



## DELIVERABLE D2.3

### PART V: PROTOTYPE DOCUMENTATION FOR THE PERFORMANCE PREDICTION COMPONENT

#### Task 2.4 Interactive and semiautomatic performance evaluation tools - Performance Prediction Component

---

Document Filename:	<b>CG2.4.2-D2.3-v1.2-USC002-PrototypeDoc.doc</b>
Work package:	<b>WP2 Grid Application Programming Environment</b>
Partner(s):	<b>CYFRONET, TUM, USC, CSIC</b>
Lead Partner:	<b>CYFRONET</b>
Config ID:	<b>CG2.4.2-D2.3-v1.2-USC002-PrototypeDoc</b>
Document classification:	<b>PUBLIC</b>

---

Abstract: This document describes the prototype developed during the first year of the project. The features of this prototype, called PPC are based on our experience in a previous tool called "Avispa" developed for a particular multiprocessor and a particular problem (iterative methods for sparse systems). Some new functionalities that are specific for Grid environments as well as other kernels are also implemented in this prototype, and will be added in future versions. We developed the prototype for the matrix-vector product and other kernels of sparse matrix algebra, as well as for the vertlq routine extracted from one of the applications of task 1.4.



**Delivery Slip**

	<b>Name</b>	<b>Partner</b>	<b>Date</b>	<b>Signature</b>
<b>From</b>	Francisco F. Rivera	USC	15/01/2003	
<b>Verified by</b>	Celso Martínez Rivero, Antonio Fuentes	CSIC, RedIRIS	07/02/2003	
<b>Approved by</b>				

**Document Log**

<b>Version</b>	<b>Date</b>	<b>Summary of changes</b>	<b>Author</b>
1.0	15/01/2003	Draft version	Francisco F. Rivera
1.1	15/01/2003	Modified title page and headers	Holger Marten
1.2	17/01/2003	Modified version with suggestions from H. Marten	Francisco F. Rivera

## CONTENTS

<b>EXECUTIVE SUMMARY .....</b>	<b>4</b>
<b>1. INTRODUCTION .....</b>	<b>5</b>
1.1. PURPOSE .....	5
1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS .....	5
<b>2. REFERENCES .....</b>	<b>6</b>
2.1. SOURCE CODE .....	6
2.2. CONTACT INFORMATION .....	6
<b>3. IMPLEMENTATION STRUCTURE .....</b>	<b>7</b>
<b>4. PROTOTYPE FUNCTIONALITY .....</b>	<b>13</b>
<b>5. USER MANUAL .....</b>	<b>14</b>
5.1. DEPENDENCIES .....	14
5.2. INSTALLATION .....	14
5.3. RUNNING.....	14
5.4. INTERFACE DESCRIPTION.....	26
<b>6. INTERNAL TESTS.....</b>	<b>27</b>
<b>7. ISSUES .....</b>	<b>28</b>

## EXECUTIVE SUMMARY

This document describes the PredPerf prototype developed. It is based on Avispa, so many of the description is from Avispa. Some new functionalities that are specific for Grid environments as well as other kernels are also implemented in this prototype. All of them are described in this document. We developed the prototype for the matrix-vector product and other kernels of sparse matrix algebra, as well as for the vertlq routine extracted from one of the applications of task 1.4. Also relevant features of these kernels are explained in the document.

---

## 1. INTRODUCTION

### 1.1. PURPOSE

The global purpose of the tool is to provide performance information about some important computational kernels when they are executed in a Grid. In some cases, this information is just predicted (execution times, communications overhead, ...), in others it is, in fact, real (number of communications, load balance, ...). The tool presents a GUI in such a way that the user can establish the features of the Grid to simulate their effects on the parallel codes, getting information about the behaviour of the kernels under different virtual hardware configurations. Note that this tool can be run in a single workstation or PC, because no Grid computations are involved.

The purpose of the prototype is twofold: on one hand, the adaptation of *Avispa* to a new heterogeneous platform, and on the other, adding new kernels into the tool.

### 1.2. DEFINITIONS, ABBREVIATIONS, ACRONYMS

Self-explanatory.

API	Application Program Interface
Avispa	A Visualization tool for Paraiso codes
CVS	Concurrent Version System
F90	Fortran 90 language
GUI	Graphical User Interface
HPF	High Performance Fortran
Paraiso	Parallel Iterative Solvers library
PIPlot	PIPlot plotting library
TCL/TK	Tool Command Language

## 2. REFERENCES

1. PARAISO and AVISPA. <http://ulla.dec.usc.es/~paraiso/>
2. V. Blanco, J.C. Cabaleiro, P. Gonzalez, D.B. Heras, T.F. Pena, J.J. Pombo and F.F. Rivera. "Performance prediction for parallel iterative solvers". Int. Conference on Computational Science. ICCS 2002.
3. V. Blanco, J.C. Cabaleiro, P. Gonzalez, D.B. Heras, T.F. Pena, J.J. Pombo and F.F. Rivera. "A performance visualization tool for HPF and MPI iterative solvers". 16<sup>th</sup> Int. parallel and distributed processing symposium (IPDPS 02).

### 2.1. SOURCE CODE

The tool will be called PPC. The current version integrated in the CVS repository is highly dependent on the Avispa tool, but we will include new features in the near future. These features are being developed in our local cluster. We use the standard TCL/TK as API, and the computational parts of the tool are coded in C and C++. In addition to PPC we included in the repository the coded kernels (code of the kernels considered by PPC).

### 2.2. CONTACT INFORMATION

**Francisco F. Rivera** ([fran@dec.usc.es](mailto:fran@dec.usc.es)) is the responsible of this work.

The technical staff is mainly composed by:

**Tomás F. Pena** ([tomas@dec.usc.es](mailto:tomas@dec.usc.es))

**José C. Cabaleiro** ([caba@dec.usc.es](mailto:caba@dec.usc.es))

**Marcos Boullón** ([marcob@usc.es](mailto:marcob@usc.es))

All of us are in:

Dpto. Electronica y Computacion

Fac. Fisica

Univ. Santiago de Compostela

15782 Santiago de Compostela

SPAIN.

Phone. +34 981 563100

Fax: +34 981 599412

### 3. IMPLEMENTATION STRUCTURE

The implementation of the tool have two different works: on one hand we have to study in detail the behaviour of the kernels to be considered by the tool, and on the other hand we have to develop the functionalities of the visualization tool, some of these functionalities depend on the kernels themselves. Therefore this section is divided into two subsections.

#### 3.1. Kernels

We are currently considering three different kernels: sparse matrix algebra problems, dense matrix - vector product and the vertlq routine. The status of the analysis of these kernels is different:

- **Sparse matrix algebra problems:**

We were adapting the HPF codes of the Paraiso library into efficient MPI versions. Therefore, a large part of the work developed to characterize the computational work was already done in a previous work, however, the communications analysis must be performed again, as the MPI behaviour is quite different from the HPF one. In particular, it is important due to the decrease in the number and size of the involved communications. These kernels can be used in some of the applications, so, as soon as we have enough good implementations of them, we can offer them to the users of WP1.

Currently, the new parallel version includes the following methods: Conjugate Gradient, Biconjugate Gradient Squared and the Generalized Minimal Residual. Others will be implemented in the near future, as well as some preconditioners and triangular solvers needed to complete the solution of a general linear system. Of course, the sparse matrix – vector routine was also implemented as it is needed for all the methods. In terms of the performance prediction model, the number of computations is obviously the same as in Paraiso, but the number and size of the communications is now much lower.

- **Dense matrix – vector product:**

This is the kernel in which we are paying more attention, and it is used to develop the visualization tool. This kernel is simple but with enough load in terms of computations and communications to be interesting as initial case of study.

