pallas

Vampir 4.0

User’s Guide 4.0.0.0

PALLAS GmbH
Hermülheimer Straße 10
D–50321 Brühl, Germany

Contents

Chapter 1

Introduction

1.1 What is Vampir?

Vampir is the market leader in performance analysis for parallel applications. Vampir provides a convenient way to graphically analyse runtime event traces produced by MPI applications. It excels in enabling the user to focus quickly at the appropriate level of detail to analyse application performance issues.

The role of parallelism in computing is motivated by the need to achieve more computing power than a single processor can provide. With a parallel programming approach users are able to reduce the time taken to solve a problem while also tackling larger problems by distribution of data, and tasks, over multiple processors. The different parallel concepts can be divided in a shared memory and a message passing approach. The message passing model allows for parallel scalability on distributed memory as well as on shared memory architectures, and presents the opportunity for unrivaled performance. However, in providing this flexibility and portability the parallel programming approach requires that the programmer must take care of explicit communication and load balancing, and this often results in complex code. The message passing model is represented by different implementations of the very successful, and widely used, Message Passing Interface (MPI). These implementations follow either the MPI-1 or MPI-2 standards that are available on almost all platforms. The MPI standard has enabled many applications to run on a large variety of parallel architectures such as shared-memory, distributed memory and, cluster architectures without (or with only little) modification of the source. The shared memory model is based on a global address space for all processors and directives are added to the serial source code to enable forking (joining) from (to) the root process at runtime. Compiler directives can make programming easier because they hide the parallel details from the programmer, but performance gains are harder to achieve with them.

Many parallel applications do not benefit from the increasing peak performance of either symmetric multi-processor systems, parallel processor systems, or clusters of workstations. This is because most applications are not designed (or programmed) to run efficiently on a large number of processors, or architectures with different interconnect fabrics. Typically they use less efficient communication patterns such as frequent global operations or blocking communication. This leads directly to a lack of scalability, or load balancing problems, or both, and poor performance of the parallel application. Efficient communication patterns are often difficult to code even for an experienced MPI programmer due to the complexity of interaction between algorithm and communication methods. Therefore, the need for performance optimisation and sophisticated tools remain key requirements for the next decade and will grow in importance. Vampir meets this requirement and its genesis is in the long-standing experience in high performance computing that

Pallas offers customers. This experience is centred on well-balanced tools and expertise in the field of parallel performance analysis.

Parallel programs can produce a large amount of trace data during a trace session. Finding performance bottlenecks requires appropriate tools that can handle this information efficiently. Text oriented tools (such as prof) that are used for serial performance analysis are disadvantageous when dealing with large amounts of sampled data. Also graphical user interfaces that represent the collective profiling data are often unable to identify problems. An event-based, trace-based performance analysis tool like Vampir can directly handle program traces and is capable of showing accumulated measurements in different types of statistical displays for arbitrary time intervals. Performance problems that are related to imbalanced code or poor synchronisation can be easily detected because they cause irregular communication patterns.

Future versions of Vampir/Vampirtrace will support other parallel programming paradigms such as OpenMP, multithreaded, and hybrid approaches (OpenMP/MPI). Vampir translates trace file information into a variety of graphical displays that help the programmer understand trace results. Examples of Vampir graphical displays include:

- timeline displays showing state changes and communication,
- profiling statistics displaying the execution times of routines,
- communication statistics indicating data volumes and transmission rates.

One of the key features of Vampir is the advanced navigation functions that easily allow zooming onto arbitrary time intervals. Thus, the programmer proceeds from a bird's eye view of the whole trace to finer granularity of detail. The profiling and communication statistics help in identifying performance bottlenecks, while the detailed timeline display gives clues as to their cause. The combined effect of these features is to lead the programmer rapidly and efficiently to the source of performance bottlenecks in the application.

Specific parallel performance problems that Vampir can help with include:

- excessive communication cost,
- load balance between CPU and cache, or memory,
- communication bottle-neck detection on the interconnect fabric.

All three of these issues determine how well a parallel application's performance scales with an increasing number of parallel processors.

