be obtained by combining (add, divide, ...) the functions of time computed by the first level.

For example, starting with a window that looks at the cycles counter and another window that looks at the instructions counter, it is possible to apply the second level of semantic functions in order to obtain a derived metric like Instructions per Cycle by dividing those two windows. Although the trace doesn't contain any counter about IPC, it could be computed from the cycles counter and instructions counter.

Quantitative analysis

Global qualitative display of application behaviour is not sufficient to draw conclusions on where the problems are or how to improve the code. Detailed numbers are needed to sustain what otherwise are subjective impressions.

The quantitative analysis module of Paraver offers the possibility to obtain information on the user selected section of the visualized application and includes issues such as being able to measure times, count events, compute the average value of the displayed magnitude,...

The quantitative analysis functions are also applied after the semantic module in the same way as the visualization module. Again here, very simple functions (average, count, ...) at this level

combined with the power of the semantic module result in a large variety of supported measures. Some examples are:

- Precise time between two events (even if they are very distant)
- Number of messages sent between time X and Y
- Average CPU utilization between time X and Y
- Number of events of type V on processor W between time X and Y
- Average CPU time between two communications
- Average execution time of the different functions

Multiple traces

In order to support comparative analyses, simultaneous visualization of several traces is needed. Paraver can concurrently display multiple traces, making possible to use the same scales and synchronized animation in several windows.

This multi-trace feature supports detailed comparisons between:

- Two versions of a code
- Behavior on two machines
- Difference between two runs
- Influence of problem size

Comparisons which otherwise are very subjective or cumbersome.

Customization

Developing a tool which fulfills the needs of every user is rather impossible. Initially Paraver aimed at supporting the projects carried out at **CEPBA** as part of our research and service activities. One objective of the design was to provide some support for expert users having new needs and willing to extend the functionalities of Paraver. For this purpose, Paraver is distributed with the possibility of personalizing the two modules that provide the expressive and analysis power.

A procedural interface is provided in such a way that a user can, with a limited effort, link into its Paraver version the actual functionality needed. Taking into account the built in semantic functions and analysis functions and their relation to the hierarchical process model and the possibility of combining them in a totally orthogonal way at each level, a user can obtain a large number of new semantics by developing very simple modules.

Another aspect where the users may have personal preferences is in setting color tables. More important is the possibility to specify the textual values associated with the values encoded in the trace. All this is achieved through a simple but powerful **configuration file**.

Configuration files are simple, but very useful files which lets to the user customize his/her environment, save/restore a session, ... Also, we can change some default options and redefine the environment.

It is important to know and use these files because they make easier the day work with Paraver.

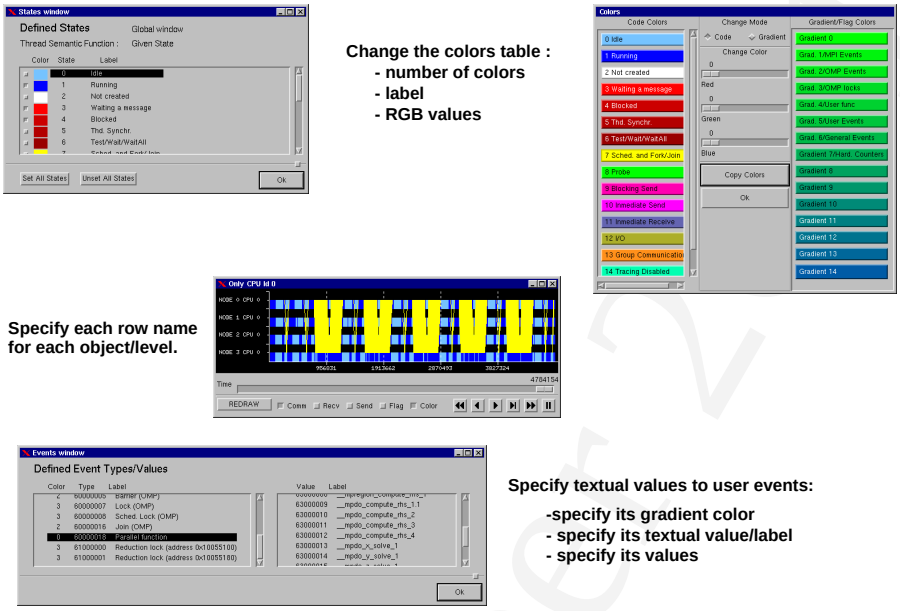


Figure 1.1: Paraver Configuration File

Large traces

A requirement for Paraver was that the whole operation of the tool has to be very fast in order to make it usable and capable to maintain the developer interest. Handling traces in the range of tenths to hundreds of MB is an important objective of Paraver.

Easy window dimensioning, forward and backward animation, zooming are supported. Several windows with different scales can be displayed simultaneously supporting global qualitative references and detailed analysis. Even on very large traces, the quantitative analysis can be carried out with great precision because the starting and end points of the analysis can be selected on different windows.

Chapter 2

Installation

Package description

The package is distributed in compressed tar format (file Paraver-XXX.tar.gz, where **XXX** is the plattform). To unpack it, execute from the desired target directory the following command line :

```
gunzip -c Paraver-XXX.tar.gz | tar -xvf -
```

The unpacking process will create four directories on the current directory (see table 2.1).

bin	Contains the binary files of Paraver distribution. This directory contains the Paraver binary, paraver file, the domapfile binary (to convert traces from ascii to binary format) and the Paraver license daemon (paraverd).
etc	Contains files related to : - the Paraver licenses. - the script to launch the Paraver daemon (paraverd.sh). - the log file for Paraver daemon (paraverd.log). - an example of a Paraver configuration file (pcf_example.pcf).

Table 2.1: Package description. Installation.

Installation script

In Paraver home unpacked base directory, there is a script named **install.sh** which creates the script to launch the Paraver license daemon.

Use it by typing :

```
install.sh  <paraver_home_directory> [system]
```

This will create the **bin/paraver.sh** shell script (see the following subsection to see how could be used).

The **system** option will install the script to automatically launch the Paraver daemon during the system boot start up (you must have root permisions to install it).

Installation script of IBM and SGI distributions

The IBM and SGI distributions contains the binaries for the two supported ABIs. On the installation script arguments, the user can specify which executable version will use.

Select it by typing :

```
install.sh  <paraver_home_directory> [system] [32|64]
```

The script creates the **bin/paraver.sh** and creates the needed softlinks to the selected executable format in *src* directory. By default, the 64-bit binaries are installed.

- the **32** option will install the package to use the new 32-bit.
- the **64** option will install the package to use the 64-bit.

Installing the license file

Before starting up the Paraver license daemon the license file has to be copied in *etc* directory as license.dat. The Paraver license daemon searches for $PARAVER_HOME*/etc/license.dat* license file. This environment variable is set up within the script.

If Paraver software packages are installed in the same directory only there could be one license.dat file for all the packages. This file must contain all the licenses, so license contents have to be copied within that file.

The license file will be sent you via e-mail[1].

Installing the Paraver license daemon (paraverd)

The Paraver license daemon can be found in *bin* directory. It is resposible for the licenses management.

To start the daemon there is a script named **paraverd.sh** in *etc* directory which will start/stop the daemon. If the user ran the installation script using the **system** option, this script has been installed in boot start up directories.

```
#! /sbin/sh

# This variable must point to the
# paraver home: example /usr/local/soft/paraver
PARAVER_HOME_DIRECTORY = <paraver_home_directory>

case $1 in

'start')

  export PARAVER_HOME=$PARAVER_HOME_DIRECTORY

  echo "Starting Paraver license Daemon ..."
  $PARAVER_HOME_DIRECTORY/bin/paraverd &
  ;;

'stop')

  /sbin/killall 15 paraverd
  ;;

*)
  echo "usage: /etc/init.d/paraverd {start|stop}"
  ;;
esac
```

[1] To obtain a license please go to the http://www.cepba.upc.es/paraver/ or contact to cepbatools@cepba.upc.es.

If you would start up manually the Paraver license daemon write :

```
paraverd.sh  start
```

To stop it write :

```
paraverd.sh  stop
```

the script stops the Paraver daemon. Paraver will no longer run until license daemon will be launched another time.

If Paraver daemon is not started, check the Paraver daemon log file (etc/paraverd.log) for further information.

Launching Paraver

To launch Paraver, the environment variable PARAVER_HOME have to be set up with the paraver home directory.

```
setenv PARAVER_HOME   <paraver_home_directory>
export PARAVER_HOME = "<paraver_home_directory>"

Example :    setenv PARAVER_HOME /usr/local/soft/paraver
```

Once this variable have been set up, start by typing :

```
$PARAVER_HOME/bin/paraver [trace_file_name.prv]
```

As arguments, the user can specify a trace file that will be loaded and one window configuration file.

Chapter 3

Where does Paraver come from?

Perhaps the greatest challenges to develop efficient parallel programs are successful debugging and performance tuning. In 1991, an environment was developed at CEPBA (European Center for Parallelism in Barcelona) to reduce costs in parallel program development and tuning: the DiP environment. DiP is based on a trace driven simulator (Dimemas) and a tool to visualize and analysis those traces (Paraver). It aims at predicting the performance of a message passing application on machines not readily available while doing most of the development on a workstation.

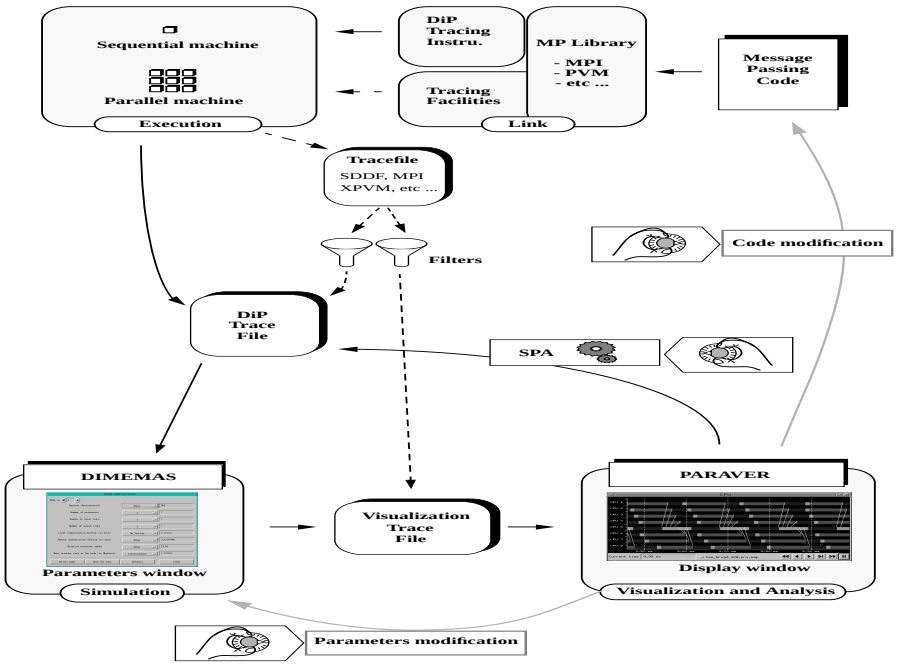


Figure 3.1: The DiP Environment Structure

The global structure of the DiP environment is described in Figure 3.1. Three tools constitute the environment core: the DiP instrumented communication library; Dimemas, a distributed memory machine simulator; and Paraver, a visualization and analysis tool. Other utilities where developed to support interaction with public domain or commercial products. Translators (filters) from several trace formats (SDDF , PICL) to the DiP trace format are some of these utilities. In this way, Dimemas and Paraver can be used as powerful post-processing analysis and prediction tools to many other environments. The design of the tools is modular, with the objective of enabling the study of other parallel program performance factors such as locality and cache

utilization, file systems,...

The DiP basic structure is similar to typical postmortem analysis tools based on traces, but there are some specific elements and design criteria that differentiate this approach from others. In the design two objectives were fixed:

- to emphasize a clear division between parts of the tool set where each module has its own functionalities
- to offer flexible mechanisms to combine those modules leading to the construction of very powerful analysis and prediction functionalities.

From these simple concepts, the environment enables complex and large application analysis.

Arcs in figure 3.1 describe the possibilities of combination of the different tools. A significant part of the power and usefulness of the environment comes from the flexibility in sequencing the individual tools execution.

The typical DiP path used to carry out studies on the influence of architectural parameters is shown here:

- The application is run with the DiP instrumented library, on a single workstation or in a parallel system.
- The instrumented library generates a tracefile with relative times, which characterizes the application.
- From this tracefile, Dimemas would rebuild the behaviour of the application on a parameterized target machine.
- A new tracefile with absolute times is generated to be visualized with Paraver.
- Based on visualization and quantitative measures given by Paraver, the user can modify the source code.

Simulation and Visualization phases can be repeated several times in order to predict and analyze the performance on different target architectures. It is an off-line process, where the user has not to compile nor execute his source code again, just modifying input parameters (architectural) of Dimemas. The quantitative statistics that can be obtained with Paraver, provide important information to guide the tuning of the program and to identify the most convenient optimization.

Dimemas and Paraver tool can be found at URL **http://www.cepba.upc.es/tools**. Dimemas and Paraver are available for IBM AIX, SGI IRIX, Tru64 AIX, HP-UX, Solaris and Linux. Paraver and Dimemas e-mail support is : **cepbatools@cepba.upc.es**

Chapter 4

Paraver Object Model

As described in the introduction, Paraver functionality is tightly coupled with the hierarchical object model targeted by the DiP environment. Paraver works with two ortogonal object models:

- The **process model** is composed by the objects that correspond to the three levels of the most frequently programming models: application, process and thread objects.
- The **resource model** represents the physical resources where the different threads are finally executed. The resource objects are tightly connected with the cluster of SMPs where applications has been executed: processor and node.

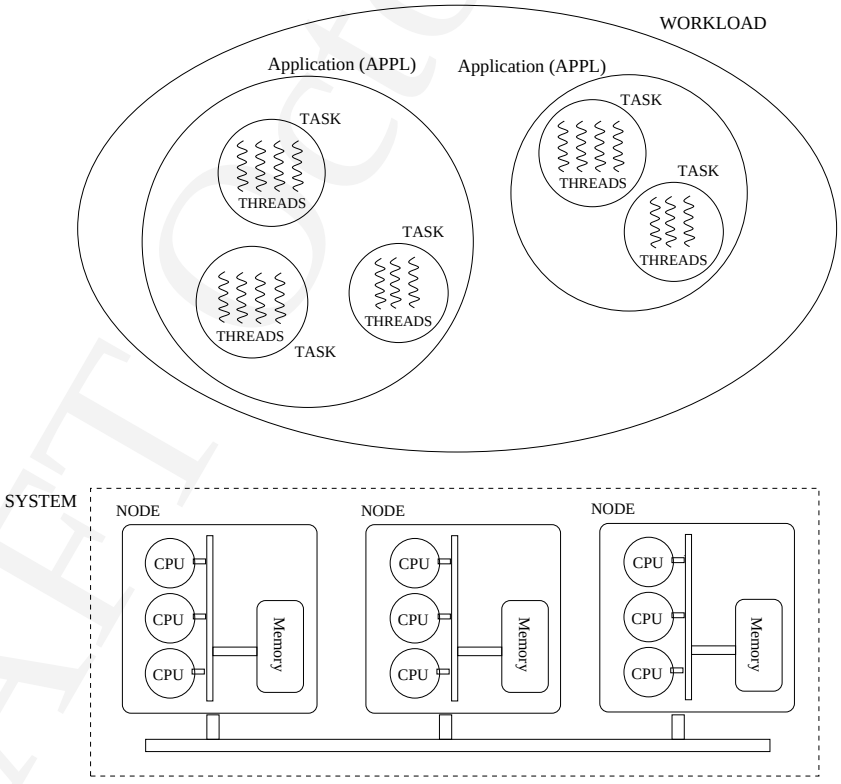


Figure 4.1: Paraver object model

To exploit all the object model levels, the tools that generates trace files for Paraver must be capable to get the process and resource information. Both shared and distributed memory

applications could be mapped on the Paraver object model.

Object Model flexibility

The Paraver **Process and Resource models** define the structure of two ortogonal type of objects for which performance indices can be computed and displayed. The actual names used for those objects derive from the standard parallel programing terminology. Nevertheless. Paraver does not assume any semantics beyond the hierarchical structure of the objects. It is possible for any user to implement instrumentation or simulation tools that map other concepts onto the thread, cpu, task, ... names. For example, the resources model could be used to represent functional units and threads could represent instruction flow. Features such as the behaviour of cluster arquithectures could be easely displayed.

4.1 Paraver Process Model

This process model is a superset of the most frequently used programming models. On a Paraver window, the type of process model object to be represented can be selected among:

- Set of applications (WORKLOAD)
- Application (APPL)
- TASK
- THREAD

A **parallel application** (APPL level) is composed by a set of sequential or parallel **processes** (named as TASK in paraver process model hierarchy). The parallel processes are composed by more than one thread while the sequential processes are mapped into one thread. The top of the process model hierarchy is the WORKLOAD level representing a set of different applications running on the same resources.

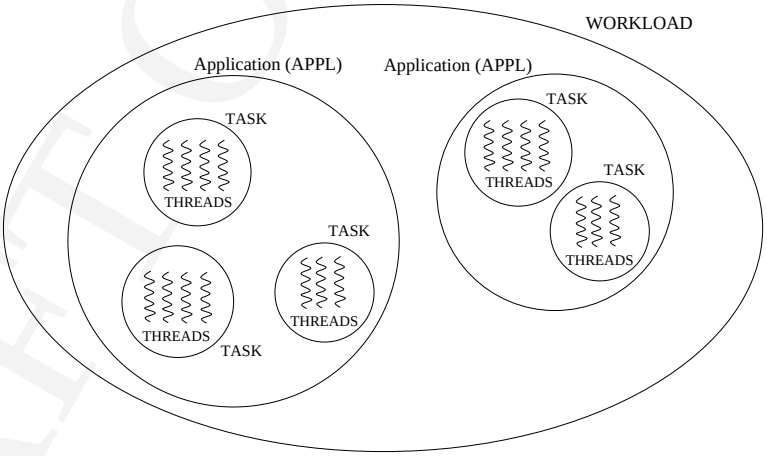


Figure 4.2: Paraver process model

This model is very flexible, and can be easily mapped to the models supported by actual communication or multi-threaded libraries.

A thread type window will display one line for each selected thread. Paraver supports several applications concurrently running on the same system thus, on an application type window, one line will be displayed for each application.

In the three-level process model an application can have one or several tasks, and each task can have one or several threads. Tasks do not share address space, thus communication between them is only done through message passing. The different threads within a task are executed within the address space defined by the task. Threads within a task can thus communicate and synchronize through shared memory.

The value represented on a line of a given type is computed from the records in the trace by the hierarchical composition of functions corresponding to each one of the levels in the model (see figure 4.3). Top levels semantic functions return a value based on the semantic values returned by the bottom levels. For example, the semantic value returned for TASK level is computed based on the semantic values of the threads of the task.

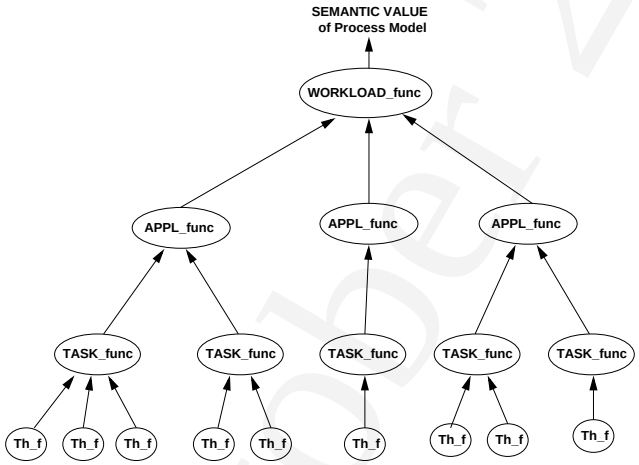


Figure 4.3: Hierarchical composition of levels

Shared and distributed memory programming models can be mapped onto the *Paraver process model*. For example, we can map the MPI programming model, the PVM programming model, the OpenMP programming model, combined MPI and OpenMP programming models, ...

Example 1: Mapping the MPI programming model.

The mapping of the **MPI programming model** onto the *Paraver process model* can be done as follows :

- each MPI process is a Paraver TASK with one Paraver THREAD
- the whole MPI application is the Paraver APPL (application object), grouping all MPI processes.

This mapping lets to have in a tracefile the execution of different MPI applications at the same time.

The **PVM programming model** can be mapped like the **MPI programming model**. Each PVM process will be a Paraver TASK.

Example 2: Mapping the combined OpenMP+MPI programming model.

The combined OpenMP+MPI programming model could be seen as a MPI application where each MPI process is composed by a set of threads which work in parallel. The mapping that could be done is a combined mapping between the two programming models :

- each OpenMP thread is mapped on one Paraver THREAD
- each MPI process composed of multiple OpenMP threads is a Paraver TASK

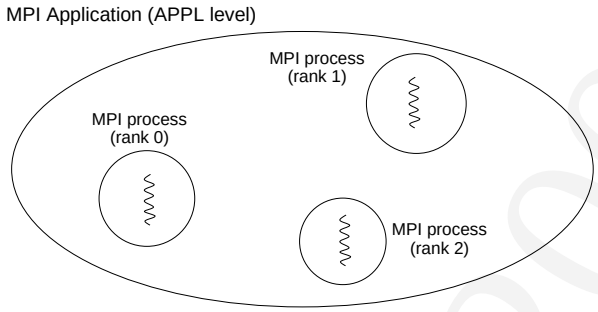


Figure 4.4: Mapping of the MPI programming model

- the whole OpenMP+MPI application is the Paraver APPL (application object).

As in the previous example, we can have a trace with different applications.

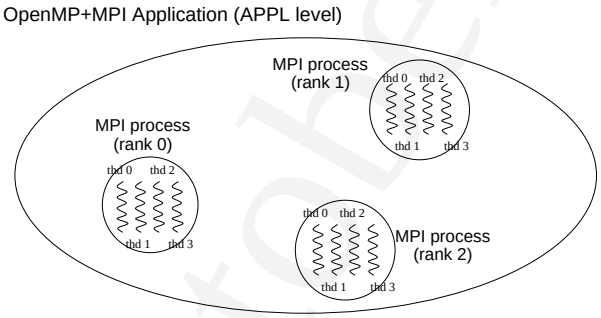


Figure 4.5: Mapping of the OpenMP+MPI programming model

4.2 Paraver Resouce Model

The resource model represents the resources where the applications are executed. On a Paraver window, the type of resource object to be represented can be selected among:

- Cluster of nodes (SYSTEM)
- NODE (set of processors)
- Processor (CPU)

The processors are the resources where the threads are executed. Processors are grouped in nodes. A task is mapped into a node and thus all its threads share their processors in that node.

The mapping is not necessarily one to one, so it is possible to have several tasks from a single application on a given node. Tasks from different applications can also be mapped into the same node.

Different resource models can be mapped onto the *Paraver resource model*. For example, a uniprocessor machine could be represented by a single node composed by one processor. A single shared memory multiprocessor with four processors could be represented as one node composed by four processors. A distributed shared memory could be represented by different nodes could be mapped by more than mode composed by different processors. At the top, the **system** level has been added in order to represent a set of SMP clusters.

For a processor type window, the valued displayed is the value returned by the thread funtion that is executing in that moment.

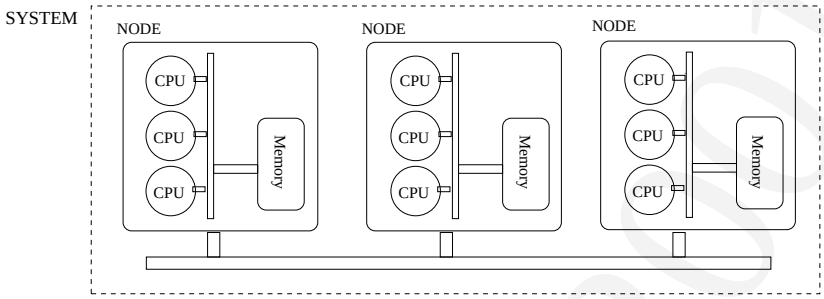


Figure 4.6: Paraver resource model

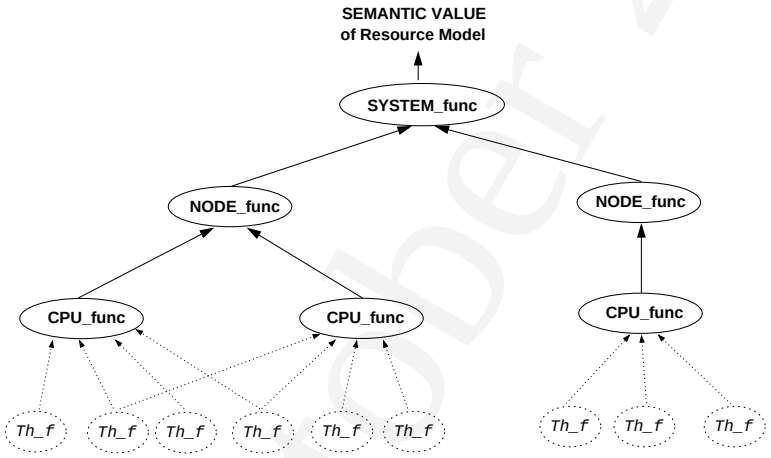


Figure 4.7: Hierarchical composition of levels

The semantic function for one node is computed based on the semantic values of the processors of that node. Typical combinations are addition, average, maximum, ... New processor semantic functions such as Active Thread or Active Thread Value have been developed that support new types of analyses. These functions return the **identifier** or **value** respectively of the thread running on the processor at the moment.

Chapter 5

Paraver Internal Structure

Paraver structure consists of three levels of modules (figure 5.1). First, working onto the trace file there is the Filter Module. This module gives to the next level a partial view of the trace file. Second, there is the Semantic Module which receives the trace file filtered by the previous module and interprets it. The Semantic Module transforms the record traces to time dependent values which will be passed to the Representation Module. The Semantic Module is the most important level because it extracts and gives sense to the record values in the trace file. The trace file has a lot of information that could be extracted and this module selects what will be extracted, this information is called, the **semantics** of the trace file.

Finally, there is the Representation Module. It receives the time dependent values computed by the semantic module and displays it in different ways. The Representation Module drives thus the whole process and offers a graphical display of the trace file.

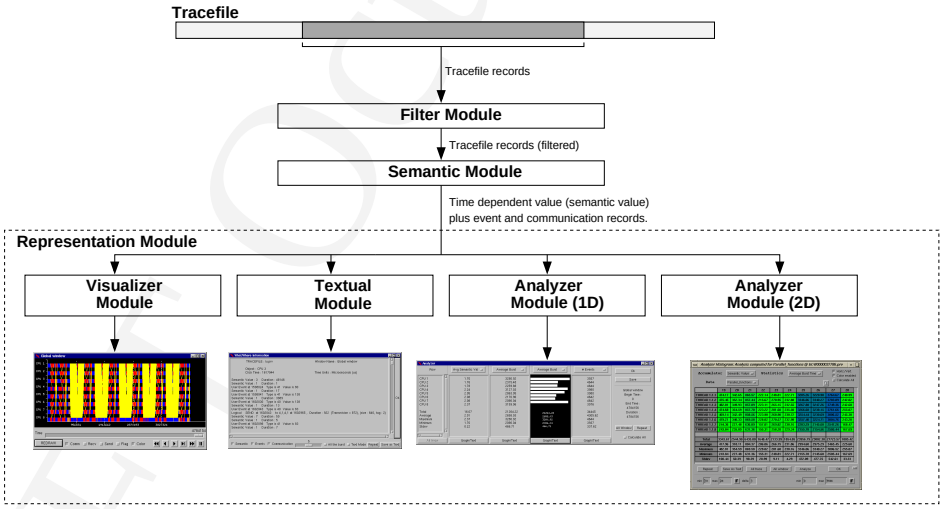


Figure 5.1: Paraver Internal Structure

5.1 Representation Module

The representation module is composed by three modules, the Visualization, the Textual and the Analyzer modules. These modules are the actual driving engine of the system. The mode of operation can be seen as demand driven. When a new value has to be displayed (for example, going to a time stepping through the animation or computing an statistic) the sequence of values or events needed is requested from the next module.

5.1.1 Visualization module

The Visualization module $\boxed{v}$ is responsible of values and events displayed on the screen. It is aware of concepts such as window sizes, display scales (time and magnitude), type of display (color or level), type of object (application, node, task, thread or processor), number of rows to display, measurement of time (even between different windows for precise measurements of long intervals) and so on. This module concerns with the creation of new windows either by a direct user command or through the zooming and cloning facilities. It takes care of the trace animation which can be displayed both forwards and backwards. It also offers the possibility to go to a position into the trace file to be displayed by specifying: an absolute or relative time, an user event type or value, a message tag or size.

5.1.2 Analyzer module

Qualitative behaviour display is not sufficient to draw conclusions on where the problems are or how to improve the code. Detailed numbers are needed to sustain that otherwise are subjective impressions.

The Analyzer module $\boxed{A}$ computes statistics on overall or even just a user selected part of the trace file.

There are some predefined statistics such as the average semantic value, number of events, number of communications (sends/receives), ... The output of the analysis functions is presented in a Paraver window either as table or as histogram.

5.2 Semantic Module

The Semantic module $\boxed{\int}$ computes the values to be transferred to the representation module. The computation of the values is based on one or several records as returned by the filter module described below. The $\int$ module is the only one that looks for the actual coding of the trace state records and for the process model semantics. As mentioned before, each window represents objects of a given type: node, processor, application, task or thread. The semantic module uses, to generate values that will be given to the visualization or analysis module, the composition of one function for each object level submitted to the one selected for the window. Examples of functions:

thread level can return:

- Useful: It takes the state trace and returns if the state value is Running (state value 1) or not (state value 0).
- State As Is: returns the state code
- Last Evt Val: This function takes the event trace and returns its event value.
- ...

task level can apply the sum or the boolean function to the values returned by the thread level function for all threads within each task.

application level can apply the sum or the boolean function to the values returned by its task members.

Each of these elemental functions are really simple but an important part of the power of Paraver lies in this module and how the user combines the different functions to generate the type of value that is relevant for his analysis. In case the system provided insuficient functions, the user can link to Paraver its own function.

5.3 Filter Module

The Filter module $\boxed{f}$ offers to the previous ones a partial view of the trace file. The user can specify that only certain events are passed to the Semantic module, like only a specific type of user event, messages of a given tag or destination, and so on. This is specified by a Filter module control window that offers a nice interface for a very flexible set of filtering options. State records are always passed to the Semantic module without filtering.

Chapter 6

Main View

When paraver is launched, it raises two windows; the Main Menu window appears on the left top corner of your screen and Global Controller window appears on the right bottom corner.