Currently, the routine was coded and it is running. It was characterized in terms of computations and communications. In fact, the communications included in it are collective: broadcasts, gathers and scatters, and they are characterized in terms of point to point messages.

- **The vertlq routine:**

The last set of kernels that we have considered are those associated to the air pollution application in task 1.4 as it will be developed by a group in our institution, and we are quite familiar with it. We know this application in detail, and we are evaluating the main kernels. All of them require dense matricial computations with a small number of communications.

The nested loop structure of the air pollution model in task 1.4 is similar to many simulations based on a finite difference method. The sequential program consists of four main nested loops: a temporal loop, and a loop for each dimension of the simulated 3D space. The temporal loop specifies the duration of the simulation process and the time simulated in each iteration. The basic modules and routines of the program are the “hor” and the “vertlq” modules. They are responsible for the horizontal transport and for the chemical reactions simulations. In the experiments of the

sequential program, the results obtained are: more than 97% of the time is consumed in the “vertlq” module, and 2.5% in the “hor” module, so we first focused on this “vertlq” module, and we will include it in the set of kernels for our prediction tool. From the computational point of view, “vertlq” presents three nested loops. All of them can be parallelized, and in fact, in the parallel implementation, the processors are distributed in such a way that the user can decide which loops are parallelized. There are no communications inside the “vertlq” module, but a global reduction must be performed before each execution of the module. So, both the computations and communications structures are quite simple, and they can be modeled easily.

To obtain a precise prediction for these kernels, we need access to the execution of them on different clusters and finally on the Grid, but from now we have the precise number of events, and we can characterize some parameters of the prediction model as the memory load balance or the number of communications, among others.

One of the initial data to be considered is the number of floating point operations (FLOPs) required for every kernel. We counted the number of FLOPs for one iteration of the iterative solvers, as the number of iterations required to achieve convergence can not be predicted. These data consist on the evaluation of simple algebraic expressions involving certain parameters of the matrix at hand, such as the number of non-zero entries,  $n$ , or the dimension of the matrix,  $N$ , as well as some information on the operations, such as the grade in the polynomial preconditioners. Sometimes, the counting is much more expensive because it depends on dynamic parameters such as the matrix pattern. In the following table, the number of FLOPs for the kernels already analyzed is shown.

KERNEL	FLOPs
Dense matrix-vector product	$2N^3/P$
Conjugate Gradient	$(19N+7n-3)/P$
Biconjugate Gradient Squared	$(23N+9n-4)/P$
Generalized Minimal Residual	$(n^2+6n+13)N/P$
Vertlq	$wm^3/P$

Where, for the vertlq routine,  $m$  is the number of points in each dimension of the simulation space (assuming an  $m \times m \times m$  3D space),  $P$  is the number of processors, and  $w$  is an integer that depends on the number of chemical reactions to be considered by the application.

The study of the communication patterns generated by the MPI routines used in the kernels is essential for predicting their overheads. The straightforward way to check these patterns is to execute the program with profiling capabilities, taking results from different executions. In the current situation, we can get only information about the number of communications performed by each processor in the parallel kernels, but presently, it is impossible to characterize their costs because the Grid environment is not yet available. The following tables show the number and size of collective and individual communications in the current versions of the parallel kernels.

KERNEL	# collective	Size
--------	--------------	------

	<b>communications</b>	
Dense matrix-vector product	3	1
Conjugate Gradient	1	n/P
Biconjugate Gradient Squared	1	n/P
Generalized Minimal Residual	2	n/P
Vertlq	1	m <sup>3</sup> /P

<b>KERNEL</b>	<b># point to point communications</b>	<b>Size</b>
Dense matrix-vector product	0	0
Conjugate Gradient	N	c/P
Biconjugate Gradient Squared	N	c/P
Generalized Minimal Residual	N	c/P
Vertlq	0	0

Where c is the average number of entries per column of the considered sparse matrix.

In the particular case of the memory balance, it depends on the sparse matrix pattern for the iterative solvers, so the solution offered by Avispa is correct in this implementation, because the data distributions are exactly the same. On the other hand, the vertlq module and the dense matrix-vector product are perfectly balanced. Other performance features depend on the systems used to execute them, and the analytical models must be established at that moment.

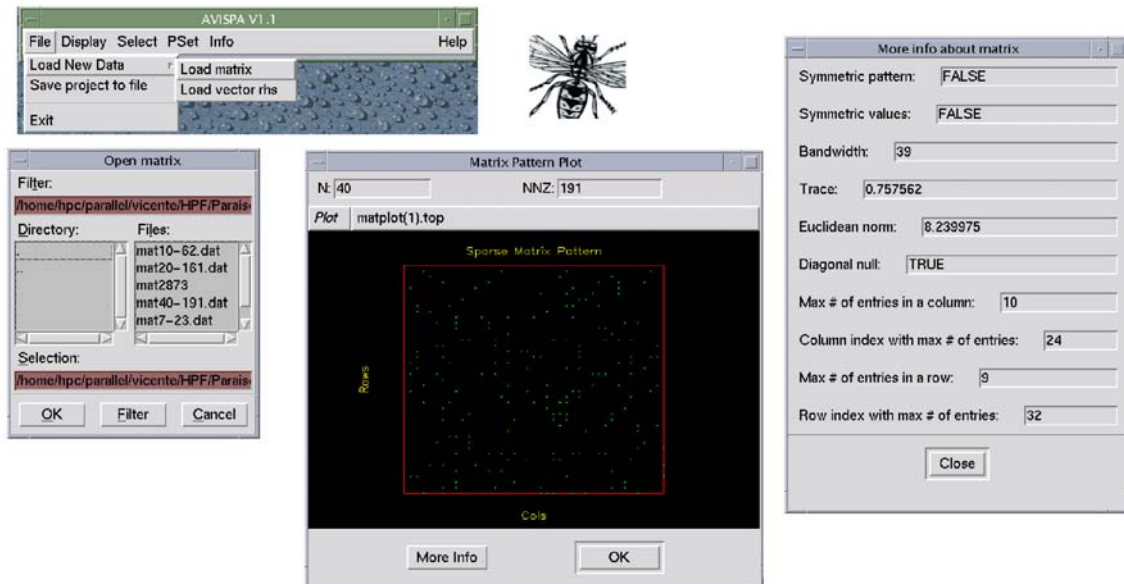
In order to characterize the collective communications, we have obtained the behavior of the broadcast, gather and scatter routines in terms of point to point communications. In this way, with the information about latency and bandwidth, the cost of these routines can be estimated.

### 3.2. The visualization tool

The visualization of the performance prediction data are based on Avispa, a visualization tool for the performance predictions of codes of Paraiso in a multiprocessor homogeneous environment. It was developed using standard TCL/TK, so it can be adapted to different GUIs.

The data to be shown in the graphical interface are application's dependent. Currently, we are adapting Avispa to a cluster and Grid environments, so the results to be shown are the ones corresponding to the iterative methods used in Avispa. Therefore, the results of the performance predictions have to be integrated into an interactive visual tool to be user-friendly for users or developers. This tool will be based on Avispa (A VISualization tool for PAraiso) for the iterative solvers, and it should be similar for other kernels, except for their particular features. In addition, for the iterative solvers, this tool must include detailed information about the matrix itself, the methods, preconditioners and kernels, as

well as other information about the execution of these codes. In fact, the visualization tool allows the user to interact in the analysis of the behavior of the codes of Paraiso under different conditions (number of processors, different distributions, different matrices, ...). Next, some examples of the capabilities of Avispa are shown, they can help the users to understand how the final product should work for their kernels.



