



Vampir 2.0

Tutorial

---

Pallas GmbH  
Hermülheimer Straße 10  
D-50321 Brühl, Germany  
<http://www.pallas.com>



---

Vampir was originally developed by the Zentralinstitut für Mathematik at the Forschungszentrum Jülich.

We wish to thank Prof. Hoßfeld, Dr. Wolfgang Nagel and Dr. Alfred Arnold for the excellent work and the good cooperation.

Vampir is a trademark of the Forschungszentrum Jülich.

Forschungszentrum Jülich 

 pallas

# Contents

<b>Contents</b> .....	<b>i</b>
<b>1 Welcome to Vampir</b> .....	<b>1</b>
1.1 Introduction .....	1
1.2 Conventions used in this Tutorial .....	1
<b>2 Vampir first steps</b> .....	<b>2</b>
2.1 Before you start .....	2
2.2 Running Vampir .....	2
2.3 Loading a Tracefile .....	3
2.4 Vampir Main Displays .....	4
2.4.1 Global Timeline Display .....	4
2.4.2 Summaric Chart Display .....	7
2.4.3 Activity Chart Display .....	10
2.4.4 Communication Statistics Display .....	12
2.4.5 Communication Length Statistic .....	15
<b>3 Analysing the NAS LU benchmark</b> .....	<b>17</b>
3.1 Statistical Analysis Based on the whole Program Execution .....	17
3.1.1 Loading the Tracefile .....	17
3.1.2 Global Displays .....	17
3.1.3 Timeline .....	18
3.1.4 Summaric Chart .....	18
3.1.5 Activity Chart .....	21
3.1.6 Communication Statistics .....	23
3.1.7 Process Displays .....	25
3.1.8 Process Timeline .....	25
3.1.9 Activity Chart .....	26
3.2 Summary .....	27
3.3 Detailed Analysis for Timeline Sections .....	28
3.3.1 Identifying a Single Time Step of the LU Sample .....	28
3.3.2 Structure of a Time Step .....	29
3.3.3 Lower Tridiagonal Solvers (jacl, blts) .....	30
3.3.4 Exercises .....	30
3.3.5 Filtering .....	31
3.4 Summary First Example .....	31
<b>4 Generation of Tracefiles (Matrix Multiply)</b> .....	<b>32</b>
4.1 Matrix Multiply (matmul) .....	32
4.2 Linking with Vampirtrace library .....	33
4.2.1 Exercises .....	33
4.3 User Defined Activities .....	33
4.3.1 Vampirtrace API .....	34
4.3.2 Instrumentation of pMxM (1) .....	34
4.3.3 Instrumentation of pMxM (2) .....	35
4.3.4 Instrumentation Hints .....	35
4.4 Summary .....	36
<b>5 Tracing “Real World” Programs</b> .....	<b>37</b>
5.1 Using a Configuration File .....	37
<b>6 Vampir – Tips &amp; Tricks</b> .....	<b>38</b>
6.1 Filter Arbitrary Symbols .....	38
6.2 Highlight Messages in the Timeline .....	38
6.3 Reduce Default Timeline Display Range .....	38
6.4 Use the Parallelism Display inside Global Timeline .....	38
6.5 Select Multiple Processes inside Timeline View .....	39



# 1 Welcome to Vampir

## 1.1 Introduction

This tutorial documents the main features and capabilities of Vampir release 2.0. It can help you quickly master the basics of using Vampir and move on to using Vampir's more specialized features by guiding the reader through a selection of useful scenarios. Along practical examples it is not a replacement for the complete reference manual distributed with the Vampir 2.0 package.

Vampir offers a lot of opportunities for every user to find his own style of using it. This tutorial gives some hints and should encourage the user to develop his own way in using Vampir.

The “*Vampir performance analysis toolkit for MPI*” consists of two packages:

<b>Vampir</b>	Tracefile visualization program with a graphical user interface (GUI) for X Windows desktops.
<b>Vampirtrace</b>	Instrumented MPI library to link with the user code for automatic tracefile generation on a parallel platform.

Chapter 2 will provide an introduction into the most important elements of Vampir by using a predefined tracefile. The user not familiar with Vampir will get an impression of important Vampir displays and their significance for program analysis.

Chapter 3 shows an in depth analysis of application hot spots and how Vampir can help the user to find these sections inside the traced application.

Chapter 4 deals with the whole process of tracefile generation with Vampirtrace's automatic MPI instrumentation and the possible use of user defined activities. The example code matrix multiplication is kept simple because the focus is on the basic technique of trace generation.

Finally some tips for tracing real world programs are given in chapter 5. Reduction of tracefile size is the central issue of this chapter closing with a more advanced “Tips & Tricks” section.

## 1.2 Conventions used in this Tutorial

- Instructions will be given in form of bullet lists for emphasis of different steps of execution.
- Clickable menus are printed **bold**.  
For example: **File / Open Tracefile**. It is activated by selecting the first menu on the left (**File**) and selecting **Open Tracefile** inside the **File** menu.
- Keyboard shortcuts are printed in verbatim-bold with a box around each key.  
For example: **ALT-O** is activated by pressing the keys **ALT** and **O** together.
- File, directory and function names are printed verbatim: `vampir`.
- Context menus are associated with a display and were activated by a right mouse button click inside the window area.



## 2 Vampir first steps

### 2.1 Before you start

Vampir will create a configuration file called `VAMPIR2.cnf` in the directory `$HOME/.VAMPIR_defaults`, with `$HOME` referencing your home directory. This file contains all configuration settings for Vampir and enables Vampir to preserve settings from one session to another. If you don't already have a configuration file, you should create the directory `$HOME/.VAMPIR_defaults` and copy the system-default file `$VAMPIR_ROOT/etc/VAMPIR2.cnf` into it.

For Vampir 1.0 users the existing configuration file `~/.VAMPIR_defaults/VAMPIR.cnf` will be automatically migrated to the new `VAMPIR2.cnf` file. So all Vampir 1.0 settings are immediately available.

If there's no configuration file at all, Vampir will create one with default values that allow to start the visualisation.

Vampir has to allocate a lot of different color cells to display all information properly. In conjunction with other color greedy applications like "Netscape Navigator" it is possible (especially on 8-bit displays) that the colormap does not have enough free cells. To avoid this, the user can invoke Vampir with the command line option `-install` to instruct Vampir to use a private colormap instead of the default.

### 2.2 Running Vampir

We assume that Vampir has been installed on the system you are using. If that is not the case, please refer to the "*Vampir Installation Guide*" for help. Let `$VAMPIR_ROOT` be the root directory of your Vampir installation, e.g. `/usr/local/vampir`.

The Vampir executable image resides in the directory `$VAMPIR_ROOT/bin`. This directory should be included in the user's executable search path (shell variable `path` or environment variable `PATH`).

To start Vampir please type :

```
% vampir
```

After Vampir has completed startup, the main window shown in Figure 2.1 is displayed:



Figure 2.1 Vampir main window

## 2.3 Loading a Tracefile

To open a trace file select the menu item **File / Open Tracefile** or press the keyboard shortcut **ALT-O**. A Motif-style file selector as shown in Figure 2.2 will pop up.

To actually open the provided tutorial tracefile, either double-click on the appropriate entry in **Files**, or enter the filename `$VAMPIR_ROOT/etc/lu.S.4.bpv` into the **Selection** field and click on the **OK** button.

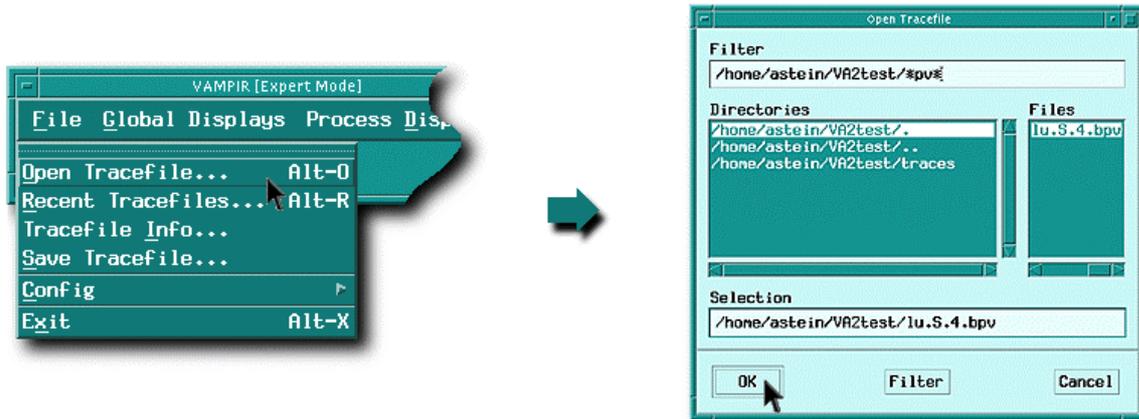


Figure 2.2 Open Tracefile

During the process of loading a tracefile the status panel inside the Vampir main windows will show a progress bar and three buttons to **Pause**, **Resume** or **Cancel** the tracefile loading. The different status panels while loading a tracefile are shown in Figure 2.3.

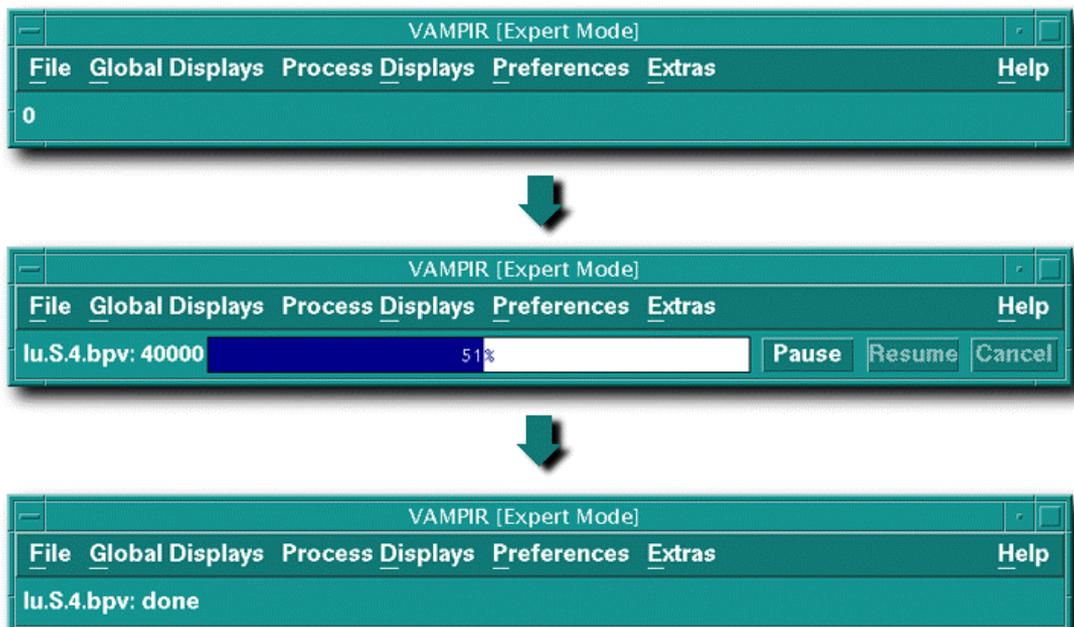


Figure 2.3 Vampir 2.0 main menu bars

## 2.4 Vampir Main Displays

### 2.4.1 Global Timeline Display

After loading the tutorial tracefile `$VAMPIR_ROOT/etc/lu.S.4.bp` the global timeline view (available via **Global Displays / Timeline** or keyboard shortcut **ALT-O**) will popup *automatically* as shown in Figure 2.4.

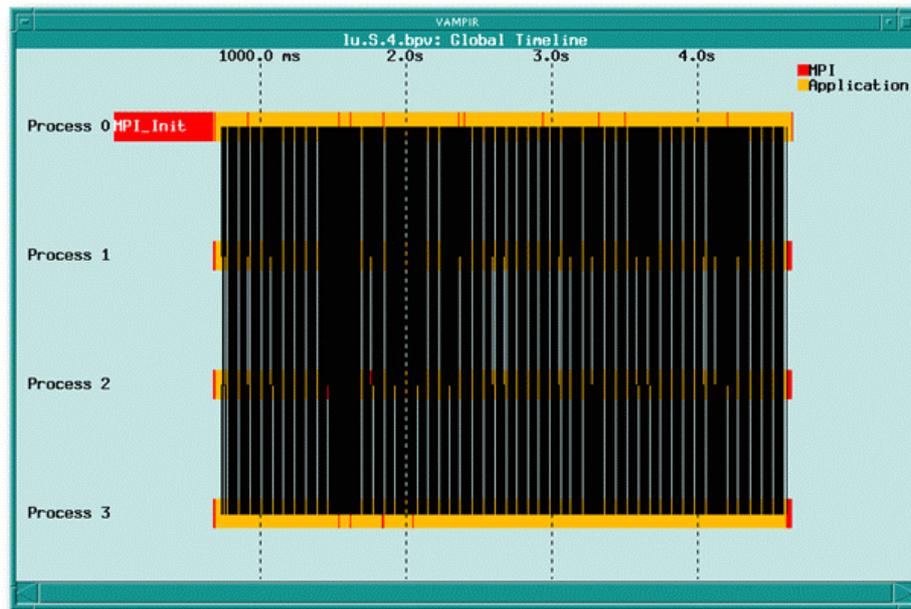


Figure 2.4 Global Timeline Display

In default mode this view presents an overview of the whole execution trace. For each process, the display shows the different states and their change over execution time along a horizontal time axis. Messages between processes are indicated as lines connecting the sending and receiving process.