This User's Guide is intended to reveal details of Vampir's features and can be used both as a learning guide and as a reference. While not all features of Vampir are covered at the deepest level of detail the coverage is sufficient to allow rapid start-up for the programmer while encouraging further exploration. To this end this User Guide is organized into a demonstration of how Vampir may be used to understand code performance in Chapter 2, followed by a quick-start section in Chapter 3. The remaining sections in this chapter cover some preliminary requirements before Vampir can be started on the user's platform.

1.2 What is new in Vampir 4.0

Vampir 4.0 adds several new features that extend the traditional event-based performance analysis. A new file format, the Structured Trace Format (STF) is used to store the event data in a more compact format as well as meta data about the events. These extensions allow:

- preview of the event data without loading it (FRAME DISPLAY, Section 3.3.5)
- fast, random access to parts of the trace (**Loading Data**, Section 3.3.5.1)
- statistical analysis of a program run without storing or loading trace data (STATISTICS VIEWER, Section 3.3.7)
- a redesigned PROCESS FILTER that makes better use of grouping (Section 3.5.2)

1.3 What is new in Vampir 3.0

To allow an efficient in-depth investigation of MPI programs and to provide support for modern SMP clusters, Pallas introduces a new generation of Vampir and Vampirtrace. Vampir 3.0 graphics performance has been drastically improved, and it incorporates new displays for the analysis of hardware or OS performance information: the values are shown in the global or local timelines, and a by-routine profile is also available. On SMP clusters, the message statistics can be computed either between processes or between nodes. The new Vampir 3.0 release includes new and improved features while keeping all the unique features of Vampir 1.x and 2.x.

The current release has many new added features, these include:

- improved performance
- better scalability
- explicit support for clustered systems allows analysis of communication between nodes
- support for hardware or OS performance counters
- unified filtering capabilities
- maximum/minimum values are available in statistic displays
- trace file specific configuration files.

As an aid to users familiar with Vampir 2.5 the following is a list of some changes especially relevant to getting started with Vampir 3.0 and in using this document:

- PARALLELISM TIMELINE has been renamed to SUMMARY TIMELINE.
- The new PROCESS PROFILE complements the ACTIVITY CHART.
- The call tree may also be displayed for all processes simultaneously (GLOBAL CALL TREE).
- The **Filter Processes** entry of the **Global Displays** menu and several context menu entries in various displays were moved to the global **Filter** menu.
- Vampir now starts with the SUMMARY CHART display by default. This default may be changed in the new PREFERENCES/DISPLAYS/STARTUP requester.
- The file format has evolved such that a trace written with 3.0 cannot be loaded in 2.5. Therefore the file suffix was changed from .bpv/.pv to .bvt/.avt. However, traces written with previous versions of Vampirtrace should, in general, still be readable in 3.0.

1.4 What Do You Need to Run Vampir?

1.4.1 Supported Platforms

Vampir is currently available on all major UNIX platforms and to see the full current list visit the Pallas website at `http://www.pallas.com/e/products/vampir/index.htm`. A graphical display running an X server (X11R5 or later) is also required.

1.4.2 Trace Generation Software

To generate trace files from MPI applications the Vampirtrace instrumentation library from Pallas must be used. Vampirtrace is the link between the MPI application and performance analysis or prediction: it gathers information about the execution behavior that can be fed into Vampir. It is easy to use since re-linking the application with the Vampirtrace library is sufficient in most cases. A powerful configuration mechanism allows customisation of Vampirtrace's operation, e.g. by filtering the trace data during runtime. The low-overhead Vampirtrace library works with all major MPI implementations in combination with programs written in Fortran 77, Fortran 90/95, Java, C and C++. The instrumentation functionality does vary across platforms with respect to automatic instrumentation and viewing source code references. Further information on availability, supported platforms and supported MPI implementations may be found in Vampirtrace documentation.

1.4.3 Downloading Vampir

The latest version of Vampir is available to download on the Pallas website. The Vampir distribution includes the Vampir binaries, example trace files, and documentation.

The Pallas website can be found at `http://www.pallas.com/e/products/vampir` where the products and Vampir hyperlinks should be followed to find the Vampir pages.

1.4.4 More Information about Vampir

Additional (and current) information about Vampir is available from the Pallas web pages. User's of Vampir are always invited to direct questions and suggestions about the Pallas tools via email to support@pallas.com.