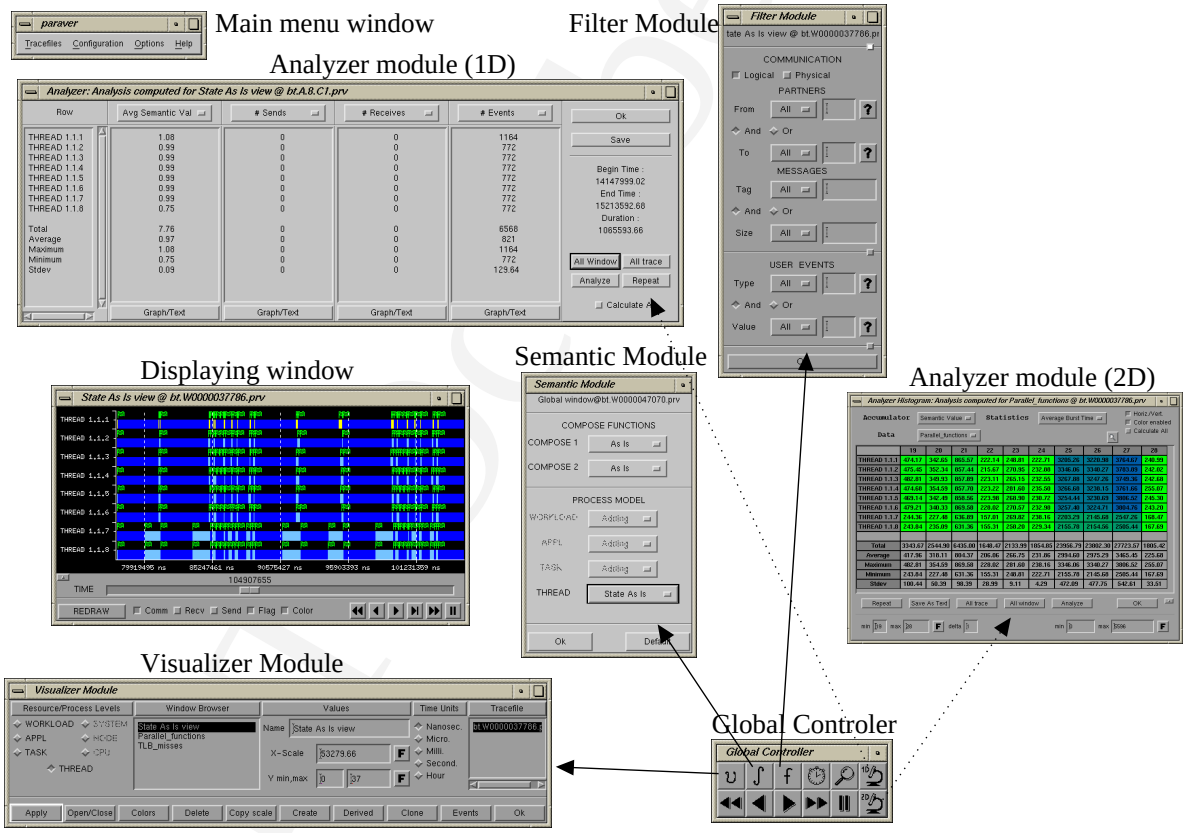


Figure 6.1: Paraver Screenshot

Once a tracefile has been loaded, the different Paraver modules can be accessed through these windows (see figure 6.1). The next chapters will describe them :

- The **Main Menu** is composed by a set of menu options which lets us to load/unload trace files, load/save window configurations, change some globals options, ... The **Global Controller** lets to work with the different modules. It is composed by a set of buttons which implement the Paraver functionalities.

- the **Filter module** is described in chapter 7 on page 33. This module lets to filter some communications and/or some events that won't be passed to the next modules.
- the **Semantic module** is described in chapter 8 on page 37. It works onto the Filter module and computes the values that will be transfered to the representation levels.
- the **Displaying windows** are described in chapter 9. They are the graphic representation of the values passed by the Semantic module.
- finally, the chapter 10 on page 75 describes the **Visualizer**, the **Textual** and the **Analyzer** modules. The **Visualizer module** manages the displaying windows. The **Textual Module** give us a textual representation of the trace file. Finally, the **Analyzer modules** lets us to get a very detailed qualitative analysis by properly selecting the FILTER and SEMANTIC modules. information.

6.1 Main Menu window

The Main Menu window is composed by a set of menu options divided in three mainly pop up menus (figure 6.2). The first menu manages the traces (load, unload, get trace information, ...), the second one is used to load/save window configurations and third one allows to change the Paraver global options (drawing speed, system colors, ...). Next sections will explain them in detail.

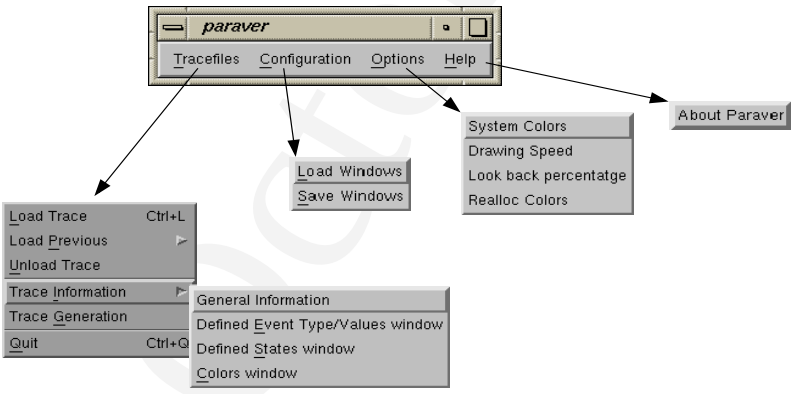


Figure 6.2: Main Menu

6.1.1 Tracefiles menu

The **Tracefiles** menu allows to manage traces.

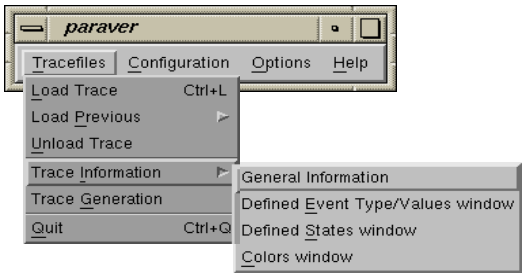


Figure 6.3: Tracefiles menu

Load Tracefile option (Ctrl+L)

The **Load Tracefile** option is used to browse the directories and load a trace file. Select a trace name in the file selection box and click the OK button. Then, Paraver begins to load it. **Gzipped traces are also accepted,** paraver will uncompress them during the loading.

While loading a trace file, Paraver raises the computing window showing the loaded percentage. Trace loading could be stopped by pressing the **Stop Loading** button to work only with the current trace percentatge that has been loaded.

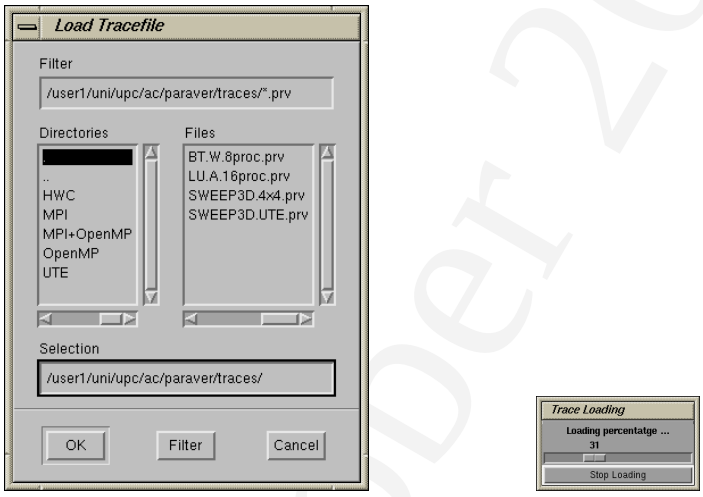


Figure 6.4: Loading a tracefile

Binary traces are loaded more quickly than an ASCII, so for big ASCII traces we strongly recommend to convert to binary format to speed up its loading (see the *prv2map* tool on page 111).

Load Previous

The **Load Previous** option contains a list of last previous loaded tracefiles. It allows quick access to previous loaded traces.

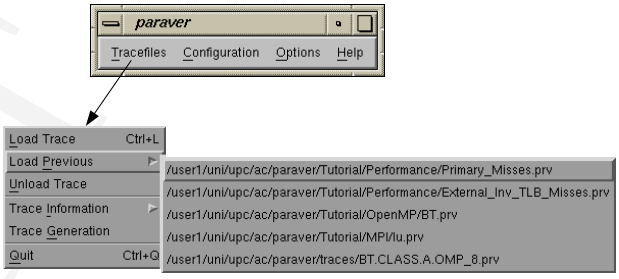


Figure 6.5: Loading a tracefile

A small hidded **.paraverdb** file is created on the user home directory. Delete it if you want to remove all list items.

Unload option

Paraver lets to unload a tracefile that won't be used. When the user selects this option Paraver raises a window where there are all the trace files that have been loaded; select the tracefile that will be unloaded and click the **Unload** button to unload it.

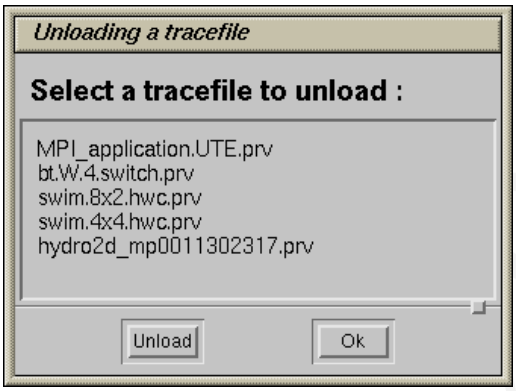


Figure 6.6: Unloading Tracefile

Paraver makes an un-mapping process, and destroys the displaying windows associated to that tracefile. The previous one in the list will be considered like the **current** tracefile.

Trace Information option

The Trace Information menu options let us to access the defined information for the current trace file: number of applications, number of tasks, number of threads, number of nodes and processors, defined user event labels, defined states, ...

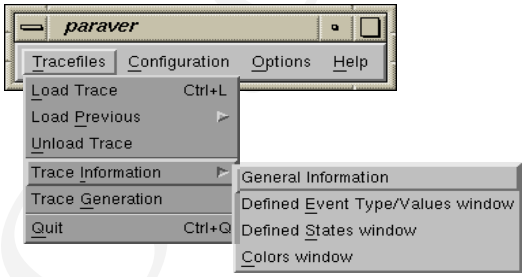


Figure 6.7: Tracefiles menu

In this chapter only the **General information** option is described, the rest will be described in next chapters. The **Define Event/Values window** raises a window that shows a list of the defined event types and all their defined values for the current tracfile, it will be desribed in detail in chapters 7 and 10. The **Define States window** raises a window that shows a list of defined states labels, it will be desribed in detail in section 8.3.1 on page 48. Finally, the **Colors window** option raises a window that shows the defined colors, see chapter 10 on page 80.

The General Information menu option raises a window containning the information about the defined objects (number of applications, tasks, threads, ...) of the current trace.

The window shows :

- the trace file name
- its duration
- the date when it was generated
- the Process model information where could be seen the total number of applications, tasks and threads and they hierarchy.

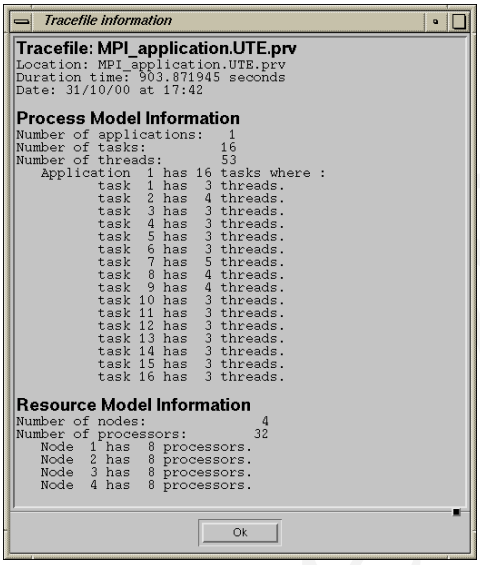


Figure 6.8: General Information window

- the Resource model information where could be seen the total number of nodes and processors for each node. If no resource information is available, the message **No resource information is available** is showed.

Quit option (Ctrl+Q)

To close Paraver.

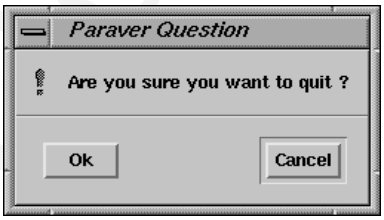


Figure 6.9: Quitting Paraver

Paraver ask for confirmation before exiting.

6.1.2 Configuration menu

The second menu is called the Configuration menu. It lets to *Load* and *Save* the window configuration files.

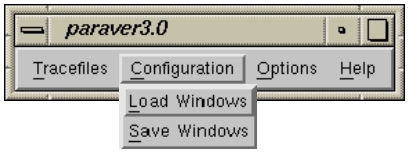


Figure 6.10: Configuration menu

Since **Displaying windows** aren't explained until chapter 9 on page 63, the window configuration files and these menu options will be explained in chapter 9.

6.1.3 Options menu

The **Options** menu allows to change Paraver global options.

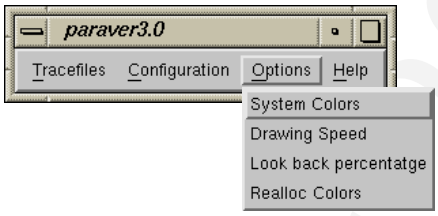


Figure 6.11: Options menu

Changing System Colors. *System Colors* option.

Paraver lets to change some colors as background and communication lines. By default, Paraver draws the Logical communications lines and icons in yellow and draws the Physical communications in red, but these colors can be changed. The change of colors will affect to all the displaying windows.

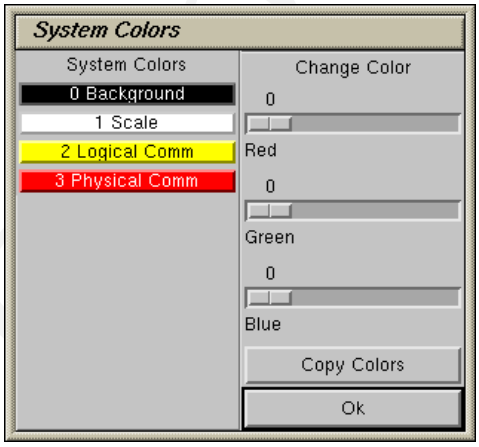


Figure 6.12: System Colors window

Change colors by clicking on the desired one (background, scale,...) and moving the scale bars of the basics (R, G, B). To recover a color just double click on the desired color and the default color will appear. To copy a color click the source color press the **Copy Colors** button and click the target color.

The **Background** (system color 0) is the background color on a displaying window and the **Scale** (system color 1) is the color of text and window axis (see figure 6.12 for details).

Changing the scrolling speed. *Speed* option.

When we are working with Play and Play back buttons (see chapter 9 on page **??**), sometimes the speed of the machine where Paraver is being executed is faster than the speed of the X Server which is processing scrolling events. As a result, the application overloads the X Server and it does not respond to any event (for example, if user tries to click the Pause button to stop the animation, it will not respond until it has processed all previous drawing events).

To solve it, we have added an option (Speed option menu) to select the number of microseconds between two drawing events that allows to define a user defined value. The *Speed* option will raise a window to select it (see figure 6.13).

Figure 6.13: Paraver scrolling Speed

By default, the value is set to 0 microseconds. To force a slow drawing, select a value greater than zero. **This value is theoretical and it depends on the X-terminal speed and the machine where it is executed.**

Look back percentatge option

Paraver tracefile is composed by a sequence of records ordered by time. The Semantic module takes these records and computes a value known as *semantic value* which gives sense to the tracefile.

To compute these values we have to look back for the correct value and in a non homogeneous big tracefile, those searches could be so slow (we are talking about traces of several megabytes). This look back behaviour only is used when computing the first semantic value, the rest values when going forward the tracefile will be computed with the records that we are going to find. If the number of records to look back isn't enough small the first semantic value could be incorrect.

Although, this first value could be incorrect, sometimes it is better not to have a well computed value (because when we compute the next value will be correct) than have to wait for one of them.

The **Look back percentatge** menu option lets to select the percentage of tracefile records that will be looked back to compute the correct semantic value.

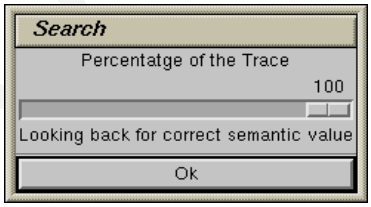


Figure 6.14: Paraver Search

A 0% or 100% values means that Paraver will look back until all the rows have their correct semantic value computed. A value greater than 0 means the percentage of records that will be looked back to compute the correct value.

Figure 6.15 shows the difference between computing the correct semantic value or not for a row. The tracefile that has been used is a non homogeneous tracefile because the processor number one only has records at the beginning and at the end of the tracefile, the rest of the tracefile is in a waiting state (red color). When we make a zoom only searching back the 10 % of tracefile records, the correct semantic value for this row couldn't be correct because to search it we should go back until the beginning. Note, that when it happens row one is painted as an idle state because the default semantic value is 0. In your example all the row has been painted as idle but usually the zoomed area has records in all the rows and only a small idle state will appear.

If we search until the beginning we obtain the correct semantic value, so the row is painted with red color.

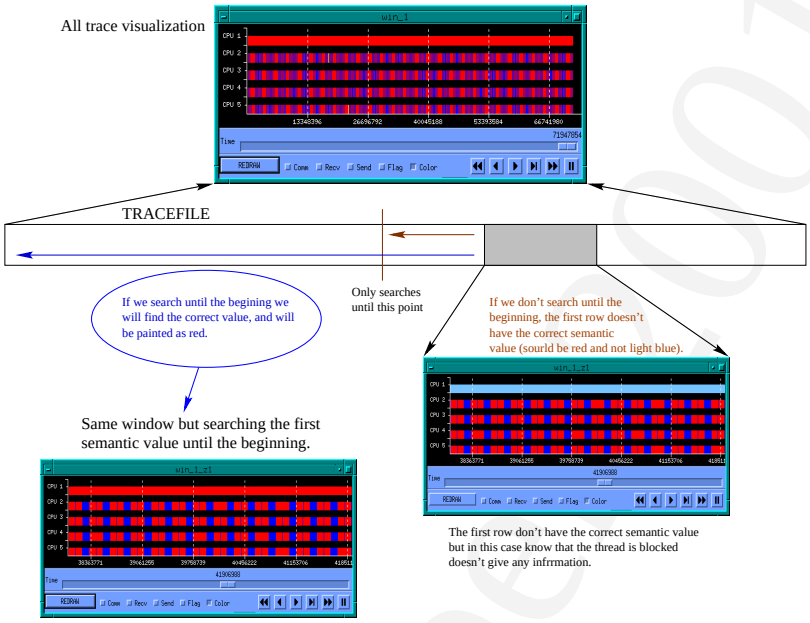


Figure 6.15: Example: Selecting the correct look back value

Realloc Colors option

Sometimes when **paraver** is launched, if other X applications are loaded (like Netscape) paraver can't alloc the color pallete and a Warning window is raised.

This option tries to realloc the color pallete used by Paraver after closing some X applications.

6.1.4 Help menu

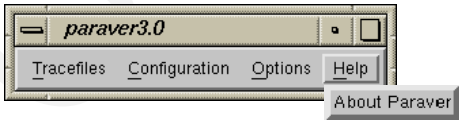


Figure 6.16: Help menu options

About option

Paraver shows its credits.

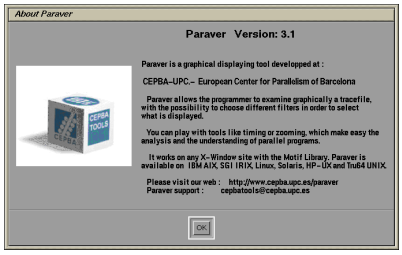


Figure 6.17: Paraver Version

6.2 Global Controller window

The Global Controller window is composed of a set of buttons which implement the Paraver functionality (figure 6.18). When a tracefile has been loaded these buttons will be enabled, if there isn't tracefile loaded these buttons do not work.

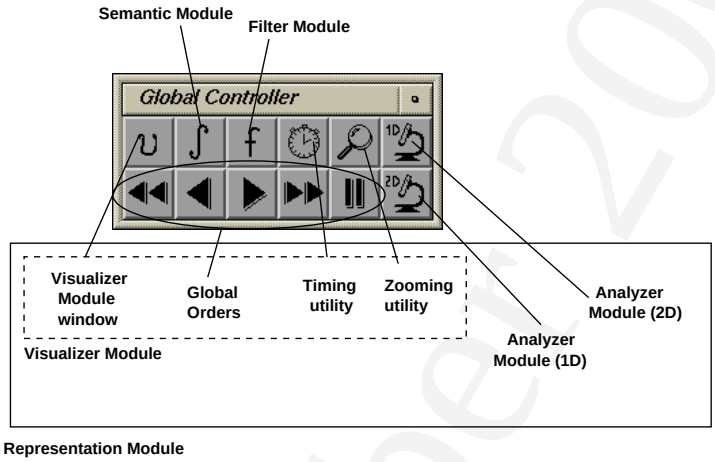


Figure 6.18: Global Controler window

Paraver has three mainly modules : the Filter Module, the Semantic Module and the Representation Module. This last module is composed of three more modules which are : the *Visualizer Module*, the *Textual Module* and the *Analyzer Module*.

In the Global Controler window, the Visualizer Module is divided in different buttons because it has some utilities that works separately. Finally, the Textual Module doesn't appear in the Global Controler because it works onto a displaying window (see chapter 9 to how does it work).

Chapter 7

Filter Module

7.1 Introduction

The Filter Module is the first module that will work onto the trace file (remember the Paraver structure 5.1 on page 19). This module lets to filter some communications and/or some events that won't be passed to the next modules.

Filtering any communication or any event means that those filtered traces won't appear in the next modules and also, they won't appear in our visualization or in our analysis.

The *Filtering Icon* (figure 7.1) raises the `Filter window` that will appear at the most right side of your screen.

Figure 7.1: Filter icon

7.2 Filter module window

The *Filter Module window* (figure 7.2) has two main areas : to filter the communications and/or to filter the events. The name of the current displaying window is shown at the top.

7.2.1 Filtering communications area.

Paraver has two types of communications, the logical (when the send/receive functions are called) and the physical (when the message is really sended or received) communications.

By default, Paraver works only with the logical communications and the physical are filtered. At the top of the communication area there are the **Logical** and **Physical** toggles to select the type of communication that will be filtered or not. Furthermore it is possible to filter communications by PARTNERS and by MESSAGES.

PARTNERS Choose the communication pattern : "From" objects (and/or) "To" objects. The objects have to be written in the text box only separated by commas. The functions to select the "partners" (rows) are:

All : All the current rows will be selected.

!= : Select all the current rows but ones written in the text box.

= : Only select the rows written in the text box.

None : No rows will be selected.

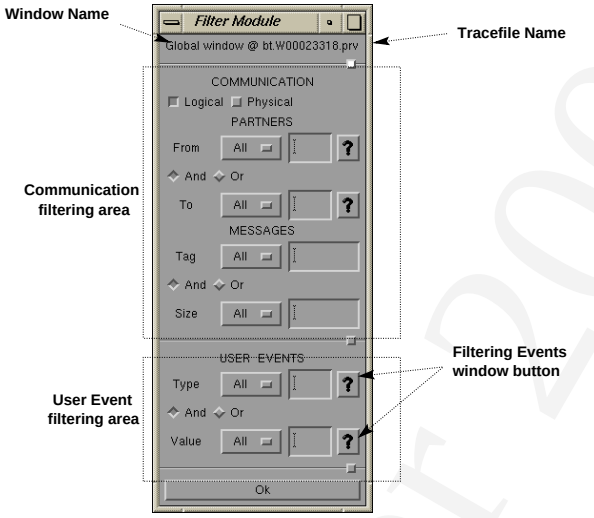


Figure 7.2: Filter Module window

In order to help users when selecting communication partners, the **Filter Object Selection window** (see figure refFilterObjectSelection) could be raised to select the *From* and *To* partners.

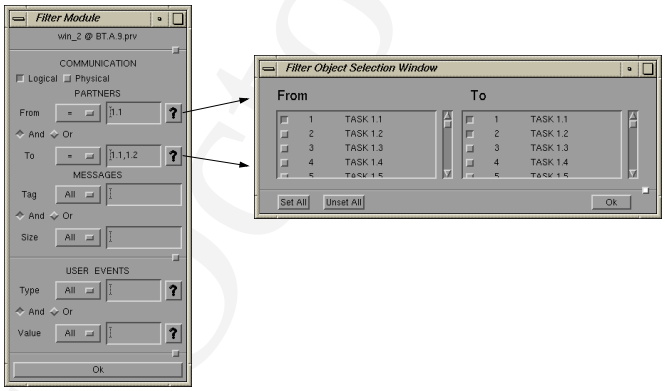


Figure 7.3: Filter Object Selection window

MESSAGES Choose the communication attributes: "Tag" tags (and/or) "Size" sizes. The tags/sizes have to be written in the text box only separated by colons.

The functions to select the "messages" are:

All : All the current messages will be selected.

!= : Select all the current messages but ones written in the text box.

> : Select all the messages greater than ones written in the text box.

< : Select all the messages less than ones written in the text box.

= : Only select the messages written in the text box.

None : No messages will be selected.

7.2.2 Filtering user events area.

The user event filtering lets to select the "Type" types (and/or) "Value" values. The types/values have to be written in the text box only separated by colons.

The functions to select the "user events" are:

All : All the current user events will be selected.

!= : Select all the current user events but ones written in the text box.

> : Select all the user events greater than ones written in the text box.

< : Select all the user events less than ones written in the text box.

= : Only select the user events written in the text box.

None : No user events will be selected.

[x,y] : Select all the user events whose type/value is a value within the interval.

When both ("PARTNERS" and "MESSAGES") filtering processes are activated, then Paraver will draw the communication traces that accomplish both filtering conditions.

Paraver only takes into account the records (selected) returned by the Filter Module.

The filtering selection can be applied on the drawing area of the current window, just by clicking on the "Redraw" button; it is not necessary to rebuild the displaying.

7.3 Selecting event types/values using the Events window

The **Filtering Events window** help us to select the user events that will be considered in the filter module. To raise it, click in one of the question mark buttons (see the *Filtering Events window buttons* in figure 7.2) in the user events area.

Sometimes, when selecting events to filter, we know the events that have to be selected but we don't remember their type or their values. This window shows a list with all the defined event types[1] and another list with the values defined for each type.

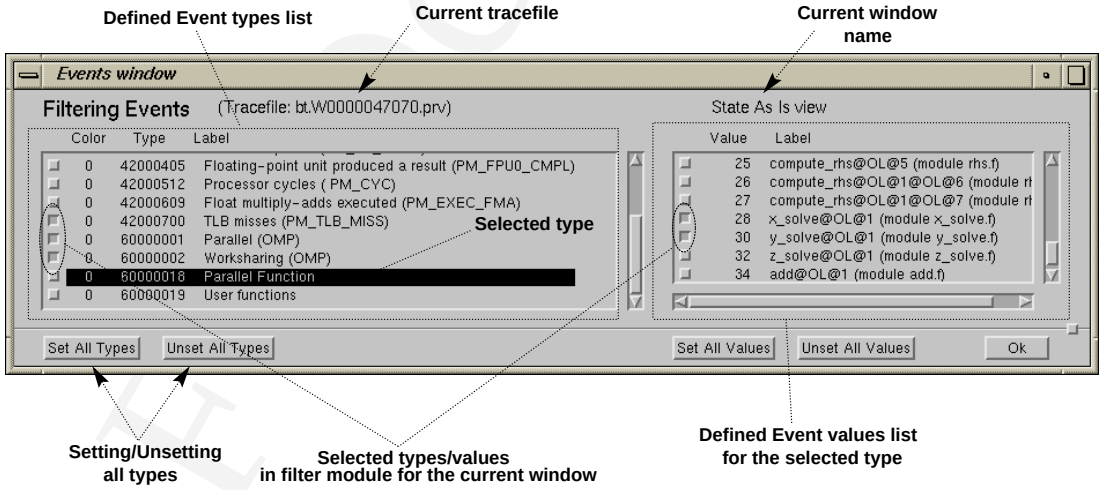


Figure 7.4: Events window

The *Defined Event Types* list shows for each event type the gradient color number which the flag will be painted, the type number and its label. Select a type by clicking its item list and you will see their defined values in the *Defined Event values list.*

[1] The Filtering Events window shows the user event types and values defined in the *Paraver Config File* (see the **Trace Generation** document). The event types could have many event values which haven't a defined label, these values don't appear in this window because they haven't been defined by the user.

Chapter 8

Semantic Module

The Semantic Module computes the values that will be transfered to the representation levels; these values are based on one or several records returned by the *Filter Module* (see chapter 7 on page 33) and are known as **semantic values**.

The representation received the semantic values and display them in a graqphical way.. For example, those values could be seen on a *displaying window* (see chapter 9) like a code colors where each value has associated a color, or can be analyzed in the *Analyzer Module*.

The Semantic Module could extract the information from the trace file in several ways, and we will have to select it to obtain the desired trace view. The semantic values are computed through the **Paraver Object Model** hierarchy (see chapter 4). The lowest level (THREAD level) gets a value from filtered trace records. Depending on the selected level where Semantic Module will work, different functions will be applied. This first level of functionality obtain values directly from record traces. A second level of functionality let to combine different views to obtain derived metrics from the first level of functionality. I could be seen as combine two different views at the same hierarchy level to obtain a new one.

Next sections will try to describe how values are collected and combined through the different levels. Section 8.1 will describe how the Semantic Module works. It will describe how values will be combined through the selected object model hierarchy to obtain the resulting semantic value at selected level. Also, it will introduce the user to the second level of functionality. Sections 8.2 and 8.3 will describe the Semantic module window and the functions implemented in each level that collect (THREAD level) or combine (upper levels) the values from record traces. Finally, the section 8.4 will describe the extended functionality DERIVED WINDOWS.

8.1 How does the Semantic Module work ?

Paraver offers some default ways to collect the values from traces. Mainly, we have three ways to get the values from traces:

- work with **Process Model** objects: THREAD, TASK, APPL and WORKLOAD.
- work with **Resource Model** objects: CPU, NODE and SYSTEM.
- obtain derived metrics by combinning different views.