Vampir implements a two-level state model. The states are grouped into activities, with a color being associated with an activity. The timeline display shows the activity colors to indicate state transitions; therefore, any state changes within an activity cannot be indicated. This method was chosen because the relatively high numbers of different states in a parallel application makes color-coding by states ineffective. Vampir displays a list of all activities occurring in the displayed part of the trace and their associated color in the upper right corner of the window.

By default, the timeline view shows the whole execution trace. Even with smaller traces this will lead to a cluttered display like that shown in Figure 2.4. To concentrate on a special part of the tracefile please invoke the Vampir zooming function by selecting an area of interest with your mouse. To zoom into a part of the timeline view, move the mouse pointer to the start of the interval you want to zoom into, press the *left mouse button*, drag the mouse to the end of the zoom interval while keeping the left mouse button down (only the x-coordinate matters). Vampir will indicate the marked region with rubber-bands. Finally release the mouse button. The timeline display will be redrawn showing just the time interval you selected, with the contents magnified accordingly. Figure 2.5 shows the process of zooming into a selected time interval.

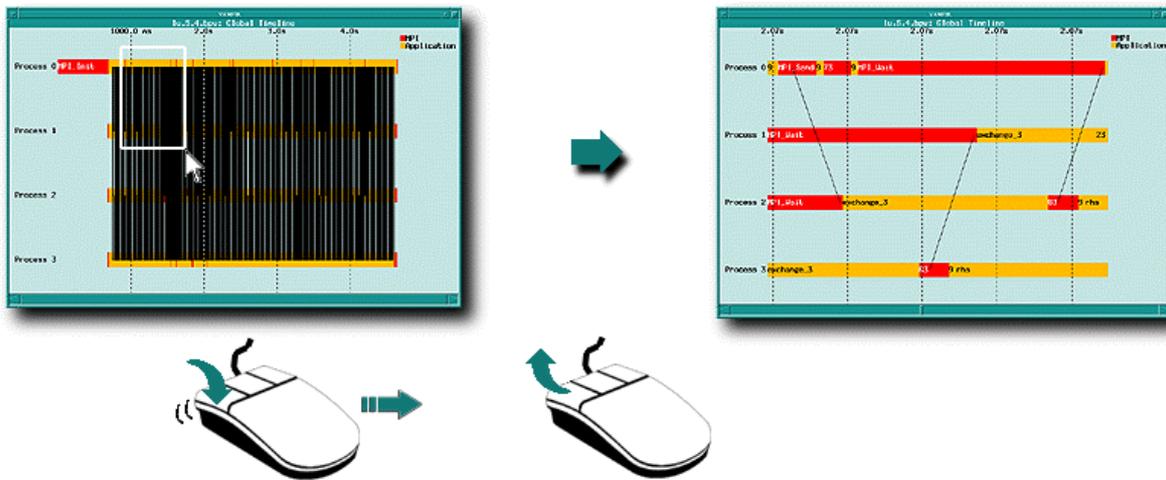


Figure 2.5 Timeline view Zoom-In

You can repeatedly zoom into arbitrary levels of detail. Zooming out step-by-step can be done with the **Undo Zoom** function of the context menu, or with the hotkey **U**. Figure 2.6 show the necessary steps to undo zoom the timeline display.

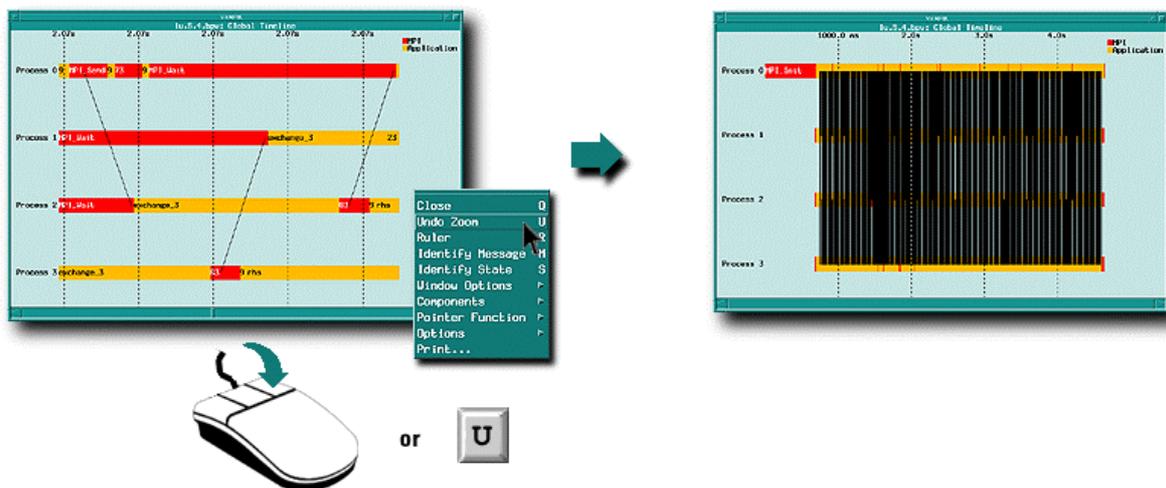


Figure 2.6 Timeline view Undo-Zoom

To get detailed information about a particular state or a single message invoke the functions **Identify State** or **Identify Message** from the timeline context menu or by pressing the timeline window hotkeys **S** for state or **M** for message. The context menu is activated by a *right mouse button* click inside the timeline window. After selecting one of the identify functions the mouse cursor shape is changed into the “target” symbol “**⊙**”. This new mouse cursor shape indicates that you have to identify the wanted state or message line by a single *left mouse button* click so that Vampir can show you further information. Please see Figure 2.7 for an illustration of these steps.

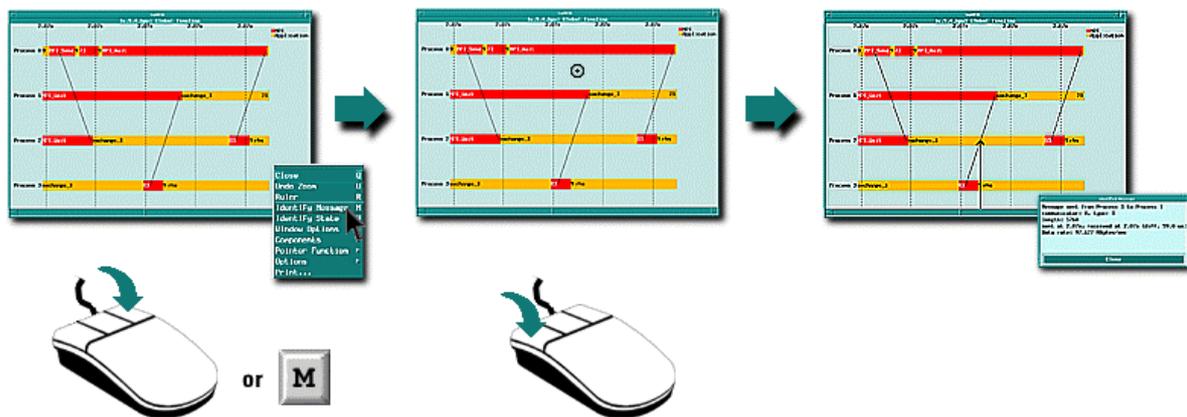


Figure 2.7 Timeline view Identify message action

The global **Timeline** view is the central display of Vampir because all other global and process specific statistic displays can be configured to use only the portion of time the timeline view displays. This option can be selected for a single display by selecting **Use Timeline Portion** from the appropriate context menu or as a global default value by setting the option **Use Timel. Portion** in the general preferences dialog available via **Preferences / Display / General**.

**Global Timeline Summary:**

- It will automatically open when pause or finished loading a tracefile.
- By default it will show the whole execution trace.
- Zooming in can be done by simply point and drag with your mouse in unlimited depth.
- Zooming out can be done step-by-step with the **Undo Zoom** function from the context menu, or by the hotkey **U**.
- Detailed information about states or messages can be shown by invoking **Identify State** or **Identify Message** from the timeline context menu or by using the hotkeys **S** (state) or **M** (message) and clicking on the state or message of interest.
- Central display to chose a timeframe for associated statistic views.
- For a complete description of all available Timeline features please read chapter 4.3.1 of the “Vampir 2.0 User’s Guide”.

## 2.4.2 Summaric Chart Display

The **Summaric Chart** display shows the sum of the time consumed by all instrumented activities over all selected processes. This is analogous to the information displayed by conventional profilers. To activate this view, select **Global Displays / Summaric Chart** from the main menu or press the hotkey **ALT-U**.

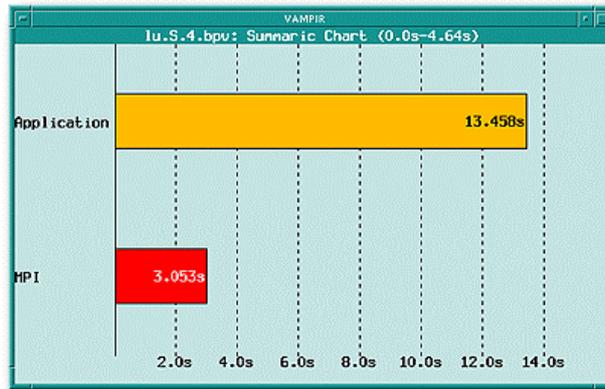


Figure 2.8 Summaric Chart View

By default, the **Summaric Chart** display shows a horizontal bar chart of those activities that occur in the time interval displayed in the window's top line, like shown in Figure 2.8. To get a statistic of a specific time interval activate the option **Use Timeline Portion** from the context menu. Now an arbitrary time interval can be chosen in the timeline display.

By default the view shows how much time is used for the application program, and how much time were used by MPI calls. An interesting question is now, which MPI functions uses up most of the runtime? To break down the time used by the activity "MPI" into the time used by single MPI functions open the context menu by pressing the right mouse button and select **Display / MPI**. Now the display shows all traced MPI functions and the time spend inside (see Figure 2.9).

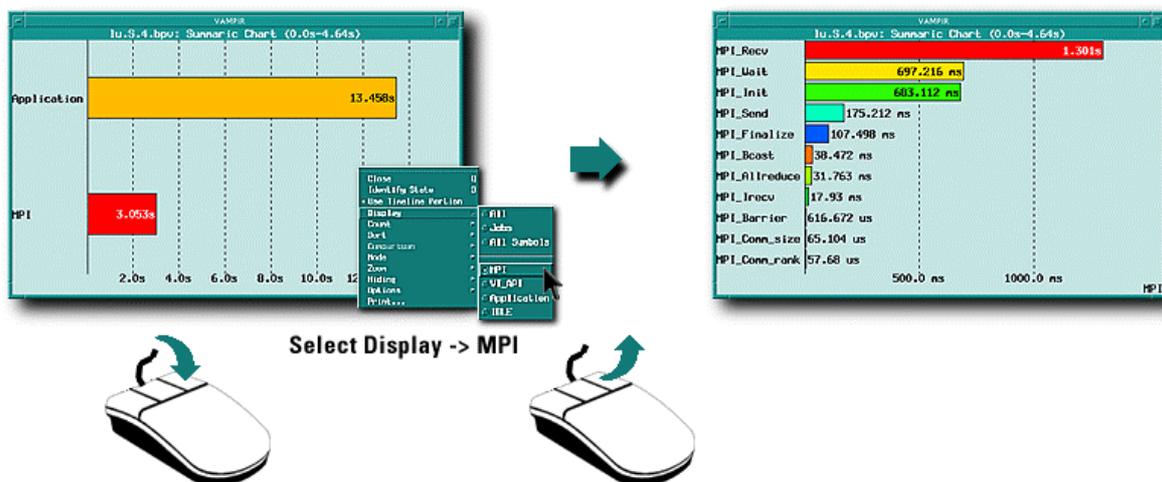


Figure 2.9 Summaric Chart Display for MPI functions

To generate an average, per process statistic open the context menu and select **Options / Per Process**. All times are now divided by the number of processes.

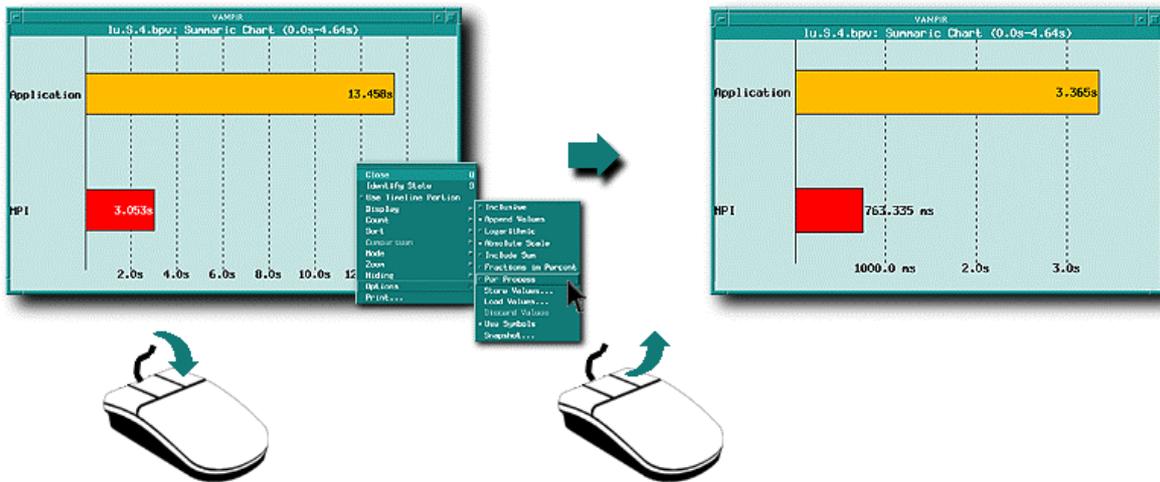


Figure 2.10 Summary Chart display – “per process”

By activating the option **Include Sum** from the context submenu **Options** an additional bar is included indicating the sum of all times shown by the display.