Chapter 2

Using Vampir to Understand Performance

2.1 MPI Performance Versus Application Performance

It is often stated that enhanced parallel performance comes from enhanced simultaneous activities. Usually this devolves into the separation of communication performance and application performance. The latter is a function on the choice of algorithm, data structure/partition, etc. whereas the former is a function of the communication costs for the chosen MPI procedures. Vampir can give insights into both categories of performance issues because, through the many displays of a trace file that it offers, it can reveal subtle performance issues. It supports the programmer both in the discovery of problems and in the quantitative analysis of performance improvements between different versions of a program after code changes have been made. This chapter demonstrates this with specific examples without providing distracting details of how the corresponding displays are obtained. Whereas the next chapter *Getting started with Vampir* explains exactly how the Vampir displays shown here are produced.

Two algorithms are discussed in the following examples. The first is a Poisson solver with a finite difference method and the second is a sort algorithm. The naive version of the Poisson solver uses `MPI_Sendrecv` to exchange data in a clock-wise strategy where the process of rank i exchanges data with the rank i+1 neighbour after message exchange with the rank i-1 neighbour is complete. As this chapter will show, this causes excessive costs for communication that can be reduced by replacing these calls with non-blocking communication. The corresponding trace files are located as follows:

```
$PAL_ROOT/examples/redblack_sndrcv.stf
$PAL_ROOT/examples/redblack_icomm.stf
```

The sort problem will serve as an example where a bad choice of the application algorithm causes a load balance problem. Two versions were implemented, both using a (qsort-like) algorithm. First, all processes sort their local parts with qsort. Second, these sorted parts are merged with an operation of linear complexity. In the first version, the merge part gathers data to the rank 0 process where all remaining work is performed. The more advanced algorithm distributes the work by organising the processes in a tree and doing a receive/merge/forward operation in each process. The corresponding trace files are located in the directory:

```
$PAL_ROOT/examples/sort_lin.stf
$PAL_ROOT/examples/sort_tree.stf
```

This demonstration shows that with Vampir/Vampirtrace it is easy to identify both the problem and quantitative metrics for measuring performance improvements. These methods also apply in more complex cases that would otherwise be hard to approach without a tool such as Vampir.

2.2 Identifying Performance Costs with Vampir

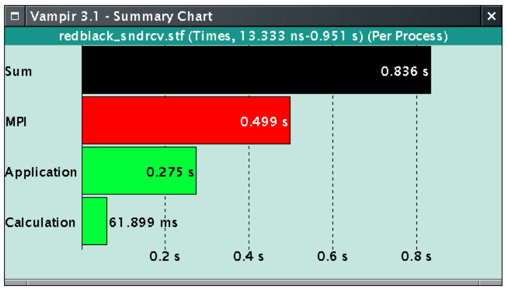
2.2.1 Statistical Overview

When Vampir is started as follows:

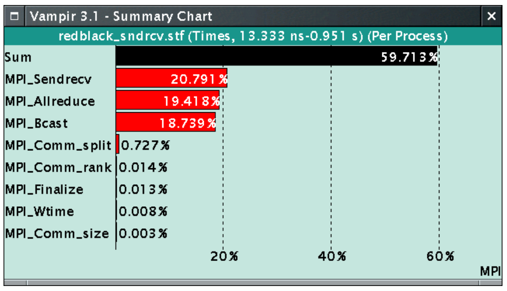
```
vampir $PAL_ROOT/examples/redblack_sndrcv.stf &
```

it opens the STF trace without actually loading event data. Instead it just loads the much smaller meta data and opens the FRAME DISPLAY to present a preview of the trace. This is explained in more detail in Section 3.3.5. To proceed to the traditional analysis of the real trace data, use the right mouse button to show the context menu and select **Load/Whole Trace**, or use the **L** keyboard shortcut.

After loading the trace data Vampir opens with the SUMMARY CHART display by default. This shows the sum of the time consumed by all activities over all selected processes for the trace file `redblack_sndrcv` in 2.1(a). This is a very condensed view of the entire program run that clearly shows more time is spent in MPI procedures compared to user code. However, even more detail is easily accessible. For example, for a better view of MPI procedures consider the summary chart display shown in 2.1(b). This display was modified to show only MPI function calls, and instead of absolute times the labels now show percentages of the total runtime. This performance metric pinpoints the dominant MPI procedures, but it is not yet clear why they take so much time.