In the first one, we get the values through the Process Model hierarchy. At the bottom level, there is the THREAD level which gets the values from traces. Top levels (TASK, APPL and WORKLOAD) combine the values returned by previous levels. For example, TASK level compute a combined value returned from all the thread values of the task.

The second one get the values through the Resource Model hierarchy. At the bottom, there is the THREAD level that passes to the processor (CPU) level the values got from record traces. The processor level gets the value from the thread that is executing on that processor. Top levels

(NODE and SYSTEM) combine the values returned by previous levels. For example, NODE level compute a combined value from values returned by all the processors of the node.

Values returned by the thread level could be: state where thread is, thread identifier, task identifier, event values, ... When working with Resource Model objects, values from traces that has not been executing in a processor are not considered.

The third one combine the different views created to get new derived views. It could be seen as combine the values returned by two hierarchies into another one.

Next sections will describe how values are computed through the object model hierarchy to obtain the semantic value. Section 8.1.1 will show how values are computed in the Process Model hierarchy, section 8.1.2 wil describe the Resource Model hierarchy and finally, section 8.1.4 will describe how derived views are obtained.

8.1.1 Working with Process model objects

When working with Paraver Process Model objects, there are four levels that we can select to work with: the application level (APPL), task level, thread level and at the top the workload level (WORKLOAD). In each level Paraver applies some functions to collect the values from the previous levels and return the corresponding value for the selected level. As we will show in section 8.1.3, before send the computed values for the selected level to the representation modules, they will be passed to another levels known as **compose levels**. These levels have been added to modify the values at the top level. We will explain in detail its uselfulness.

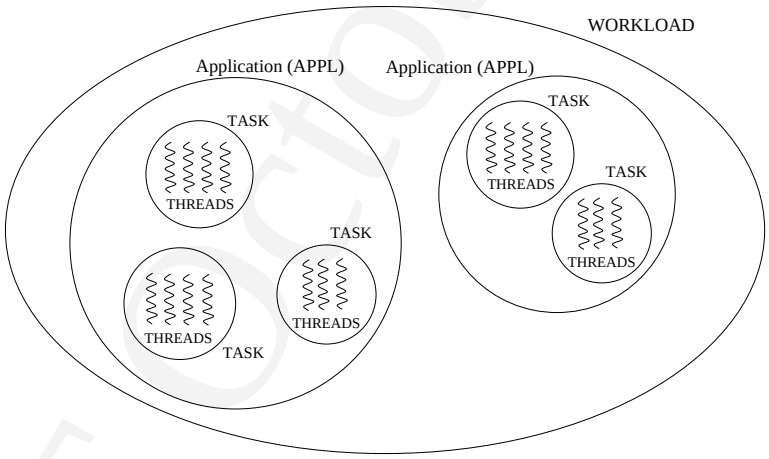


Figure 8.1: Paraver process model

When working at APPL level, the semantic value returned through the process model is a combined value of all the values computed for all the objects within the application object. Values returned at APPL level are computed as:

1. each thread within a task computes a value from record traces. These values are passed to the task where thread belongs
2. the tasks receive the values returned by its threads and computes a combined value that is passed to the application where task belongs
3. finally, the application receive the values returned by its tasks and computes a new combined value, *semantic value of Process Model*, that is passed passed to the next level (see figure 8.2).

When working with levels TASK or THREAD levels the semantic value is computed as :

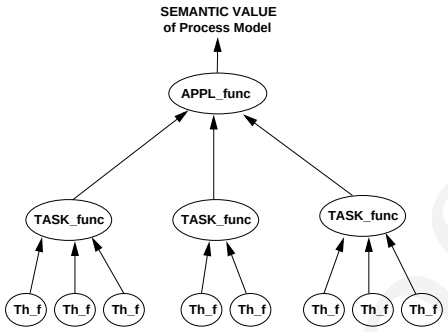


Figure 8.2: Semantic Value computed at APPL level

- when working at TASK level, each task returns a combined value, *semantic value of Process Model*, with all the values passed by its threads (see figure 8.3 left) to the next level.

- when working at THREAD level no upper levels are applied, each thread returns the result of apply the selected function, *semantic value of Process Model*, to the next level (see figure 8.3 right).

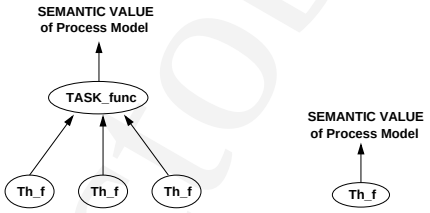


Figure 8.3: Semantic Value computed at Task and Thread levels.

When working at the WORKLOAD level, the semantic value returned through the process model is a combined value of all the application values (see figure 8.4). The WORKLOAD level has been added in order to combine the value returned for each application at the most top hierarchy level. These level is useful when trace cotains more than one application.

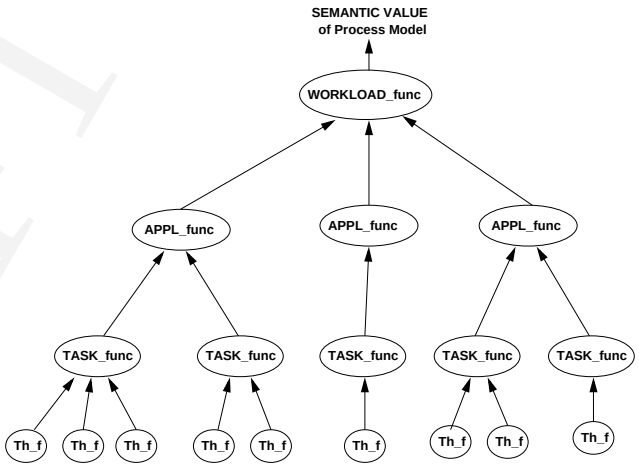


Figure 8.4: Semantic Value computed at WORKLOAD level

8.1.2 Working with Resource model objects

The Paraver Resource Model has three levels: the node level (NODE), the processor level (CPU) and at the top, the system level (SYSTEM). In each level Paraver applies some functions to collect the values from the previous levels and return the corresponding value for the selected level. As we will show in section 8.1.3, before sending the computed values for the selected level to the representation modules, they will be passed to another levels known as **compose levels**. These levels have been added to modify the values at the top level. We will explain in detail its uselfulness.

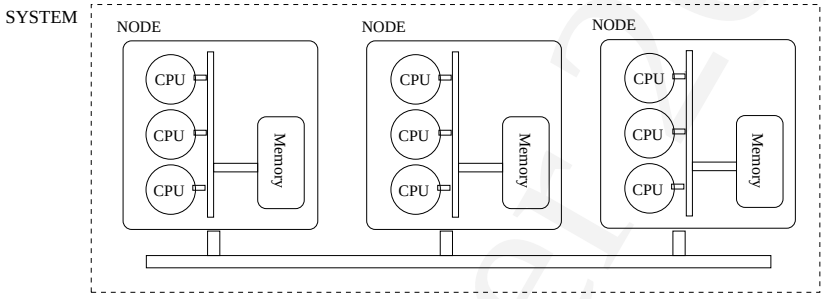


Figure 8.5: Paraver resource model

When working at NODE level, the semantic value returned through the resource model is a combined value of all the values computed for all the processors within the node (see figure 8.6 left). The CPU level works over the THREAD level, **only one thread could me mapped on a cpu at a time**.

When working at CPU level, the semantic value returned through the resource model is the value returned by the thread that is executing on the processor (see figure 8.6 right). Only one thread value is returned at a time to the processor, the values returned by applying the THREAD level function is returned to the processor where it is executing. Depending on the view that would be obtained we could select at THREAD level functions that:

- return the state of the thread. The processor view will show the thread activity that is executing on the processor.
- return thread, task or application identifier. The processor view will show the thread identifier that is executing in the processor. This type of view shows the threads that have been executing on the processor.

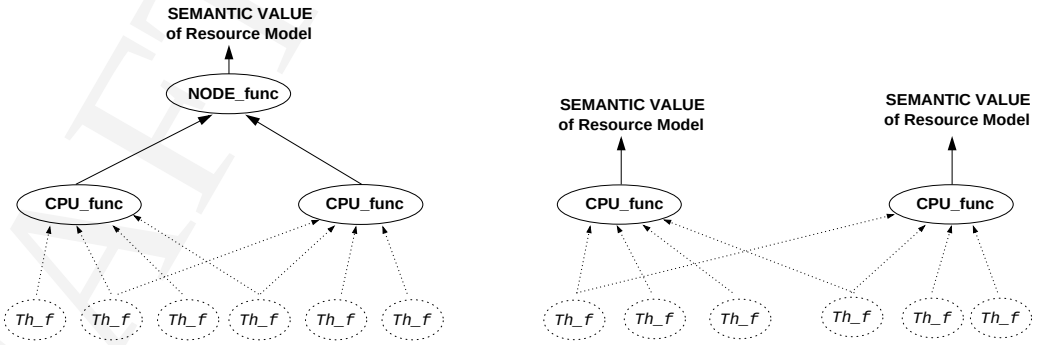


Figure 8.6: Semantic Value computed at NODE and CPU levels

When working at the SYSTEM level, the semantic value returned through the resource model is a combined value of all the node values (see figure 8.7). Different metrics could be obtained using

in this level: number of processors active in the system during the execution, number of processors idle, number of processors doing an specific thread activity, ... This level has been added in order to get a combined from the values returned for each node, a global SYSTEM value.

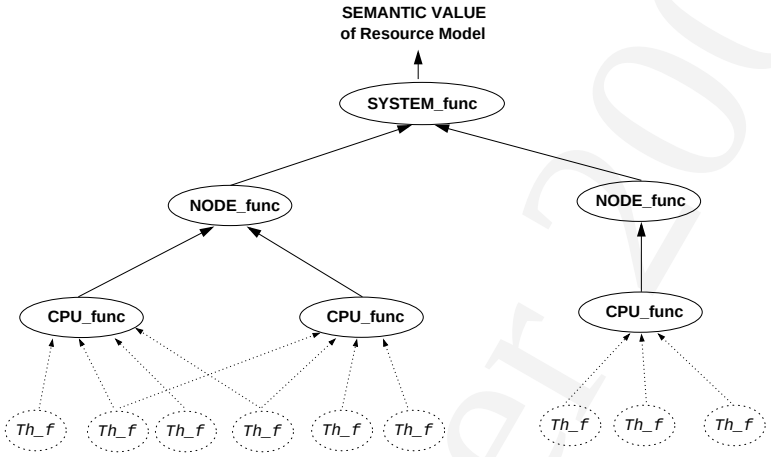


Figure 8.7: Semantic Value computed at SYSTEM level

8.1.3 Compose levels

The values returned through the process or resource models are processed by two extra levels that has been added on the top of the hierarchies, the **compose levels**. These levels allow to apply some functions to the values before send them to the representation modules (figure 8.8).

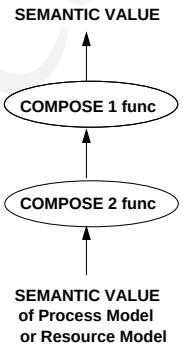


Figure 8.8: COMPOSE levels

First, the Semantic module applies the selected COMPOSE 2 function to the value returned through the object model hierarchy. The value returned by the functions is passed to the next compose level. Finally, the Semantic module applies the selected COMPOSE 1 function to the value returned by the COMPOSE 2 level.

Section 8.3.4 will describe the functions that have been added to the **compose levels**. There are some views where is needed to select one of these COMPOSE functions to modify the values at the top of the hierarchies. For example, we could filter some values (Select Range function), divide the values (Divide and Mod functions), ...

8.1.4 Derived views

The **derived views** have been added to extend the Paraver functionality. The goal is to obtain a derived metrics by combining different views of the same trace file (see figure 8.9).

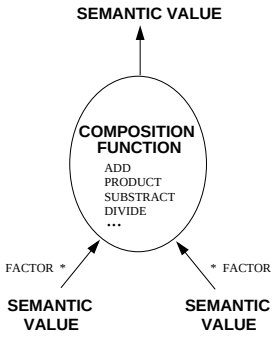


Figure 8.9: Combining two semantic values into a new one.

The **derived views** combine (add, divide, substract, ...) the values returned by the Semantic module of two object model hierarchies to a new one. A weight could be applied to the semantic values of each source hierarchy by multiplying a FACTOR value to each branch.

These type of views are useful to obtain derived metrics that could not be extracted directly from trace records through the Semantic module and it is needed a combination of extracted trace parametres to get the desired information.

Supose that we have obtained next two views:

- in the first one, the Semantic module is returning for each thread the performance counter of executed instructions
- in the second one, the Semantic module is returning for each thread the performance counter of consumed processor cycles

A derived metric that could obtained from the two performance counters, executed instructions and processor cycles, is the **Instructions per Cycle (IPC)** value. Record traces only contain the performance counter information but using the derived views to compute it is as easy as to divide the *semantic values* returned by the two views.

After the values are combined, two new COMPOSE levels also could be applied to the resulting value. **It is important to note that when working with derived views the combined semantic values are the resulting of computing the two selected source hierarchies including the compose levels too. In fact derived views are combining two semantic values that have been returned after applying the section 8.1.3).**

8.2 Semantic window

The Semantic Icon (figure 8.10) on the *Global Controller* window raises the Semantic Module window. Depending on the type of view that we are using the Semantic window looks different.

Figure 8.10: Semantic icon

Working with Process/Resource model objects

The Semantic Module window (figure 8.11) is used to select what information will be extracted from the trace file and how it will be interpreted. It has two main areas : the PROCESS/RESOURCE model area with the object model levels, and the COMPOSE area where we can select the compose

functions that will be applied to the resulting values. The current window and tracefile names are shown at the top of the Semantic window.

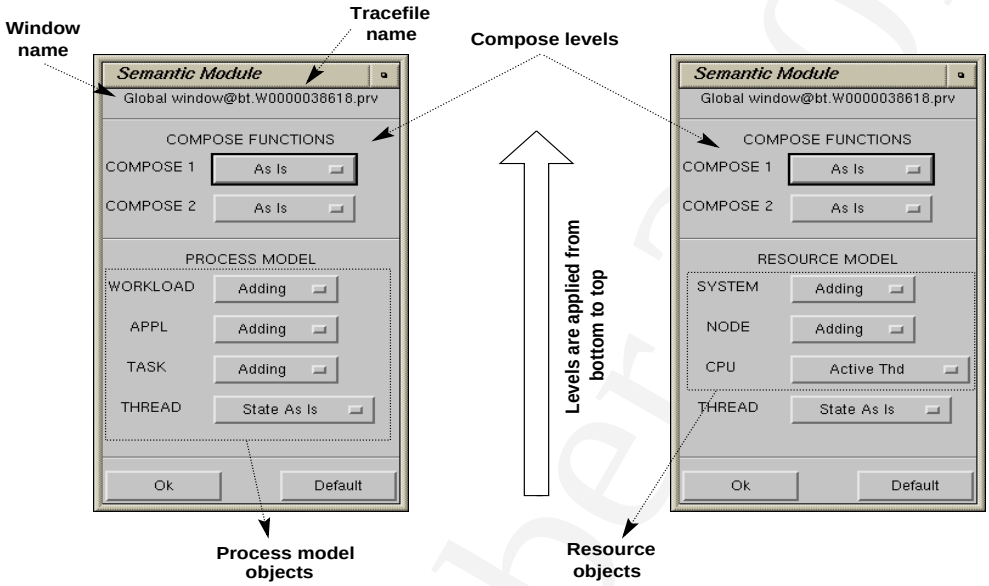


Figure 8.11: Semantic Module window

In each level there are some functions to manage the values from the trace file. These functions are applied from bottom to top and Paraver disable the levels which are forbidden in according to the process model where the current window is working. For example, at CPU level, you only can select functions in CPU and THREAD levels, the NODE and SYSTEM functions are disabled. If you worked at APPL level, you could work with APPL, TASK and THREAD functions but WORKLOAD level is disabled.

Depending on the level where window is working process or resource objects will be shown:

- When working with **Process model objects** (see figure 8.11 left) only the process objects are showed: THREAD, TASK, APPL and WORKLOAD levels. For example, we could conclude that left Semantic window in figure 8.11 is working at WORKLOAD level because all lower levels are enabled. The THREAD selected function is the *State As Is* function and the upper levels return the *Adding* of all the input values (we will describe the selected functions behaviour in next sections).
- When working **Resource model objects** (see figure 8.11 right), the resource objects are showed: CPU, NODE and SYSTEM levels plus the THREAD level. In the example of figure 8.11, we could conclude that right Semantic window shows is working at SYSTEM level because all lower levels are enabled. The THREAD selected function is the *State As Is* function and the selected CPU function *Active Thd* returns to the next level the value of the current thread (state value because State As Is function is selected) without modify it. The next upper levels add (*Adding* function) the values of the previous levels.

The object level where a window is working is selected in the Visualizer module and it will be described in chapter 10 .- **Representation module** on page 75.

Working with Derived views

When working with **Derived views**, the Semantic window looks completely different. As we explained in section 8.1.4, derived views combine different views (applying a factor to each one) to create a derived one. Since views are represented in a displaying window (see chapter 9 on page

63 to see a detailed description about **Displaying windows**), combine two views is to combine two displaying windows.

When working with this type of windows, the **Semantic window** tries to reflect that the view is a combination of two source views. Figure 8.12 show how **Semantic window** looks like when working with a derived window. It shows the two source window from values will be combined, the composition function that will be applied and the factors that will be applied to the source hierarchy values.

In the COMPOSE area, we can select the compose functions that will be applied to the resulting values. The current window and tracefile names are shown at the top of the Semantic window.

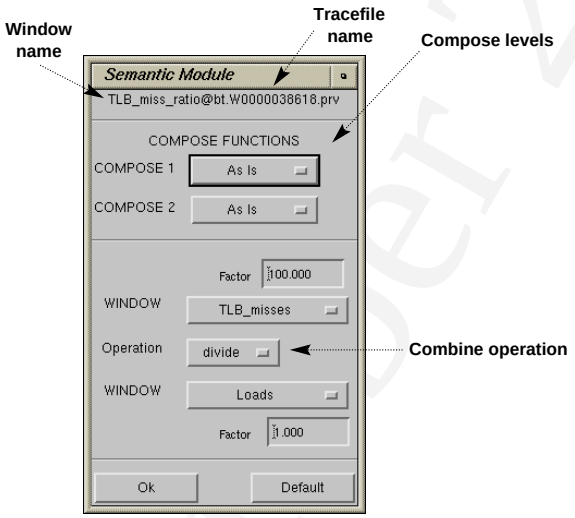


Figure 8.12: Semantic Module window

From figure 8.12 we can conclude that the resulting window **TLB_miss_ratio** window is the resulting window of dividing (*divide* operation):

- the values computed from *TLB_misses* window (multiplied by a factor of 100)
- by the values computed from *Loads* window (multiplied by a factor of 1)

The resulting values are not modified at COMPOSE levels because **As Is** operation has been selected (we will describe the COMPOSE functions behaviour in next sections).

8.3 Semantic functions

By default, Paraver has some semantic functions in each level that could be used in different ways to extract the information from the trace records. As the THREAD level is the lowest level and appears when working with process or resource object models, we first in section 8.3.1 will describe the functions that have been implemented in that level. Later, section 8.3.2 will describe the functions implemented in each process object level and section 8.3.3 will describe the functions implemented in each resource object level. Finally, section 8.3.4 will describe the functions implemented at COMPOSE level.

Thread functions return a value that could be extracted from record traces, top levels combine or modify the values from the previous level and return it to the next ones. Selecting the appropriate functions at each level we could obtain the desired information.

8.3.1 Thread functions

This is the lowest, but the most important level. This level decide in what we are going to work such as states, events or communications and what we are going to do with them.

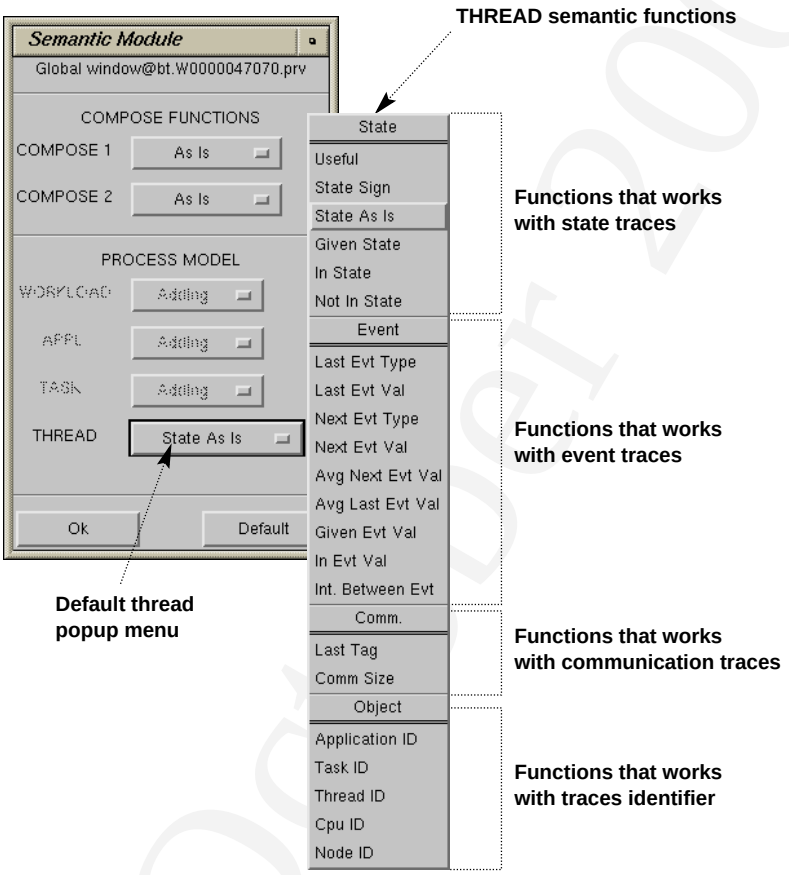


Figure 8.13: Thread level functions

Thread functions work over the trace file (remember that first, it had been filtered by the Filter Module). This level is always enabled because it works with all the defined levels in the Paraver Object Model as the lowest level.

These THREAD functions have been grouped depending on kind of information that they extract:

- the first group (**State** group) has functions that work with state traces; Functions work with the state value of the state record.
- the second group (**Event** group) has functions that work with event traces; some functions work with event types and some with event values.
- the third group (**Comm.** group) is composed by functions that work with communication record.
- the fourth group (**Object** group) is composed by functions that work with object identifiers traces. It returns the object (thread, process, ...) where record trace belongs.

Next subsections will describe the functions that have been implemented on each group and the type of information that return to the next level.

Functions that work with state traces

- **Useful :** It takes the state trace and returns if the state value is Running (state value 1) or not (if the state trace isn't *Running* it returns an *idle state* or value 0). Only state values 1 are returned, the rest of state values are returned as an idle state (state value 0). See figure 8.14.

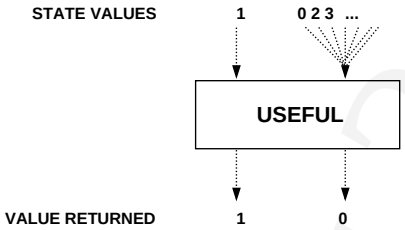


Figure 8.14: Useful function

- **State Sign :** It takes the state trace and returns a state value 1 (*running state*) when state value isn't an idle state (state value 0) and if state value is 0 (idle state) a 0 is returned.

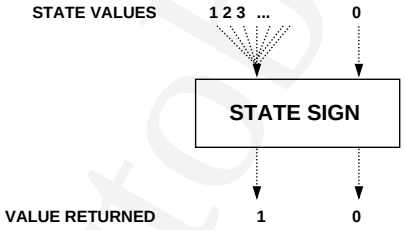


Figure 8.15: State Sign function

- **State As Is :** It takes the state trace and returns its state value. This function does not modify the state value, it just returns the state value.

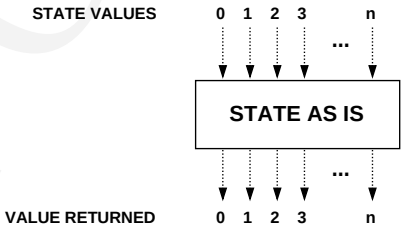


Figure 8.16: State As Is function

- **Given State :** It takes the state record and returns its state value if it is selected and idle otherwise. Is like the State As Is function but filtering only the selected states, these states are returned, but the states which hasn't been selected will be converted to an idle state (state value 0).

  To select the state values that will be returned as is, when this function is selected it raises a window to fill those state values. The user can select more than one state putting commas between them (note figure 8.17).

- **In State :** It takes the state trace and returns if the state trace is in one of the states selected or not. Is like the State Sign function but only returns as a running states the

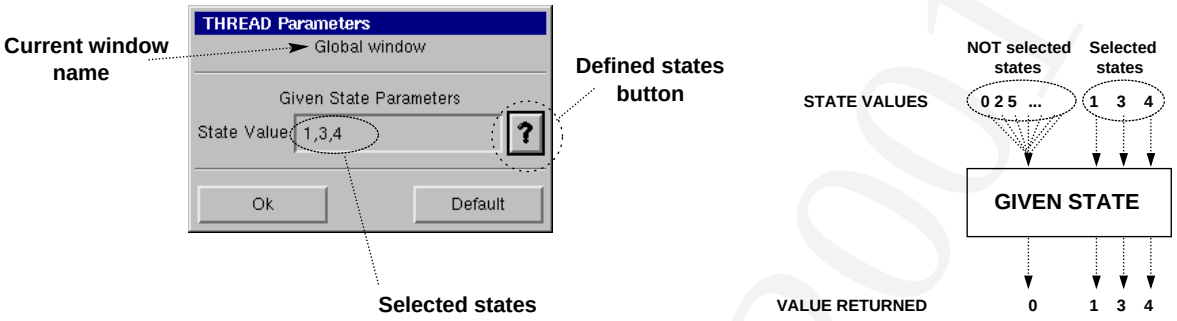


Figure 8.17: Given state parameters window

selected state values, the selected values are converted to the a running state, the rest are returned as an idle state.

When this function is selected it raises a window to fill the state values. The user can select more than one state putting commas between them (see figure 8.18).

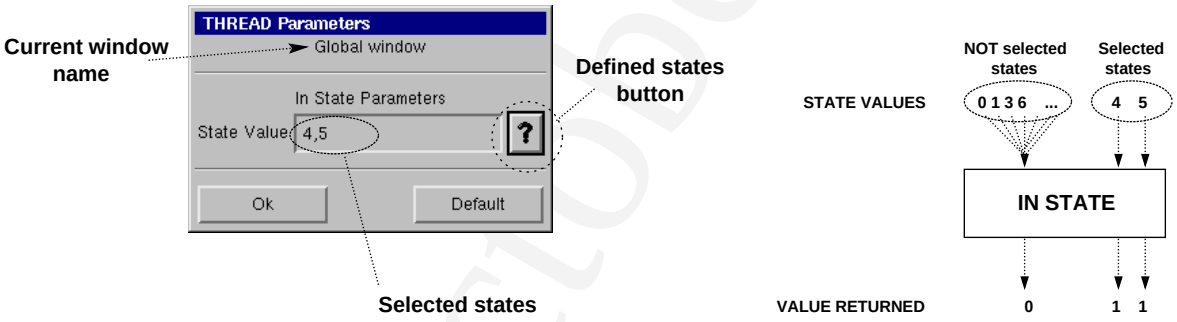


Figure 8.18: In State parameters window

Figure 8.19 shows the difference between the *In State* and *Given State* functions when we are selecting only the state value 3. The *In State* function transform this state value to a running state (state value 1) which by default is painted in dark blue, the others states values are converted to an idle state (state value 0) and are painted in light blue. The *Given State* function doesn't change the state value 3, and returns its state value (value 3)but the rest are converted to an idle state (value 0). By default, the state value 3 is painted in red as you could see in the displaying window.

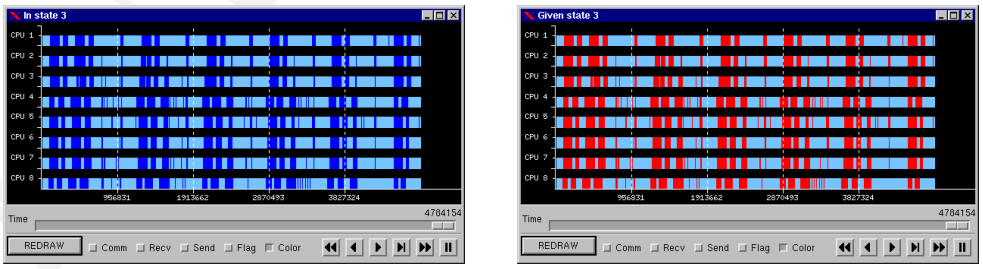


Figure 8.19: In State vs Given State

- **Not In State :** It takes the state trace and returns if the state trace isn't in one of the sates selected or not. When this function is selected a new window appears to select the states. The user can select more than one state putting commas between them.

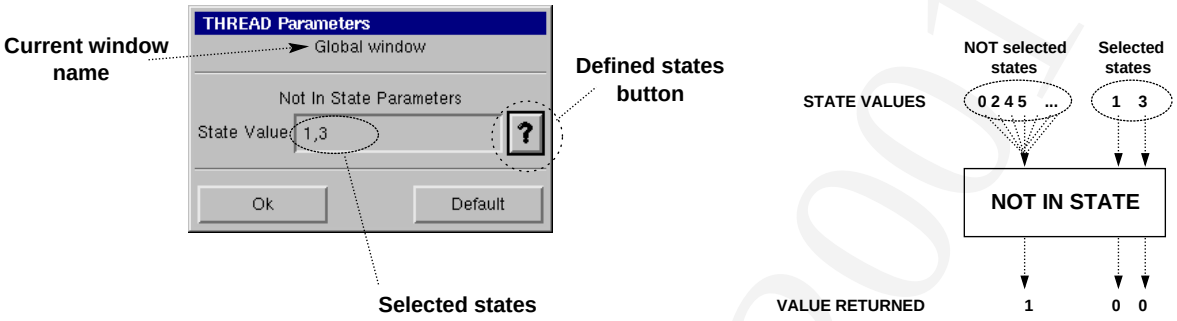


Figure 8.20: Not In State parameters window

Selecting states using the Defined states window

When selecting individual states in functions *Given State*, *In State* and *Not In State* we can use the **Defined States** window (figure 8.21).

This window contains a list with all the defined states showing their color, their value, their label and a toggle button to select/unselect the state. Next to the text box to fill the selected states in each *THREAD PARAMETERS* window there is a question mark button to raise this helpful window (see figures 8.17, 8.18 and 8.20).