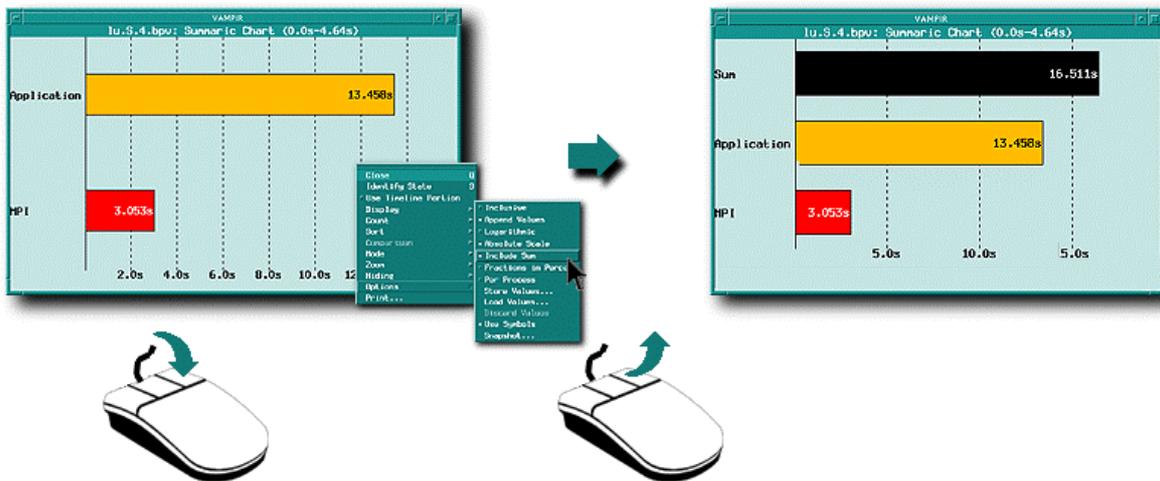


Figure 2.11 Summary Chart display – Include Sum

**Summaric Chart Display Summary:**

- By default it will show the sum of all times consumed by all processes.
- It is possible to look (“zoom”) into arbitrary activities by selecting a special activity from the **Display** submenu reachable by the context menu
- It is possible to get an average “per process” statistic by selecting **Options / Per Process** from the context menu.
- To see the relation between single bars and the total time used, by inserting a special “Summation” bar via **Options / Include Sum**.
- For a complete description of all features please read chapter 4.3.2 of the “*Vampir 2.0 User’s Guide*”.

### 2.4.3 Activity Chart Display

The **Global Activity Chart** display (available via **Global Displays / Activity Chart** or keyboard shortcut **ALT-A**) shows a statistic about the time spent in each activity individually for each process defined in the tracefile. It's default appearance is shown in Figure 2.12.

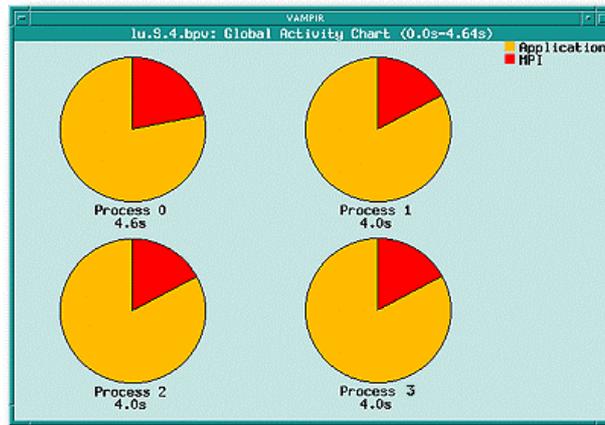


Figure 2.12 Global Activity Chart

With the default pie chart display you can recognise load imbalance at a glance in the traced program by comparing the different time consumption of activities over all processes (in our example MPI). Vampir can assist the user by visualising the actual trace data in different chart modes. Depending on the users preference the activity chart display can be switched to the so called "Histogram" mode. Please see Figure 2.13 for details.

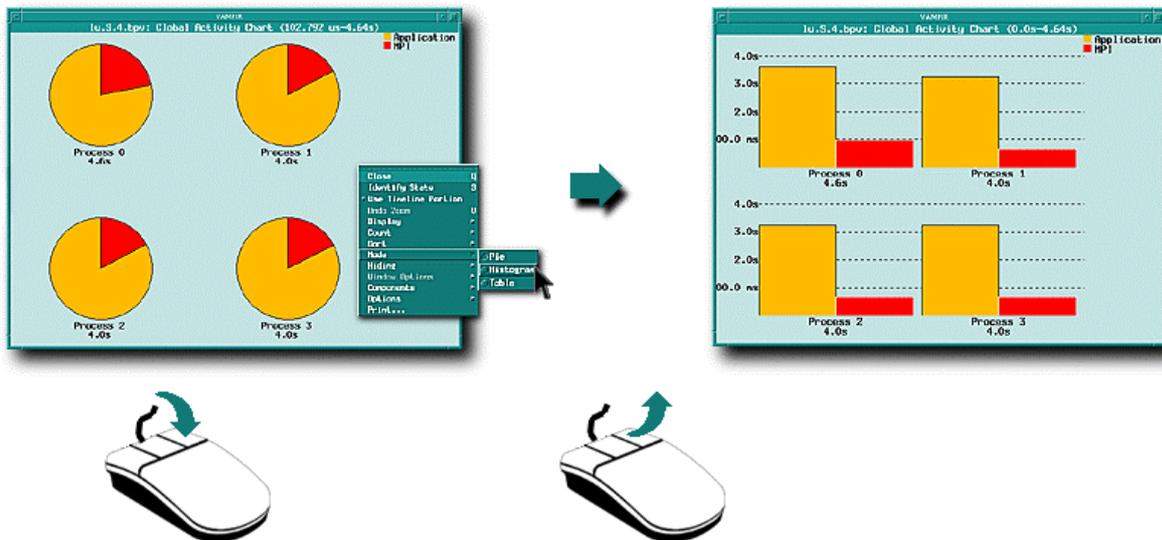


Figure 2.13 Global Activity Chart in histogram mode

To focus on a single activity, for instance **MPI**, please open the context menu with a right mouse button click inside the **Activity Chart** and select the activity **MPI** from the **Display** menu cascade (as you did before for the Summaric Chart display). A new set of pie charts is drawn, showing only the symbols of the selected activity MPI (see Figure 2.14). So you can easily compare different activities or symbols of all processes.



Figure 2.14 Global Activity Chart for MPI functions

#### Activity Chart Display Summary:

- By default it will show all activities for each process in a single pie charts.
- It's main purpose is to detect load imbalance. Different chart types assist the user in detecting the imbalance.
- It is possible to look ("zoom") into arbitrary activities by selecting a special activity from the **Display** submenu reachable by the context menu)
- For a complete description of all feature please read chapter 4.3.3 of the "*Vampir 2.0 User's Guide*".

### 2.4.4 Communication Statistics Display

The global **Communication Statistics** display shows a communication matrix describing the messages that were passed between each pair of sender and receiver. The default view will show the absolute number of bytes communicated between processes (see Figure 2.15).

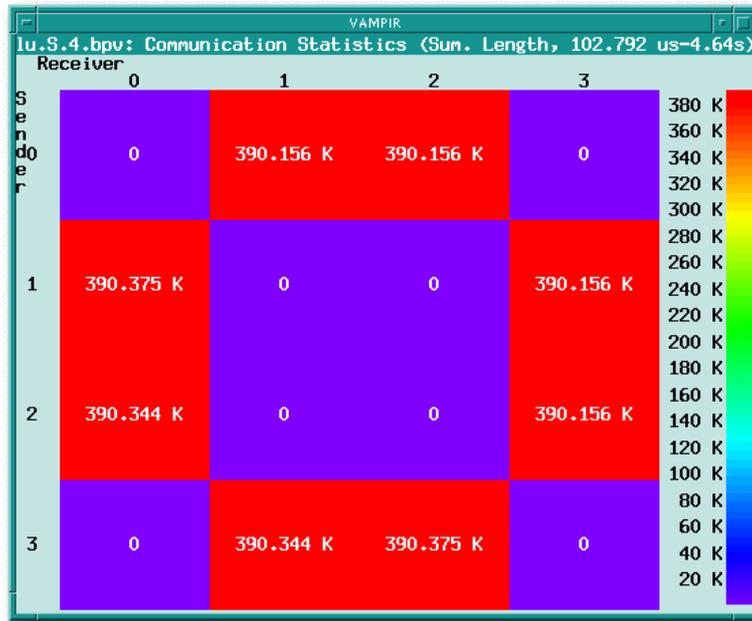


Figure 2.15 Communication Statistics View

While in default mode, the sender / receiver matrix will show the total amount of data communicated between the processes. To get the minimum, average or maximum bandwidth or message length please select the appropriate item of the **Count** submenu cascade from the communication statistics context menu.

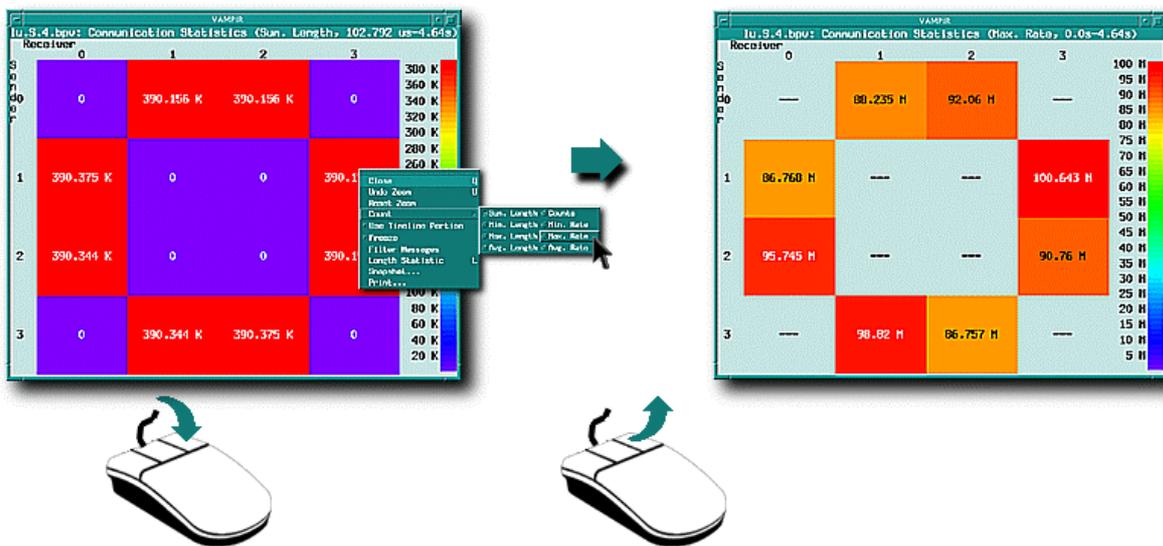


Figure 2.16 Communication Statistics – Max. Rate

There are two ways to reduce the amount of processes shown in the communication matrix and to focus on special processes and their communication behaviour. If you would like to see a continuous block of processes like the upper 2x2 sub matrix from the display, a selection is done by simply mark the area with your mouse (like zooming inside the timeline display). To select a sub matrix move the mouse pointer to the upper left corner of the wanted area, *press and hold the left mouse button*, move the mouse pointer to the lower right edge of the sub matrix (Vampir will indicate the marked region with rubber bands) and *release the left mouse button*. Now, only the selected sub matrix will be shown. This action is called *zooming*. To undo these selection please use the **Undo Zoom** function from the context menu or the corresponding keyboard shortcut **U** (see Figure 2.17 for details).