(a) with absolute times



(b) with relative times for MPI

Figure 2.1: SUMMARY CHART display

2.2.2 Runtime Analysis

An in-depth-analysis of the runtime behaviour addresses the issue as to why MPI procedures dominate. Because Vampir has access to a trace that exactly describes when an event occurred in the application, it can draw these changes over time in so-called Timeline Displays. One such timeline is the SUMMARY TIMELINE, which shows all analysed state changes for each process (over the chosen time interval) in one display as a histogram over time bins. Within each time bin the number of processors in either an MPI (red) or Application (green) state is shown as a stacked column bar chart. An example of the summary timeline is shown in 2.2 for `redblack_sndrcv` and from this display the overall structure of the program becomes clear. First there is a setup phase where new process become active and wait for others to get ready, then a phase of several iterations with increasing and decreasing MPI activity. Because a histogram is used the SUMMARY TIMELINE display works well even for long trace times.

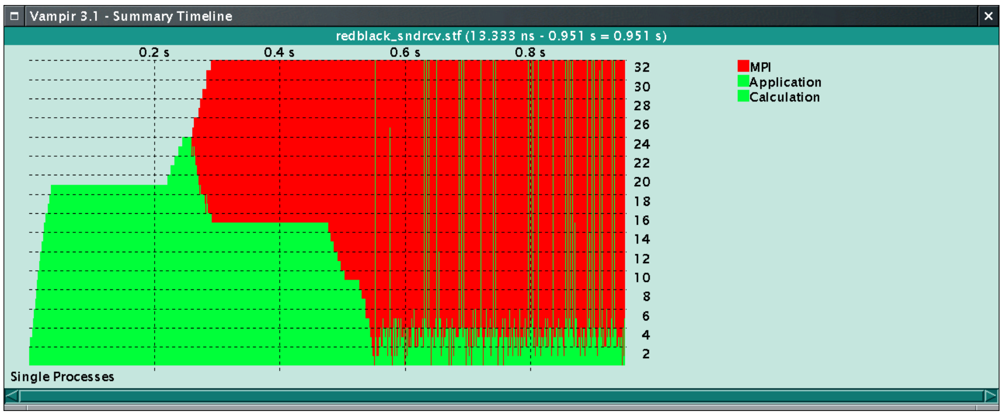


Figure 2.2: SUMMARY TIMELINE display

On the other hand, the GLOBAL TIMELINE by default shows all events and thus supports detailed analysis at the function (or procedure) level. Because the whole program run is visible in 2.3(a), there is only enough space to print the name of the long-running `MPI_Bcast` and `User_Code` states at the beginning of the run. The color map has used red for MPI, green for the application, but also introduces purple for collective operations. In this example there are many `MPI_Allreduce` calls and this is why purple dominates (however, an option is available to turn off display of collective procedures). It is possible to identify individual states by clicking on them with the left mouse button. This action marks them in the timeline display and opens a display similar to that shown in the MPI example of 2.4 for individual processes.

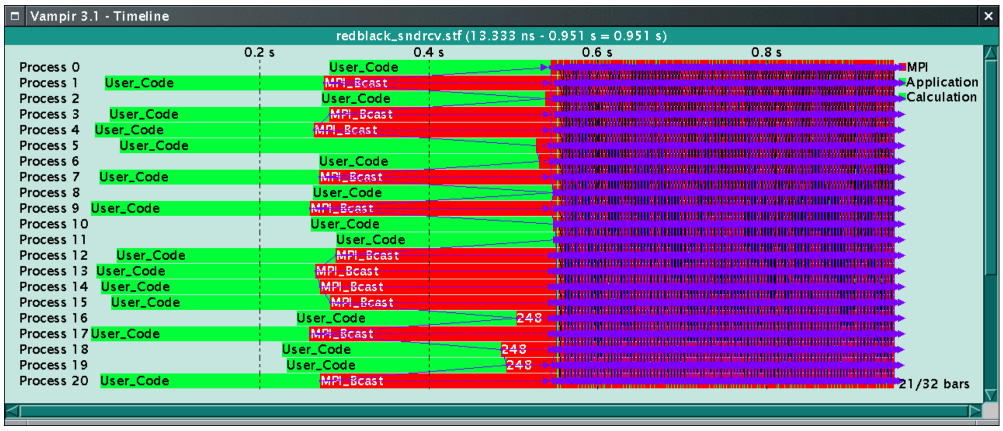
However, the performance situation looks quite different after magnification of part of the timeline in 2.3(a) with the zooming feature of Vampir. The result is in 2.3(b) which shows a few iterations in the application. At this level of granularity the MPI symbols for `MPI_Allreduce` and `MPI_Sendrecv` are visible. The purple horizontal arrowheads indicate the beginning and end of time intervals for collective MPI communication. Also visible are messages between different processes shown as thin black lines connecting different points in the respective time series.