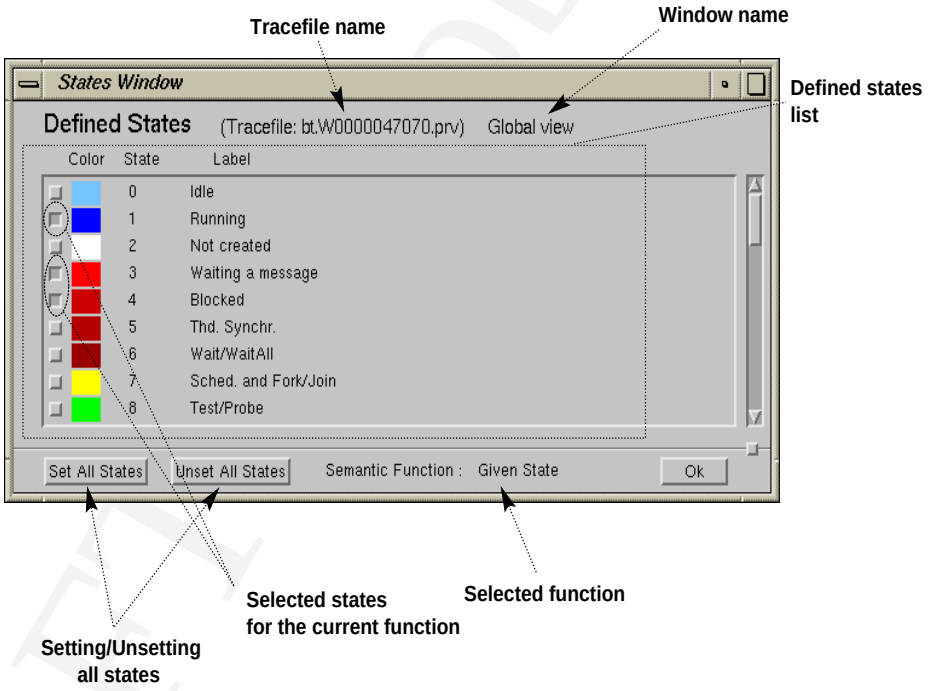


Figure 8.21: Defined States window

At the top of the window, next to the **Defined States** label there is the current window name and the current thread semantic function. Select/Unselect the desired states by clicking in each toggle button, if the toggle button is set, the state is selected and appear in the states selection of the current window. When the toggle is unset, the state isn't selected. Also, you can write the desired states separated by comas in the parameters window of each function, changes are applied and toggle buttons will be selected/unselected. Note, that changes in the selection of states only will affect the current window and the current thread semantic function.

At the bottom of the window, the buttons **Set All States** and **Unset All States** lets to

select/unselect all the states only by clicking them. The **OK** button will close the window.

The number of states, the color and its label can be changed (see Trace Generation document) to obtain a customized user environment.

Functions that work with event traces

- **Last Evt Val :** This function returns the las value of the event until a new event is found in the trace[1].

  Figure 8.22 shows an example of the values returned by this function. The user events are painted as flags (see figure 8.22). Note, that when the user event with value 107 is encountered the value returned is 107, so the value goes from 4180 to 107. When the next event is encountered the value of the the event is returned (event value 9683) so the value goes from 107 to 9683.

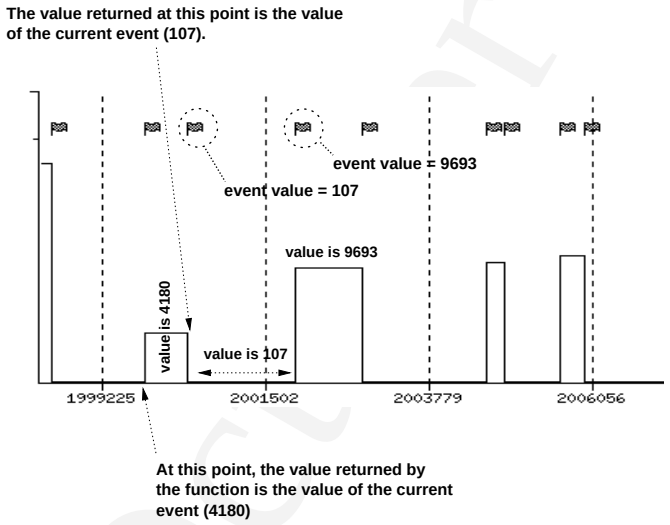


Figure 8.22: Example of Last Event Val

- **Last Evt Type :** It takes the event trace and returns its event type. Is like the **Last Evt Val** function but it returns the event type instead of the event value.

- **Next Evt Val :** When an event trace is encountered this function searches for the next event and returns its value. So from the current event to the next event, the value returned will be the value of the next event.

  This function will be very useful when for example, the user events are counting something that had happened within the interval between the current time and the next event time, because it lets to paint the interval with the counting value.

  Figure 8.23 shows the same example that figure 8.22 but using the *Next Evt Val* function. When the current event is the event with value 107, the value returned is the value of the next event (9683). The value 107 has been returned when the previous event has been encountered.

  Suppose that event values are counting the primary data misses. Each event marks the data cache misses that has been occurred during the interval. If we use the **Last Evt Val** function (figure 8.22), we can't see the value in the correct interval, note that the 9683 cache misses are painted in an incorrect interval, they should be painted in the interval from event with value 107 to event with value 9683.

[1] Remember that **Filter Module** is applied before the Semantic Module, so only selected events will be passed to the semantic function. Filter the desired events to obtain your desired visualization.

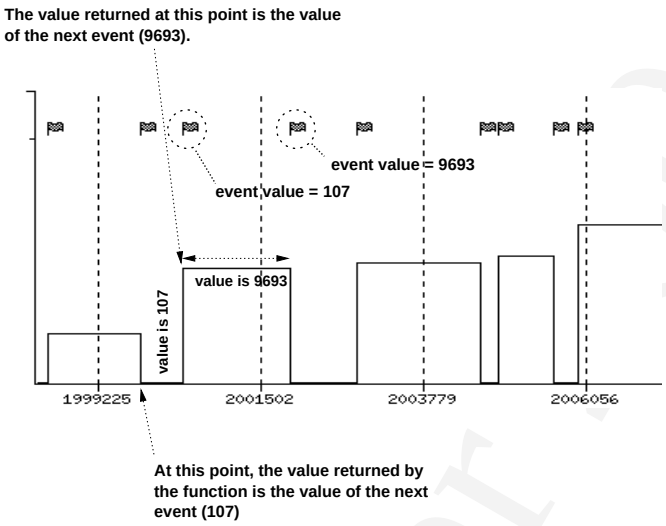


Figure 8.23: Example of Next Event Val

If we use the **Next Evt Val** function, the value is painted in the correct interval. Note the difference between the two examples. The user has to select what will be the correct visualization.

- **Next Evt Type :** When an event trace is encountered this function searches for the next event and returns its type. Is like the **Next Evt Val** function but it returns the event type instead of event value.
- **Avg Next Evt Val :** This function works like the *Next Evt Val* (working with the value of the next event) but returning a value **in function of** the interval duration.

$$\frac{value_{i+1} * factor}{t_{i+1} - t_i}$$

Where:

  - $value_{i+1}$ is the value of the next event.
  - t_{i+1} is the time of the the next event (time expressed in microseconds).
  - t_i is the time of the current event (time expressed in microseconds).

The value of the next event ($value_{i+1}$) is multiplied by a **factor** (by default, value 1000). This **factor** could be used to change the units of the returned value, for example, can be used to obtain the value per seconds instead of value per milliseconds. By default, we obtain a value per milliseconds because factor is 1000 and we are working with microsecond precision.

Figure 8.24 shows the same example than figure 8.24 but using the Avg Next Evt Val instead of Next Evt Val. Note that value 181 is obtained through the next event value (value 187) multiplied by factor (value 1000) and divided by the duration of the interval (591 microseconds).

- **Avg Last Evt Val :** This function works like the previous one but works with the current event value instead of the next.

$$\frac{value_{i} * factor}{t_{i+1} - t_i}$$

Where:

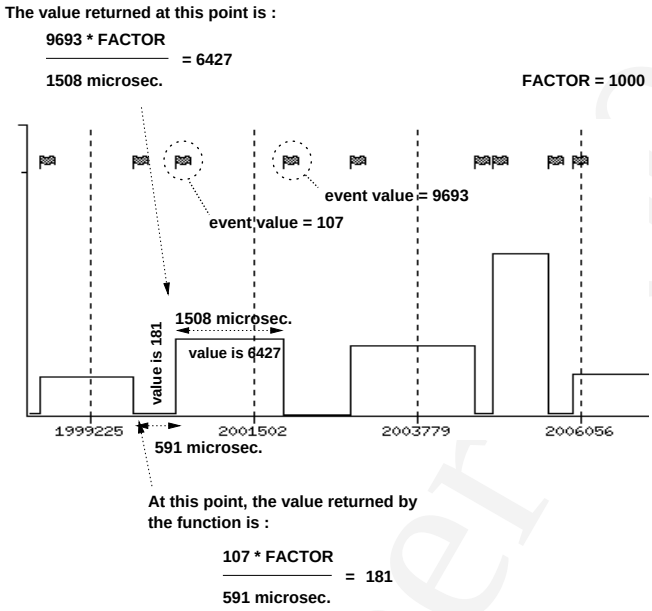


Figure 8.24: Example of Avg Next Event Val

- $value_i$ is the value of the current event.
- t_{i+1} is the time of the the next event (time expressed in microseconds).
- t_i is the time of the current event (time expressed in microseconds).

As a result, the function will return the value of the current event in function of time.

- **Given Evt Val :** It takes the event trace and returns its event value if it has been selected or an idle state if not. To select the event values that will be returned, when this function is selected Paraver raises a window to select these event values. As the state parameters windows, more than one value could be selected.

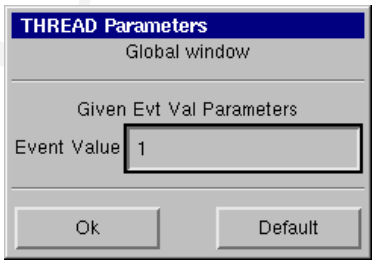


Figure 8.25: Given Event Value parameters window

- **In Evt Val :** It takes the event trace and returns a running state if it has been selected or an idle state if not. To select the event values that will be returned, when this function is selected Paraver raises a window to select this event values. As the state parameters windows, more than one value could be selected.

  The difference between this function and *Given Evt Val* is the same that the difference between *Given State* and *In State*; the previous one returns the event value, and this one returns it as a running state (value 1).

- **Int. Between Evt :** This function returns as a value the time between the current event trace and the next event trace. When an event trace is encountered, search for the next

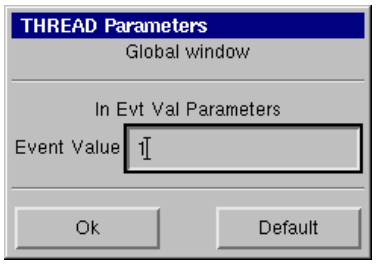


Figure 8.26: In Event Value parameters window

event and substracts the two times. Figure 8.27 shows how it works and which values are returned.

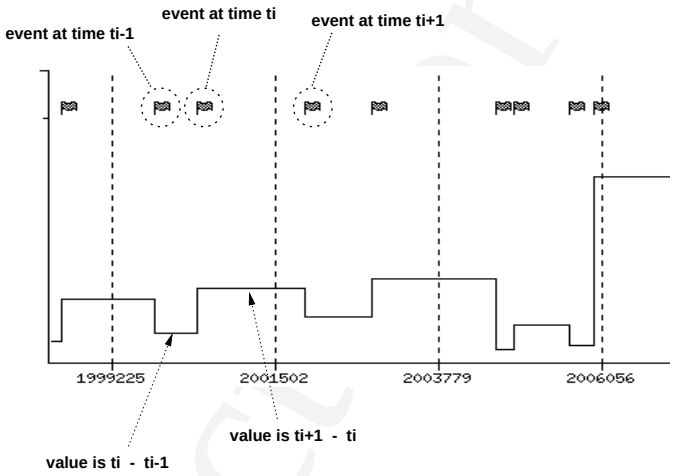


Figure 8.27: Example of Int. Between Evt

Functions that work with communication traces

- **Last Size :** Returns the communication size while it is active. From logical communication ot physical the communication size is returned as value.
- **Last Tag :** If the communication trace is a physical receive returns its message tag.

Functions that work with object identifiers

- **Application ID :** Returns as a value the application identifier where thread belongs. For example, this function is useful to see which applications have been running on a processor.
- **Task ID :** Returns as a value the task identifier where thread belongs. The task identifier is a global numbering assigned to all tasks from all applications. For example, this function is useful to see which tasks have been running on a processor.
- **Thread ID :** Returns as a value the thread identifier. The thread identifier is a global numbering assigned to all threads from all applications. For example, this function is useful to see which threads have been running on a processor.
- **Cpu ID :** Returns as a value the processor identifier where thread is running. The processor identifier is a global numbering assigned to all processors from all nodes. This function is useful to see the processors where a thread has been running during the execution.

- **Node ID :** Returns as a value the node where thread is running.

8.3.2 TASK, APPL and WORKLOAD functions

TASK, APPL and WORKLOAD levels return a combined value of all the values returned by the previous level. Next points will describe the functions implemented on Process Object level.

TASK functions

When the TASK functions are enabled it means that we are working at TASK or APPL levels. The TASK level works over the THREAD level.

In this level, each task receives the values from its threads, takes all these values and computes a combined value. The value will be passed to the next level (if window is working at TASK level, the value is passed to the COMPOSE levels). The TASK level is enabled when working at TASK level and APPL level.

Figure 8.28: Task level functions

The functions implemented at this level are :

- **Adding :** It adds all the input values and returns the sum.
- **Adding Sign :** It adds all the input values and returns if the sum is greater than zero.
- **Average :** It returns the average of the input values.
- **Maximum :** It returns the maximum value of all the input values.
- **Minimum :** It returns the minimum value of all the input values.
- **Thread i :** This function takes all the input values and returns the value of the **thread i**[2]. The thread identifier is selected in the parameters window (8.29) that will be raised when **Thread i** function is selected.

[2]Tasks could have different number of threads, if a selected thread number not exists within a task, a value 0 is returned for that thread.

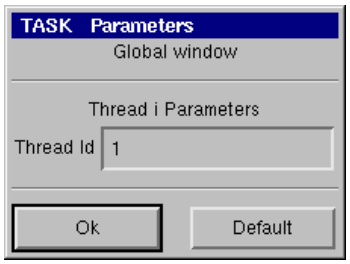


Figure 8.29: Thread i parameters window

APPL functions

The APPL functions return values which refers to the application level, these values are computed with all the values from the previous level, in this case this level is the TASK level. As the previous level, these functions compute a combined value which will be passed to the next level (if window is working at APPL level, the value is passed to the WORKLOAD level).

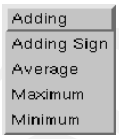


Figure 8.30: Appl level functions

The functions implemented at this level (see figure 8.30) are :

- **Adding :** It adds all the input values and returns the sum.
- **Adding Sign :** It adds all the input values and returns if the sum is greater than zero.
- **Average :** It returns the average of the input values.
- **Maximum :** It returns the maximum value of all the input values.
- **Minimum :** It returns the minimum value of all the input values.
- **Add Tasks :** It only adds the input values of the selected tasks.

WORKLOAD functions

When working at WORKLOAD level, APPL functions, TASK functions, and THREAD functions are enabled. The WORKLOAD functions receive the values from all the applications defined in the trace (APPL level).

In this level, thr workload level receives the values from all the applications, takes all these values and computes a combined value that will be passed to the next level (COMPOSE levels).

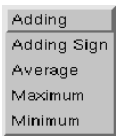


Figure 8.31: Node level functions

The functions implemented in this level (see figure 8.31) are:

- **Adding :** It adds all the input values and returns the sum.
- **Adding Sign :** It adds all the input values and returns if the sum is greater than zero.
- **Average :** It returns the average of the input values.
- **Maximum :** It returns the maximum value of all the input values.
- **Minimum :** It returns the minimum value of all the input values.

8.3.3 CPU, NODE and SYSTEM functions

The CPU level returns the value of the thread that is executing on that processor so only one value is received from bottom level. On the other hand, NODE and SYSTEM levels return a combined value of all the values returned by the previous level. A node receives the value of all its processors and the system receives the value from all the nodes.

It is important to note that the lowest level (THREAD level) when a resource object level is selected only considers the record traces which have been executing on a resource (the processor identifier of the record has been set), the rest are not considered. Next points will describe the functions implemented on Resource Object levels.

CPU functions

When working at CPU level, the CPU functions and the THREAD functions are enabled. The CPU functions receive the value returned from the previous level, THREAD level.

CPU functions are used to act on the value returned by the thread that is executing and return it to the next level. If no thread is executing at a time, a zero value (zero value corresponds to the idle value) is returned.

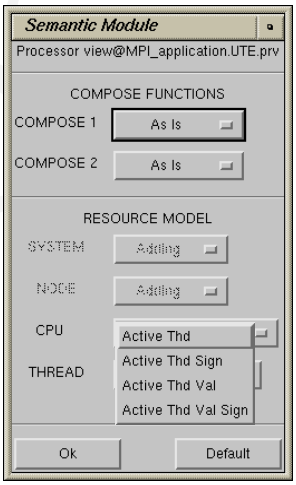


Figure 8.32: Cpu level functions

The functions implemented by paraver at this level are :

- **Active Thd :** Returns the value of the active thread as is, without modify it. If there isn't any thread executing on the processor, a zero is returned.
- **Active Thd Sign :** Returns the sign of the value passed by the active thread. If value passed by the thread is greater than zero, the value 1 is returned. Otherwise, return the idle state (value 0).

- **Active Thd Val :** If the value passed by the thread level has been selected, returns it as is. If it hasn't been selected, a zero value is returned. The function acts as filter, only selected values will be passed to the next level, the rest will be transformed to zero.

  A window is raised to select the values that will be passed to next next levels (see figure 8.33).

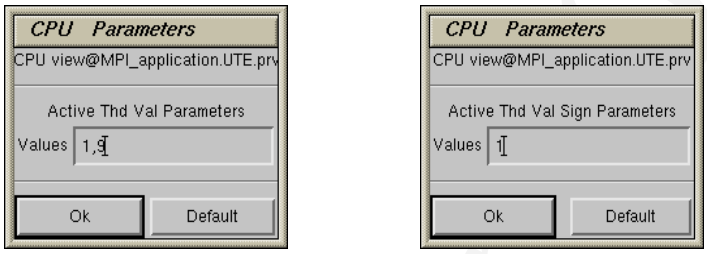


Figure 8.33: Active Thd Val and Active Thd Val Sign selection values windows

- **Active Thd Val Sign:** If the value passed by the thread level has been selected, returns its sign. If value is greater than zero, the value 1 is returned instead of the value as is. If it hasn't been selected, a zero value is returned. The function acts as filter, only the sign of selected values will be passed to the next level, the rest will be transformed to zero.

  A window is raised to select the values that will be passed to next next levels (see figure 8.33).

NODE functions

When working at NODE level, the NODE functions, the CPU functions and the THREAD functions are enabled.

The NODE functions receive the values from all its processors (CPU level) and computes a combined value that will be passed to the next level. The NODE level is also enabled when working at SYSTEM level.

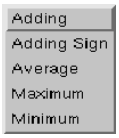


Figure 8.34: Node level functions

The functions implemented in this level (see figure 8.34) are:

- **Adding :** It adds all the input values and returns the sum.
- **Adding Sign :** It adds all the input values and returns if the sum is greater than zero.
- **Average :** It returns the average of the input values.
- **Maximum :** It returns the maximum value of all the input values.
- **Minimum :** It returns the minimum value of all the input values.

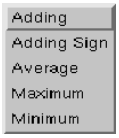


Figure 8.35: System level functions

SYSTEM functions

When working at SYSTEM level, the SYSTEM functions, the NODE functions, the CPU functions and the THREAD functions are enabled. The SYSTEM functions receive the values from all the nodes defined in the trace (NODE level) and computes a combined value that will be passed to the next level (COMPOSE levels).

The functions implemented in this level (see figure 8.35) are:

- **Adding :** It adds all the input values and returns the sum.
- **Adding Sign :** It adds all the input values and returns if the sum is greater than zero.
- **Average :** It returns the average of the input values.
- **Maximum :** It returns the maximum value of all the input values.
- **Minimum :** It returns the minimum value of all the input values.

8.3.4 Compose functions

The compose functions are applied at the top level, when all the object levels have been computed.

The two compose levels which are applied from bottom to top. First, it is applied the COMPOSE 2. The level which receives as input the value computed by the object model functions. the result is passed to the COMPOSE 1.

The two compose levels have the same functions and can be combined to obtain the final value that will be returned by the Semantic Module to the Representation Module.

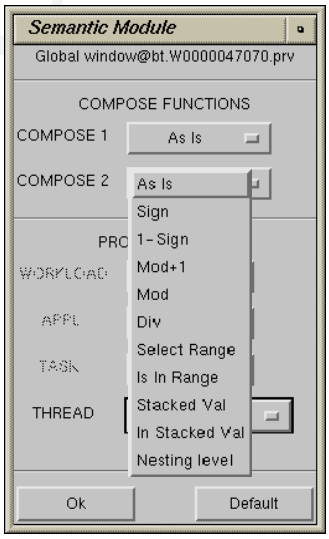


Figure 8.36: Compose level functions

The implemented functions are :

- **As Is :** Not change the value returned from the previous level. The value remains intact.
- **Sign :** This function returns the sign of the value returned from the previous level. This function return 1 (a running state) if the value is greater than zero and 0 if it is equal.
- **1-Sign :** This function is the complementary of the function Sign. It returns 0 (an idle state) if the value is positive and 1 otherwise.
- **Mod+1 :** This function returns the module of the value returned from the previous level. When the user select this function a new window appears to select the divider. By default the divider is the greatest integer number on the machine which does not change the value. The value returned by this function will be within the interval [1...divider] because adds to the result a 1.
- **Mod :** This function returns the module of the value returned from the previous level. When the user select this function a new window appears to select the divider. By default the divider is the greatest integer number on the machine which does not change the value. The value returned by this function will be within the interval [0...(divider-1)].

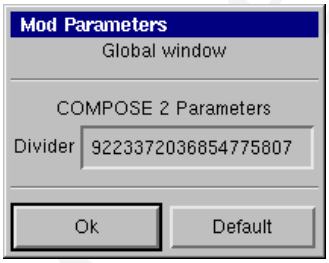


Figure 8.37: Mod window

- **Div :** This function returns the division of the value returned from the previous level. When the user select this function a new window appears to select the divider. By default the divider is 1 which not change the value.
- **Select Range :** This function returns the value returned from the previous level if it is in the selected range, if not returns 0. This function only lets to pass the values that are between the interval, the rest are converted to an idle state (value 0). When the user select this function a new window appears to select the range, the user has to select the maximum and minimum value. By default the maximum value is the greatest integer number on the machine and the minimum is 0.

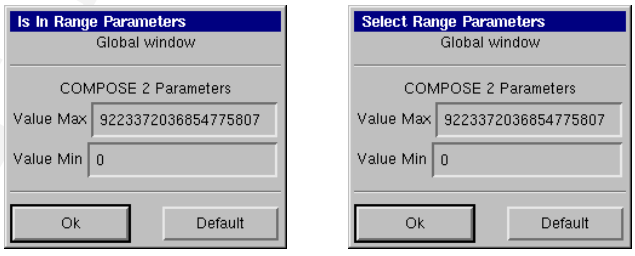
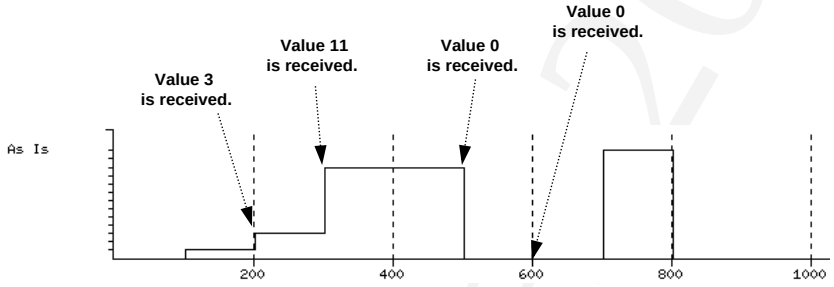


Figure 8.38: Is In Range/Select Range windows

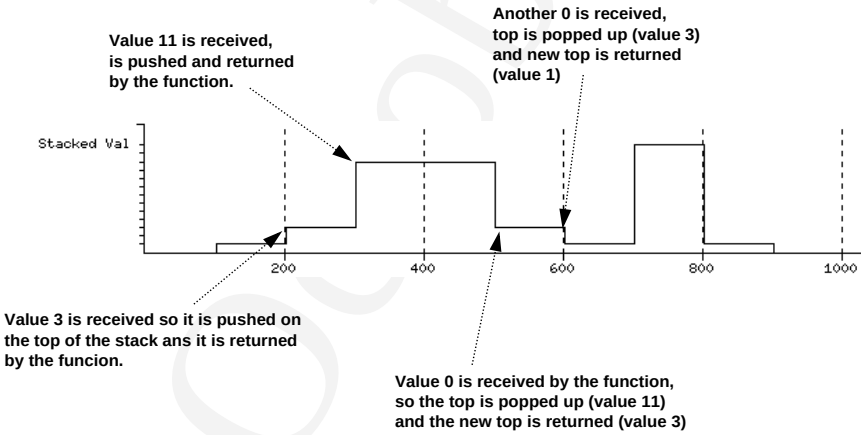
- **Is In Range :** This function works more or less as **Select Range** but returns 1 if the value returned from the previous level if it is in the selected range, if not returns 0. When the user select this function appears a new window to select the range, the user have to select

the maximum and minimum value. By default the maximum value is the greatest integer number on the machine and the minimum is 0.

- **Stacked Val :** This function works as a stack. Stores the values returning in each moment the top value. Values greater than 0 are pushed, when the function receives a value equal to 0 pop up the top of the stack and then, the top is returned. Note that the current (top of the stack) has been returned before the last change. The goal is to remember the previous states.



(a) When working with As Is function the value received is returned.



(b) When working with Stacked Val function values are pushed/popped.

Figure 8.39: Stacked Val function behaviour

Supose that values are function identifiers and zero values are the return of the functions. Using the Stacked Val function we could obtained a displaying where after returning of function 11, the displaying returns to the function id 3.

It is important to note that too construct the stack of values, Paraver searches back all the Look Back Percentatge (see section 6.1.3 on 29. Depending on the selected value the process to construct the stack could be slow. If a 100% is selected, each time that the stack needs to be reconstructed, the Semantic Module has to construct it from the trace beginning. Select the desired value that not affects the results.

- **In Stacked Val :** This function works like *Stacked Val* function but only the selected value is considered. When the user select this function a new window appears to select this selected value.

  Values greater than 0 are pushed, when the function receives a value equal to 0 pop up the top of the stack. Once selected value has been pushed on the stacked it is returned although other values were pushed after it. When selected value is popped up, a zero is returned.

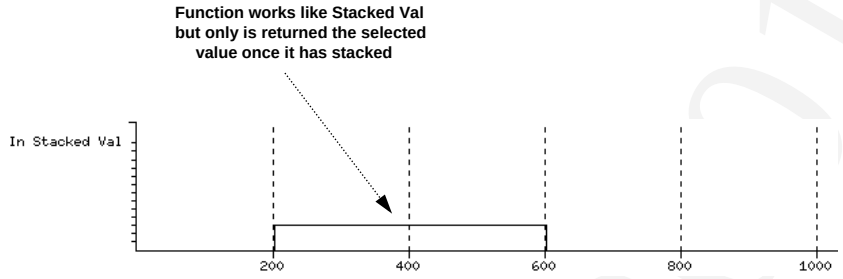


Figure 8.40: In stacked Val function behaviour

For example, if in the function of time showed in Stacked Val example (see figure 8.39) we select the *In Stacked Val* (selecting value 3) function instead of *Stacked Val* we will obtain figure 8.40. Note that where value 3 is received it is pushed and is returned until its corresponding zero is received. When the function receives and pushes the value 11, value 3 is still returned.

As Stacked Val function, supose that values are function identifiers and zero values are the return of the functions. Using the In Stacked Val function we could obtain a displaying that shows the execution of function id 3. The calls within function id 3 code are not displayed.

- **Nesting level :** This function *Stacked Val* function. Values greater than 0 are pushed, when the function receives a value equal to 0 pop up the top of the stack. The returned value is the number of stacked values (lenght of the stack) which correspods to the nesting level of the value that function *Stacked Val* will return if it was selected.

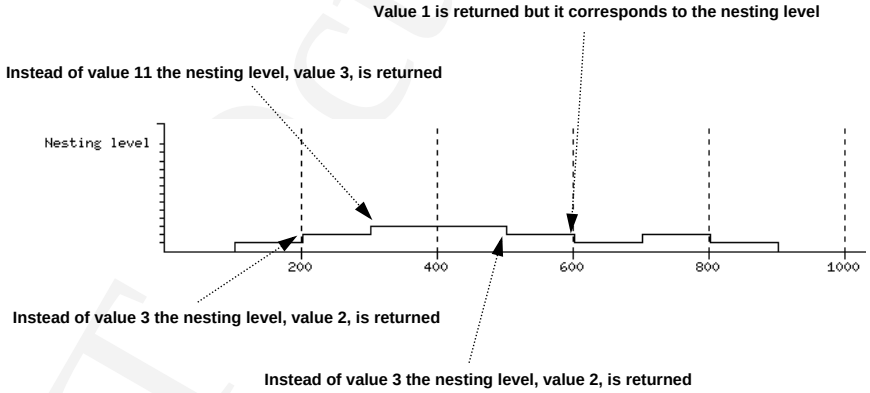


Figure 8.41: Nesting level function behaviour

As previous examples, supose that values are function identifiers and zero values are the return of the functions. Using the Nesting level function we could obtain a displaying that shows the execution nesting levels of the function stack calls.

8.4 Derived views

The Semantic Module offers the possibility to combine different trace views in order to obtain new views that could not be directly derived from trace records. For example, the IPC (Instructions per Cycle) metrics shown in section 8.1.4 could not be directly obtained from trace file records, we can combine the executed instructions and processor cycles counters that have been coded in the trace records to get it.

As we showed in section 8.2, the Semantic Module window when working with this type of views looks different (see figure 8.42). The process and resource model area where semantic functions of the active levels could be selected changes.