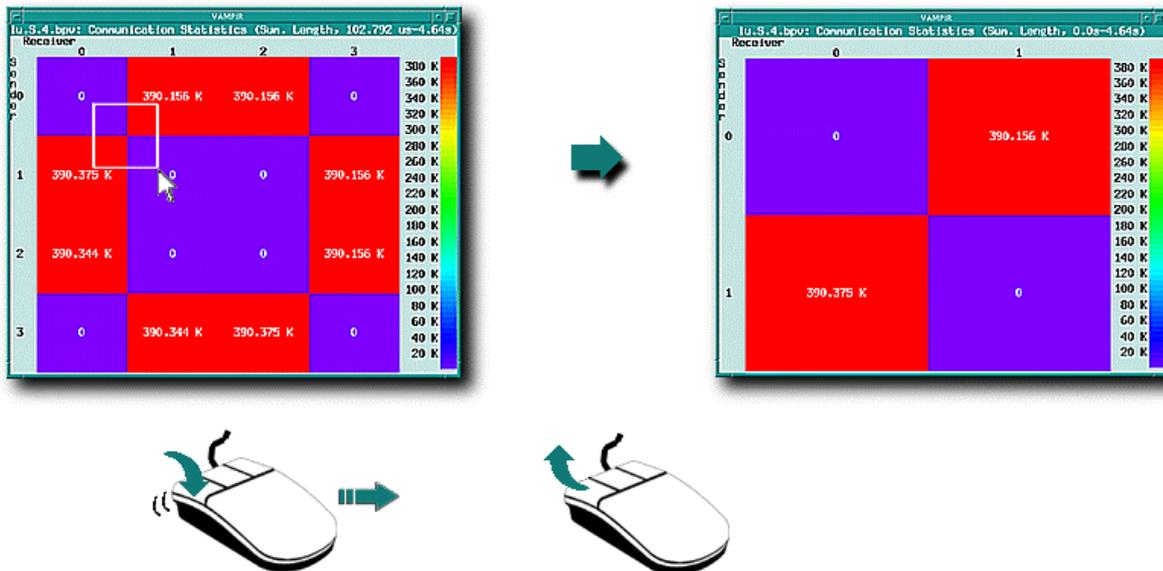


Figure 2.17 “Zoom” into communication statistic display

To select arbitrary processes from the communication matrix please use the **Global Displays / Filter Processes** dialog (see Figure 2.18).



Because this is a global filter, the current selection affects all other global displays !



Figure 2.18 Global Process Filter

To filter messages by their tag or communicator invoke the local message filter dialog by selecting **Filter Messages** from the context menu. By default all communicators and tags are selected and so *not* filtered. To actually filter messages deselect unwanted items and click on the **Apply** button. The communication statistic will be recalculated with the local message filter in mind. Now the communication matrix will show only unfiltered messages. To deactivate the local filter select all communicators and tags and click on the **Apply** button. The message filter dialog is shown in Figure 2.19.

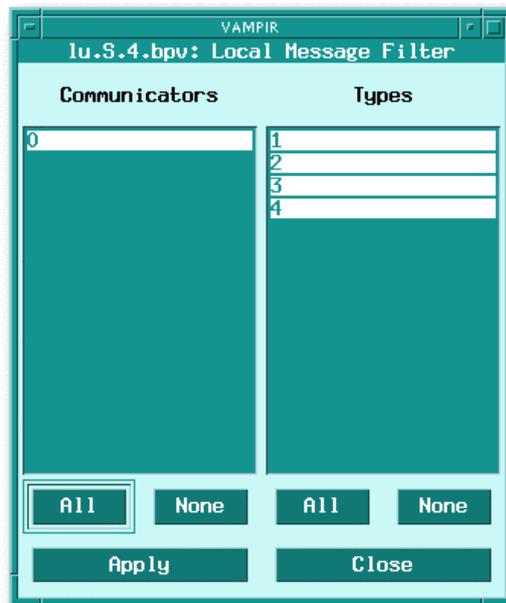


Figure 2.19 Local message filter dialog



The **Communication Statistics** display is the *only* display that can be opened several times simultaneously and all displays can be configured independently. This is important for the communication analysis to see different communication statistics at a time.

#### Communication Statistics Display Summary:

- By default it will show the total amount of data passed between the processes.
- The display can show min/max/avg bandwidth or message length.
- A subset of processes can be filtered locally with your mouse or globally with the **Global Displays / Filter Processes** dialog.
- Different groups of messages can be filtered locally with the **Filter Messages** dialog.
- It can be opened several times simultaneously.
- For a complete description of all feature please read chapter 4.3.5 of the *"Vampir 2.0 User's Guide"*.

### 2.4.5 Communication Length Statistic

The **Length Statistics** display is an addition to the **Communication Statistics** display. To invoke it, open the **Communication Statistics** display click on the **Length Statistic** entry of the context menu. Now wait for the cursor to change into *cross-hairs* (“+”), and then mark the wanted area of the communication matrix by pressing the *left mouse button* and drawing a rectangle on the communication matrix. After the left mouse button is released, a **Length Statistics** window like Figure 2.20 will be opened. It shows a histogram of the distribution of message lengths.

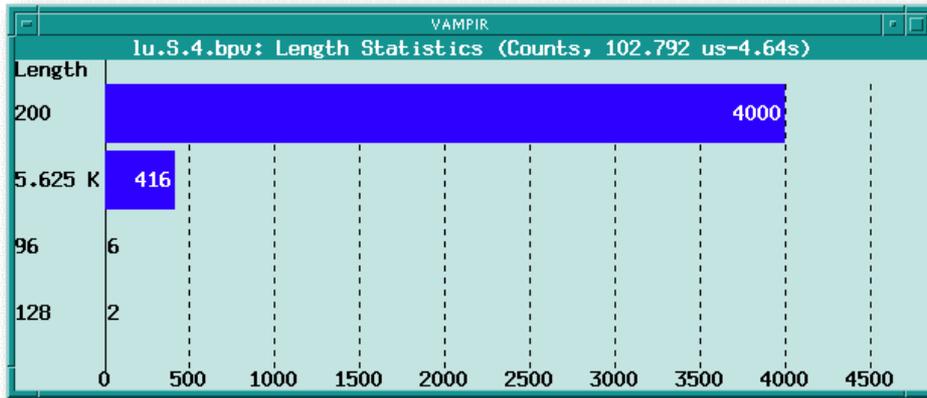


Figure 2.20 Communication length statistic view



Instead of other context displays the **Length Statistic** view behaves like a normal top level statistic display. So it is independent from the parent **Communication Statistic** window and would not be closed automatically when the user closes the former parent view.

In default mode the **Length Statistics** view shows how many messages with a particular length were passed, because messages are grouped by their length in default mode. These information helps you to decide if the traced application is latency (mostly short messages), or bandwidth (mostly large messages) bound. The actual achieved min/max/avg bandwidth can be shown by selecting one of the items from the **Display** sub menu cascade, reachable via the context menu.

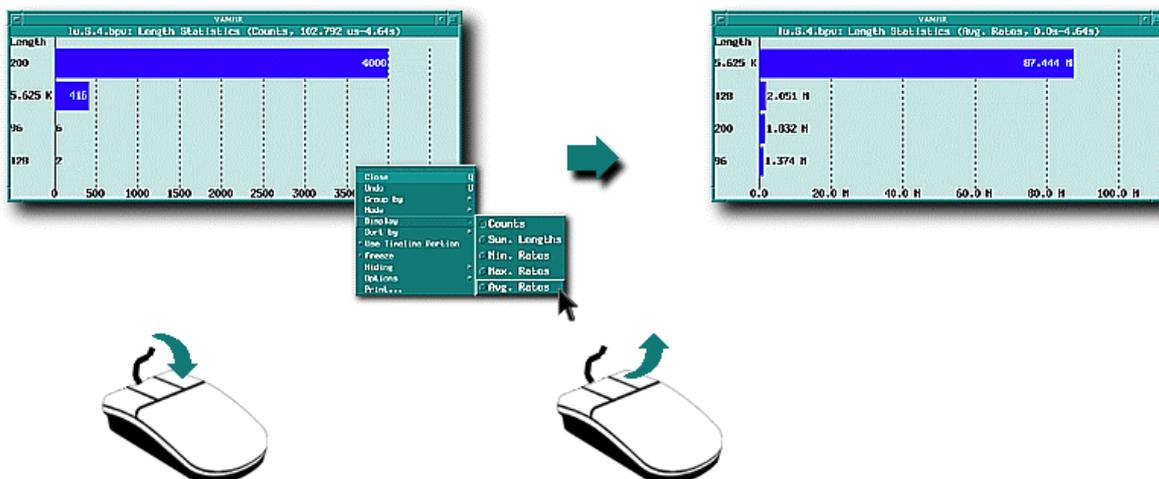


Figure 2.21 Length Statistic View – Avg. Rates

**Communication Length Statistic Summary:**

- Invoked by the **Communication Statistics** context menu. Thereafter independent from the parent display.
- The length statistics can be grouped by message length (default), message type (tag) or by the message communicator.
- The histogram can show message counts, the amount of data passed or min/max/avg rates (bandwidth).
- For a complete description of all feature please read chapter 4.3.5 - *"Length statistic display"* of the *"Vampir 2.0 User's Guide"*.

## 3 Analysing the NAS LU benchmark

The NAS LU benchmark is part of the NAS benchmark suite which can be found at: <http://www.nas.nasa.gov/NAS/NPB>. Documentation is available at the same location. The example is an implementation of a fluid dynamics code employing the SSOR method. Limited concurrency is achieved by using a so called wave front strategy. The limitation is due to inherent sequential calculations.

All tracing was done on the Cray T3E (450Mhz) machine located at the *FZ-Jülich*. The program was compiled with the makefile flags `CLASS=W NPROCS=8`. In order to keep the tracefile small only 10 time steps were used.

The first part of the analysis (section 3.1) will employ most of the important Vampir displays in order of their appearance in the main menu **Global Displays**.

After this basic analysis, appropriate displays will be demonstrated for a more detailed view of the program (section 3.3). In particular, filtering (section 3.3.5) will be stressed.

### 3.1 Statistical Analysis Based on the whole Program Execution

In order to get a first impression of the programs timing, load balancing and message passing performance we will relate analysis to the whole program execution. In section 3.3 details are revealed by magnification.

#### 3.1.1 Loading the Tracefile

- A file browser is activated by the menu: **File / Open Tracefile**  
A valid tracefile is provided under: `$TUTORIAL/lu.w.8.bpV`  
Select this file and load it by clicking the **OK** button.
- In later sessions one can use **File / Recent Tracefiles** to select a tracefile that has been analyzed in a previous session.

#### 3.1.2 Global Displays

The Global Displays menu contains a collection of Vampir displays providing views of global properties of a program. Here, "Global" means referring to either all processes or a selected group of processes at the same time (the next major menu **Process Displays** will focus on *single* selected processes). The key view is **Timeline**, a display of the program's time evolution.



As predefined in the configuration file, all other global displays refer to the time interval shown in the Timeline window. This feature is not default. The context menu of most displays provide a **Use Timeline Portion** switch which turns this feature on or off. Globally this feature is turned on by selecting the **Use Timeline Portion** in the **Preferences / Displays / General** menu.

### 3.1.3 Timeline

The **Global Displays / Timeline** menu has already been activated automatically by the **File / Open Tracefile** menu. Figure 3.1 shows the timeline window as it should appear now. The black vertical lines are so called message lines. Due to a quite coarse resolution, they appear in clusters. These 10 blocks correspond to a code inherent sub structuring into so called time steps. Blue bars at the beginning mean suspended tracing (`TRACE_OFF`).

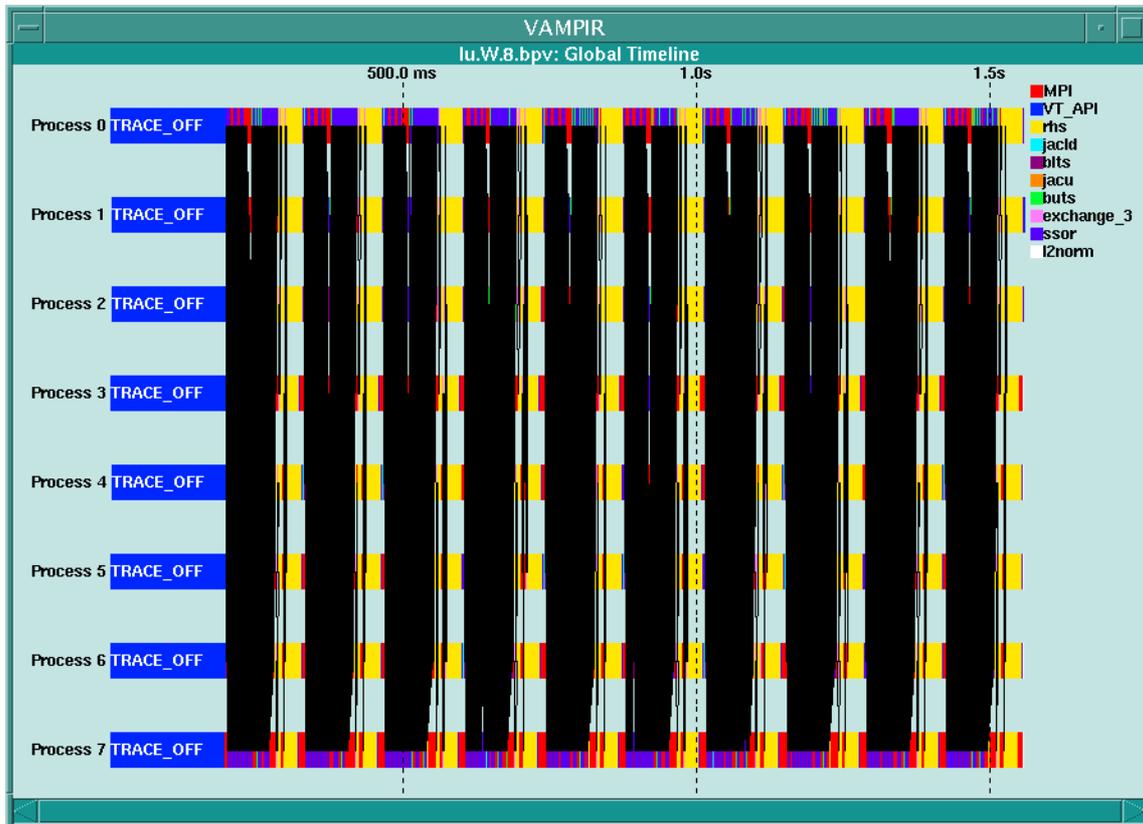


Figure 3.1 Timeline window

The blue bars at the beginning of the timeline can be hidden by zooming into the timeline. Point with the left mouse button on the right end of one of the blue bars, drag the rubber box to the end of the timeline and release the mouse button (if the first try fails, zooming can be undone by typing `U`). The original view is restored by typing `A`).