For the case of `redblack_sndrcv` 2.3(b) shows there is a clear asymmetry of the time interval for MPI procedure calls as a function of processor rank (increasing for `MPI_Sendrecv`, decreasing for `MPI_Allreduce`). This asymmetry is due to the choice of algorithm with a clockwise message passing algorithm that is contingent on ascending rank order (as described later). Higher rank processes wait longer for completion of `MPI_Sendrcv` because more processes of lower rank must first complete their message transactions. Conversely, `MPI_Allreduce` has the inverse behaviour where the lowest rank process starts the operation soonest and must wait for the highest rank process to complete first.

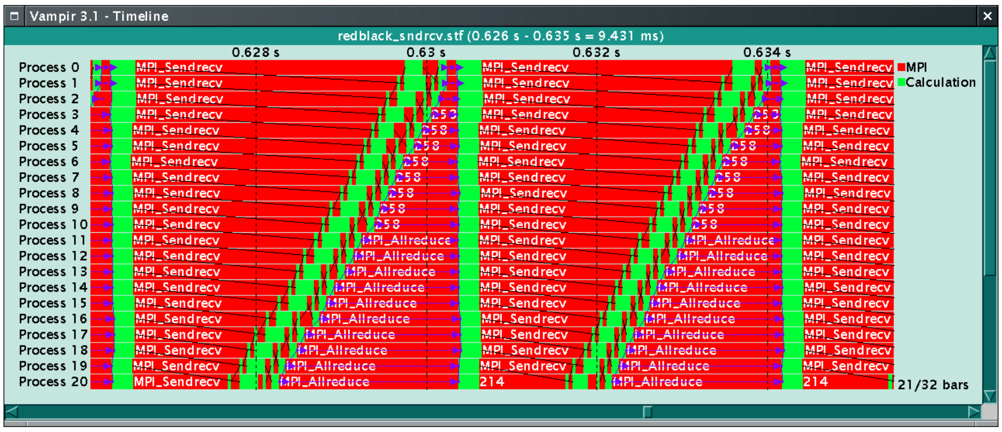
2.2.3 Evaluation of Alternative Data Exchange Implementations

In evaluating alternative coding methods trace files may be compared in Vampir by either:

- using multiple main windows to open multiple trace files simultaneously, or
- using only one main window with selected displays for multiple trace files.