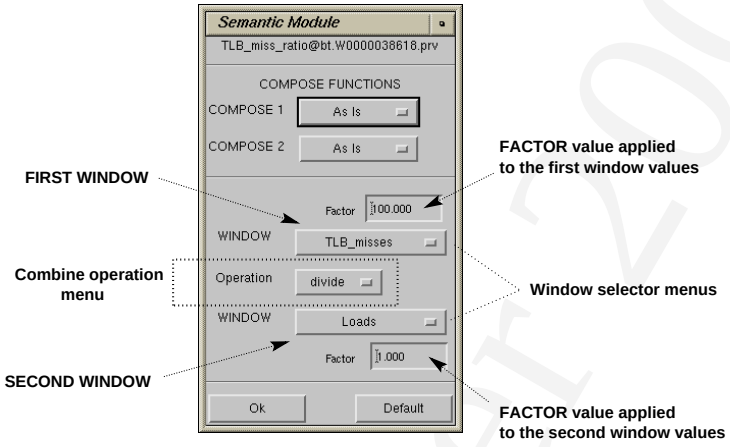


Figure 8.42: Semantic window format for a derived window.

Since semantic values are graphically represented on displaying windows (next chapter 9 will describe the displaying windows), combine two semantic hierarchies could be seen as the combination of two displaying windows (see figure 8.43). Section 10.1 on chapter 10 will describe how to create a derived window.

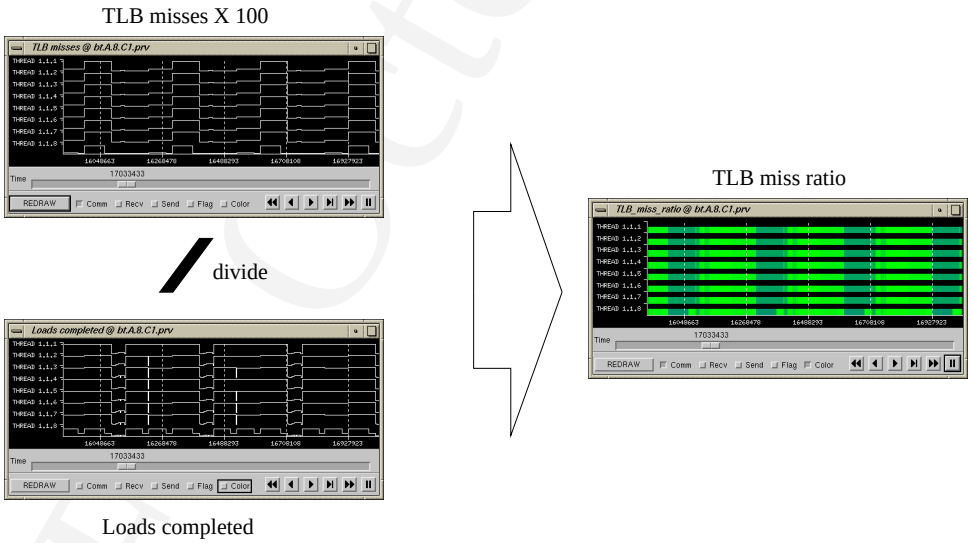


Figure 8.43: Obtainning a derived metrics from TLB misses and Loads completed

Chapter 9

Displaying Windows

The graphic representations of Paraver are pictured on the displaying windows. The windows show Gant timing diagrams of value evolution from the different status and events.

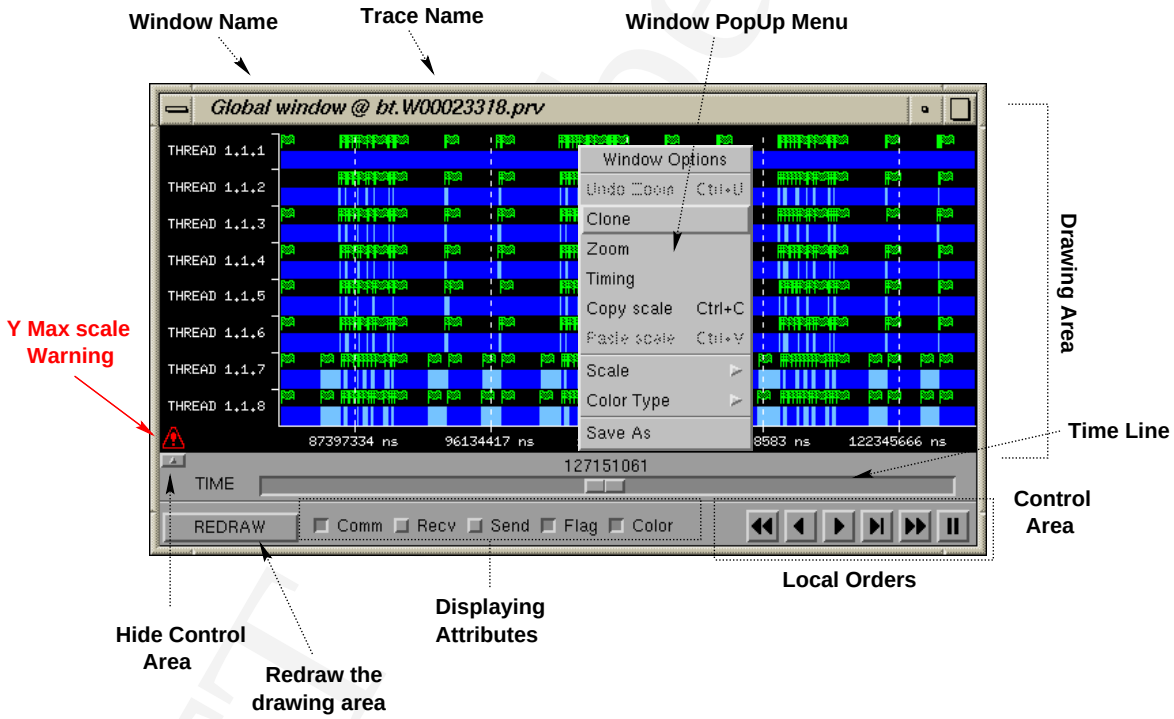


Figure 9.1: A displaying window

The Displaying Window has two areas: the **drawing area** where Paraver draws the representation of the objects (CPU, Tasks, ...), and the **control area**, at the bottom, where the user can manage that representation.

The Drawing Area has some hidden functions:

- the **click** utility (see Textual Module in section 10.2 on page 85): when user clicks in this area using mouse left button, a window is openned to display the trace records around that point. It shows a textual display of the values displayed in the window.
- select window: when user clicks in this area using mouse middle button, the focus is changed to the window. This implies that all Paraver windows (filter, semantic, visualizer, ...) are updated to the values selected for the window.

- popup menu: when user clicks in this area using mouse right button, a popup menu is displayed. This menu contains a quick access to the most common utilities.

The Control Area has several buttons to manage the drawing area: the time line, the local orders, the displaying attributes and the redraw button.

9.1 Control Area

9.1.1 Redraw button

Every time that a property of a displaying window is modified (for example, the filtered events) or the semantic function displayed but also turning on and off the displaying attributes, the drawing area must be redrawn to show the changes.

The REDRAW button is very useful because it lets the user to refresh the drawing area, taking into account the displaying attributes which have been modified.

9.1.2 Time Line

Figure 9.2: Time Line : Scale bar

The scale-bar shows the current time of the last trace displayed. It runs to the right most side of the displaying window, if you go forward, and to the left most if you go backward.

The scale-bar can be moved to shift the timing limits of the drawing area.

9.1.3 Local Orders

The Local Orders are used to manage the drawing area of the displaying window. It works like a tape player, where the trace file is the tape. All the information was sequentially recorded on the tape. Paraver can read this tape (trace file) and display its information.

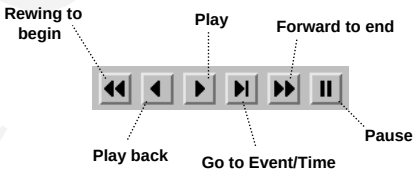


Figure 9.3: Local Orders : Tape recorder buttons

In accordance with the tape player design we use next typical functionalities:

Rewind

To rewind the trace file to the beginning. The beginning of the trace file will be displayed on the drawing area according to the scale that we are working.

Play Back

To move back (and display) in the trace file (scroll-back). When this button is pressed the drawing area begins to scroll back the trace file.

Play Forward

To move forward (and display) in the trace file (scroll-on). The drawing area begins to scroll on the trace file. The same problem than *Play back* could appear; to solve it see Scrolling Speed option in section 6.1.3 on page 28.

Go to Event/Time

To go to a given time or user event.

Figure 9.4: Go to Time or User Event

- Go Time : If the user wants to move to a certain point, Paraver moves the trace file to the time written in the Time text box. The time specified can be computed :
  - Rel : The time is taken like a Relative time to the current time.
  - Abs : The time is taken like an Absolute time (It is the default).
- Go Type : Move to the user event which "event" is equal to the text box "Type".
- Go Value : Move to the user event which "value" is equal to the text box "Value".
- And : Move to the user event which "event" and "value" are equal to the text box "Type" and "Value" respectively.
- Or : Move to the user event which "event" or "value" are equal to the text box "Type" or "Value" respectively.

If the user wants to find an user event, Paraver moves the trace file to the first occurrence of this event. The options let the user look for an event to the end, to the beginning, to the next or to the previous occurrence. Move to Event options are :

- $|\rightarrow$ Searching forward for the next event occurrence.
- $\leftarrow|$ Searching backward for the last event occurrence.
- $\rightarrow$ Searching forward for the next event occurrence.
- $\leftarrow$ Searching backward for the next event occurrence to the beginning.

The drawing area is centered in the selected point.

Forward to end

To go to the end of the tracefile. The ending of the tracefile will be displayed on the drawing area according to the scale we are working with.

Pause

To stop the tracefile (stop the scrolling).

> **In some X Servers the difference between the speed of the machine where paraver is executing is greater than the speed of the X Server which is processing the scrolling events, as a result, the X Server can't process all the scrolling and drawing requests from the application; when this happens the buttons loose sensibility and if you press for example the pause button, the animation doesn't stop until all the scrolling has been processed. This problem occurs because the application executes faster than the X Server. To avoid this problem see the Scrolling Speed option in section 6.1.3 on page 28.**

9.1.4 Displaying Attributes

The displaying attributes display a view of trace records. They can be mixed to get the most expressive displaying window. After each new updating the displaying area must be redrawn, so the user should press the "Redraw" button to see the final view.

Color attribute

The Color toggle change the visualizing representation (by default it is enabled). The **color** is the default representation where each visualized value has a specific color which can be modified in the *Colors window* (see "Visualizer" in section 10.1). Remember that the *Code* color scale is limited, so if there are values that do not match with any color, it should be selected the representation without color (see below).

If the color toggle is disabled the representation is shown as a function instead of color code (Figure 9.5).

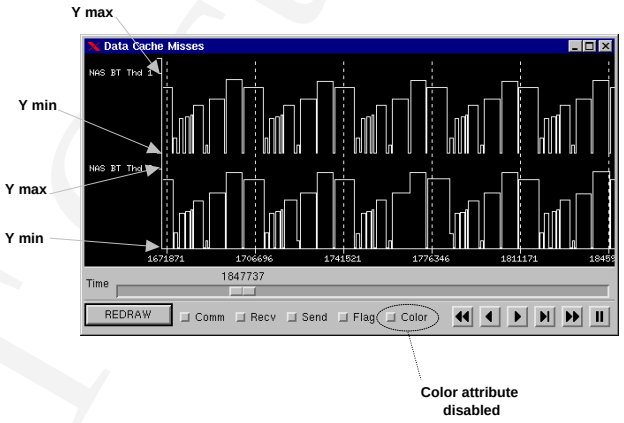


Figure 9.5: Window without Color: Function display

The representation without color is very useful when the range of values to show is very wide, or when the user wants to see the behaviour of a value in a time line.

The range of values are computed from the "Y max" and "Y min" values (see "Visualizer" in section 10.1). If the user does not take care about this, the view could be confused due to low precision in the drawing. Each displayed object are drawn between this limits, values less than Y min are painted as Y min, and values greater than Y max are painted as Y max.

Flag attribute

Paraver draws a flag icon each time that a user event appears when the Flag toggle button is enabled.

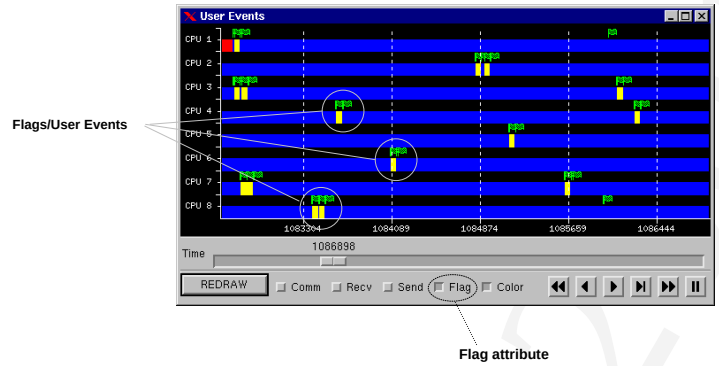


Figure 9.6: User Event Flags

These flags are colored by user event type. The user can decide which gradient color will be used to paint a user event type. By default, all event are painted using the first gradient color.

Comm attribute

When the communication attribute is enabled, Paraver draws a line each time a complete communication appears. The line is shown when the message is actually received. Its final time will be the lower time between the logical and physical receive.

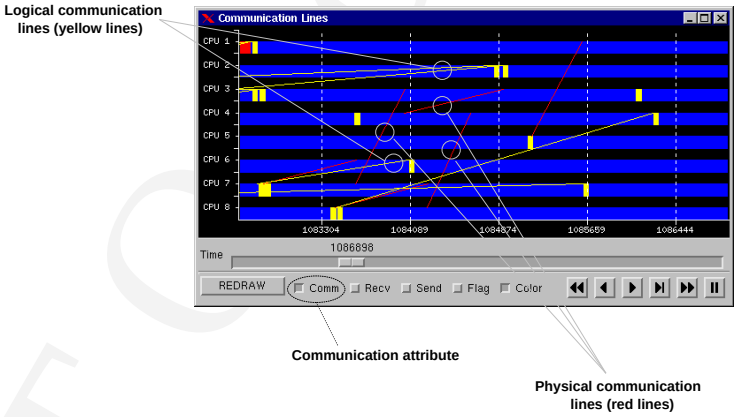


Figure 9.7: Communication Lines

Both types of communication (logical and physical) are drawn in different colors. By default, logical communications are painted in yellow and physical in red. These colors can be changed (see Changing System Colors in section 6.1.3 on page 28).

Send and Recv attribute

Communication can be shown as line or as icons like the user events. Paraver draws an incoming arrow icon each time that a receive record appears and an outgoing arrow icon each time that a send record appears.

The logical and physical communications have the same icons, but the are drawn with different colors like the communication lines.

Remember that logical communication means when the user wants to send/receive, and the physical means when the message is actually sent/received.

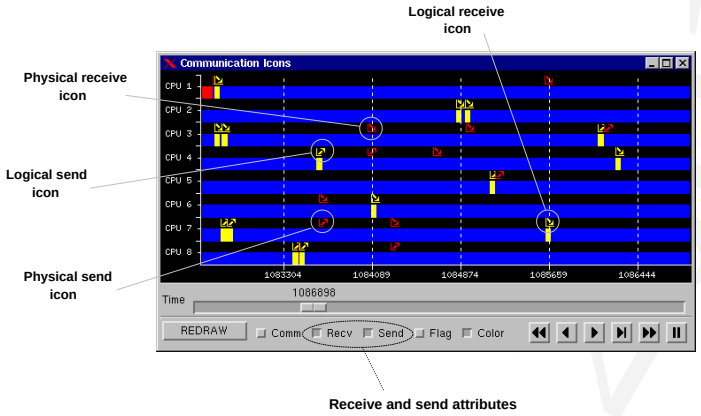


Figure 9.8: Send and Receive Icons

9.2 Window PopUp Menu

A click in the drawing area using the mouse right button will raise the window popup menu. This menu has the most common features used when working with Paraver windows.

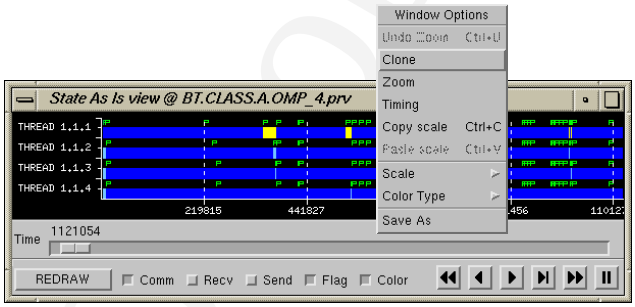


Figure 9.9: Window PopUp Menu

Clone menu option

The **Clone** menu option creates an identical copy of the window where it has the same scale, object representation, filtering options, semantic functions selected, ... The window name is the source window name plus the ending _cX to distinguish the two windows; where the X is a number to distinguish the different clones onto the same window.

Zoom menu option

Quick access menu option to the **Zoom utility**. The Zoom utility is used to magnify a specific part of the displaying window, go to the section 10.1.4 on page 83 to see a detailed description.

Undo Zoom menu option

Undo the zoom that has been done restoring the scale and window limits from original window. This button is disabled if last zoom has been undoned or no zooms have been done on that window. Go to the section 10.1.4 on page 83 for the **Zoom utility** details.

Timing menu option

Quick access menu option to the **Timing utility**. The Timing utility offers the possibility of measuring a specific part of the displaying window, go to the section 10.1.4 on page 83 to see a detailed description.

Copy/Paste scale option

The **Copy/Paste scale** option copies the window limits (begin time and end time) and window X-scale to another window. Select the **Copy scale** option (Ctrl+C) in the source popup window menu and the **Paste scale** (Ctrl+V) option in all the target window where scale have to be copied. The target window will be redrawn with the new limits. The copied limits are saved so the paste operation could be applied to different windows.

Scale menu option

The **Scale menu** option is used to change window scale values. The three functions that acts onto the scale, modify the scale value and redraw the window to display the new scale value.

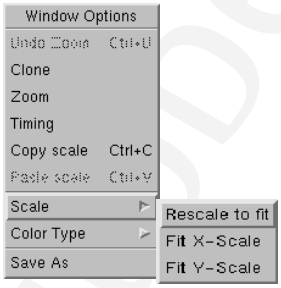


Figure 9.10: Scale Menu

- Rescale to fit: Computes the X scale value to fit all window (useful when window width has been resized). Figure 9.11 shows how after applying the Rescale to fit option the same displayed section is showed but rescaled to fit all the drawing area.

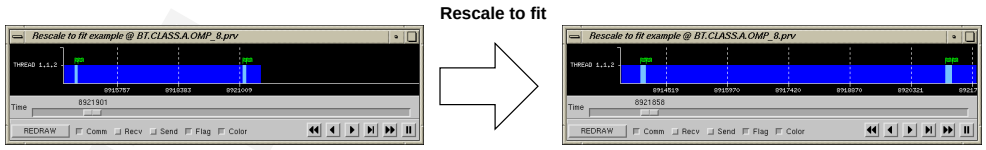


Figure 9.11: Rescale to fit example

- Fit X-Scale: Computes the X scale value to fit all the trace file in the displaying window. After the X scale is computed, the window is redrawn showing all the trace file.
- Fit Y-Scale: Computes the Y maximum scale value to fit all displayed values. After the maximum is computed, the window is redrawn with the new Y maximum value.

Warning message : If on the left bottom corner of the drawing area appears a red exclamation sign like showed in figure 9.12 it means that something could be wrong in the displaying. Click the symbol to see what happens.

In figure 9.12 the warning symbol is advising that Y maximum scale does not fit all the values and user should recompute it.

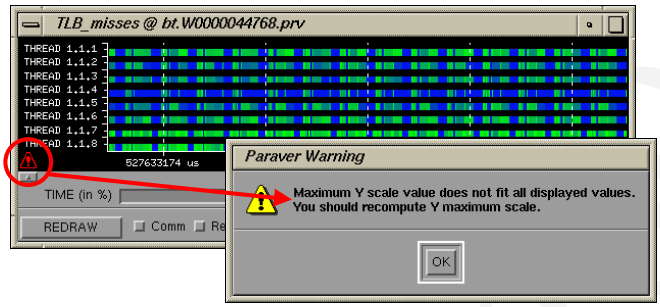


Figure 9.12: Warning message

Color Type menu option

If we use a color visualization, by default the **code colors** are used. If values are greater than available code colors, we usually use a visualization without color where values are viewed as a time line (see section 9.1.4).

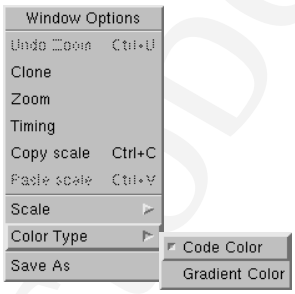


Figure 9.13: Color Type Menu

Paraver offers another type of color visualization, the GRADIENT VISUALIZATION.

When using the gradient visualization, values between **Y max,min** (see section 10.1.2) are painted using the gradient colors. Values between the selected scale (from Y-min to Y-max) are grouped into several groups (one for each gradient color).

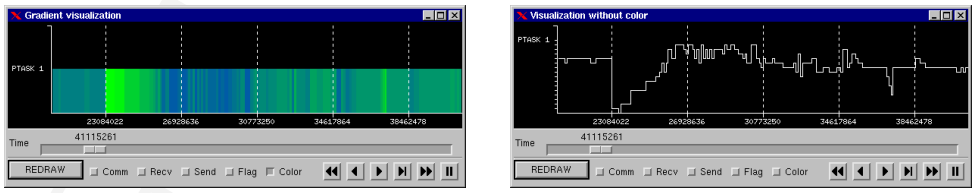


Figure 9.14: Gradient visualization

Lower values are painted with a lower gradient color, upper values are painted with an upper gradient color. Figure 9.14 shows a gradient visualization versus a non color visualization. Note that dark regions correspond to the upper values and light regions are lower values.

The Color Type menu option lets us to change the color scale used. Select the **Code color** option to use the code colors and select **Gradient color** visualization to the gradient visualization.

Save As menu option

The Save As menu option lets to save the window to a file. The file where windows are saved are known **window configuration files** (see next section 9.3 for a detailed description on window

configurations).

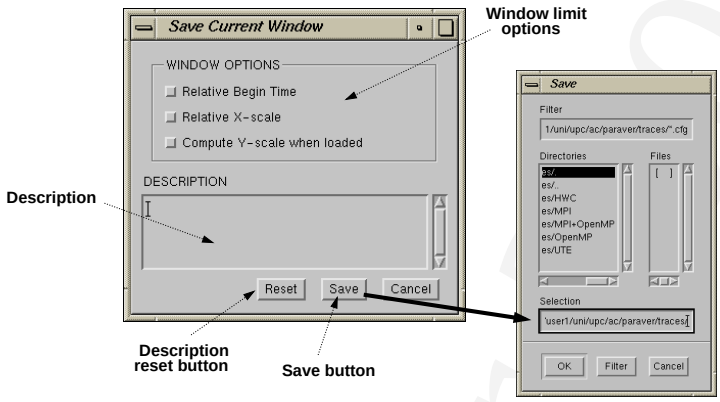


Figure 9.15: Save As menu option

When **Save As** option is selected ,a window is raised to select some saving options and to offer the posibility to write a description of the file (see figure 9.15). The WINDOW OPTIONS and the DESCRIPTION usefulness of the text are described in next section.

To proceed the saving, press the **Save** button to raise the selection box to select the file where window will be saved (see figure 9.15).

9.3 Window Configuration Files

Paraver is a flexible tool that lets the user drive into the application analysis. Different type of application views could be extracted from the tracefile depending of application type and tracing parameters. Sometimes, create these views could be a tedious process because more than one parameter selection is involved. The problem is increased when working with derived metric, different base views are involved to create the derived one.

Paraver offers a way to save the created view to files in order to load it later for the same or another trace. Files were windows are saved are known as **window configuration file**. Later, the same view could be loaded and the created window will display all the same selected parameters as when it was create. Since window configuration files could be used to load the view into a different trace file, Paraver offers a way to adjust the view to the new trace duration. Create the desired trace view is as easier as load a file.

Windows could be saved by selecting the **Save windows** option in Configuration menu (see figure 9.16) or the **Save As** menu popup option. The **Load windows** option lets to load them.

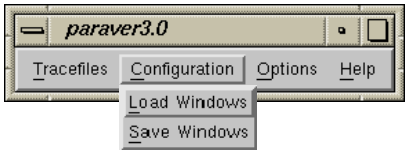


Figure 9.16: Configuration menu

What is a Window Configuration File (.cfg) ?

The **window configuration files** are used to save the information related to one or more paraver windows in order to load them the next time. These files ends using the extension **.cfg**.

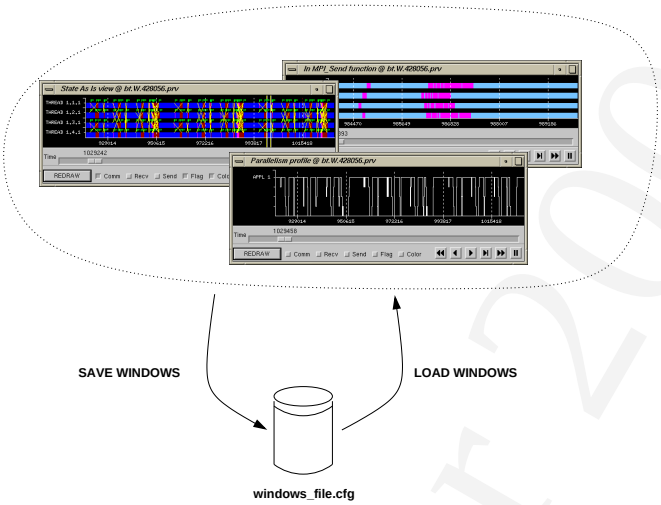


Figure 9.17: Window files are used to save created windows to a file.

The main goal of window configration files is to obtain created views another time by only loading a file (without the needed to reconstruct it).

The next sections (9.3.1 and 9.19) will explain how to save and load the displaying windows.

9.3.1 Saving paraver windows

To save windows in a **window configuration file** when working with Paraver, go to the menu option **Save windows**, the figure 9.18) will be raised in order to save them.

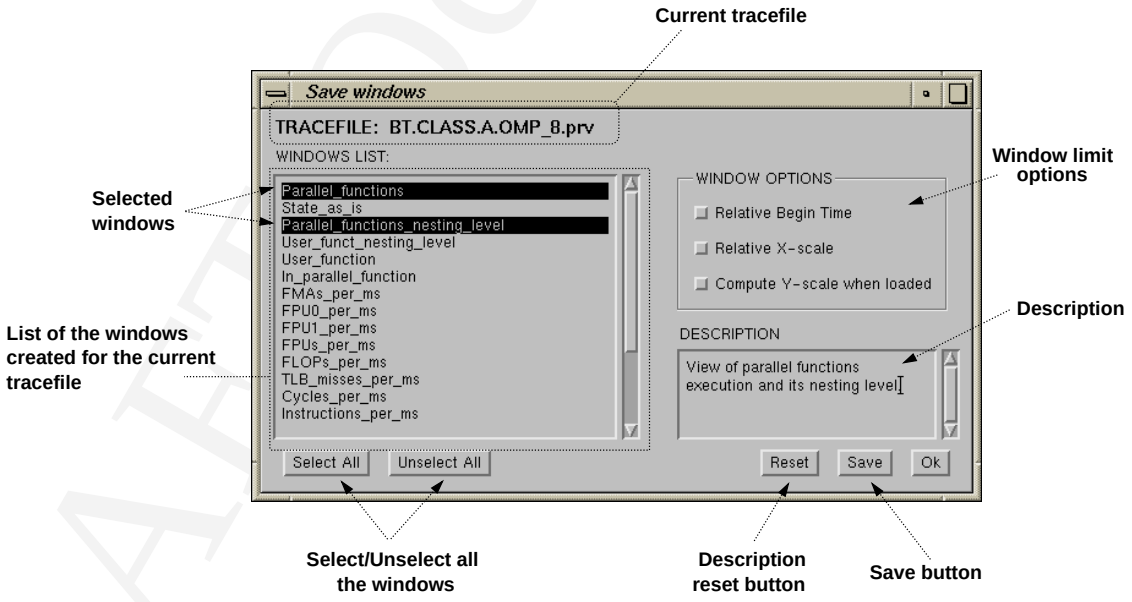


Figure 9.18: Window to save the paraver windows.

The name of the current tracefile and a list of all the created displayiong windows for that tracefile are listed. Select the windows by selecting/unselecting their names and click the SAVE button; it will raise a file selection box to select the file where windows will be saved. The SELECT

ALL/UNSELECT ALL buttons can be used to select/unselect all the windows. When all desired windows have been saved, close the window by cliking the OK button.

Since window configuration files can be used to load them into trace files with different sizes and durations, some window options could be added to the file in order to make them more general.

WINDOW OPTIONS

Working with window configuration files is very useful because the user could load the predefined desired trace views. In order to make them more flexible we have added some options that could be specified at saving time. By enabling them, the configuration files could be used in different trace files because limits are saved relative to the current trace limits. The options that could be specified are:

- Relative Begin Time: If it is enabled, the next time that configuration file is loaded the beginning window time is relative to the new tracefile duration. For example, if the beginning time of the window was about the 10 % of the trace duration, when it is loaded on a different trace, its beginning time will also be about the 10 % of the new trace duration.
- Relative X-scale: If it is enabled, the next time that configuration file is loaded the X scale window is relative to the new tracefile duration. For example, if the window window was showing about the 30 % of the trace duration, when it is loaded on a different trace, it will also show about the 30 % of the new trace duration.
- Compute Y-scale when loaded: If it has been enaled, the next time that configuration file is loaded the maximum Y scale value is computed in order to fill all displayed values. This option is useful when the range of all displayed values will change (for example, a window displaying the data cache misses profile easely can change the range of displayed values on another trace file).

DESCRIPTION

The user can add a description to the file in order the describe the windows contained in that trace. File description is displayed at loading time (see next section **Loading paraver windows** 9.3.2).

9.3.2 Loading paraver windows

To load a window configuration file, go to the **Load windows** menu option, the figure 9.19 will be raised.

The window lets to browse the directories and window configuration files. On the left top of Load windows window (see figure 9.19) there is a menu which contains the list of all the loaded traces, it used to select the tracefile where loaded files will be applied.

User defined directories

Next to the Load windows window, the User directories popup menu contains a list of directories where there are the window configuration files. The user could specify a set of directories where he/she has created some window configuration files. The user must to specify them by using the environment variable **PARAVER_CFGS_DIR** (see the **Environment variables** description on page 109).

The window configurations directory provided with Paraver distribution also is added. The SELECT DIRECTORY enables the user to browse the directories, by selecting this option, the user could specify the desired location in CURRENT DIRECTORY label.

The CURRENT DIRECTORY is shown over the file selection list. Window cofiguration files are shown in bold face (note, that the extension has been removed from the filenames).