### 3.1.4 Summaric Chart

- Activation of the **Global Displays / Summaric Chart** menu will pop up a new window with timings that are accumulated over all processes. This so called Summaric Chart can be compared to conventional profiling for sequential programs.
- Note that Timeline and Summaric Chart window will change accordingly when zoom into the timeline is applied. Type `A` to restore the original timeline and observe the accompanying change in the Summaric Chart (see Figure 3.2). Zoom back into the timeline to hide the `TRACE_OFF` symbol.

- Return to the Summaric Chart window. Timings can be shown per process and related to the sum of all shown activities. In order to get these options, employ a context menu (click with the right mouse button inside the Summaric Chart window) and activate **Options / Per Process** and **Options / Include Sum** (see Figure 3.3)
- The MPI activities shown in the Summaric Chart window consists of all traced MPI symbols. In order to investigate only MPI timings, use the context menu **Display / MPI**. Several different MPI functions will be displayed (see Figure 3.4)
- Note that tracing events can be ordered hierarchically into activities and symbols. MPI is an activity that consist of several symbols. User defined activities like `rhs` consist of only one symbol.
- The Summaric Chart reported summed or averaged timings over all processes. Use the context menu **Close** to deactivate it.

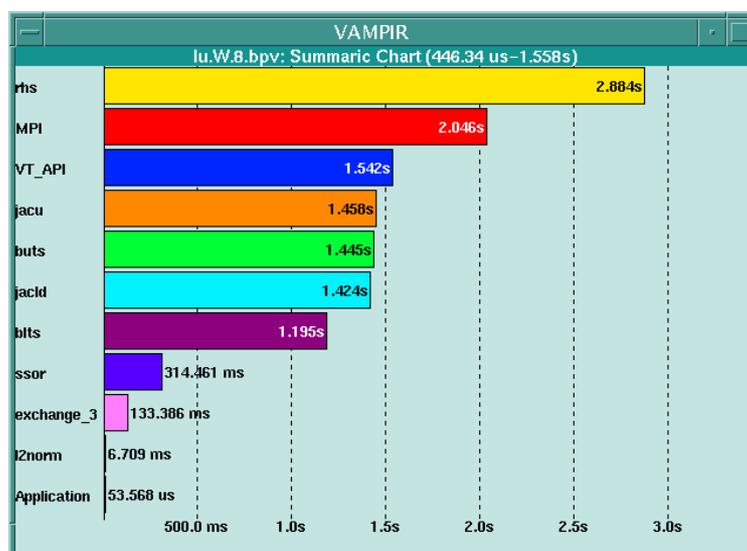


Figure 3.2 Summaric Chart display in default mode

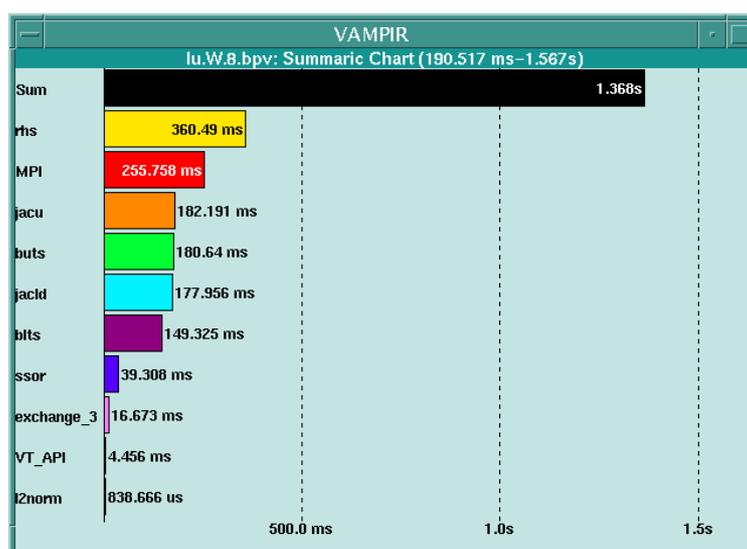


Figure 3.3 Summaric Chart - Sum and Per Process selected

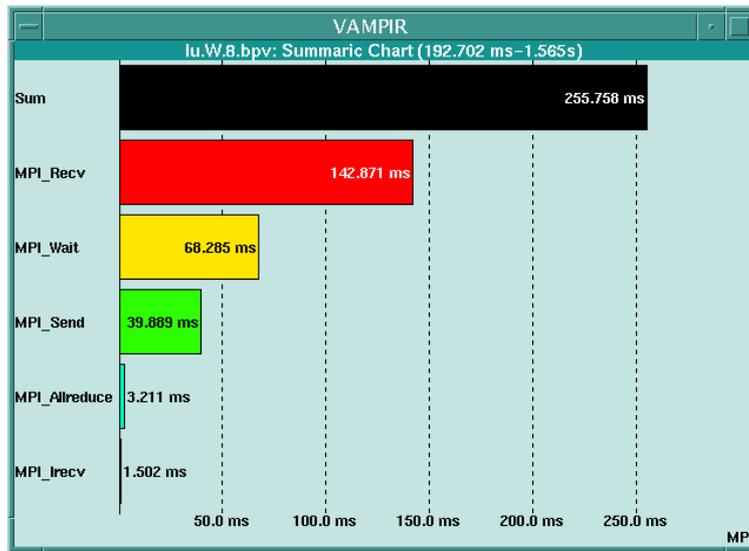


Figure 3.4 Summaric Chart – **Display / MPI** selected

### 3.1.5 Activity Chart

- Selection of **Global Displays / Activity Chart** pops up a window that consists of pie charts for every process. Load imbalances can be identified by this view (see Figure 3.5).
- Select the context menu **Mode / Histogram** for a histogram representation of the current view.
- The context menu **Display / rhs** will filter only `rhs` activity. It has already been mentioned that `rhs` consists of only one symbol so this view is quite simple. One identifies a non perfect load balancing. The timings of `rhs` are ordered in the following way: (0,1,2),(4,5,6),(3),(7).
- Part of the imbalance for the current example are induced by a not perfectly distributed grid. The timing order is due to the fact that the top processes (0,1,2,3) get one more row of the grid and the right processes (3,7) get one column less (see Figure 3.6).
- Filter MPI activities by aid of **Display / MPI**. The MPI timings show a reversed order of timing because short computation time means more waiting time in blocking MPI routines.

So far we have only displayed routine or code block timings. The next views will provide message passing performance information. Close the **Global Displays / Activity Chart** first.

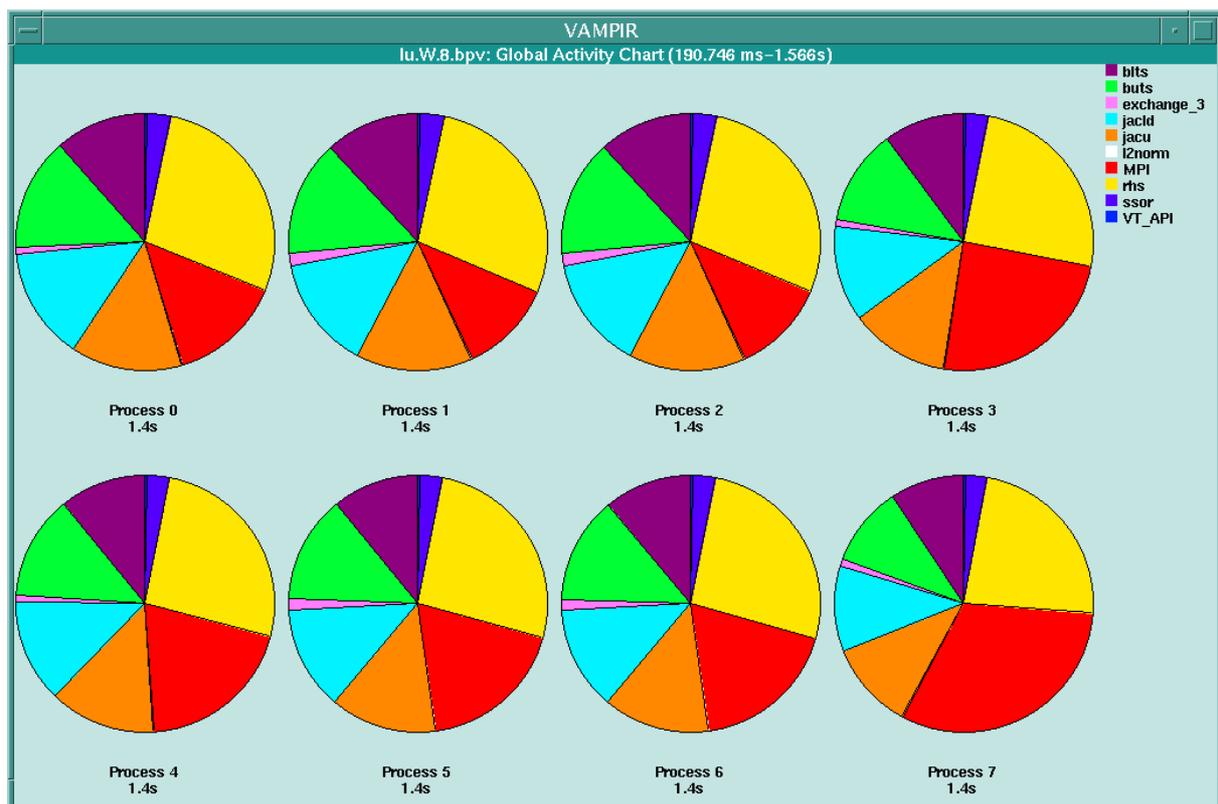


Figure 3.5 Activity Chart – default view



Figure 3.6 Activity Chart – Histogram Mode

### 3.1.6 Communication Statistics

- Activation of the menu **Global Displays / Comm. Statistics** shows a communication matrix displaying sent / received data volumes for all process pairs.
- A typical 2D message passing pattern for next neighbour communication is displayed (see **Global Displays / Activity Chart** figure to identify the 2x4 process grid). One sees a perfectly symmetric pattern for the amount of sent data.
- There are eight different views for the communication matrix that can be found under the context menu **Count**. Default is **Sum. Length** (sum of all message length in bytes). Try out some of the other!
- For optimization issues the average message passing rates in [MB/sec] are more interesting. Select the context menu **Count / Avg. Rate** to get the rates (last item in the Count menu). See Figure 3.8.
- A more detailed analysis of the highly non symmetric rates will follow later. Rates are quite low, compared to the peak rates of the machine (Cray T3E). The most likely reason is that message lengths are quite short.
- To verify this assumption, employ the context menu **Length Statistics**. Click on this menu, point to the (0,0) matrix element, drag the mouse to the (7,7) element and release the mouse. See Figure 3.9.