**Figure 1**

Figure 1 shows the selection of a sparse matrix to be analyzed by the tool. In particular, it shows the menu to load the matrix, that launches the window to select the matrix file from disk. The user can see the pattern of the matrix and some additional features in two different windows as shown in this figure.

As soon as the matrix and the right hand side vector of the system were selected, the user can choose the number of processors, method and preconditioner, and the parallel version of the codes. Figure 2 displays examples of how some of these selections can be done in Avispa. In addition, a window displaying statistics about the number of FLOPs is included in this figure, and another one provides theoretical information about the selected method.

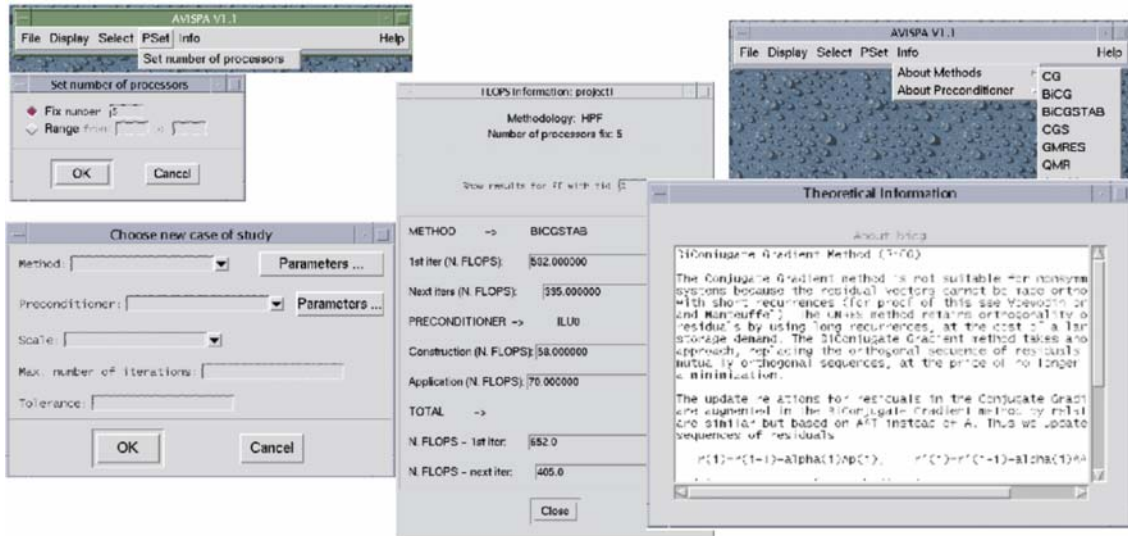


Figure 2

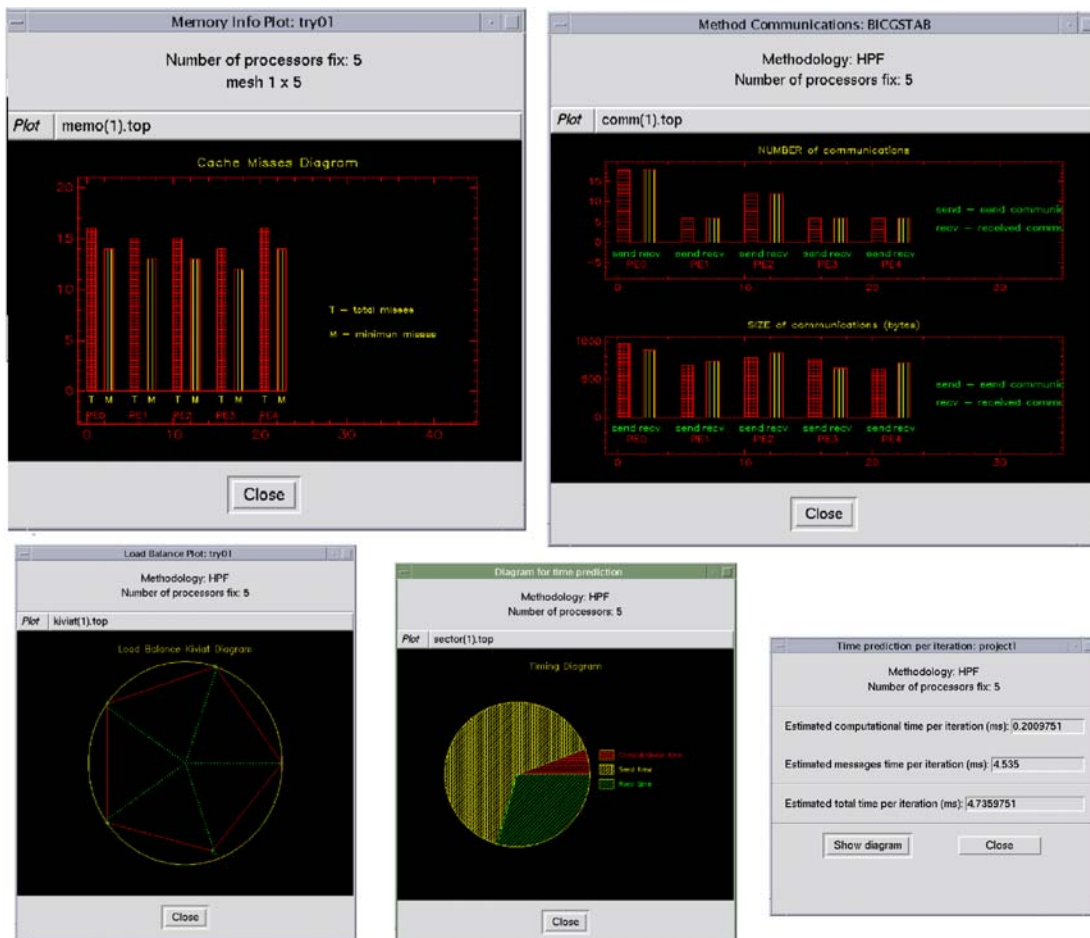


Figure 3

Among the large amount of performance prediction data, some examples are shown in figure 3. In particular, this figure displays information about perprocessor cache misses, and predicts number and size of communications, both sent and received messages. It also shows the predicted load balance using a Kiviat diagram, the ratio between computation costs and communication overheads in a sector diagram, and a window summarizing the total execution time per iteration.

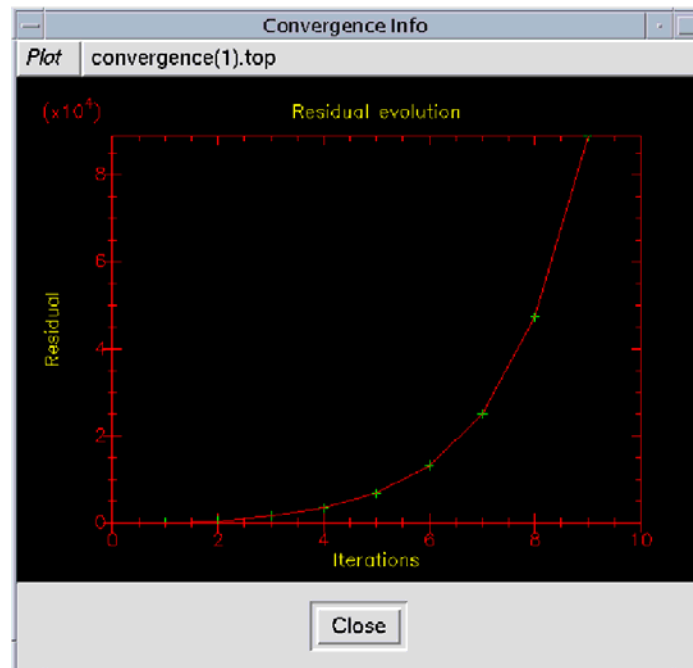


Figure 4

As an additional feature, a small number of iterations can be executed in order to analyze the residual evolution, and so, get a first approach about the convergence of the methods. Figure 4 shows the residuals for the first 10 iterations of a method that does not converge. This is a typical example of a feature that is application-dependent that can be included in the tool in an easy way.