(a) whole trace

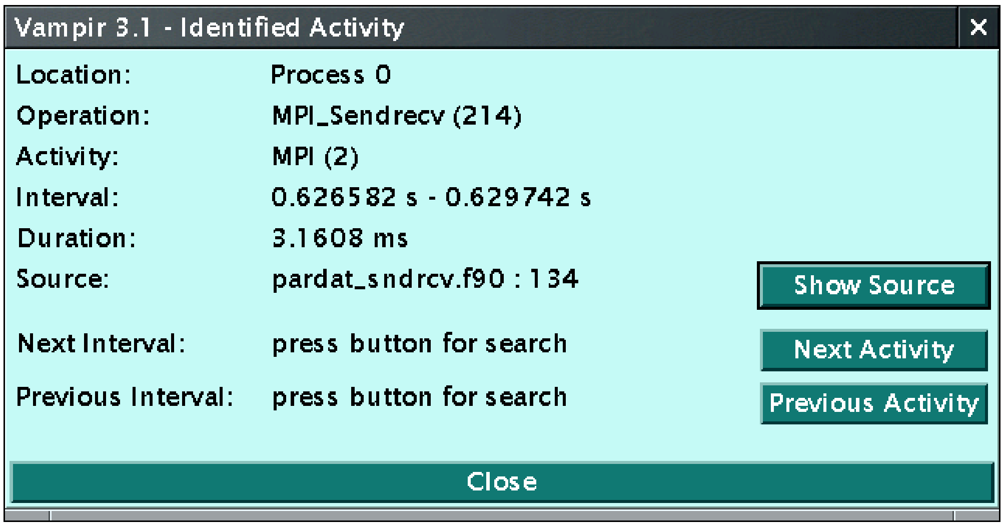


(b) a few iterations

Figure 2.3: GLOBAL TIMELINE display for redblack_sndrcv

In the first comparative method all Vampir displays (or dialog boxes) may be compared side-by-side while in the second only some displays can be used to show results from multiple trace files. This chapter uses the first comparative method for trace files, and the next chapter explains the second method.

It is clear from the previous analysis that something needs to be done to improve the performance of the code. For Poisson solvers with finite difference methods the standard MPI monographs compare blocking and non-blocking communication strategies for halo exchange. The first MPI case discussed above (`redblack_sndrcv`) uses blocking `MPI_Sendrecv` procedure calls while an alternative MPI case, introduced below (`redblack_icomm`), replaces the `MPI_Sendrecv` calls with `MPI_Isend` and `MPI_Recv` pairs, together with a `MPI_Waitall` procedure call. These are implemented in decoupled blocks where non-blocking sends are performed in the same order messages are received. Therefore performance differences are to be expected when the two MPI cases are compared, even though they are functionally equivalent. The standard MPI monographs make generalized comments about such performance differences, but only by using Vampir is a quantitative, in-depth, understanding of the specifics of performance differences possible. The analysis described below demonstrates how this understanding is reached by using Vampir to


Vampir 3.1 - Identified Activity

Location: Process 0
Operation: MPI_Sendrecv (214)
Activity: MPI (2)
Interval: 0.626582 s - 0.629742 s
Duration: 3.1608 ms
Source: pardat_sndrcv.f90 : 134
Next Interval: press button for search
Previous Interval: press button for search

Figure 2.4: IDENTIFIED ACTIVITY dialog

compare the first approach with the improved one.

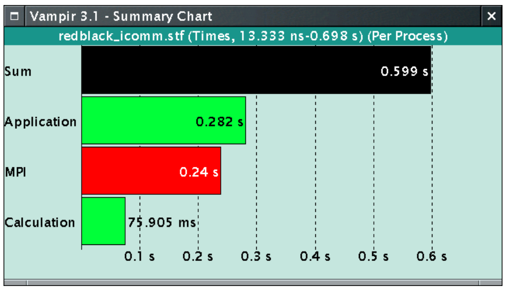
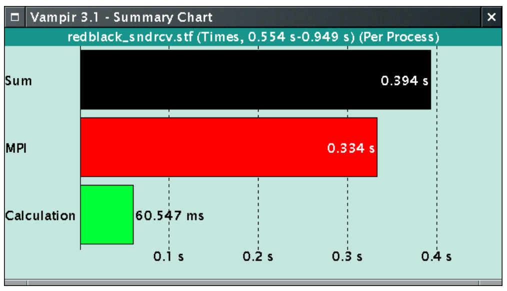


Figure 2.5: SUMMARY CHART display for redblack_icomm

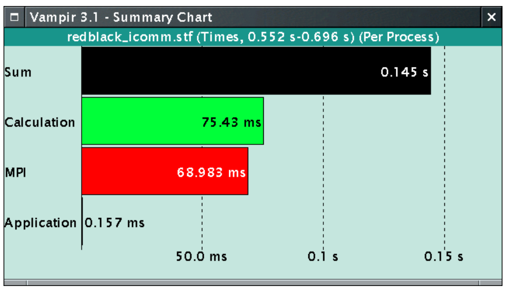
A comparison of the respective SUMMARY CHART displays 2.1(a) and 2.5 shows that the case of `redblack_icomm` consumes only 0.599 seconds compared to 0.836 seconds for `redblack_sndrcv`. It is also obvious that the dominance of the MPI time portion has been reversed. By switching to displaying function counts, one also finds that the dominating calls in redblack_icomm `MPI_Comm_split` and `MPI_Bcast` are only called once and can be ignored, as their relative importance decreases for higher number of iterations and bigger problem sizes.