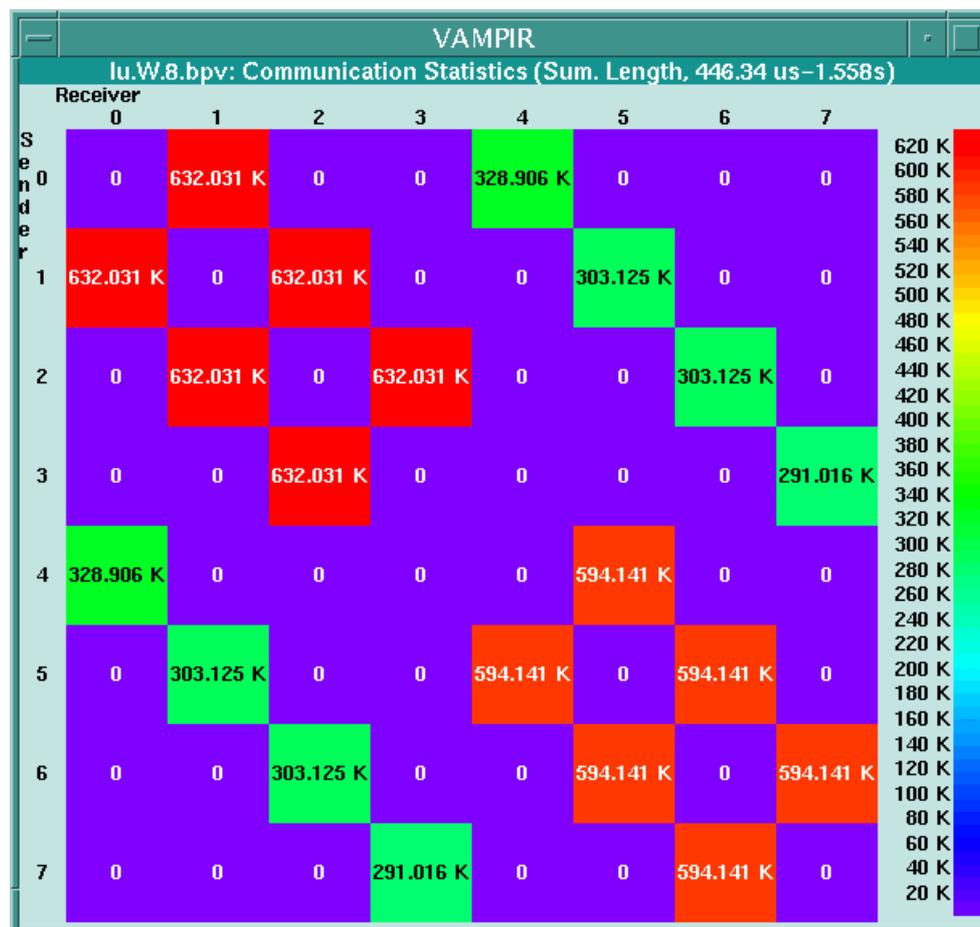


Figure 3.7 Communication Statistics – Default View

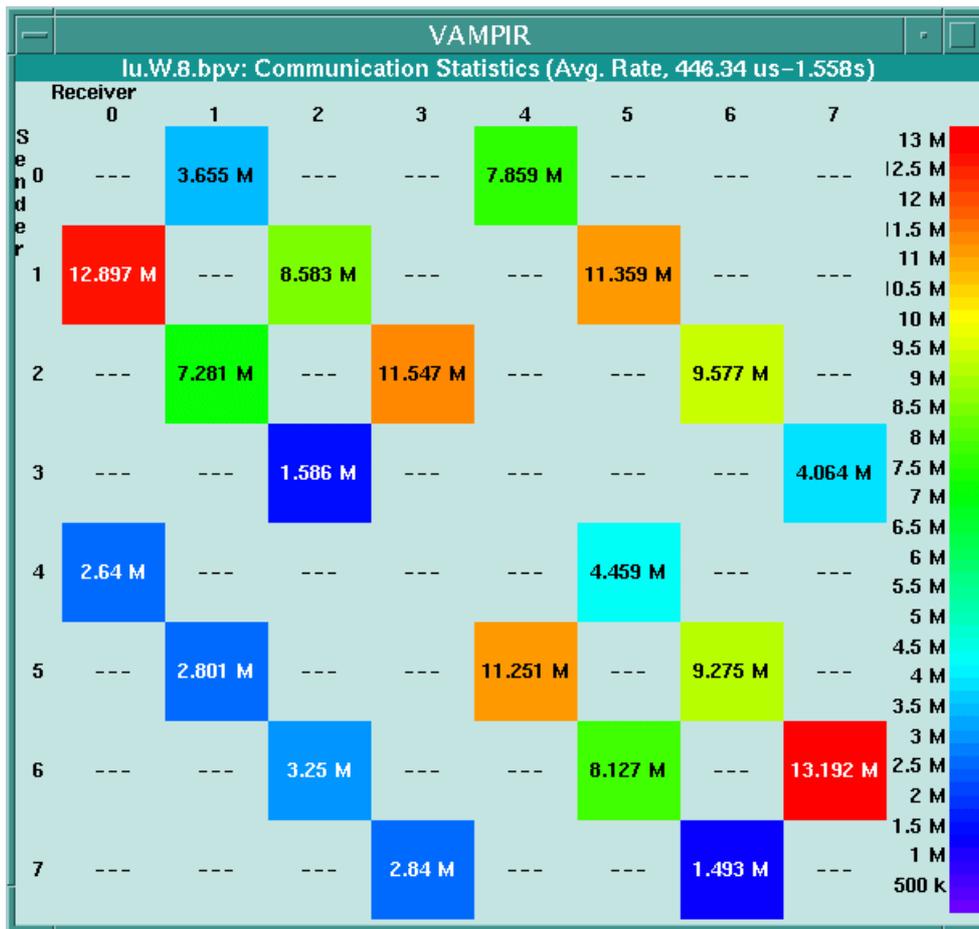


Figure 3.8 Communication Statistics – Message Passing Rates

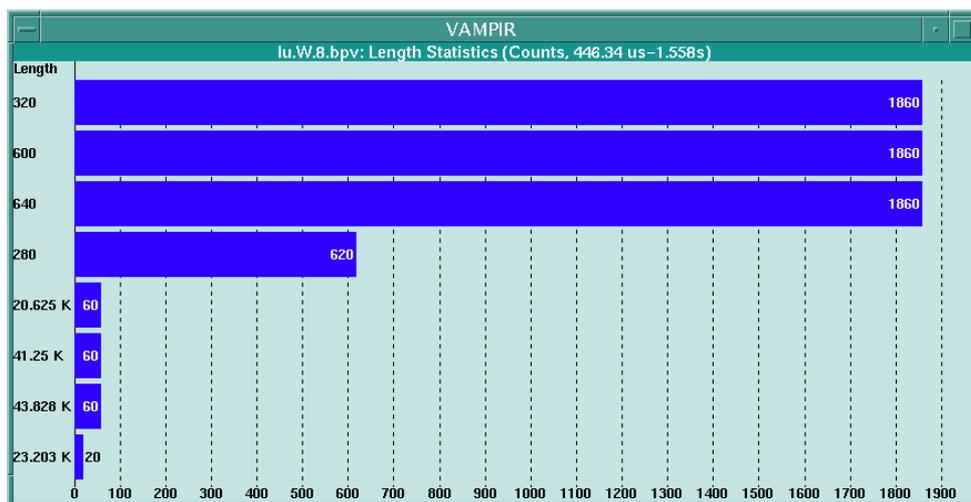


Figure 3.9 Communication Length Statistics

### 3.1.7 Process Displays

The third main menu **Process Displays** is valuable for viewing individual selected processes. It contains the menus **Timeline** and **Activity Chart** that directly correspond to their Global Displays counterparts. If no process has been selected so far the Process Display's submenus are invalidated.

- To activate the **Process Displays**, select at least one process. A process is selected by clicking the left mouse button on it's Timeline process bar. Selected processes are marked by a bounding box surrounding the process bar. Please select only process #0 at first because in the following a window will be opened for all selected processes.

### 3.1.8 Process Timeline

- Click on **Process Displays / Timeline** to open a new timeline window for each selected process (Figure 3.10). The new timeline chart displays a list of activity names on the left. The colored bars to the right display the time intervals for which the selected process is executing the corresponding activity.
- An interesting view can be activated by the context menu: **Options / Guess Calls**. It shows a graphical calling tree representation of the activities of this process (Figure 3.11). The numbers on the left side of the timeline represent a calling tree hierarchy level. The activities shown in the timeline can be identified by aid of the color legend on the left side.
- The 10 time steps are easily recognised.

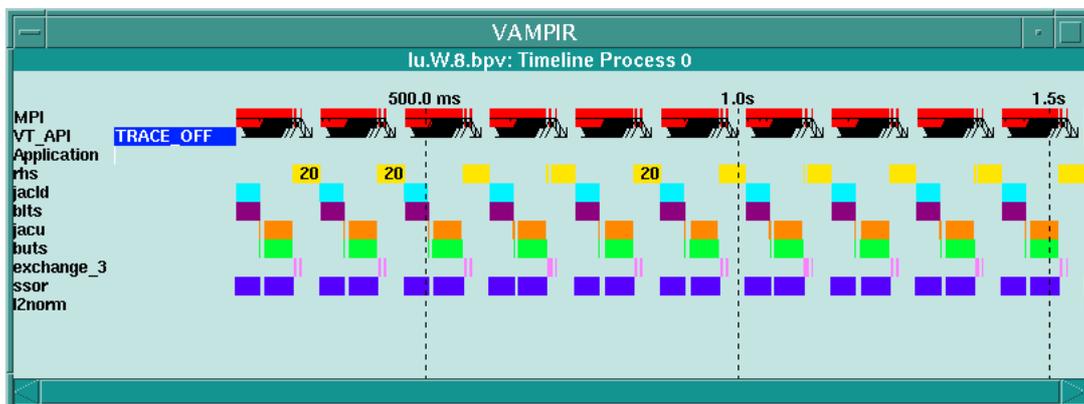


Figure 3.10 Process Displays – Timeline Default View

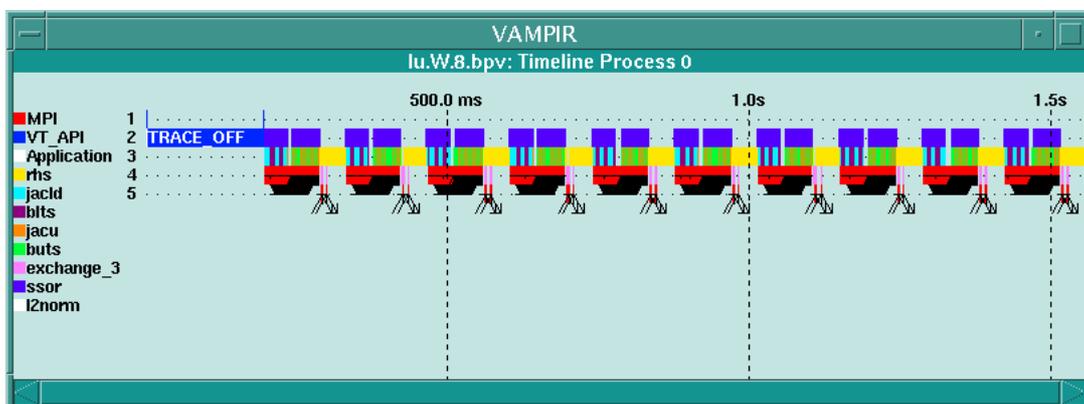


Figure 3.11 Process Display – Timeline, **Guess Calls** Mode

### 3.1.9 Activity Chart

- The **Process Displays / Activity Chart** is completely analogous to the **Global Displays / Summaric Chart** (see section 3.1.4). Compare the two.
- It is interesting to compare timings for individual processes with each other and with the global mean values (new Vampir 2.0 feature). In order to compare timings, an **Activity Chart** has to be saved as an ASCII file and reloaded by another Chart.
- Use the context menu of the Activity Chart for process 0 and select: **Options / Store Values**. A file browser pops up and a directory and filename can be selected. Save the file under the name `Chart_0` (e.g.) in the directory `VAMPIR-tutor/LU-trace` (e.g.).
- Select process 7 and open **Process Displays / Activity Chart**. Note that the context menu **Comparison** is deactivated.
- Open the context menu of the Activity Chart for process 7 and use the menu **Options / Load Values**. A file browser pops up, and the file `Chart_0` can be loaded that has been stored before (see Figure 3.12).
- Assure that the context menu **Comparison** is active because two timings are to be compared.
- Activate **Comparison / Difference** in order to get the difference timing(7) – timing(0) for all activities. Note that the histogram bars are all right directed independent of the sign of the difference.

Exercise: employ **Global Displays / Summaric Chart** to generate a timing average over all processes (see section 3.1.4) and store it to the file `Chart_A` (e.g.). Load it into the last Chart and try some of the comparison modes.

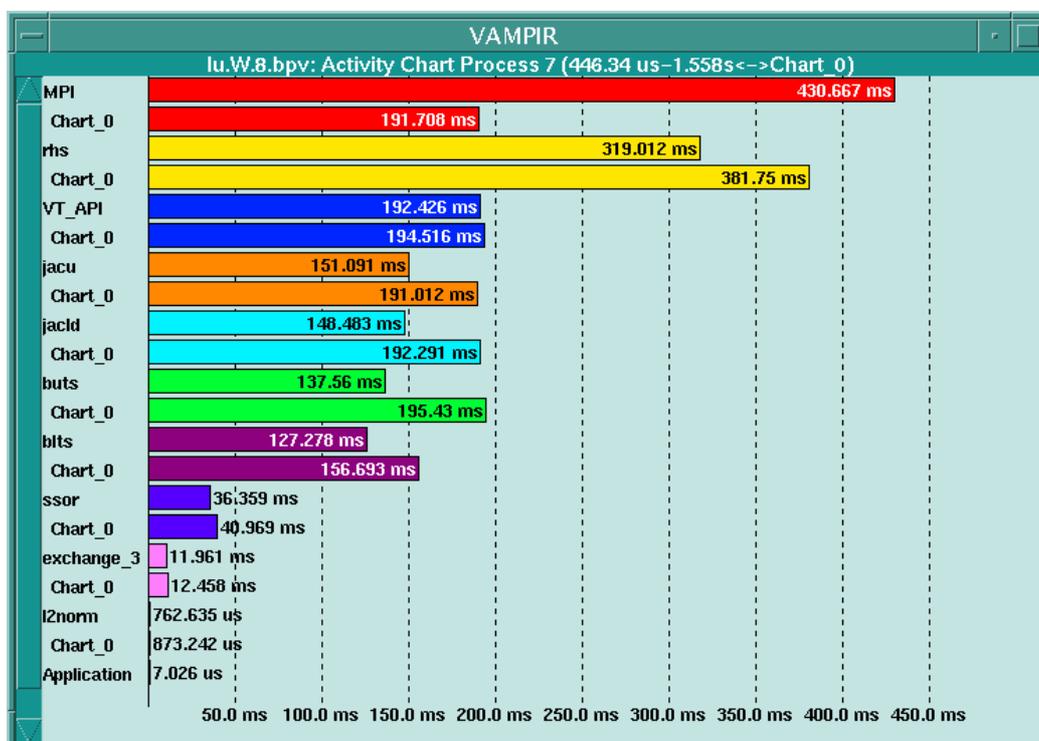


Figure 3.12 Process Activity chart – comparison mode

## 3.2 Summary

A brief survey over all **Global Displays** menus handled so far is as follows:

<b>Timeline</b>	graphical display of time evolution for all processes. All other displays can be related to the selected timeline portion.
<b>Summaric Chart</b>	timings summed over processes (suited for first profiling).
<b>Activity Chart</b>	comparison of individual process timings (valuable for evaluation of load balancing).
<b>Communication Statistics</b>	statistics over global message passing lengths and rates or individual messages.