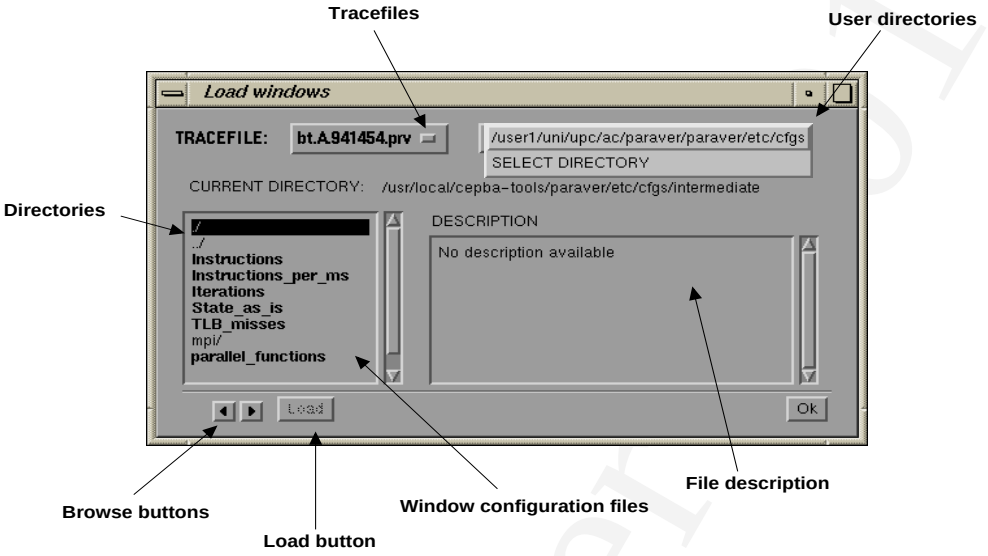


Figure 9.19: Window to save the paraver windows.

Browsing the files

Only window configuration files (files with extension **.cfg**) and directories are displayed in the list. Window configration files are shown in bold face format without the **.cfg** extension. Directory names end by using the slash character ("/").

The user could browse the different directories by double clicking the directory name or using the browse buttons placed in left bottom.

When a window configuration file name is selected, its description is showed in **description** text box. Load it by double clicking it or by clicking the **Load** button.

Derived windows combine different trace views in order to obtain a new one. In fact, derived windows, combine two or more displaying windows. When they are saved to a window configuration file, all its structure is also saved. When they are loaded all the structure is also created, so all dependetn windows are created but only the top window is popped up. The rest are closed until **Open** button in Visualizer Module (see chapter 10) is pressed to raise them.

Chapter 10

Representation Module

Once the trace file is processed by the Filter and Semantic modules, the computed time dependent values are passed to the **Representation Module**. This module is the responsible to give a graphical visualization and a very detailed qulitative analysis of these values.

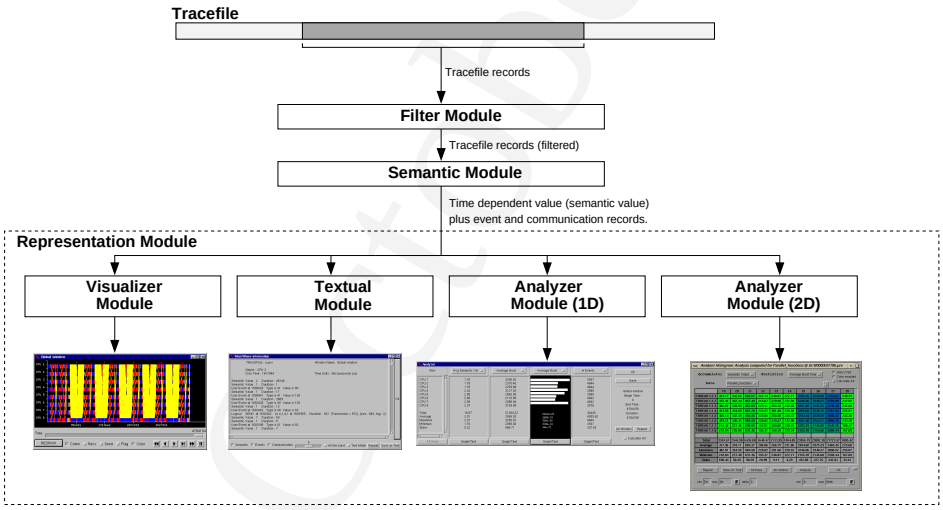


Figure 10.1: Paraver Internal Structure

The Representation Module is composed by three modules : the visualizer module, the textual module and the analyzer module (see figure 10.1)

- The **Visualizer Module** (section 10.1) will give us a graphical visualization of the trace file. It is responsible to manage the displaying windows (see chapter 9) and some utilities to work with, like *Zooming* or *Timing*.
- The **Textual Module** (section 10.2) give us a textual representation of the record traces around specific points. It displays in text mode what we are seeing in a displaying window.
- The **Analyzer Module** (section 10.3) lets us to get summarized information. Very detailed qualitative analysis can be done by properly selecting the Filter and Semantic modules combined settings.

Except the Textual Module, which is accessed by a click into the displaying window, the different Visualizer and Analyzer Module utilities could be accessed through the Global Controler window (see figure 10.2).

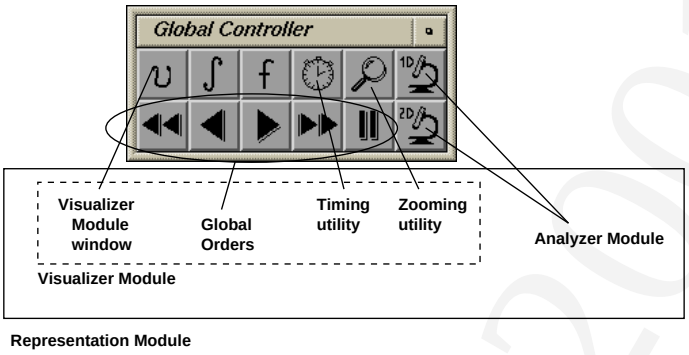


Figure 10.2: Representation Module functions at GlobalControler window

10.1 Visualizer Module

The visualizer module allows the user to display the computed values by the Semantic Module in a displaying window (see chapter 9 for the *Displaying windows* description).

Each window shows a particular view of the trace file with its time interval, scale, object representation and even its trace file.

In the Visualizer module could be included other utilities like *zooming*, *timing*, ... that let the user to work with the different trace views.

10.1.1 Visualizer Module window.

When a trace file has been loaded, if the user press the visualizer icon (Figure 10.3) in the *Global Controller* window, the **Visualizer Module** window is raised (see figure 10.4).

Figure 10.3: Visualizer icon

The **Visualizer window** is used to manage and modify the *displaying windows*. We can select which type of object will be displayed, which window will be the current one, its parameters, its time units and the trace file that will display. We can create and destroy windows, clone, copy,...; these operations will be explained in detail.

The *Visualizer Module* window is composed by five areas plus a group of buttons to manage the displaying windows. These five areas are:

- First, at the left top corner, the Resource/Process Levels area composed by the *Level* button and some *toggle* buttons is used to select the object type that will be displayed in a displaying window. The **Resource/Process Levels** button hides a window that will be raised when this button is pressed (see below, in section 10.1.3).
- Next to the *Resource/Process Levels area*, the Window Browser lists the names of all created displaying windows. To select a displaying window, the user just has to click the name of that window and it will be considered like the **current** one.
- Between, the *Window Browser* and the *Time Units* area, there is the Values area where there are some values attached to a displaying window. These values are the *window name* (that could be modified), the *X scale* (number of microseconds by pixel, explained below) and the *Y scale* (see section 9.1.4 on page 66). The **Values** button has a hidden functionality explained below.

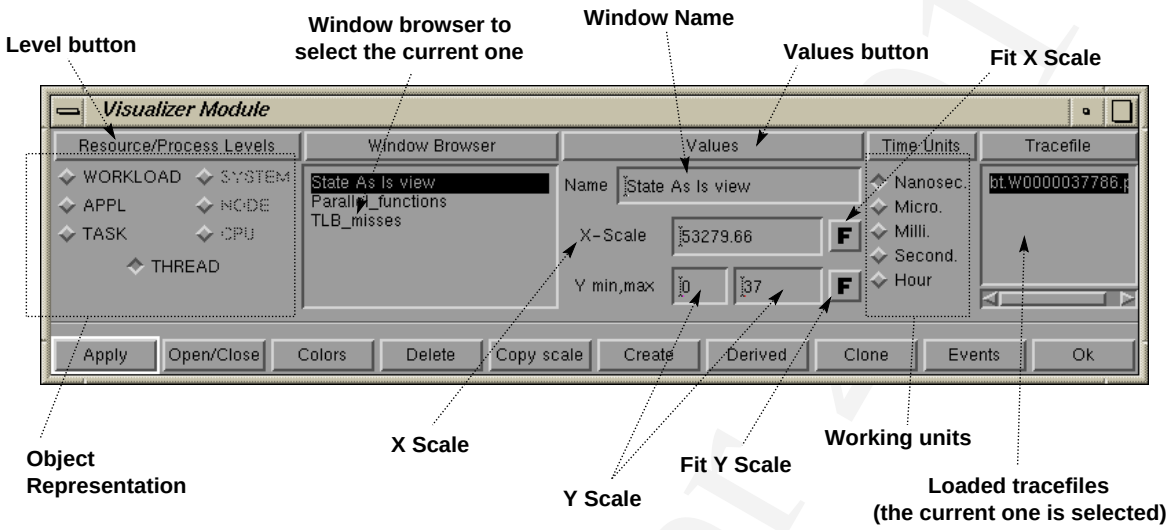


Figure 10.4: Visualizer window (Visualizer Module)

- The Time Units area lets to select the time units which a displaying window is going to work.
- On the right side of the window, the Trace file area lists the names of all the trace files that has been loaded, the current one is highlighted. To change it, the user just has to click the name of the trace file and it will be considered the current one.

At the bottom of the window there are some buttons to manage a displaying window. Section 10.1.2 will describe them.

The Visualizer Module parameters will affect to the displaying windows. Displaying windows:

- have a window name : The *Window name* text box at Values area sontains the window name that will appear at the top of the displaying window and in the window browser list. By default, paraver gives a different name for each window but the name doesn't identify the window. It could be possible to have different windows using the same name.
- are working on a **object representation level**. Seven object levels could be selected: THREAD, TASK, APPL, WORLOAD, CPU, NODE or SYSTEM. On the left side of the Visualizer Module window there is the *Resource/Process Levels area.* This is the visualization object in accordance to the object model that we are going to work. By default, the THREAD level is selected. Remember, that the level where window works will affect on the hierarchy of functions applied at Semantic Module.
- are working in a specific scales:
  - X scale : The X Scale is the number of microseconds by pixel. If it is a high number we will be working with a higher scale; when the scale number is reduced a more detailed visualization will be displayed. By default, this parameter has its own value and it is rarely modified by the user. When the user creates the first window, paraver puts a value to fill all the trace file in the displaying window and the *Zooming utility* computes the new value for the zooms.
    If the $\boxed{F}$ button next to the X scale is pressed, Paraver will change the X scale to a value that fits all the trace file in the displaying window.
  - Y scale : minimum (Y min) and maximum (Y max) of representation limits. If the value is out of these boundaries then an overflow/underflow occurs and Paraver draws the maximum/minimum value. The Y scale is very useful when we are working with a no color visualization (see section 9.1.4 on page 66).

If the [F] button next to the Y scale is pressed, Paraver will change the Y maximum scale to a value that fits all displayed values.

- with specific time units. Although trace is specified in microseconds precision, the user could select the desired time units just selecting its toggle button. There are four options : **Micro.** (microseconds), **Milli.**(milliseconds), **Second** and **Hour**. When the Textual and Analyzer modules will work onto the window, they will use the time selected units.

- and are displaying a view of a specific trace file. On the right side of the Visualizer Module window there is a list with the loaded trace files. User should select the trace file that will be displayed in the window.

10.1.2 Visualizer Module window buttons

Paraver lets to create, destroy, clone,...a displaying window. These operations could be done through the managing buttons at the bottom of the *Visualizer* window. We are going to see how those operations could be done.

Creating and destroying displaying windows

- CREATE button: Before creating a new window some parameters should be checked. These parameters will affect to the displaying window.

  When all the parameters have been checked you can create the new window just clicking the Create button. It is possible to generate as many windows as the user wants, just limited by the memory of the X-server.

  Also, new windows could be created using the *Zooming utility* and the Clone button (see below).

  The values filled on the text boxes can be cleaned when the Values button bar is pressed. The defaults could be filled by clicking this button bar again.

- DERIVED button: The **Derived button** creates a derived window that is a combination of two other source windows.

  To create it, thw two source windows have to be seleceted in the **Window browser** list (see figure 10.5)

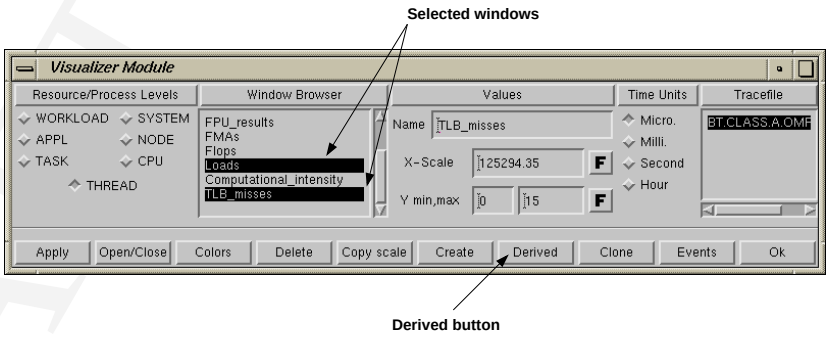


Figure 10.5: Creating a derived window. Visualizer module.

Select the first window by clicking its name in the Window browser list and just by pressing the CTRL key, select the second one by se;ecting its name. As shown in figure 10.5, the two window names have to be selected and just by pressing the Derived button, the new window will be created.

- CLONE button: When the user wants to make an identical copy of a window, the CLONE button should be used. To create the new window, select the source window like the current one and press the CLONE button. This will create an identical window, where it has the same scale, object representation, filtering options, semantic functions selected, ...; the window name is the source window name plus the ending _cX to distinguish the two windows; where the X is a number to distinguish the different clones onto the same window.

- DELETE button: To destroy a window, select the window that should be destroyed and press the DELETE button. The window will be destroyed and it will disappear from the window browser.

Managing the displaying windows

- APPLY button: Paraver lets us to modify the selected parameters of a created displaying window. If you want to change any parameter that you had filled, change it and press the APPLY button, the new selected parameters will be applied to the window and the drawing area will be rebuilt. Every time that you modify the parameter of a window you must apply the changes.

- OPEN/CLOSE button: The OPEN/CLOSE button hide and unhide a displaying window. When a opened displaying window is closed it disappears from the screen but it remains created, also, its window name remains in the window browser. When you want to unhide a closed window, select it like the current one in the window browser and click the OPEN/CLOSE button; the window will be opened.

- COPY VALUES button: To copy the current displaying window parameters to another created window, select the source window.

  Press the COPY VALUES button and then, select the target displaying window by clicking another name in the **Window Browser** column. It works like the cloning button but when copying values the target window must be created.

Defined Event TypesValues window. *Events button*

The *Events button* raises the **Defined events window** (see figure 10.6). This window shows a list of the defined event types and all their defined values for the current tracefile.

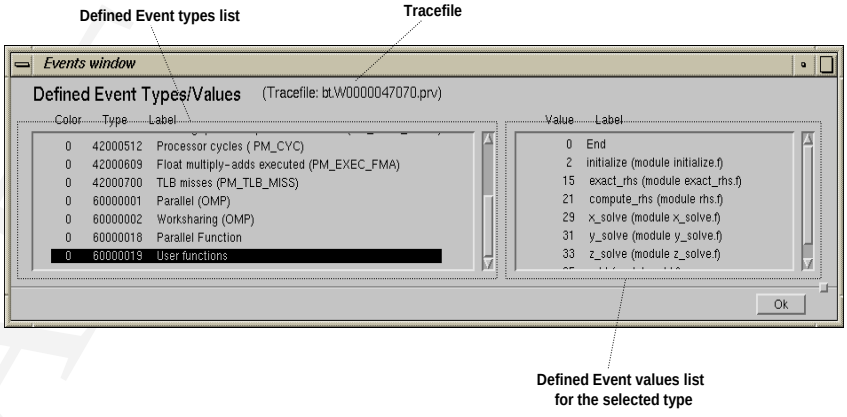


Figure 10.6: Events window

The event type list shows the gradient color number which the flag will be painted, the event type number and its associated label. Select a type by clicking, and their defined values will be displayed.

The user can define the labels for each event type and his defined values using the **Paraver Configuration Files** (see **Trace Generation** document at URL http://www.cepba.upc.es/paraver).

Changing window colors. *Colors button*

The **Colors** button raises the Colors window where the user can change the code or gradient colors, used for **Code** and **Gradient** displaying. The user can change the displaying mode and the RGB values of each color.

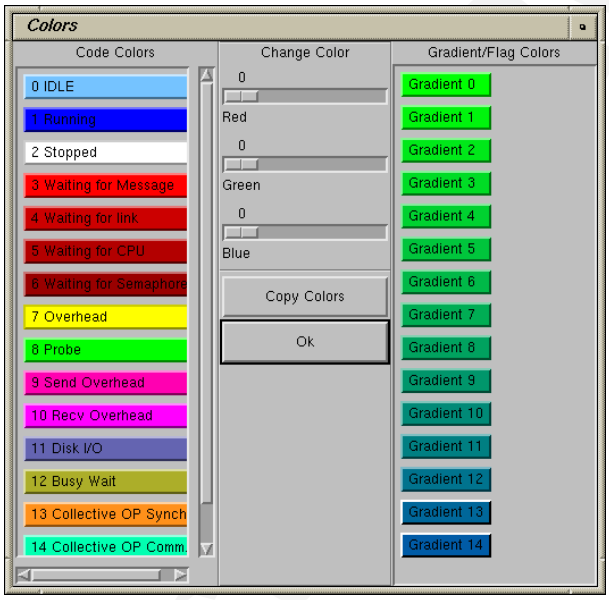


Figure 10.7: Color Menu: "Code" and "Gradient" colors

Change colors by clicking on the desired one and move the scale bars of the basics (R, G, B). To recover the color just double click on the desired color. To copy a color click the source color press the COPY COLORS button and click the target color. The change of colors will affect all the displaying windows.

The labels of code colors and gradient colors can be modified to get a correct information about the meaning of each one for the user. Also, the code colors and gradient colors can be defined in the **Paraver Configuration File** (see **Trace Generation** document at URL http://www.cepba.upc.es/paraver).

10.1.3 Selecting objects that won't be displayed. *Level button*

Each displaying window will be associated with a level of visualization. This level fixes the type of object to work with and what will be displayed.

By default, paraver works with all the objects in a level and all are displayed in a window. Through the *Object Selection windows* Paraver offers a way to select the objects that the user wants to see.

The Level button (figure 10.4) raises a window where the user could select and unselect the objects that will be displayed in the drawing area according to the trace file and level where the window is working. This window is called the **Objects window** (figure 10.8).

Also, these windows have another utility, they lets to change the object names that appears in the Y axis. By default, paraver has a default name for each object. For example, the thread objects are named with his number of application, number of task and thread identifier within the task (THREAD 1.4.1). The user can change these names with his/her own.

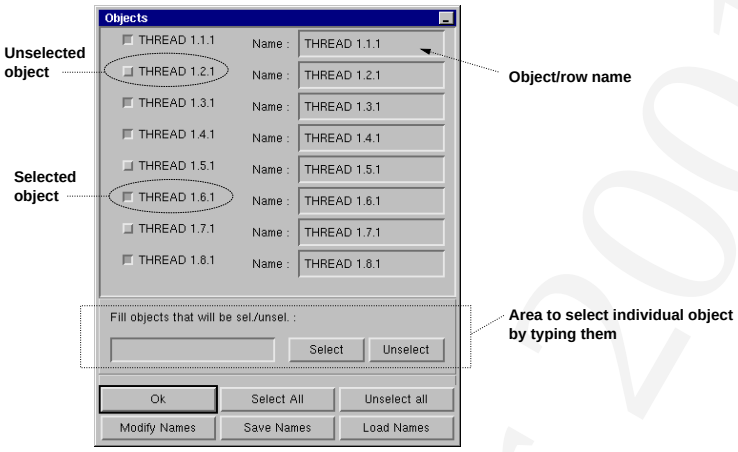


Figure 10.8: Level button: Objects window

The Objects window (figure 10.8) is composed by a row of objects where we can select if an object will be displayed or not (toggle buttons) and its associated object/row names (that could be changed). Also, we can select/unselect individual objects or a range of objects by typing them. All these functionality will be explained in next points.

Selecting/Unselecting displayed objects

The user can select what objects will be displayed. This selection match with the object list in the Y axis of the drawing area. By default, paraver displays all the objects in a level. To unselect the objects that won't be displayed disable their toggle buttons and click the Apply button on the *Visualizer* window to apply the changes.

There are two buttons to help us when selecting all or unselecting all the objects. To select all the objects, click on the button Select All, and to unselect all the objects, click on the button UnSelect All.

Also, to make easier the object selection there is a text box where the user can fill the objects that will be selected, single objects or intervals, and select/unselect them only by clicking select/unselect button.

- APPL and NODE levels: since APPL and NODE levels are at the top of the hierarchy, we can select the objects only by typing its number:
  - selecting single objects : 23,24,25
  - selecting intervals : 23-25,40-43
  - both, single and interval selections : 23,24-26,30,31
- TASK level: when working at TASK level we have to select the task by typing its application and its task number within the application like:
  - selecting single objects : 1.3,1.4,1.5
  - selecting intervals : 1.3-1.5
  - both, single and interval selections : 1.3,1.4-1.7,2.1,2.3-2.6
- NODE level: when working at CPU level we have to select the processor by typing its node and the processor number within the node like:
  - selecting single objects : 1.3,1.4,1.5

  - selecting intervals : 1.3-1.5
  - both, single and interval selections : 1.3,1.4-1.7,2.1,2.3-2.6

- THREAD level: selecting/unselecting objects by typing them when working at THREAD level implies to type the application, task and thread identifiers like :
  - selecting single objects : 1.1.3,1.2.1,1.3.1
  - selecting intervals : 1.1.3-1.2.1
  - both, single and interval selections : 1.1.3-1.1.7,2.1.1,2.1.3-2.1.5

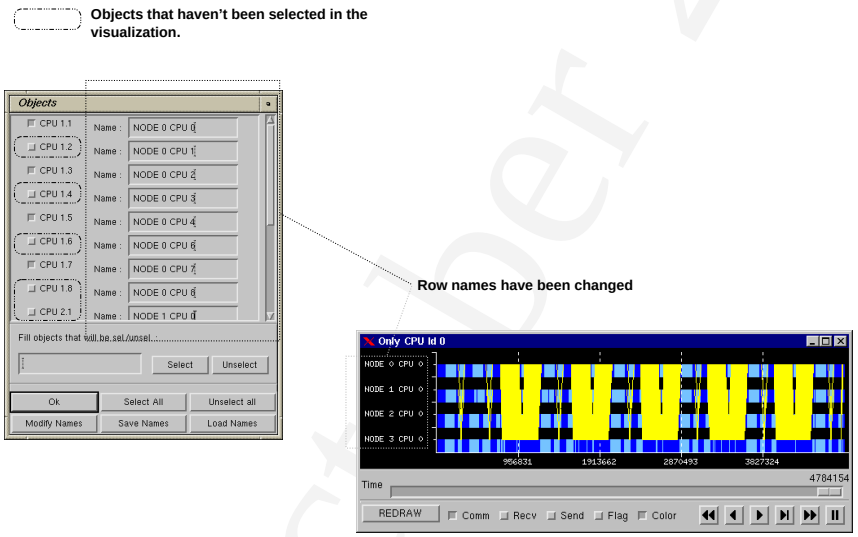


Figure 10.9: Objects window: Selecting/unselecting objects and changing its name

Modifying the object names

The user can modify the object names for the current trace file. These names will be displayed in the Y axis of any displaying window and in the analyzer window. To change them, in the Objects window a text box appears next to each select/unselect object button, which contains the row name, fill in each of those text box the new names for each object and when once finished press the button Modify Names in the **Objects window** (figure 10.9). Then, Paraver apply the changes to all the displaying windows of the selected trace file and stores internally the new names. The new windows that will be created for this trace file will have the new names.

- The button Save Names allows the user to save the current names for the trace file. Press this button to store the names into a file. It will raise a file selection box to select the filename where those names will be stored. By default, paraver offers a filename which is composed by the trace file name plus the .row extension. If the names are saved in this configuration names file offered by Paraver, the next time when the trace file will be loaded the names configuration file will be loaded at the same time.

- The button Load Names allows the user to load a configuration names file while Paraver is running. The new names are applied automatically after the load of this file.

Note, that a configuration names file are specific for a trace file, if this file is reused for other trace files, they must have the same number of objects in each level.

10.1.4 Working with windows

An easy way to measure. Timing utility.

This utility offers the possibility of measuring a specific part of the displaying window. It computes the elapsed time between two points in the displaying window. Both points, the beginning and the end, can be selected from any displaying window.

Figure 10.10: Timing icon

To measure a specific part on the displaying window (see also figure 10.11):

1. Click onto the **Timing icon** (figure 10.10) on the *Global Controller* window. The **Timing** window will appear and when the cursor passes over a displaying window it looks like a vertical line.

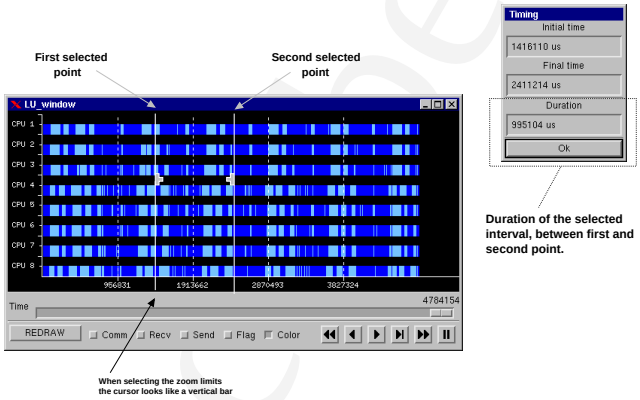


Figure 10.11: Paraver Timing Utility

2. Select the initial and final points in the drawing area of the window. Note that when cursor is moving, the text boxs on **Timing** window are changing to the time where the cursor is located.

3. Timing results will be written in the the **Timing** window The *Initial and Final time* text boxs show the initial and final points. The *Duration* text box shows the duration of the selected interval in the units where window is working.

This Timing tool is very important to provide quantitative measures and it is implicitly used by several utilities like the zooming or even the static analyzer.

Magnifying specific parts. Zooming utility

This utility offers the possibility of magnifying a specific part of the displaying window. It works like the *Timing utility* but the result is a new detailed zoom of the selected area.

Figure 10.12: Zooming icon

To magnify a specific region of a displaying window (see also figure 10.13):

1. Click onto the **Zooming icon** (figure 10.12) on the *Global Controller* window. The **Timing** window will appear and when the cursor passes over a displaying window it looks like a vertical line.

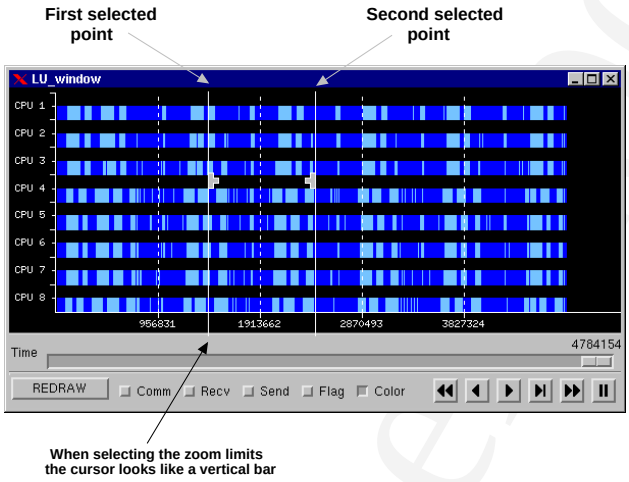


Figure 10.13: Zooming utility : Source window

2. Select the initial and final points in the drawing area of the window using the **left mouse button**. Note that when cursor is moving, the text boxs on **Timing** window are changing to the time where the cursor is located.

3. When the initial and final points have been selected, Paraver creates a new window with the same characteristics, (except the scale and name) that contains a zoom of the selected area (figure 10.14).

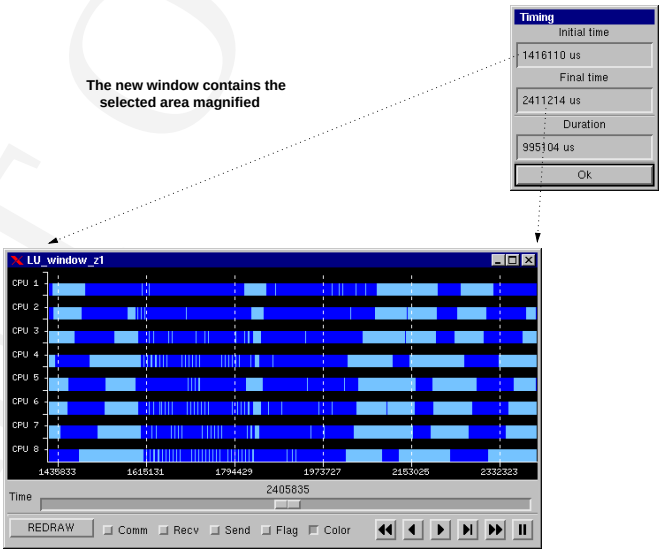


Figure 10.14: Zooming utility : Zoomed window result

A new window entry is added in the *Window Browser* on the Visualizer Module window. The name of the new window is like the previous one but adding the ending $\_zX$ which told us that this window is a zoom. The X is a number which tell the zoom level number.

If the initial and final points are selected using the **middle mouse button** the current window is reused to fill the new zoomed view. Thus, there is not created a new one. When a window has been reused to fill the new zoomed view, the zoom could be undoned by selecting the **Undo zoom** window popup menu option (see section 9.2 on page 68).

Work with all the windows at the same. Global Orders.

The buttons in the *Global Controller* window, which look like tape recorder buttons, make the same functions as the displaying window buttons. The difference is that all these functionalities have a global action, all the displaying windows are managed together.

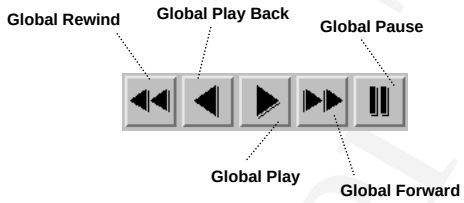


Figure 10.15: Global Controller Buttons

For instance, if you press the global **Play**, then all the displaying windows will make a **play** too. Therefore, you will see all these windows scrolling at the same time.