The parameters that characterize the hardware are visualized as a triangular matrix that stores the latency, the bandwidth between each pair of nodes and the computational power of each node. In the present version we are considering just two levels: the LAN level and the cluster level, so we need a triangular matrix for each. This can be generalized to more complicated situations. The information provided by these matrices can be established by default or provided by monitoring or even modified by the user himself in an easy way. Therefore, hypothetical cases can be considered in which the network is faster or the number of nodes increases, and so on.

---

## 4. PROTOTYPE FUNCTIONALITY

The results obtained up to now agree with the provisions and are as expected. Two brief use cases can help to understand the functionality of the tool.

- **Use case 1, for a real user of the tool:**

1. As soon as the user executes the tool, he can obtain information about any of the kernels on real monitored features in the Grid, basically latencies, bandwidths, FLOPS of each node, ...
2. In particular, predicted runtimes are depicted.
3. Some particular points can be extracted, like the load balance, the number and size of the communications, ...
4. Some specific features can be obtained, like the pattern of a sparse matrix, or the fact that a particular matrix is symmetric, ...
5. He can modify the performance features on the Grid to check the behaviour of the kernel in other Grid circumstances: with more nodes, or faster links, ...
6. All of these studies are done in any local workstations or PC, because Grid computations are not needed.

- **Use case 2, for the developers of the performance models:**

1. The developer executes enough times the kernel in the Grid under different hardware features.
2. The number of FLOPs and communications, as well as their size are obtained by studying the kernels.
3. The behaviour of the communications is established, in particular, this is important for collective communications.
4. A statistical analysis of data from the above steps is done in order to get an analytical model for the computations costs and the communications overheads.
5. The analytical models are included in the visualization tool.

## 5. USER MANUAL

### 5.1. DEPENDENCIES

To compile PPC, it's necessary a F90 version of Paraiso Library. Due to this, a Fortran 90 Compiler is required. As the current repository of PPC in CVS is mainly based on Avispa, this dependency will be removed soon in new versions.

PPC requires Tcl/Tk with Incr Tcl extensions and Megawidgets and a plotting library for graphs and charts (PIPlot, a 2D/3D-plotting library). Current version is working with Tcl/Tk 8.0.5, ITcl 3.0.0 and PIPLot, version pplot-990122. This software is installed in RedHat 6.2 by default. Also PIPlot is available for Linux as a RPM package.

### 5.2. INSTALLATION

1. Go to Paraiso HOME (PARAHOME) and type "make F90" to construct a F90 version of Paraiso.
2. Edit PPCmake.inc to set environment variables and type "make PPC".
3. To execute PPC, set PARAHOME and PATH environment variables:
  - a. sh style:
    - i. export PARAHOME=\$HOME/Paraiso
    - ii. export PATH=\$PARAHOME/PPC/bin:\$PATH
  - b. csh style:
    - i. setenv PARAHOME \$HOME/Paraiso
    - ii. setenv PATH \$PARAHOME/PPC/bin:\$PATH

### 5.3. RUNNING

In this section the use of PPC is shown in detail. This document covers the basics for using PPC: how to prepare an application for PPC, run it, view its performance data, and find performance bottlenecks in the application. PPC is easy to install and its usage is fairly self-explanatory. The figures in this description are from Avispa, but they are exacty the same for PPC.

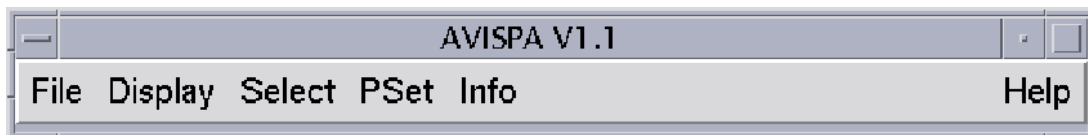


Figure 5: Main control window

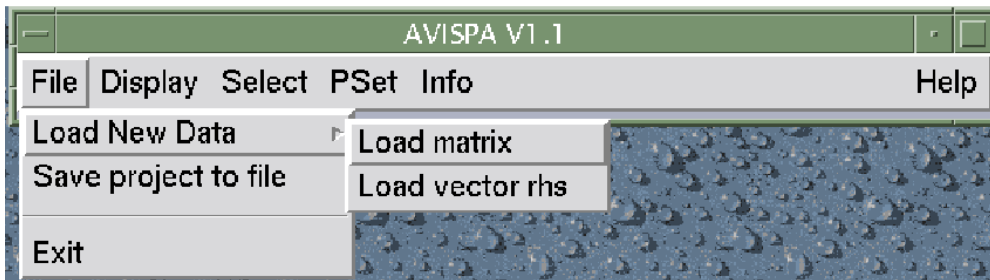


Figure 6: Main menu to load a matrix and a vector from disk

PPC starts running and displaying the main control window shown in Figure 5. In this window there are six buttons. The first one is used for loading new data to work with. The menu to load them is shown in Figure 6. Note that the last option of this menu is the way to exit PPC.

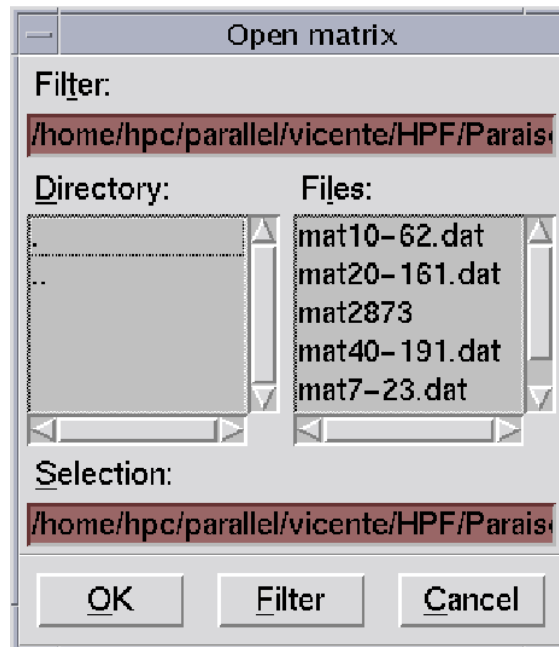


Figure 7: Dialog to load a sparse matrix from a file

When buttons *Load new data* and *Load matrix* are clicked, the dialog in Figure 7 is opened. This allows you to choose a file from disk in which a matrix is stored. A similar dialog appears when buttons *Load new data* and *Load vector* are selected to load a vector from disk.