A brief survey of all **Process Displays** menus is as follows:

<b>Process Timeline</b>	shows inclusive timings for individual processes or a graphical calling tree for all instrumented activities.
<b>Process Activity Chart</b>	profiling for individual processes. Comparison for individual process timings is gained by employing the context menus <b>Options / Store</b> and <b>Options / Load</b> for charts together with the context menu <b>Comparison</b> .

### 3.3 Detailed Analysis for Timeline Sections

At this point all major display menus of Vampir have been shown. Now, in combination with zooming they will be used to get a more detailed understanding of the underlying program.

At first, in the LU example, beginning and end of a single time step will be identified. Then, the structure of a time step will be analyzed in more detail. In particular, the technique of filtering will be explained.

#### 3.3.1 Identifying a Single Time Step of the LU Sample

- Select process 0 (click on the process bar). Reactivate **Process Displays / Timeline** together with the context **Options / Guess Calls**.
- We have already identified 10 time steps. The highest hierarchy level (2) during the time stepping is occupied by the driver routine `ssor`. After the `rhs` section the time evolution pattern repeats.
- Zoom into the fourth time step. It begins after the third call to `rhs` and ends after the fourth occurrence of `rhs` (see Figure 3.13 and Figure 3.14).

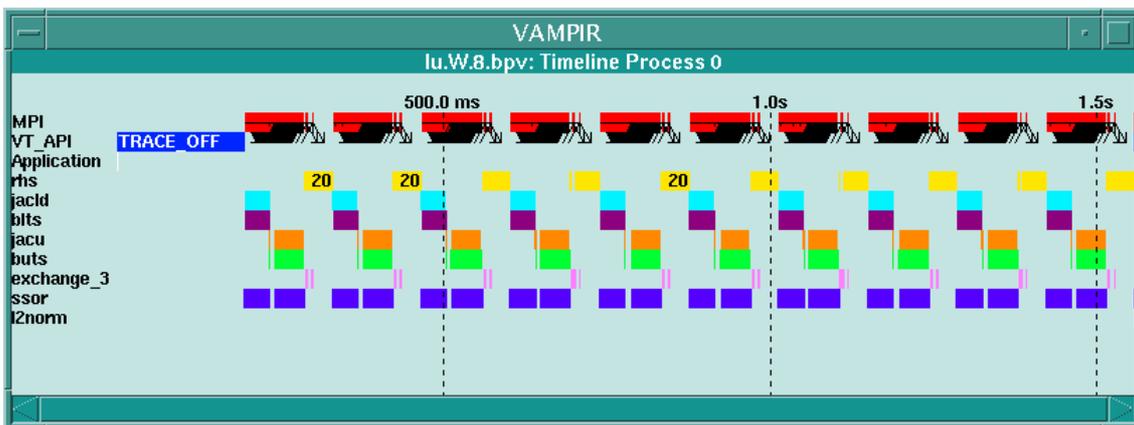


Figure 3.13 Process Timeline Display – Default Mode

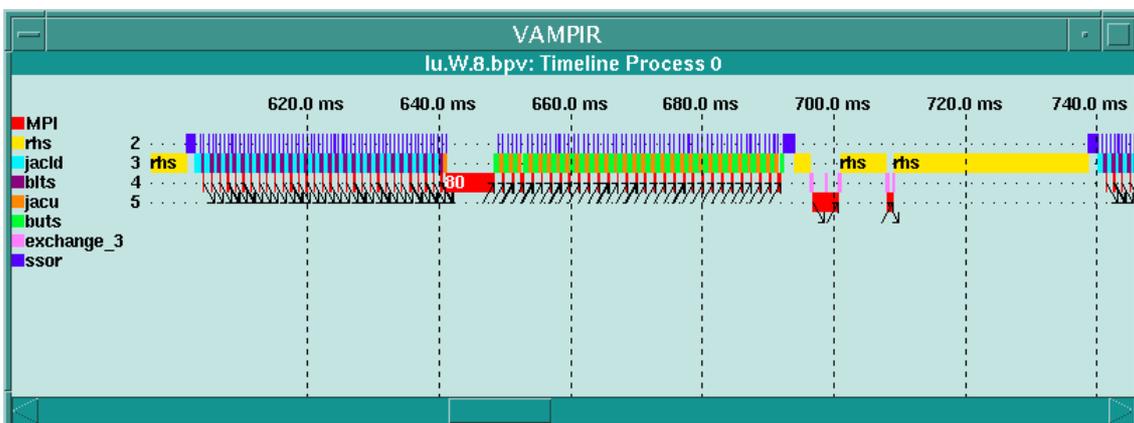


Figure 3.14 Process Timeline Display – Zoom In

### 3.3.2 Structure of a Time Step

- Resume the original view of step 4 (Figure 3.14).
- To get more information, zoom further into the timeline. Depending on the zoom resolution, the sections are either tagged by their name, by a symbolic number or just by a unique color. By increasing the zoom depth, it is always possible to reveal the symbol name.
- In order to undo zooming type `U` on the keyboard.
- To undo all zooming type `A`.
- Three major code blocks can be identified: alternating calls to `jacld` and `blts`, alternating calls to `jacu` and `butts` and `rhs` at the end.
- Use also **Global Displays / Timeline** in order to recognize these substructures.
- Exercise: zoom into time step 4 of **Global Displays / Timeline**, compare the time stamps with the stamps of **Process Displays / Timeline**. Note: the single processes' starting points of a time step are not synchronized.

### 3.3.3 Lower Tridiagonal Solvers (jacld, blts)

- Zoom into the first part of time step 4 with alternating calls to `jacld` and `blts`. Try to get about 4 to 5 occurrences of each routine per process (Figure 3.15).
- One sees a frequent occurrence MPI symbol 80. To find out which function symbol 80 stands for use the **Global Displays / Timeline** context menu **Identify State**. Select this menu and click on the symbol you want to know more about.
- Get more information about a single message: Select **Identify Message** in the context menu and click on a message line.
- One of the two identification menus can be made permanent by using the context menu **Pointer Function**. Dynamically updated with mouse movements, status / messages are analyzed wherever the mouse points to.

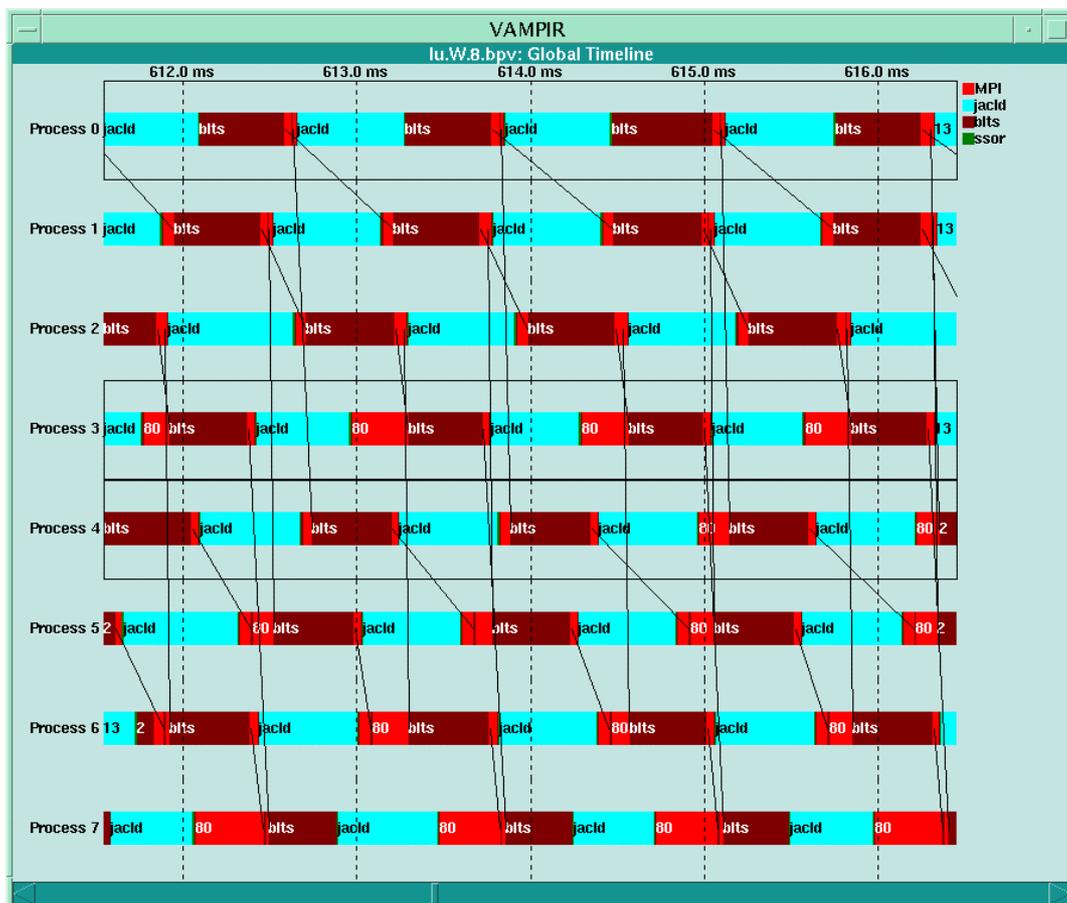


Figure 3.15 Four stages of the forward wave (lower tridiagonal solver)

### 3.3.4 Exercises

- Analyze the message passing patterns of the 3 major blocks a time step consists of (`jacld/blts`, `jacu/buts`, `rhs`). Use **Global Displays / Comm. Statistics**.
- Why are message passing rates highly non symmetric, especially for process 0 and process 1?

- There is a sequential order in the starting point of the message passing patterns:  
 For `blts`, the order is: 0 (1,4) (2,5) (3,6) 7.  
 For `buts`: 7 (3,6) (2,5) (1,4) 0.  
 Try to recognize these orders in the Timeline chart (hint: scroll to the beginning of the alternating patterns).

### 3.3.5 Filtering

One major concept of Vampir is filtering of processes and messages. The resulting views become clearer as activities of minor relevance are ignored.

- To filter a selection of processes, use the **Global Displays / Filter Processes** menu or select a set of processes and use the timeline context menu: **Options / Filter by Selection**.
- To filter a selection of messages (different tags or communicators), use the timeline context menu: **Options / Filter Messages**.

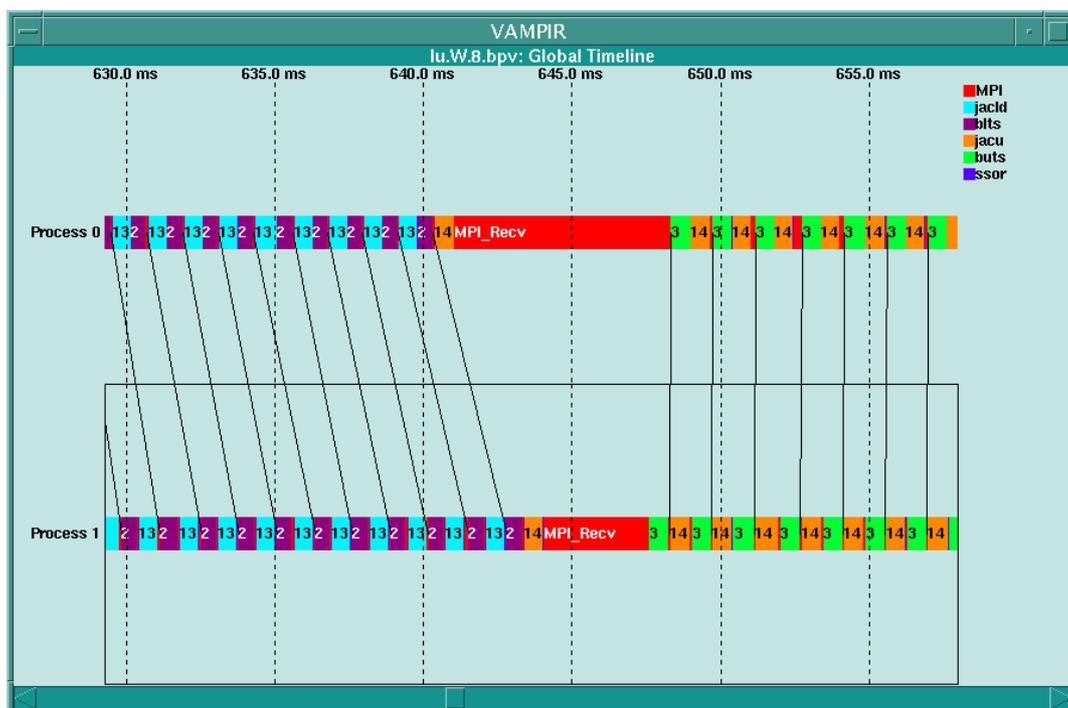


Figure 3.16 Filtering of process 0 and 1

## 3.4 Summary First Example

- Most of the major Vampir displays were presented by analyzing the NAS LU Benchmark. The level of detail was increased step by step, beginning with a global view of the entire program. This should be an appropriate approach in general.
- Zooming has been demonstrated in order to relate all views to a single time step or even more detailed structures.
- Finally filtering has demonstrated how to get clearer views by concentrating on a selection of processes.