The Global Orders are used when the user wants to compare the evolution of two displaying windows at the same time.

10.2 Textual Module. What/where information.

Paraver offers a simple and quick way to get information about the tracefile from the displaying window. A click in the drawing area using the left button (see chapter 9 on page 63) gets a textual display of the displayed values around the selected point.

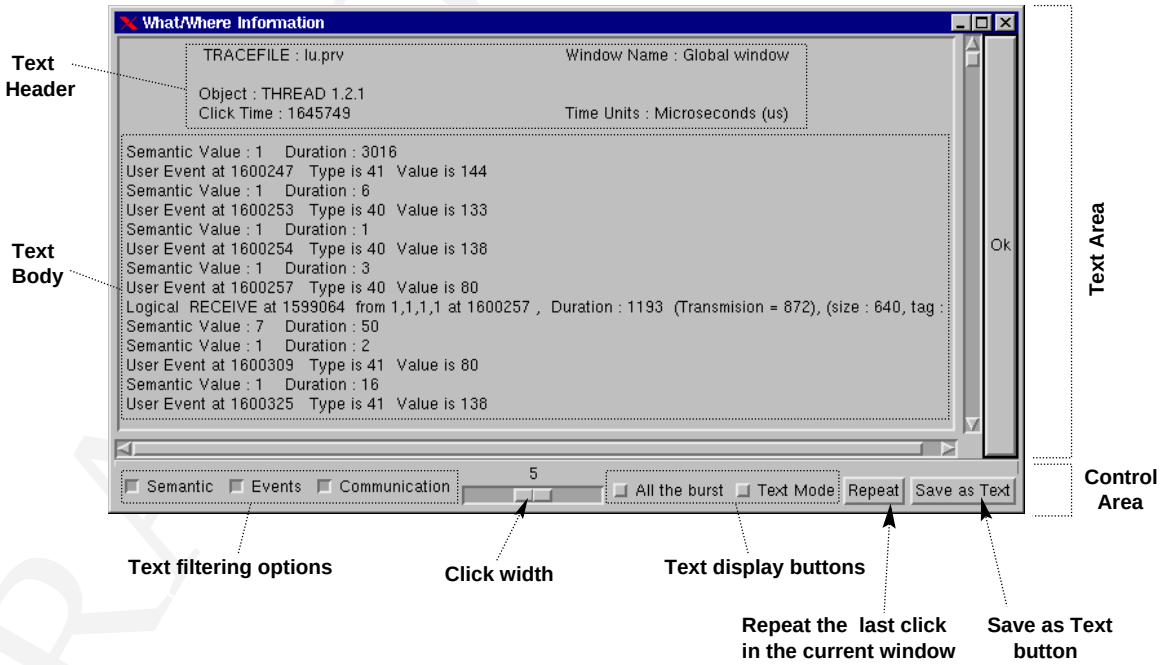


Figure 10.16: Paraver What/Where Information

This textual information is displayed in a window called *What/Where window* (Figure 10.16) and is composed:

- by the **Text Area**, where the textual information is displayed.
- and the **Control Area**, where the user can select what he/she wants to see.

This window is resizable and has scrolling bars to shift the information. The textual information displayed in the Text Area is composed by a header and a body which have a predefined format.

10.2.1 Text area

Text header

The text header gives information about where the click has been done. It shows the *tracefile name*, the *window name*, the *object* where the click has been done and the *timing point* selected. Also, the header shows the current window *time units*.

TRACEFILE : *tracefile name* Window Name : *window name*
Object : *object*
Click time : *timing point* Time Units : *current time units*

Text body

The text body shows the trace information around the selected point. There is one line for each type of textual information : semantic values, events and communications. The information line format for each type is :

- Semantic value format line is :

  ```
  Semantic Value : <value>    Duration : <burst time>
  ```

  where the *burst time* is the duration of the burst with the same semantic *value*.

- Event format line is :

  ```
  User Event at <time>    Type is <type>  Value is <value>
  ```

- Communication format line is :

  ```
  Logical/Physical SEND/RECEIVE at <time> to/from <object> at <time>,
  Duration : <communication time> (Transmission = <transmision_time>),
  (size : <size>, tag : <tag>)
  ```

10.2.2 Control Area

At the bottom of the window there are some buttons to select what the user want to see. These buttons are divided in two groups : `Text Filtering` buttons and `Text displaying` buttons.

Figure 10.17: What/Where Control Area

The `Text Filtering` buttons are the toggle buttons on the left and they are used to filter the traces found by the click process. The displayed information is the one that arrives to the Filtering Module. For example, if the user don't select any type of communication the user won't

see any communication line. Note that this utility collects the traces around the selected value; if the user are working in a high scale of visualization, the number of lines raises and the clearness is lost. Select an adjusted scale to get the desired level.

Their functionality are :

- `Semantic` : This toggle button is used to display or not to display the semantic value and its duration. By default, this button is enabled but disabling this button the information received about the semantic value burst, won't be painted.
- `Events` : This toggle button is used to display or not to display the information about the events. By default, this button is enabled. Disabling the button the event information won't be painted.
- `Communication` : This toggle button is used to display or not to display the information about the communications occurred around the click point. By default, this button is enabled.

The `Scale bar` lets to fix the width of the click selection in current time units. Paraver multiplies the 'X-scale' value (see "Visualizer" in section 10.1) of the displaying window to the scale bar number, to compute that width.

The `Text displaying` buttons are the toggle buttons on the right of the scale bar, they are used to select how the information will be displayed. Their functionality are :

10.2.3 All the burst

Sometimes, when the user is working in a lower scale (like figure 10.18), the click utility doesn't give all the information of the current burst because the width of the click is lower than the burst. To solve this problem there is the button "All the burst". If this button is selected the click search and display all the information of the burst clicked filtering it if it is necessary in accordance to the toggle buttons.

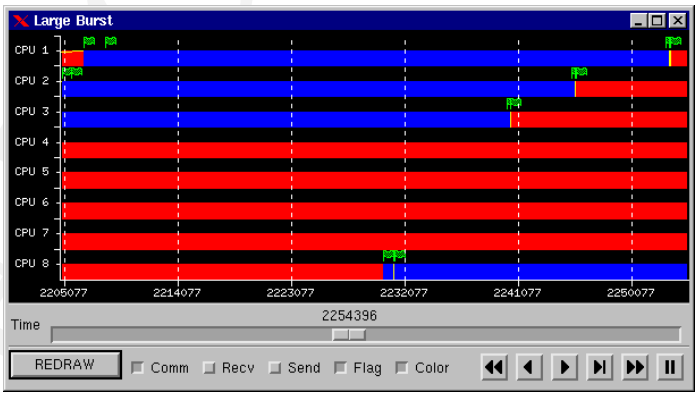


Figure 10.18: Window at lower scale

The click information in a window with a lower scale gives more information with all the burst enabled. In the previous example, the click with all the burst disabled in the processor number one (figure 10.19) gives a poor information because the scale is tiny and the burst is big. As a result, the click width is tiny, too. With all the burst enabled (figure 10.20) Paraver searches back and forward to all the burst, and the result is all the information related for this burst. Note, that the click utility has searched until the limits of the burst (blue zone).

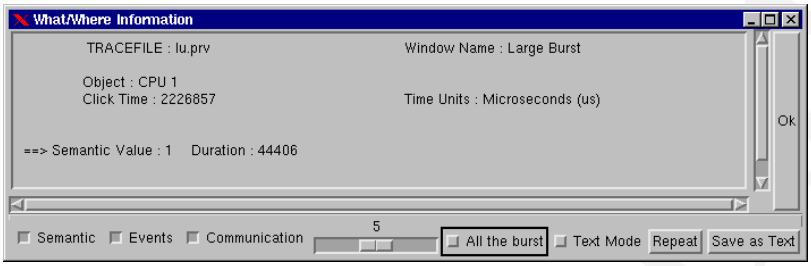


Figure 10.19: What/Where with all the burst disabled

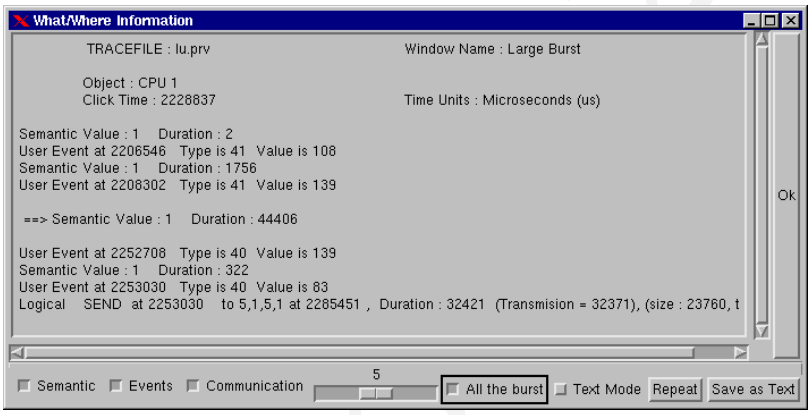


Figure 10.20: What/Where with all the burst enabled

10.2.4 Text Mode

There are two information modes available that affect to the `Semantic Value` and `User Event` information. The Text Mode can be selected just pressing the button *Text Mode* at the bottom of this window.

If the `Text Mode` toggle is disabled (default mode), the *semantic value*, the *user event type* and *user event value* are shown as numbers (Figure 10.21).

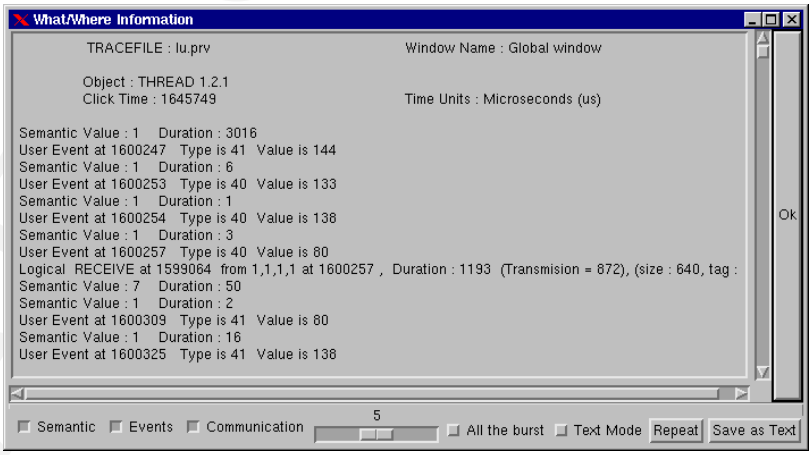


Figure 10.21: Paraver What/Where Information shown as numbers

But if the *Text Mode* toggle button is enabled these values are shown as labels (if they are defined). When these values are shown as labels the format line is :

- Semantic value format line when we are working with labels is :

```
<semantic value label>   Duration: <burst time>
```

- Event format line when we are working with labels is :

```
User Event at <time>  <label type>  <label value>
```

The figure 10.22 shows the same click that figure 10.21 but with the *Text Mode* information enabled. Numbers are shown as labels.

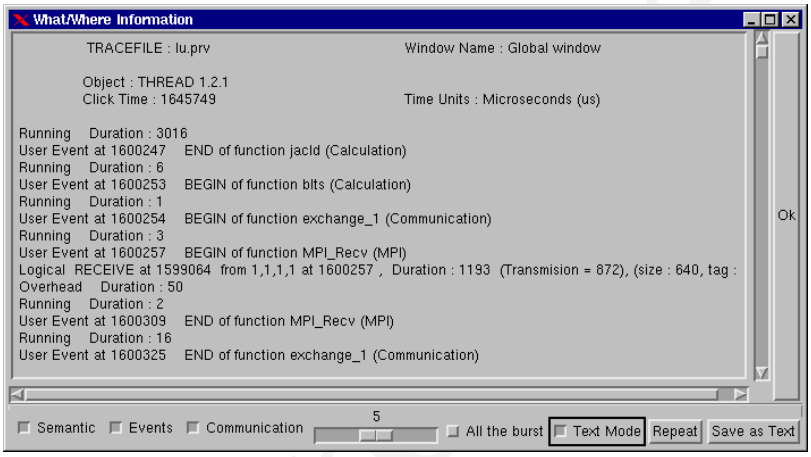


Figure 10.22: Paraver What/Where Information shown as labels

Note, that the *semantic value* has been converted to labels like `Running` (semantic value 1), overhead (semantic value 2), the *user event type* has been converted to labels like `Begin of function` (event type 40) and `End of function` (event type 41). Finally, in our example the *user event values* have been converted to function names which had been defined in the configuration file.

Since the semantic value is a computed value from trace records through object model hierarchy, not ever could be associated to a corresponding label. There are some trace views where semantic value is obtained directly from a defined record value:

- the semantic values of the **State As Is** view could be associated to the defined state labels because the returned values from the semantic module are directly record state values. For example:

```
Running   Duration: 3016
```

- the semantic values of the **Last Evt Val** view could be associated to the labels of the last processed event:
  - when an event has a defined defined type label (*User function*) and has a defined value label (*x_solve*), the event line will be shown as:

```
User function   x_solve  Duration: 3016
```

  - when an event has a defined type label *TLB misses* but doesn't have a defined value label for value the *4273498*, the event line will be shown as:

```
TLB misses   4273498  Duration: 3016
```

- the semantic values of the **Last Evt Type** view could be associated to the type label of the last processed event: For example:

```
User function   Duration: 3016
```

- the semantic values of the **Next Evt Val** view could be associated to the labels of the next event. For example:
  - when an event has a defined type label (*User function*) and a defined value label (*x_solve*), the event line will be shown as:

```
User function   x_solve  Duration: 3016
```

  - when an event has a defined type label *TLB misses* and doesn't have a defined value label for value *4273498*, the event line will be shown as:

```
TLB misses   4273498  Duration: 3016
```

- the semantic values of the **Next Evt Type** view could be associated to the defined event type label of the next event. For example:

```
User function   Duration: 3016
```

If there is no label, the text mode will appear as an unknown plus the numerical value:

- Semantic value format line when we are working with labels is :

```
Unknown for value  <value>    Duration :  <burst time>
```

- Event format line when we are working with labels is :

```
User Event at  <time>   TYPE <type>   VALUE <value>
```

10.2.5 Repeat button

The **Repeat** button repeats the last click done in the current window. This button is very useful when we have changed any parameter that will affect the displayed information and we want to obtain the information on the same point.

10.2.6 Save as Text button

The What/Where window lets the user to save the textual information around the click point in a plain text. To save it, click onto the button Save as Text on the bottom right corner of the window and a selection file box will be raised to select the filename where this information will be saved.

Figure 10.23: Analyzer icons: (a) Analyzer icon 1D (left) (b) Analyzer icon 2D (right)

10.3 Analyzer Module

This **Analyzer Module** let us to analyze a subset or the full trace file. The selected traces are pre-processed by the Filter Module and Semantic Module modules, before entering in the Analyzer Module.

Very detailed qualitative analysis can be done by properly selecting the Filter Module, Semantic Module, and Analyzer Module combined settings.

10.3.1 Analyzer Module 1D

Analyzer window

When the analyzer icon (figure 10.23 (a)) is pressed, the Analyzer window is raised.

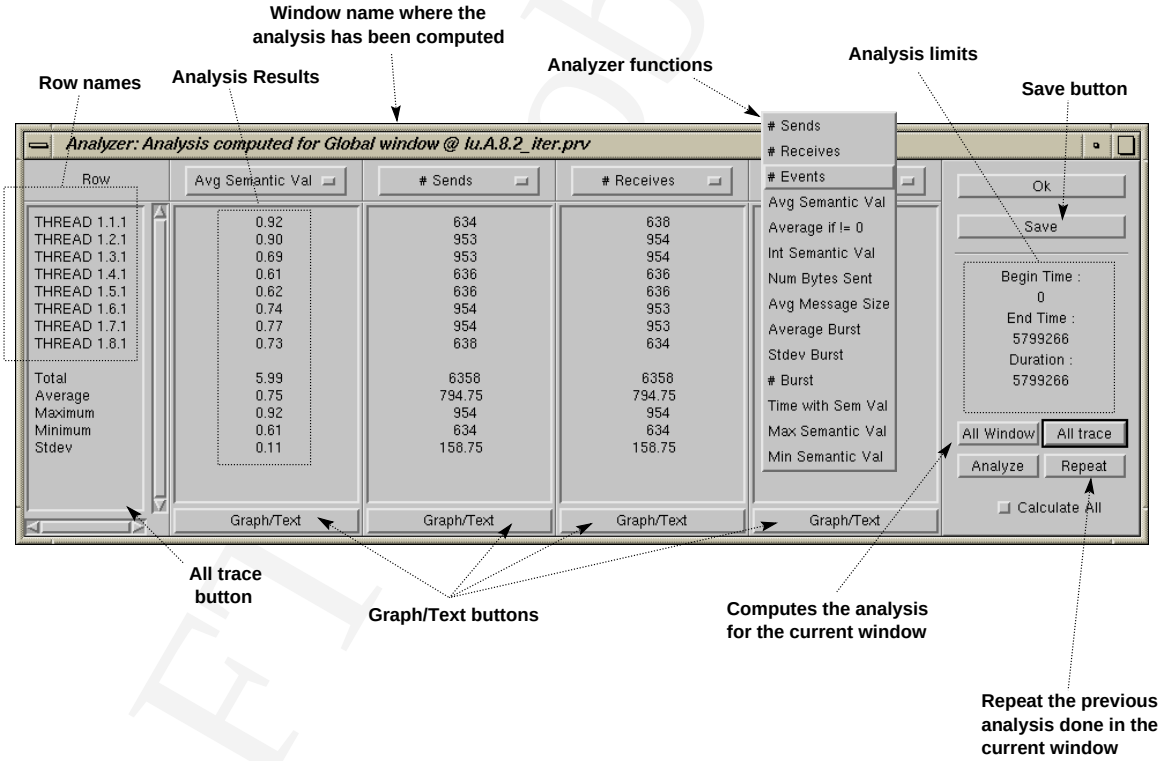


Figure 10.24: Analyzer window

The Analyzer Module window can show four analysis at the same time, one in each row. At the top of each row there is a pop up menu to select the function that will be displayed in that row. This pop up menu contains the default functions and the user defined functions.

On the right side of the window there are the limits and the window name were the analysis has been done plus some buttons that will be explained below.

The *Analyzer results* contains the result of the applied function plus some information between the different rows that take part in the analysis like *average*, *variance*, ...

Calculate All **option**

The analyzer module can compute the results for all the analyzer functions and not only for the four displayed functions. Then, when we select a new function in a row its analysis has already been done and the analyzer only has to show the results without compute the analysis another time (the user doesn't have to wait for the function results). Enabling the Calculate All button when the analysis is working it computes the results for all the functions.

Sometimes, when we are working with large traces, to compute all the analyzer functions will be so slow and usually when we are doing an analysis, we are only interested in a specific group of functions. With Calculate All button disabled the analysis is computed only for the four displayed functions. If the *Analyzer Module* only have to compute the four displayed rows when we are working with large traces, the analysis will go more quickly. The other functions which aren't selected in a row don't have its results computed and if they are selected in a row after the analysis, they will appear as Funct. Not Calculated (figure 10.25). To compute them, the same analysis could be done just pressing the Repeat button (remember enable the Calculate All button if you want compute the result for all the analyzer functions).

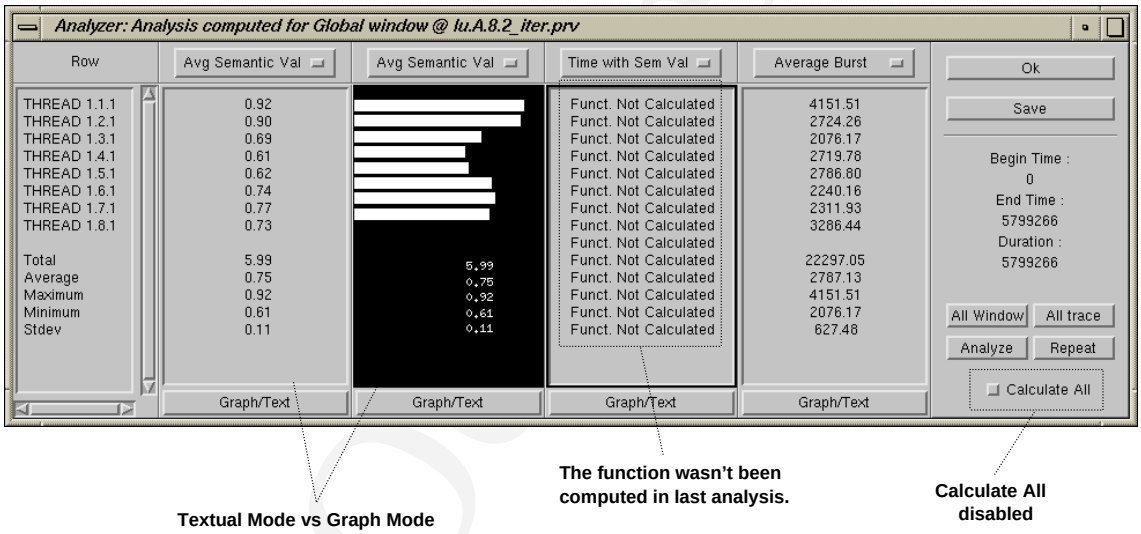


Figure 10.25: Text mode vs. graphical mode

Graph/Text representation. *Graph/Text* **buttons.**

Paraver can display the results in text or graph modes. By default, Paraver displays the results in text mode, but the graph mode give us a **bar graph** display.

Each row has a Graph/Text button to change the display mode. The **Graph** display shows the values as bars. When all the values to display are less than 1, the maximum bar length is taken as 1 and all the bar lengths for the values are scaled to 1. This feature could be seen in function **Avg Semantic Val** on figure 10.25 where the bar lengths are scaled to the maximum value which it is taken as 1.

If there is a value greater than 1, all the lengths are scaled to this value.

If the number of rows that have taken part in the analysis is greater than the row size, the rows can be scrolled in the two display modes.

All window **button**

Compute the analysis for the current window using the selected window limits (begin/end time) and computing the analysis for all the displayed rows.

All trace **button**

Compute the analysis for all the trace.

Analyze **button**

Compute the analysis for a selected region. After press the button, when the cursor will go into the drawing area of a displaying window it looks like a little corner, the region must be selected by clicking two points in the same or different windows. The *Analyze* button works like **Analyzer icon**.

Repeat **button**

Sometimes, when an analysis has been done, the user would do the analysis onto the same limits but changing a specific parameter. The Repeat button makes the analysis with the same limits that the last analysis. The analysis only could be repeated on a window where an analysis had been done, and it will be made using the same limits of last time.

If no analysis has been computed in the current window, the *Repeat* button is disabled.

Saving the analysis in a file. *Save* **button**

The save button lets us to save the analysis, which we have been done, in a plain text file. When this button is pressed it raises a file selection box asking for the file name where it will be saved.

The save option saves the four displayed columns in the analyzer window plus the limits where the analysis was done. The text file organization is more or less like the window appearance and look like this :

```
Begin Time : 0.00     End Time : 4784156.00    Duration : 4784156.00


Row         Time with Sem Val   Avg Semantic Val     # Sends        # Receives
CPU 1            3188830.00           0.67            324.00          328.00
CPU 2            3167106.00           0.66            488.00          489.00
CPU 3            3143421.00           0.66            488.00          489.00
CPU 4            2703920.00           0.57            326.00          326.00
CPU 5            2885524.00           0.60            326.00          326.00
CPU 6            2879705.00           0.60            489.00          488.00
CPU 7            2868487.00           0.60            489.00          488.00
CPU 8            2683333.00           0.56            328.00          324.00

Total           23520326.00           4.92           3258.00         3258.00
Average          2940040.75           0.61            407.25          407.25
Maximum          3188830.00           0.67            489.00          489.00
Minimum          2683333.00           0.56            324.00          324.00
Variance          189803.37           0.04             81.26           81.26
```

How to make an analysis

Paraver lets to make an analysis for all the trace file or a subset. Before doing the analysis we should enable or disable the **Calculate All** button to compute all the analyzer functions or only the selected in each column.

To make an analysis for all the trace file, first press the **Analyzer icon** on the *Global Controller* window if **Analyzer window** is not raised, and click onto the All trace button, wait a moment, an the analysis will be written in the analyzer columns. If the **Analyzer window** is raised, press the All trace button to compute the analysis .

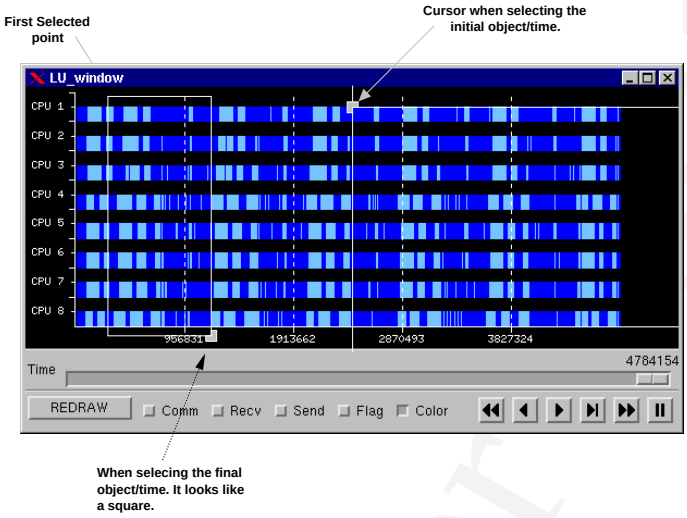


Figure 10.26: Cursor when selecting a region to make an analysis

To analyze a subset, press the **Analyzer icon** (if **Analyzer window** is not raised, if is raised you can press the **Analyze** button); then, when the cursor will go into the drawing area of a displaying window it looks like a little corner, select a region clicking two points in the same or different windows. While Paraver is computing the analysis, a window (see figure 10.38) rows are marked in a computing state and the working window is raised until the analysis finishes. To **cancel** the analysis, click on to the *Cancel button*.

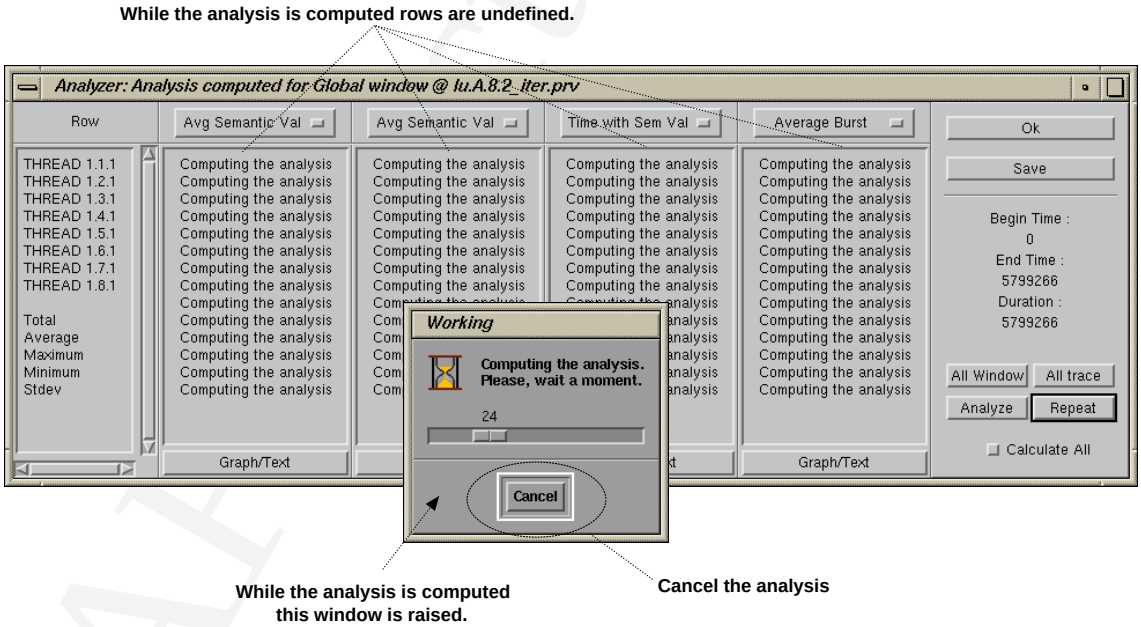


Figure 10.27: Computing the analysis

The *Timing* window shows the limits when the user is selecting the two points. Each point determines an initial/final object (CPU, ptask, task or thread row), and a beginning/ending time. The second point may be selected on another window, but the analyzer only takes into account the timing and the object row, the others features like the trace file will be omitted. When the points has been selected, wait a moment and the analysis will be displayed on the *Analyzer window*.

User analyzer functions

The *Analyzer Module* has some user functions in the pop up submenus of each row which can be used to make the analysis of the trace file. By default, the *Analyzer Module* has implemented some functions which could be used to do a very qualitative analysis.

Function Avg Semantic Value

This function computes for each selected row the average value along the selected area. Adds the duration of each burst multiplied by their values and divide the result by the duration of the selected area. The Mathematical Formula for each row is :

$$\frac{\sum_{i=1}^{n\_bursts} (t_i * value_i)}{Selected \quad Time}$$

Where:

- n_bursts – is the number of bursts within the selected area for each row (if the selection cuts a burst, only the selected burst time is used to compute the result).
- t_i – is the duration of burst i.
- $value_i$ – is the value of burst i.
- $SelectedTime$ – duration of the selected interval.

Function Average if != 0

This function computes for each selected row the average value different of 0 along the selected area. Adds the duration of each burst multiplied by their values and divide the result by the time of the selected area with a value greater than 0. It is different from the previous function because the time with value 0 is not considered in the result. The Mathematical Formula for each row is :

$$\frac{\sum_{i=1}^{n\_bursts} (t_i * value_i)}{\sum_{i=1}^{n\_bursts} (t_i)} \quad where \quad value_i > 0$$

Where:

- n_bursts – is the number of bursts within the selected area for each row greater than 0 (if the selection cuts a burst, only the selected burst time is used to compute the result).
- t_i – is the duration of burst i.
- $value_i$ – is the value of burst i.

Function Int Semantic Val

This function computes for each selected row the integral of the values in the selected area.

It adds the duration of each burst multiplied by their values. It is different from the **Avg Semantic Val** because the adding is not divided by the duration. The Mathematical Formula for each row is :

$$\sum_{i=1}^{n\_bursts} (t_i * value_i)$$

Where:

- n_bursts – is the number of bursts within the selected area for each row greater than 0 (if the selection cuts a burst, only the selected burst time is used to compute the result).
- t_i – is the duration of burst i.
- $value_i$ – is the value of burst i.