Figure 8: Project set menu

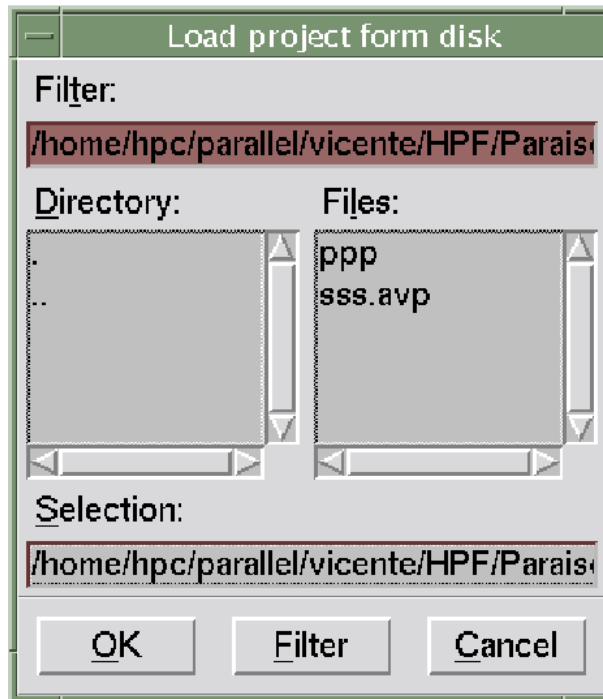
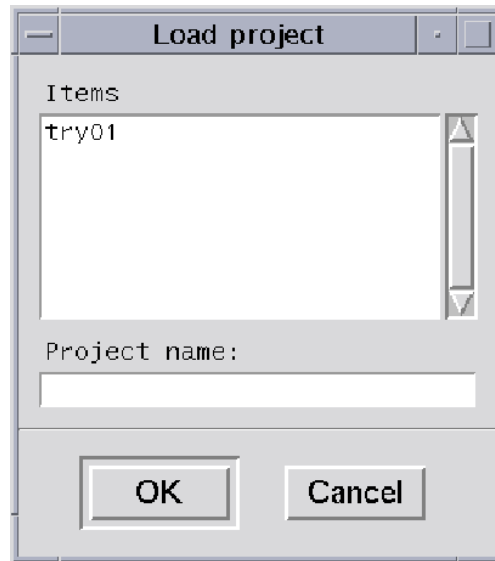


Figure 9: Dialog to select a project from disk

Before running the display of performance information, a project must be started. Figure 8 shows the way to load a project from the main menu. There are two possibilities, to create a new project or to load a previous one. A previous project can be selected from disk if it was previously saved. If this last option is selected, the dialog Figure 9 is opened. To load a project not saved on disk but used previously in this session, the dialog of Figure 10 is opened. Note in Figure 6 the button to save on disk the target project.



**Figure 10: Dialog to load a project from the current session**



**Figure 11: Window to create a new project**



**Figure 12: Window to accept a default name for a new project**

In Figure 11 the dialog to select a name for a new project is shown. In this example, the name *try01* was selected. When no name is typed, Figure 12 shows the window that automatically appears to accept a default name suggested by PPC.

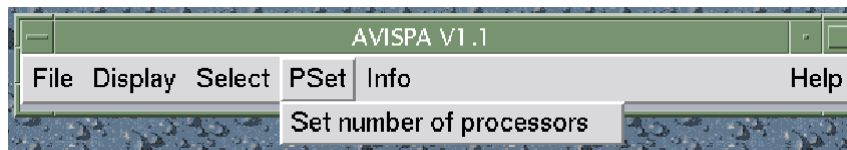


Figure 13: Menu to select the number of processors

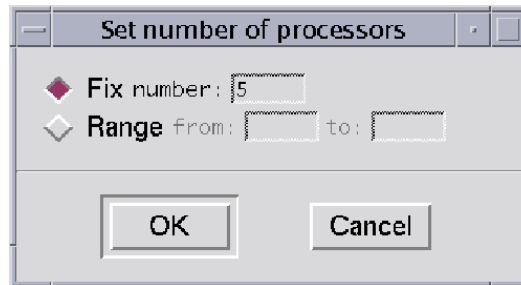


Figure 14: Dialog to select the number of processors

Before showing performance results, the number of processors must be fixed. The menu to be selected for that is shown in Figure 13. When this selection is done, the dialog of Figure 14 appears in a new window. Note that there are two possibilities to select the number of processors: a fixed value or a range of numbers. In the example of this figure five processors were selected. If a range was chosen, performance results for any number of processors in the range can be visualized to study the scalability of the codes. By default, the number of processors is one.

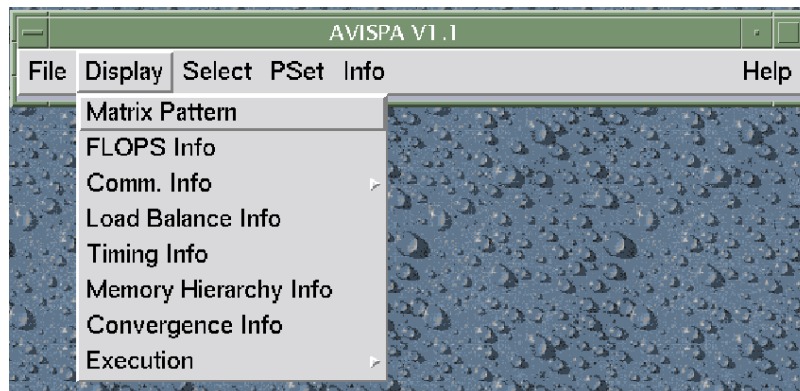
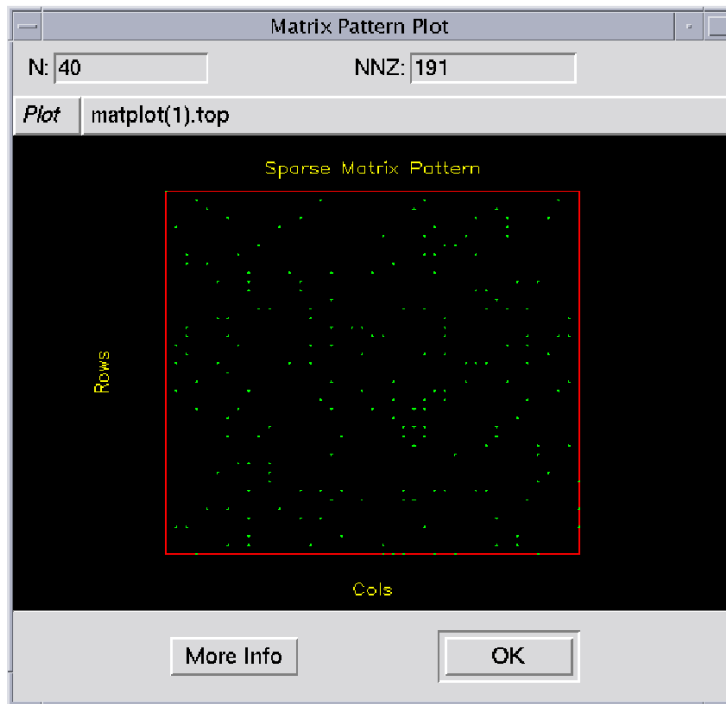
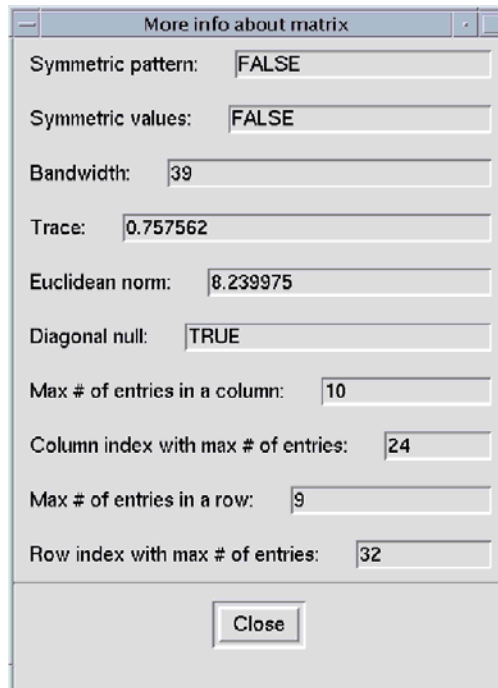


Figure 15: Menu to select the information to be showed

At this point, PPC is ready to start running the visualization. You can now select the *Display* button from the PPC main control window to start a visualization process, as shown in Figure 15. Data presentation modules include bar graphs, Kivi diagrams, 2-D scatter plots, matrix displays, polar plots, etc.