## 4 Generation of Tracefiles (Matrix Multiply)

There are basically two levels of tracefile generation:

- Default (achieved by just a modified link step, see section 4.2). All MPI\_XXX activities are traced, the remainder code is all over marked as “user code”.
- User defined. The user can select arbitrary code sections (and their symbol names) to be traced. See section 4.3.

Some implementations allow for **automatic** instrumentation. This feature should be used with care because tracefile size and runtime overhead may become considerable.

With a second example, parallel matrix multiply, both levels of tracefile generation will be demonstrated now. Some brief explanation will be given about the usage of the corresponding program.

An instrumented version of matmul source code is in `VAMPIR-tutor/matmul-instr/`. Some sample tracefiles reside in `VAMPIR-tutor/matmul-traces/`. The digit in the tracefile name corresponds to the exchange mode defined below.

### 4.1 Matrix Multiply (matmul)

The source of matmul is in the directory: `VAMPIR-tutor/matmul`.

- Compile the source code by using the `makefile`, which first has to be adapted to the current machine.
- An input file controls the program execution. It requires the following five integer entries:

```
N M K           ! (NxK) matrix multiplied by (KxM) matrix
output exchange ! file output mode, data exchange mode
```

- Output flag:

```
0:    no output
1:    matrices are printed in files (for diagnostics)
```

- Exchange mode:

The basic communication step is an exchange of parts of one matrix operand. The program offers three choices of how to perform these exchanges:

```
1:    MPI_Isend / MPI_Recv / MPI_Wait
2:    MPI_Sendrecv
3:    MPI_Isend / MPI_Irecv / MPI_Waitall
```

Run matmul by typing: `mpirun -np 4 matmul` (the specific mechanism for running MPI programs may be different on your machine. The line given above, however, is taken from the MPICH implementation and is utilized by many vendors). The standard output will provide some timing information.

## 4.2 Linking with Vampirtrace library

- In order to link a MPI program to Vampirtrace one has to change the link line, typically like

```
f90 -o prg.x prg.o -L(LIB_PATH) -lmpi
```

into:

```
f90 -o prg.x prg.o -L$(LIB_PATH) -L$(VT_PATH) -lVT -lpmpi -lmpi
```

with `VT_PATH` = location of the library `libVT.a`

- Note that the order of libraries is important because the Vampirtrace library overwrites the MPI functions of `libmpi`. The MPI standard demands the availability of the library `libpmpi.a`, which provides different names (used by `libVT`) for the standard MPI interfaces. Consult the Vampirtrace User's Guide for more information (e.g. "C support").
- Exercise: change the link line in `makefile` in order to link Vampirtrace in the way explained above (inquire the path of `libVT.a`). Running the program will now generate a tracefile: `matmul.bpv`.
- Transfer this tracefile to your workstation where the Vampir GUI (Graphical User's Interface) is installed.
- Load the tracefile by aid of the **File / Open Tracefile** menu. Use the `Lu.cnf` configuration file.

### 4.2.1 Exercises

- Find all MPI calls in the source file `VAMPIR-tutor/matmul/matmul.f` and relate them to your tracefile. Why does the timing of process 0 differ substantially from that of other processes?
- Zoom into the matrix multiplication section (hint: the matrix multiply is done by subroutine `pMxM`. This routine is enclosed by barriers).
- The matrix multiply is done in  $p-1$  sweeps ( $p$ : number of processes). Find the beginning and the end of one sweep in the tracefile.
- Try other exchange modes and other matrix sizes (new tracefiles). Is there a timing difference for different exchange modes on your machine? It is interesting to compare the nonblocking (exchange mode 1 or 3) with blocking communication (mode 2). Is your machine really able to overlap message passing with CPU activity? Try to answer this question by using: **Global Displays / Communication Statistics** and the runtime given on standard output.

## 4.3 User Defined Activities

In order to make tracefiles more intelligible it is useful to have user defined activities in addition to mere MPI activity. It makes sense to start by instrumenting top level subroutines that consume most of the time. The way of achieving this will be explained now.



### 4.3.1 Vampirtrace API

- Definition of a symbol:

```
SUBROUTINE vtsymdef( code, symbol, class, err )
INTEGER           code           ! Symbol code (positive integer)
CHARACTER*(*)    symbol         ! Symbol name (e.g. init_exchange)
CHARACTER*(*)    class          ! Class of symbol (e.g. exchange)
INTEGER          err            ! Error code
```
- Start of tracing a symbol:

```
SUBROUTINE vtbegin( code, err)
INTEGER          code           ! Symbol code (positive integer)
INTEGER          err            ! Error code
```
- End of tracing a symbol:

```
SUBROUTINE vtbegin( code, err)
INTEGER          code           ! Symbol code (positive integer)
INTEGER          err            ! Error code
```
- Suspend tracing:

```
SUBROUTINE vttraceoff()
After a call to this routine all tracing will be ignored.
```
- Reactivate tracing:

```
SUBROUTINE vttraceoff()
This call is used to activate tracing again.
```

The latter two routines are useful for limiting the number of tracing events and therefore the tracefile size (see also section 5).

### 4.3.2 Instrumentation of pMxM (1)

“Global” code instrumentation:

Symbols and their internal code are declared right after the initialization of MPI (MPI\_Init). Begin and end of every traced code block have to be marked with the corresponding symbol code:

```
PROGRAM matmul                               ! Main contains MPI_Init
...
! End of declarations
INTEGER vt_err
...
CALL MPI_INIT(ierr)
CALL vtsymdef( 1, "pMxM", "pMxM", vt_err )      !
Definition of symbols
CALL vtsymdef( ....)

....
END

SUBROUTINE pMxM( ... )
```

```
INTEGER vt_err
CALL vtbegin( 1, vt_err )
      ! Code to be timed
CALL vtend( 1, vt_err )
END
```

This technique has the advantage that all symbol definitions reside in one central place. There is a better overlook of which symbol codes are in use. The disadvantage is that definition and code blocks are in different files.

### 4.3.3 Instrumentation of pMxM (2)

“Local” code instrumentation:

```
SUBROUTINE pMxM( .... )
! End of declarations

INTEGER vt_err, vt_code
PARAMETER ( vt_code = 1 )      ! Use different codes for other symbols
LOGICAL vt_not_def
DATA      vt_not_def  /.TRUE./
SAVE      vt_not_def

IF (vt_not_def) THEN          ! The symbol is only defined once
  CALL vtsymdef( vt_code, "pMxM", "pMxM", vt_err )
  vt_not_def = .FALSE.
ENDIF

CALL vtbegin( vt_code, vt_err )
      ! Code to be timed
CALL vtend( vt_code, vt_err )
```

This technique keeps the symbol declaration and the code block in the same file. The flag `vt_not_def` is used to make sure that the declaration is called only once. The disadvantage is that one has to look after the right symbol code values (When a symbol code is used twice the first definition will be overwritten).

### 4.3.4 Instrumentation Hints

- To define or change the color of classes use the menu: **Preferences / Activities / Colors**.
- An instrumented program can be used without Vampirtrace by linking the dummy library `libVTnull.a` found in the same location as `libVT.a`
- The C preprocessor may be exploited to switch between instrumented and non instrumented versions of the code.
- Vampir overhead:  
Compare the timings of original, MPI instrumented, User + MPI instrumented and “`vtnull`” versions of `matmul`. Check the (expectedly low) overhead produced by the different scenarios.



## 4.4 Summary

- The easiest method of generating a Vampir tracefile is to link the program with the Vampirtrace library. All MPI activities are traced automatically.
- User defined tracing activities are gained by inserting certain Vampirtrace API calls in the source code. These calls can be invalidated by aid of a dummy library.

## 5 Tracing “Real World” Programs

Using Vampir for realistic large scale applications may introduce problems that have not been discussed so far. One limiting factor is the availability of disk space on the parallel production machine and the workstation where the Vampir GUI is supposed to run. Here are some hints of how to handle such difficulties, if occur.

- Careful instrumentation  
For a systematic instrumentation a conventional profiler may first be used to identify the routines that consume most of the time. It is a good starting point to instrument the top level routines first. After having achieved a good survey over the program’s time evolution interesting details may be instrumented additionally.
- Suspend tracing  
The most direct way to limit the tracefile size is to suspend tracing for non interesting sections of the program. By a call of the Vampirtrace API routine `vttraceoff()`, all tracing is suspended until resumption by a call of `vttraceon()`. It is often a good idea to suspend tracing of the initialization phase as it was done in the LU example.
- Configuration file  
There is another way of tracefile size reduction with no source code changes and recompilation. A configuration file can be used to filter activities or symbols. The user can control the tracefile generation by certain commands in the configuration file. The file is used when an appropriate environment flag is set. An example is given now.

### 5.1 Using a Configuration File

A configuration file (e.g. `vt_config`) is used by Vampirtrace when the environment variable `VT_CONFIG` is set:

```
export VT_CONFIG=vt_config    ! ksh
setenv VT_CONFIG vt_config    ! csh
```

The file `vt_config` may be named by an absolute or a relative path. A configuration file example is as follows:

```
# this is a comment
# redirection of the tracefile to the /tmp directory
# /tmp may provide more disk space
# Tracefile name= /tmp/matmul_16.bpv
Logfile-name /tmp/matmul_16.bpv
# suspend all MPI tracing
Activity MPI off
# Single MPI symbols can be reactivated
Symbol MPI_Isend on
```

The configuration file shown above will change the default tracefile name e.g. `matmul.bpv` into `/tmp/matmul_16.bpv`. It will suppress all MPI tracing other than `MPI_Isend`. Consult the *Vampirtrace User’s Guide* for more information about the configuration file.

## 6 Vampir – Tips & Tricks

### 6.1 Filter Arbitrary Symbols

To get rid of unwanted symbols from all Vampir displays regardless of their associated activity, invoke the symbol grouping dialog by selecting **Preferences / Activities / Symbol Grouping** from the main menu. Select the unwanted symbol and chose “(none)” from the drop down box at the bottom of the window. Now the remapped symbol will be suppressed in all Vampir displays. This means also that all statistics are recalculated without the remapped symbols.

This is only a temporary mapping for the current Vampir session. To make it permanent it is necessary to rewrite or save the actual tracefile. First verify that Vampir is in “**Expert Mode**”. Select **File / Save Tracefile** from the main menu, activate the checkbox “**Mod. Symbols**”, chose a filename and click on the **OK** button to save the trace data.

### 6.2 Highlight Messages in the Timeline

To highlight message lines inside the timeline display open the message style dialog via **Preferences / Messages / Display Style** and define communicator, tag ranges and how to draw the affected message lines. To emphasize long messages set the value in the **Big Msg Limit** input field to a certain value (in bytes) so that all bigger messages are drawn with bold lines. These settings are saved in the Vampir configuration file for future sessions.

### 6.3 Reduce Default Timeline Display Range

Analysing really huge tracefiles could be annoying because the visualisation of all trace events inside timeline view takes a while. If you already know that only a particular interval is valuable, you can configure Vampir to show only these interval by default. To set the interval open the timeline preferences dialog **Preferences / Displays / Timelines** and select “**Range**” from the radio button group “**Default Timeline Range**” and enter the start and end of the interval into the “**from**” and “**to**” input boxes in the chosen units (clocks or seconds) and click on the “**Apply**” button to activate these settings. Now if you re-open the timeline view, only the selected range is displayed.

Inside the timeline window the keyboard shortcut **A** or context menu **Window Options / Adapt** switches back to the complete traced range, and the shortcut **D** or context menu **Window Options / Default** goes back to the chosen timeline range.



Please keep in mind that Vampir will save these settings in the global configuration file. So in future sessions the selected display range will lead to irritating displays especially in conjunction with different tracefiles.

### 6.4 Use the Parallelism Display inside Global Timeline

To get a first impression of the degree of parallelism in your traced program activate the integrated parallelism display inside the timeline window. To activate the integrated parallelism display select **Components / Parallelism Display** from the timeline context menu. By default the activity MPI will be shown. To change the visualised activity open the timeline window configuration dialog via **Preferences / Displays / Timeline** and enter a desired activity into the textfield named **Parallelism Activity**. After pressing the **Apply** button the new activity will be used inside timeline view.

---

## 6.5 Select Multiple Processes inside Timeline View

To select multiple process bars which are disposed in a row move the mouse pointer to the right or left side of the timeline window, either above the process labels or activity legend, so that the mouse pointer is beside the first process you wanted to select. Then press the left mouse button. While keeping the button pressed, drag the mouse to the last process you wanted to select. Vampir will draw a black line and when you release the mouse button all processes aside your line will be selected.

To deselect a single process click on the appropriate process bar with the left mouse button. To deselect all processes click with your middle mouse button inside timeline window.