Function Average Burst

This function computes for each selected row the average duration time for all the selected bursts with a value greater than 0. Adds the duration of each complete burst and divide the result by the number of selected bursts. The Mathematical Formula for each row is :

$$\frac{\sum_{i=1}^{n\_bursts} (t_i)}{n\_bursts} \quad where \quad value_i > 0$$

Where:

- n_bursts – is the number of complete bursts within the selected area for each row (incomplete burst aren't used to compute the result).
- t_i – is the duration of burst i.
- $value_i$ is the value of burst i.

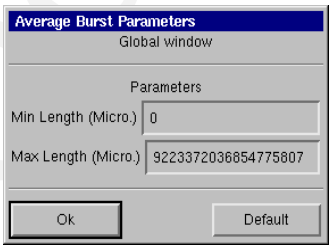


Figure 10.28: Average Burst Parameters window

By default, all selected bursts will be used to compute the average value, but the user can select the maximum and minimum duration of the burst that will be considered. This feature is useful to filter outliers.

To select it, when the user selects this function, a window (called *Average Burst Parameters*) is raised, where the user can put the minimum and the maximum duration times.

Function # Burst

This function computes number of bursts in the selected area for each row with a *value* greater than zero. If the selected region cuts a burst with a value greater than zero, that burst will not be added to the result.

As the previous function, by default all the bursts will be used but, the user can select the minimum and the maximum length that will be considered.

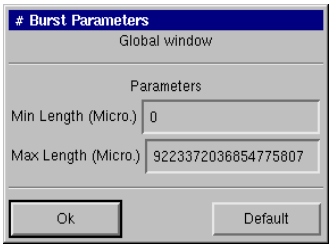


Figure 10.29: # Burst Parameters window

Function Stdev Burst

This function computes for each selected row the standard desviation of the duration time for all the selected bursts with a value greater than 0. The Mathematical Formula for each row is :

$$\sqrt{\frac{\sum_{i=1}^{n\_bursts}(t_i)^2}{n\_bursts} - \left[\frac{\sum_{i=1}^{n\_bursts}(t_i)}{n\_bursts}\right]^2} \quad where \quad value_i > 0$$

Where:

- n_bursts – is the number of complete bursts within the selected area for each row (incomplete burst aren't used to compute the result).
- t_i – is the duration of burst i.
- $value_i$ is the value of burst i.

As the previous functions, by default all selected bursts will be used to compute the average value. The user can select the maximum and minimum duration of the burst that will be considered.

To select it, when the user selects this function, a window (called *Stdev Burst Parameters*) is raised, where the user can put the minimum and the maximum duration times.

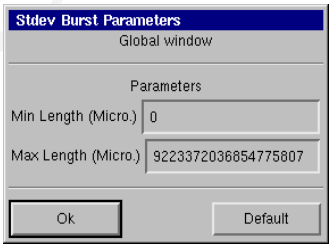


Figure 10.30: Stdev Burst Parameters window

Function Time with Sem Val

This function computes for each selected row the standard desviation of the duration time for all the selected bursts. The Mathematical Formula for each row is :

$$\sum_{i=1}^{n\_bursts}(t_i)$$

Where:

- n_bursts – is the number of bursts within the selected area for each row (if the selection cuts a burst, only the selected burst time is used to compute the result).
- t_i – is the duration of burst i.
- $value_i$ is the value of burst i.

As the previous functions, by default all selected bursts will be used to compute the average value. The user can select the maximum and minimum duration of the burst that will be considered.

To select it, when the user selects this function, a window (called *Stdev Burst Parameters*) is raised, where the user can put the minimum and the maximum duration times.

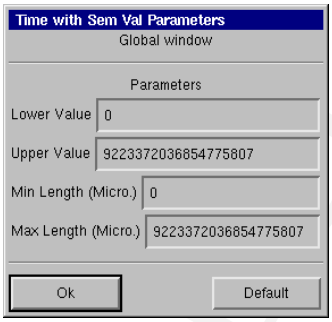


Figure 10.31: Time with Sem Val Parameters window

Function Avg Message Size

This function computes for each selected row the average message size along the selected area. Adds the message size of each sended message and divide the result by the duration of the selected area. The Mathematical Formula for each row is :

$$\frac{\sum_{i=1}^{n\_messages} (size_i)}{n\_messages}$$

Where:

- $n\_messages$ – is the number of messages in the selected area. Only logical sends are considered to compute the result.
- $size_i$ – is the size for message i.

Function Num Bytes Sent

This function computes for each selected row the number of send bytes in the selected area. Adds the message size of each sent message. The Mathematical Formula for each row is :

$$\sum_{i=1}^{n\_messages} (size_i)$$

Where:

- $n\_messages$ – is the number of messages in the selected area. Only logical sends are considered to compute the result.
- $size_i$ – is the size for message i.

Function # Sends

This function computes the number of outgoing communications within the selected area. If *logical* and *physical* communications aren't filtered, the computed value will be the addition of logical sends and physical sends.

Function # Receives

This function computes the number of incoming communications within the selected area. If *logical* and *physical* communications aren't filtered, the computed value will be the addition of logical receives and physical receives.

Function # Events

This function computes the number of events in the selected area.

Function Max Semantic Val

This function computes the maximum value within the selected area.

Function Min Semantic Val

This function computes the minimum value within the selected area.

10.3.2 Analyzer Module 2D

The *Analyzer Module 2D* allows to merge the semantic function of two windows or to analyze the communications for each source-target pair. It includes features such as being able to measure times, averages, number of bytes, ...

Figure 10.32: Analyzer 2D icon

Analyzer window

When the analyzer 2D icon (figure 10.33) is pressed, the Analyzer window is raised.

Three menus (**Accumulator**, **Statistics** and **Data** menus) select the type of information that will be computed and displayed in the *Matrix of results* where by default:

- the matrix rows shows the different paraver objects (thread, task, ...) that has taken part in the analysis. In figure 10.33, rows are displaying threads.
- the matrix columns shows the selected accumulator type of information. For example, in figure 10.33, the columns show the semantic values where analysis results have been accumulated.

The analysis compute the quantitative information for the **Accumulator** type of information using the **Statistics** function and taking the data from selected *Data* window.

Total, average, maximum, minimum and stddev statistics per column are showed bellow the matrix of results.

On the right top corner, there are the next options:

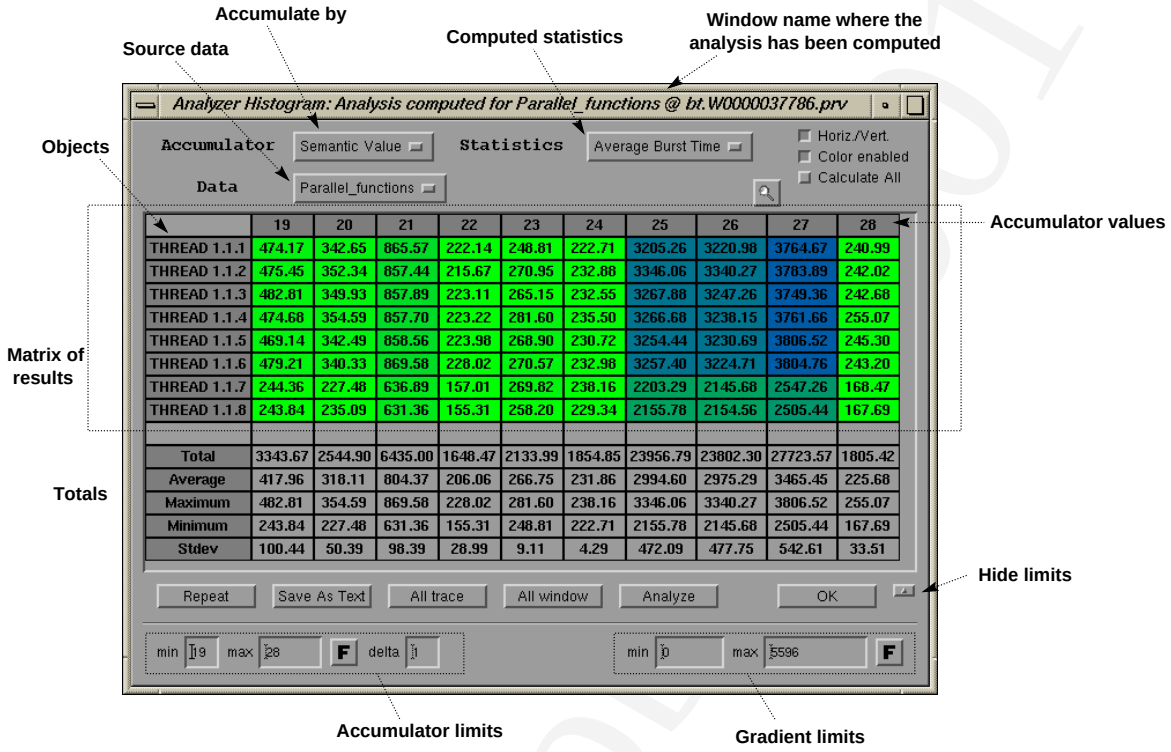


Figure 10.33: Analyzer 2D window

- **Horiz./Vert.**: Interchanges rows and columns. When it is unselected, paraver objects are showed on columns and accumulator selected information is showed on rows. By default, *horizontal* is selected.

- **Color enabled**: Fill each cell using a gradient color. Values between selected **Gradient limits** are grouped into different groups and one different gradinet color is used for each group of values. Thus, cells with greater values are filled with dark colors and cells with lower values are filled with light colors. By default, color mode is selected.

- **Calculate All**: The analyzer module can compute the results for all the analyzer functions and not only for the four displayed functions. Then, when we select a new function in a row its analysis has already been done and the analyzer only has to show the results without compute the analysis another time (the user doesn't have to wait for the function results). Enabling the Calculate All button when the analysis is working it computes the results for all the functions.

  Sometimes, when we are working with large traces, to compute all the analyzer functions will be so slow and usually when we are doing an analysis, we are only interested in a specific group of functions. With Calculate All button disabled the analysis is computed only for the four displayed functions. If the *Analyzer Module* only have to compute the four displayed rows when we are working with large traces, the analysis will go more quickly. The other functions which aren't selected in a row don't have its results computed and if they are selected in a row after the analysis, they will appear as Funct. Not Calculated (figure 10.25). To compute them, the same analysis could be done just pressing the Repeat button (remember enable the Calculate All button if you want compute the result for all the analyzer functions).

Accumulator, Statistics and Data menus

The **Accumulator**, **Statistics** and **Data** menus select the type of information that will be computed by the 2D analysis.

The **Accumulator** menu select the type of information where statistics will be computed:

- **Semantic Value**: The semantic values displayed in the analyzed window will be considered. Thus, we will obtain an **object x semantic value** table.
- Object level where window is working. Thus, we will obtain an **object x object** table.

The **Accumulator limits** select the minimum and maximum accumulator values that will be considered on the analysis. Also it is possible to group the accumulator values by specifying a **delta** value.

The **Statistics** function is the type of quantitative information that will be computed for each pair of row-column (object and each accumulator value). Depending on the accumulator type of information, the **Statistics** menu contains a different set of functions:

Accumulator	Available statistics
Thread Task ...	#Sends #Receives Bytes sent Bytes received Average bytes sent Average bytes received
Semantic Value	Time %Time #Bursts Integral value Average value Maximum Minimum Average Burst Time Stddev Burst Time

Table 10.1: Statistics functions

The **Data** menu is used to select the source window where data is taken. Depending on the quantitative analysis that we want to compute, we can select a different current window as source data window.

Object x Object available statistics

The available statistics when selecting the *Accumulator* object type are showed in table 10.1. Next figure 10.34 shows the results of an object x object analysis where *Bytes sent* statistics function has been selected. Window shows the number of bytes sent by each pair-to-pair of processes.

- Function # SENDS: This function computes the number of outgoing communications within the selected area. If *logical* and *physical* communications aren't filtered, the computed value will be the addition of logical sends and physical sends.

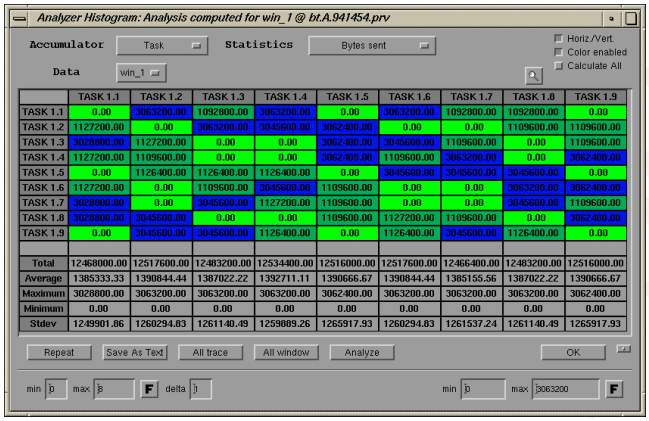


Figure 10.34: Object x Object analysis

- Function # RECEIVES: This function computes the number of incoming communications within the selected area. If *logical* and *physical* communications aren't filtered, the computed value will be the addition of logical receives and physical receives.

- Function BYTES SENT:

  This function computes for each selected row the number of send bytes in the selected area. Adds the message size of each sent message. The Mathematical Formula for each row is :

$$\sum_{i=1}^{n\_messages} (size_i)$$

  Where:

  - $n\_messages$ – is the number of messages in the selected area. Only logical sends are considered to compute the result.
  - $size_i$ – is the size for message i.

- Function BYTES RECEIVED:

  This function computes for each selected row the number of received bytes in the selected area. Adds the message size of each received message. The Mathematical Formula for each row is :

$$\sum_{i=1}^{n\_messages} (size_i)$$

  Where:

  - $n\_messages$ – is the number of messages in the selected area. Only logical receives are considered to compute the result.
  - $size_i$ – is the size for message i.

- Function AVERAGE BYTES SEND: This function computes for each selected row the average number of bytes send along the selected area. Adds the sent bytes by of each sended message and divide the result by the duration of the selected area. The Mathematical Formula for each row is :

$$\frac{\sum_{i=1}^{n\_messages} (size_i)}{n\_messages}$$

Where:

- $n\_messages$ – is the number of messages in the selected area. Only logical sends are considered to compute the result.
- $size_i$ – is the size for message i.

- Function AVERAGE BYTES RECEIVED: This function computes for each selected row the average number of bytes received along the selected area. Adds the received bytes by of each received message and divide the result by the duration of the selected area. The Mathematical Formula for each row is :

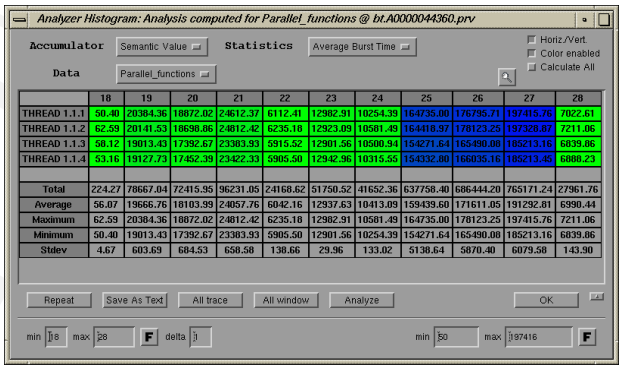
$$\frac{\sum_{i=1}^{n\_messages} (size_i)}{n\_messages}$$

Where:

- $n\_messages$ – is the number of messages in the selected area. Only logical receives are considered to compute the result.
- $size_i$ – is the size for message i.

Object x Semantic Value available statistics

The available statistics when selecting the *Accumulator* semantic value type are showed in table 10.1. Next figure 10.35 shows the results of an object x semantic value analysis where *Average Burst Time* statistics function has been selected. Each value represent the execution of a function so window is showing the average duration of each function per thread in the selected region.



	18	19	20	21	22	23	24	25	26	27	28
THREAD 1.1.1	50.40	20384.36	18872.02	24012.37	6112.41	12982.91	10254.39	164735.00	176795.71	197415.76	7022.61
THREAD 1.1.2	62.59	20141.53	18698.86	24012.42	6235.18	12923.09	10581.49	164418.97	178123.25	197328.87	7211.06
THREAD 1.1.3	50.12	19013.43	17392.67	23383.93	5915.52	12901.56	10500.94	154271.64	165490.08	185213.16	6839.86
THREAD 1.1.4	53.16	19127.73	17452.39	23422.33	5905.50	12942.96	10315.55	154332.80	166035.16	185213.45	6808.23
Total	224.27	78667.04	72415.95	96231.05	24168.62	51750.52	41652.36	637758.40	686444.20	765171.24	27961.76
Average	56.07	19666.76	18103.99	24057.76	6042.16	12937.63	10413.09	159439.60	171611.05	191292.81	6990.44
Maximum	62.59	20384.36	18872.02	24012.42	6235.18	12982.91	10581.49	164735.00	178123.25	197415.76	7211.06
Minimum	50.40	19013.43	17392.67	23383.93	5905.50	12901.56	10254.39	154271.64	165490.08	185213.16	6839.86
Stdev	4.67	603.69	684.53	658.58	138.66	29.96	133.02	5138.64	5870.40	6079.58	143.90

Figure 10.35: Object x Semantic Value analysis

- Function TIME: This function computes the time in each value on the selected area.
- Function % TIME This function computes the percentatge of time in each value on the selected area.

- Function # Bursts

  This function computes number of bursts of a value in the selected area If the selected region cuts a burst, that burst will not be added to the result.

- Function Integral Value

  This function computes for each value the integral time in the selected area. It adds the duration of each burst multiplied by their values. It is different from the **Average Value** because the adding is not divided by the duration. The Mathematical Formula for each row is :

$$\sum_{i=1}^{n\_bursts} (t_i * value_i)$$

  Where:

  - n_bursts – is the number of bursts within the selected area for each row greater than 0 (if the selection cuts a burst, only the selected burst time is used to compute the result).
  - t_i – is the duration of burst i.
  - $value_i$ – is the value of burst i.

- Function Average Value

  This function computes for each selected row the average value along the selected area. Adds the duration of each burst multiplied by their values and divide the result by the duration of the selected area. The Mathematical Formula for each row is :

$$\frac{\sum_{i=1}^{n\_bursts} (t_i * value_i)}{Selected \quad Time}$$

  Where:

  - n_bursts – is the number of bursts within the selected area for each row (if the selection cuts a burst, only the selected burst time is used to compute the result).
  - t_i – is the duration of burst i.
  - $value_i$ – is the value of burst i.
  - $SelectedTime$ – duration of the selected interval.

- Function Maximum

  This function computes the maximum value within the selected area.

- Function Minimum

  This function computes the minimum value within the selected area.

- Function Average Burst Time

  This function computes for each selected row the average duration time for all the bursts. Adds the duration of each complete burst and divide the result by the number of selected bursts. The Mathematical Formula for each row is :

$$\frac{\sum_{i=1}^{n\_bursts} (t_i)}{n\_bursts} \quad where \quad value_i > 0$$

  Where:

- n_bursts – is the number of complete bursts within the selected area for each row (incomplete burst aren't used to compute the result).
- t_i – is the duration of burst i.
- $value_i$ is the value of burst i.

- Function STDEV BURST TIME

  This function computes for each selected row the standard desviation of the duration time for all the bursts. The Mathematical Formula for each row is :

$$\sqrt{\frac{\sum_{i=1}^{n\_bursts}(t_i)^2}{n\_bursts} - \left[\frac{\sum_{i=1}^{n\_bursts}(t_i)}{n\_bursts}\right]^2} \quad where \quad value_i > 0$$

  Where:

  - n_bursts – is the number of complete bursts within the selected area for each row (incomplete burst aren't used to compute the result).
  - t_i – is the duration of burst i.
  - $value_i$ is the value of burst i.

All window **button**

Compute the analysis for the current window using the selected window limits (begin/end time) and computing the analysis for all the displayed rows.

All trace **button**

Compute the analysis for all the trace.

Analyze **button**

Compute the analysis for a selected region. After press the button, when the cursor will go into the drawing area of a displaying window it looks like a little corner, the region must be selected by clicking two points in the same or different windows. The *Analyze* button works like **Analyzer 2D icon**.

Repeat **button**

Sometimes, when an analysis has been done, the user would do the analysis onto the same limits but changing a specific parameter. The Repeat button makes the analysis with the same limits that the last analysis. The analysis only could be repeated on a window where an analysis had been done, and it will be made using the same limits of last time.

If no analysis has been computed in the current window, the *Repeat* button is disabled.

Saving the analysis in a file. *Save As text* **button**

The save button lets us to save the analysis, which we have been done, in a plain text file in **Matrix** or **Row** format. When this button is pressed it raises a window to select the plain text format. Once is selected, a file selection box is raised.

Matrix Format The text file organization is more or less like the window appearance. It looks like:

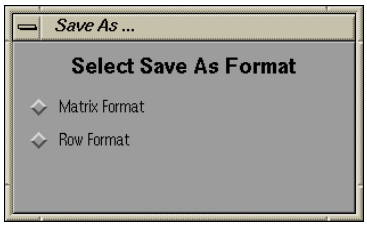


Figure 10.36: Save As Format selection format

```
Begin Time : 0.00 End Time : 5091389.00 Duration : 5091389.00


Objects/Intervals    22      23      24      25       26       27       28
THREAD 1.1.1       333.75  458.59  398.37  6164.21  6291.35  7064.83  390.32
THREAD 1.1.2       340.98  492.48  413.95  6198.00  6201.06  7245.71  392.86
THREAD 1.1.3       343.48  498.05  412.29  6205.40  6153.44  7105.86  395.43
THREAD 1.1.4       244.42  506.43  455.46  4108.66  4090.92  4821.41  280.68

Total 93.06     1262.63 1955.55 1680.07 22676.27 22736.77 26237.81  1459.29
Average 23.26      315.66  488.89  420.02  5669.07  5684.19  6559.45   364.82
Maximum 24.60      343.48  506.43  455.46  6205.40  6291.35  7245.71   395.43
Minimum 21.94      244.42  458.59  398.37  4108.66  4090.92  4821.41   280.68
Stdev 0.96        41.28   18.18   21.34   901.04   921.21  1005.70    48.62
```

Row Format The Row Format text file organization look like:

```
Begin Time : 678851.87 End Time : 4751963.07 Duration : 4073111.20


Row Column Value
THREAD 1.1.1 22 54124.00
THREAD 1.1.1 23 74205.00
THREAD 1.1.1 24 64856.00
THREAD 1.1.1 25 1003621.13
THREAD 1.1.1 26 1030621.00
THREAD 1.1.1 27 1146174.07
THREAD 1.1.1 28 63149.00
THREAD 1.1.2 22 55304.00
THREAD 1.1.2 23 80285.00
THREAD 1.1.2 24 66829.00
THREAD 1.1.2 25 1008870.13
THREAD 1.1.2 26 1010874.00
THREAD 1.1.2 27 1175880.07
THREAD 1.1.2 28 63598.00
THREAD 1.1.3 22 55729.00
THREAD 1.1.3 23 80303.00
THREAD 1.1.3 24 66450.00
THREAD 1.1.3 25 1010074.13
THREAD 1.1.3 26 1000502.00
THREAD 1.1.3 27 1152158.07
THREAD 1.1.3 28 63923.00
```

```
THREAD 1.1.4 22 39569.00
THREAD 1.1.4 23 82428.00
THREAD 1.1.4 24 73786.00
THREAD 1.1.4 25 668224.13
THREAD 1.1.4 26 666074.00
THREAD 1.1.4 27 782763.07
THREAD 1.1.4 28 45438.00


Total 22 204726.00
Total 23 317221.00
Total 24 271921.00
Total 25 3690789.53
Total 26 3708071.00
Total 27 4256975.27
Total 28 236108.00
Average 22 51181.50
Average 23 79305.25
Average 24 67980.25
Average 25 922697.38
...
```

How to make an analysis

Paraver lets to make an analysis for all the trace file or a subset. Before doing the analysis we should enable or disable the **Calculate All** button to compute all the analyzer functions or only the selected in each column.

To make an analysis for all the trace file, first press the **Analyzer 2D icon** on the *Global Controller* window if **Analyzer window** is not raised. Select the type of information that have tobe computed and click onto the All trace button, wait a moment, an the analysis will be showed. If the **Analyzer window** is raised, press the All trace button to compute the analysis .

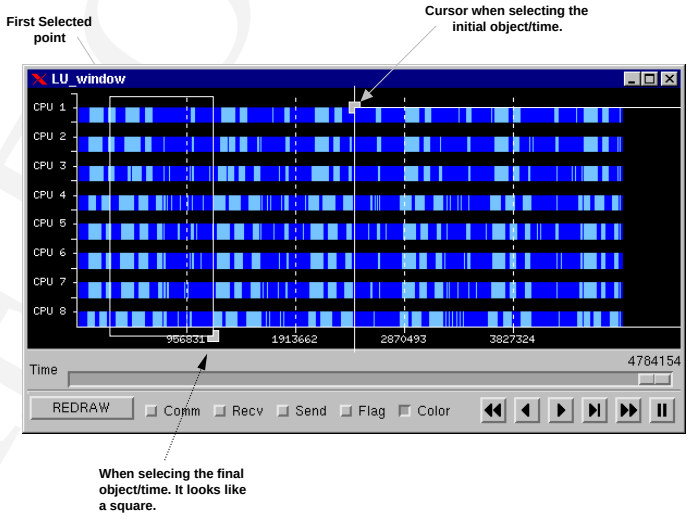


Figure 10.37: Cursor when selecting a region to make an analysis

To analyze a subset, press the **Analyzer icon** (if **Analyzer window** is not raised, if is raised you can press the **Analyze** button); then, when the cursor will go into the drawing area of a displaying window it looks like a little corner, select a region clicking two points in the same or

different windows. While Paraver is computing the analysis, a window (see figure 10.38) rows are marked in a computing state and the working window is raised until the analysis finishes. To **cancel** the analysis, click on to the *Cancel button*.

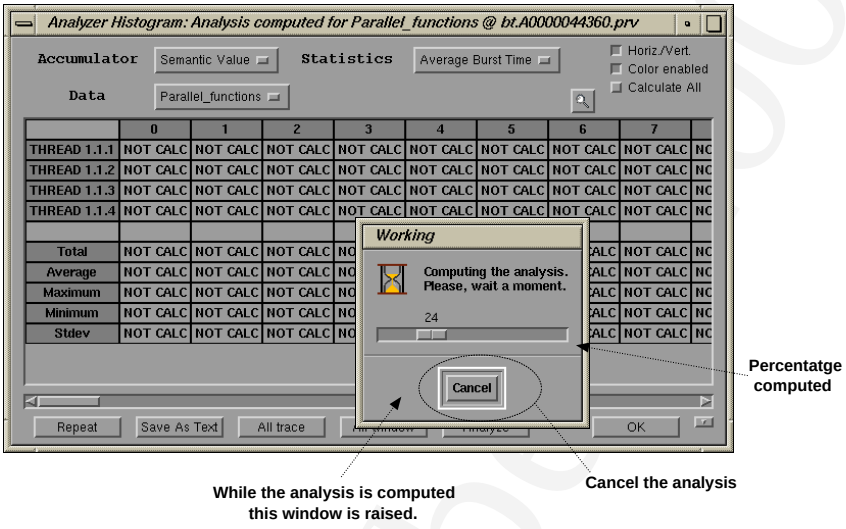


Figure 10.38: Computing the analysis

The *Timing* window shows the limits when the user is selecting the two points. Each point determines an initial/final object (CPU, ptask, task or thread row), and a beginning/ending time. The second point may be selected on another window, but the analyzer only takes into account the timing and the object row, the others features like the trace file will be omitted. When the points has been selected, wait a moment and the analysis will be displayed on the *Analyzer window*.

Appendix A

Environment Variables

PARAVER_HOME

The environment variable PARAVER_HOME must contain to the Paraver installation directory. For example, Paraver searched the license file as $PARAVER_HOME/etc/license.dat

```
setenv PARAVER_HOME /user/user1/traces
```

PARAVER_CFGS_DIR

Variable used to specify a list of window configuration files directories. The specified directories will be added to the **Load Windows** *User directories* popup menu described in chapter 9.3 on page 71. Directories must be and separated by a colon and must be specified using the full path name. For example, if we set the environment variable to:

```
setenv PARAVER_CFGS_DIR /user/user1/ompitrace_cfgs:/user/user1/ute_cfgs
```

we are specifying the directories */user/user1/ompitrace_cfgs* and */user/user1/ute_cfgs* that will be added to the *User directories* menu.

PARAVER_DIR

Variable used to specify a directory that contain the Paraver trace files. The contents of the environment variable will be used to set the initial directory of **Load Tracefiles** menu option in Paraver Main Menu window.

```
setenv PARAVER_DIR /user/user1/traces
```

PARAVER_CONFIG_FILE

Variable used to specify a default **Paraver Configuration File** for all tracefiles that will loaded.

```
setenv PARAVER_CONFIG_FILE /user/user1/traces/default_file.pcf
```

When this environment variable is set up, the specified configuration will be added to each trace that will be loaded. To get more information about the **Paraver Configuration File** see the Trace Generation manual at URL: http://www.cepba.upc.es/paraver.

TMPDIR

Variable used to specify the directory where Paraver temporal files will be placed. By default, temporal files are placed in the current directory.

```
setenv TMPDIR /tmp
```

Appendix B

Binary Trace Format

Paraver accepts two trace format types: ASCII and BINARY. The ASCII trace format is described in **Paraver Tracefile Description** document (see **http://www.cepba.upc.es/paraver)**. The binary format is an internal Paraver trace format to speed up the trace loading. A tracefile using the binary format only can be obtained using the a provided tool (it can be found in Paraver distribution) from an ASCII trace file. The BINARY trace file names usually end using the extension **.map**.

The loading of the ASCII format can be slow for trace files of several hundreds of MB. Loading trace files in binary format is much faster. Paraver offers an utility, **prv2map**, to convert ASCII trace files into BINARY trace files.

prv2map description

```
Usage: prv2map [-xEvt [all|<evt_list>]] [-xComm [all|<comm_list>]]
               <prv_trace> -o <output_map_trace_name>

Description: Converts an ASCII Paraver trace to the binary format.
Options:
       -h                                  Get this help.
       -xEvt  [all | type1,type2,...]      Exclude events.
       -xComm [all | tag1,tag2,...]        Exclude communications.
       -sT     starting time of PARAVER trace file '.map'
               by default, time (T in us.) is zero.
       -eT     ending time of PARAVER trace file '.map'
               by default, time (T in us.) is the total time.
```

prv2map converts ASCII into binary records, and it generates a new mapping. This new mapping lets Paraver to map the trace file into memory, therefore the loading process is very quickly. The mapped trace file is also better managed by the memory system, so Paraver can display very big trace files.

prv2map also act as a filter. It lets to filter events or communications in order to reduce the output binary trace files. A list of event types that will be excluded in the final binary trace files could be specified. Also, a communication list could also be specified.

The binary trace file could not be portable between different platforms because the binary representation may be different in each machine.