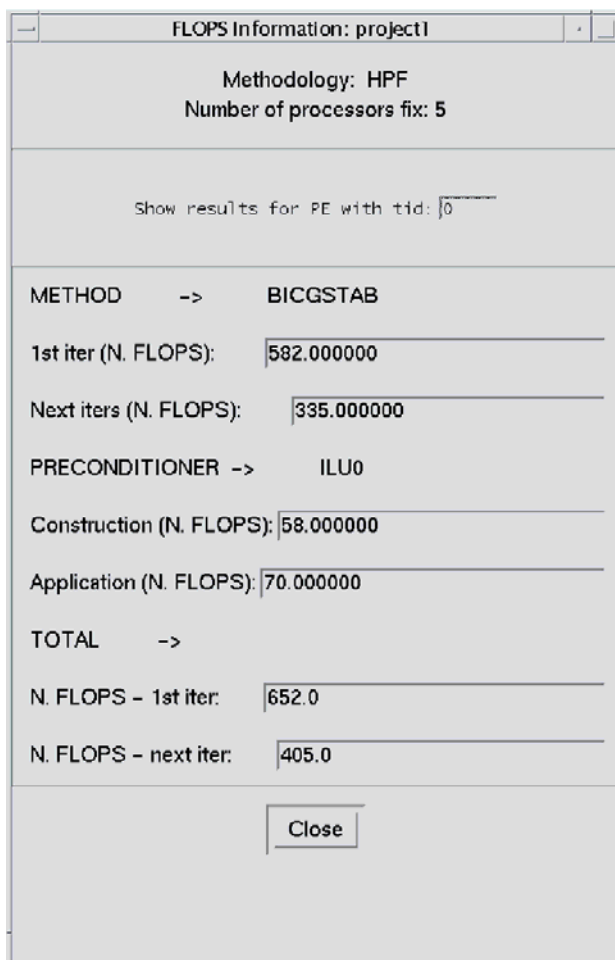
**Figure 16: View of the sparse matrix information**



**Figure 17: View of more information about the sparse matrix**

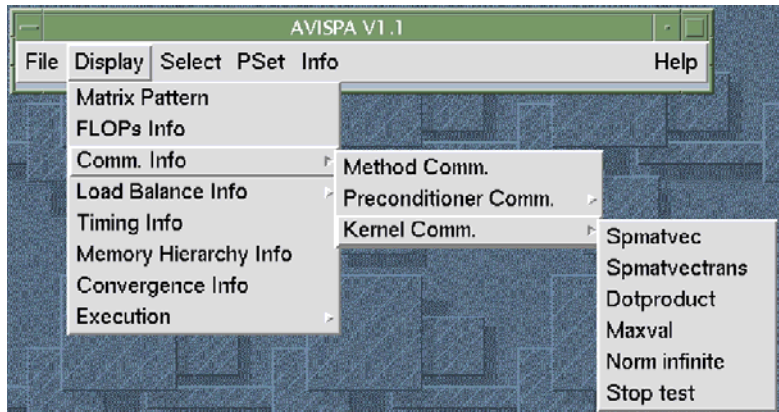
The first option in the *Display* menu is for showing information about the sparse matrix. The information is shown in a new window like the one in Figure 16. In this figure, the number of rows and the number of non-zero entries are displayed (in our example these numbers are 40 and 191 respectively). Additionally, the pattern of the matrix is also shown and an advise if the matrix is symmetric or not. The display can take some seconds to proceed.

The button called *More Info* in the window of Figure 16 offers more information about the sparse matrix. This information includes: if the matrix is symmetric or not, both in terms of the pattern and in terms of the values, the bandwidth, the trace, the Euclidean norm, if the diagonal is null, the maximum number of entries in a column and its index, and finally, the, maximum number of entries in a row and its index. This information could be valuable in different applications, but it is not usual, this is the reason because it is shown in a secondary window. An example of this window is shown in Figure 17.



**Figure 18: Information about the number of FLOPs**

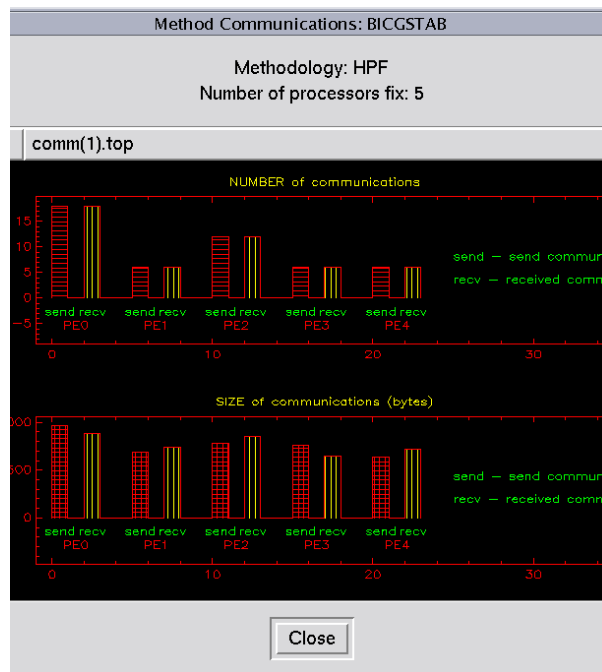
Option *FLOPs Info* from the *Display* menu is used to check computational information in terms of the number of FLOPs to be executed in each processor. Figure 18 shows an example of these results. The user can select a particular processor from whom the information is displayed. Integer numbers from 0 to the number of them minus one identifies processors. Then, the number of FLOPs for the first and for any other iteration of the method are displayed, as well as the number of FLOPs needed for the construction of the preconditioner and for its application. The total number of FLOPs for any iteration is finally shown also for the first and the next iterations.



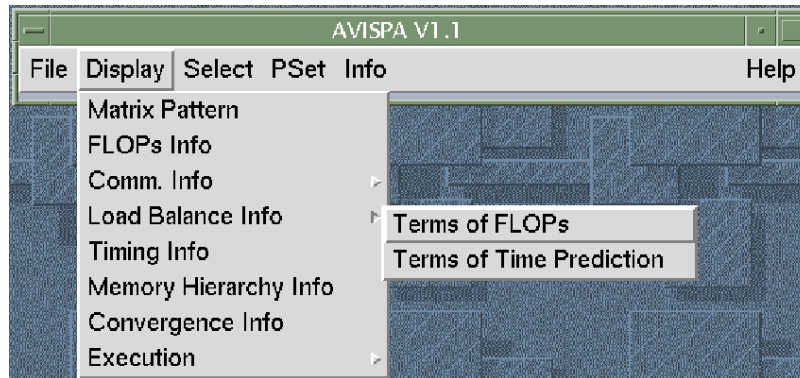
**Figure 19: Menu to select communications information about the kernels**

The menu to select the view of information about communications is shown in Figure 19. With this information the user can check the influence of the communications in a particular kernel. To choose the kernel for which the metric will be collected, you select resources by clicking on the appropriate button on these menu.

As an example, see Figure 20. This view shows the sends and receives executed on each processor. This view provides information regarding both, the number of messages sent and received, and the number of bytes sent and received. Two horizontal bars represent each processor, with different colors for message activity: number and size of sent and received messages. Heights of the bars are used to represent the number and size of the messages.

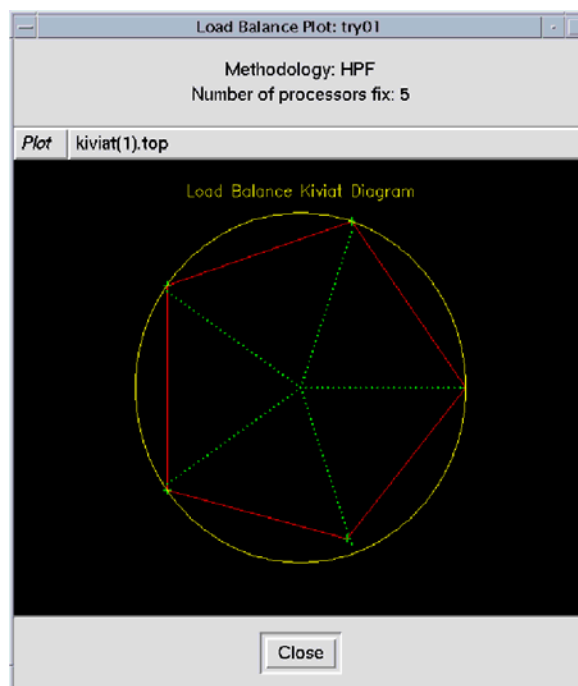


**Figure 20: Bars information about communications**



**Figure 21: Menu to select the kind of balance information to be shown**

If option *Load Balance Info* in Figure 21 is selected, two possibilities are found: to show it in terms of the number of FLOPs or to show it for the estimated execution time. A window like the one of Figure 22 automatically appears for the first option. It shows a Kiviat diagram for the FLOPs in each processor.



**Figure 22: Kiviat diagram of the number of FLOPs for 5 processors**

If the second option is selected, then a window like the one in Figure 23 appears. It shows a Kiviat diagram for both, the number of FLOPs and the estimated communication time (for send and receive messages) for any processor. Using this view, the user can check the influence of these events on the load balance. Both views are quite important to detect unbalanced situations.

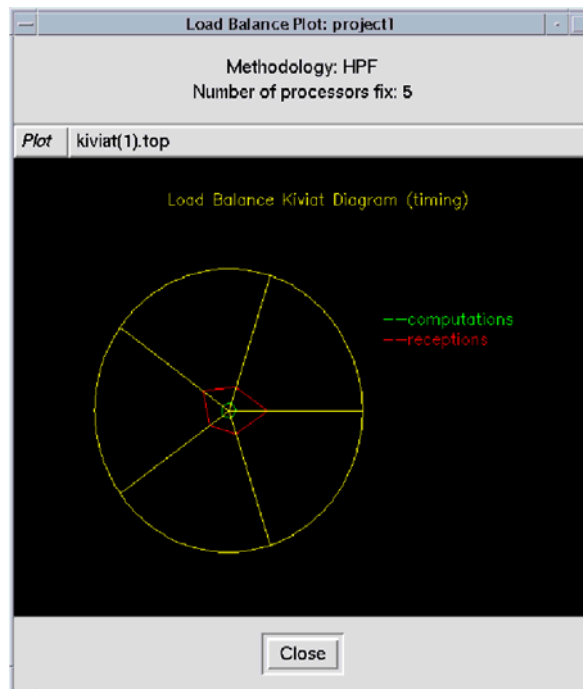


Figure 23: Kiviat diagram of communications and computations

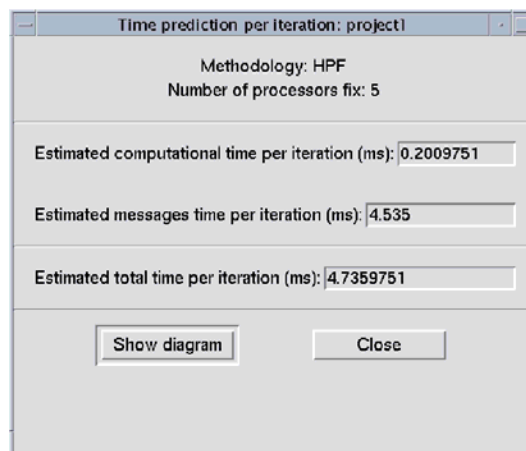


Figure 24: Estimated time per iteration

The next option in the menu of Figure 15 is to get an estimated runtime for an iteration of the kernel. This option is called *Timing Info*. As soon as this option is selected, a window like the one in Figure 24 appears showing this information. Button *Show diagram* can be used to view this information in a sector diagram like the one in Figure 25.

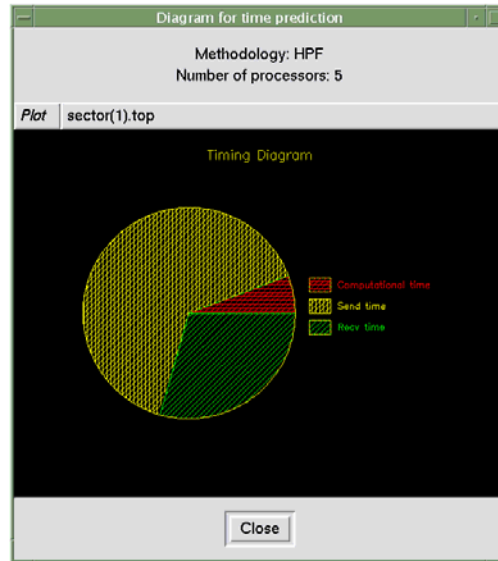


Figure 25: Estimated runtime due to execution and communications

Finally, the option called *Help* (Figure 26) in the main menu is used to get information about PPC. An example of this information is shown in Figure 27.

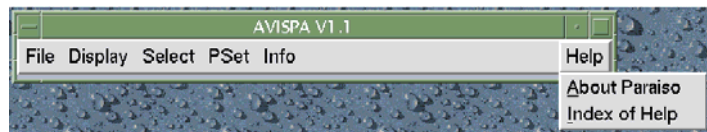


Figure 26: Help option

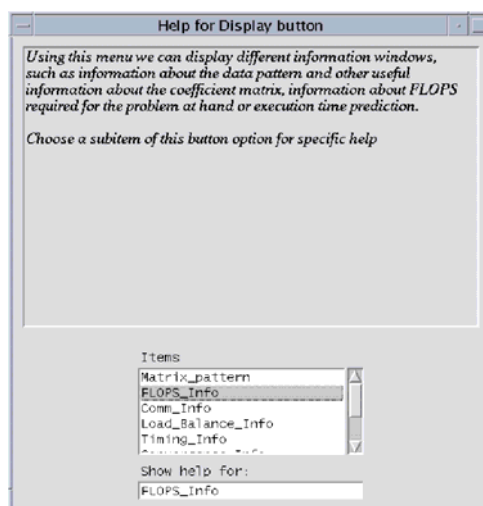
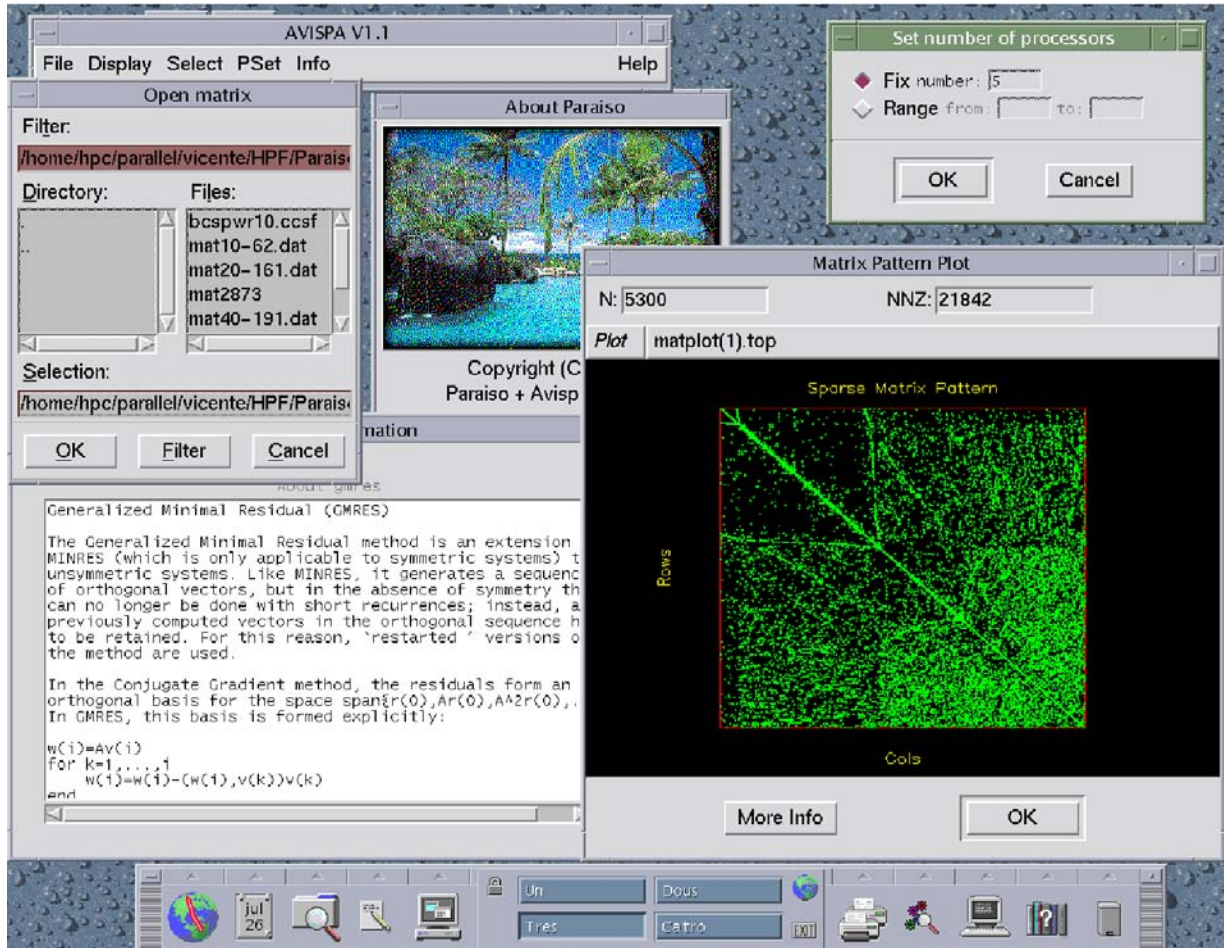


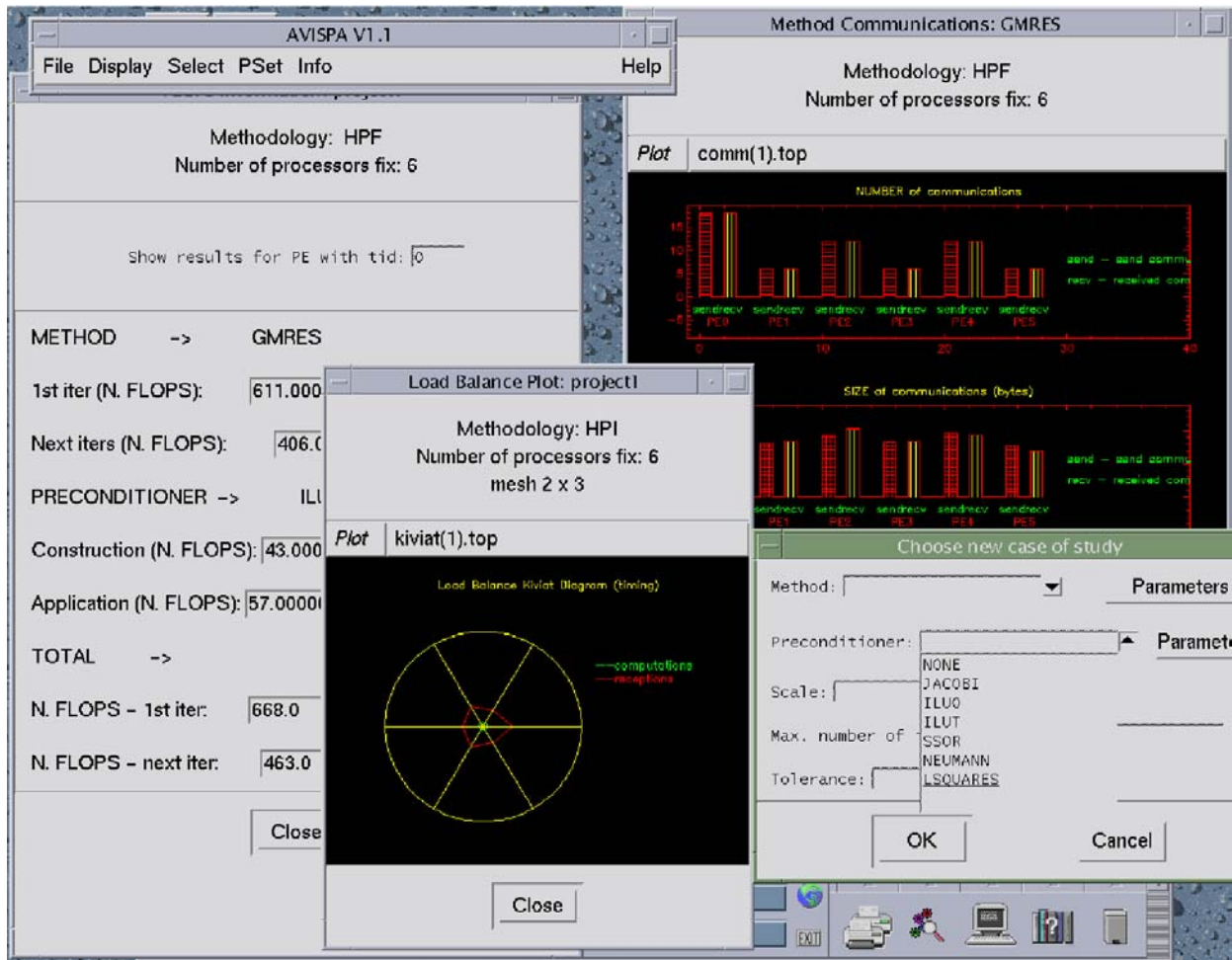
Figure 27: Example of help

The snapshots in Figure 28 show an example of the tool, in particular it shows the main menus, the menu for selecting a sparse matrix from a file, the pattern of this matrix, the help window and the window for selecting the number of processors to execute the parallel code.

And, in Figure 29, note the performance consultant window that shows the statistics for each process, a Kiviati graph showing the load balance and the histogram of the number and length of the messages to be sent and received by each processor.



**Figure 28: A view of a working session**



**Figure 29: A view of a working session**

**5.4. INTERFACE DESCRIPTION**

The prototype does not provide any API.

## 6. INTERNAL TESTS

The internal tests about the visualization prototype meet the functionalities, but no tests were done about the accuracy of the performance prediction. They were only validated in the local cluster.

## 7. ISSUES

The prototype has a dependency with the Paraiso library. This dependence is going to be removed in future versions.