Nevertheless, to obtain a quantitative measure of communication-to-computation load balance in the body of the application, the zoom feature of Vampir may be used to exclude the `MPI_Bcast` and `MPI_Finalize` from the global timeline display. In Vampir performance information viewable in different displays is linked in a context sensitive way. Therefore the corresponding summary chart displays, shown in 2.6(a) and 2.6(b), are the result after zooming in the timeline display. When looking at these statistics, it becomes obvious that the optimization was successful: time wasted in MPI was reduced from 0.344 to 0.068 seconds.

Finally, to confirm the change in communication pattern, and the performance improvement, 2.7 shows magnification with the zooming feature of Vampir for the `redblack_icomm` timeline display, similar to 2.3(b) for `redblack_sndrcv`. The contrast in communication patterns with `redblack_sndrcv` is clearest if the zooming time interval for both zoomed displays is chosen



(a) redblack_sndcv



(b) redblack_icomm

Figure 2.6: SUMMARY CHART display, just the iterations, absolute times

to show a few iterations. The display for `redblack_icomm` shows that the data exchange is done by all processes at the same time and more quickly. However, the ratio of communication to computation still poses a problem, which can possibly be addressed by increasing the work to be done by each process.

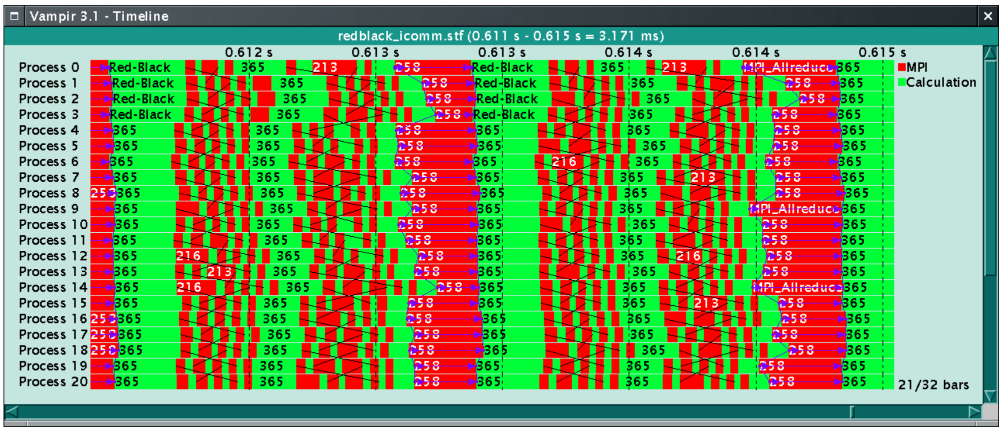


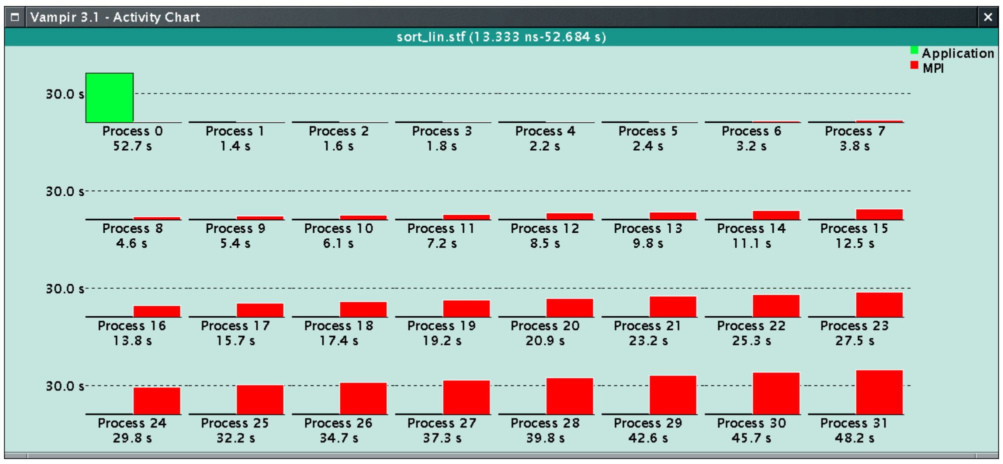
Figure 2.7: GLOBAL TIMELINE display for redblack_icomm, a few iterations

2.3 Identifying Load Balance Problems with Vampir

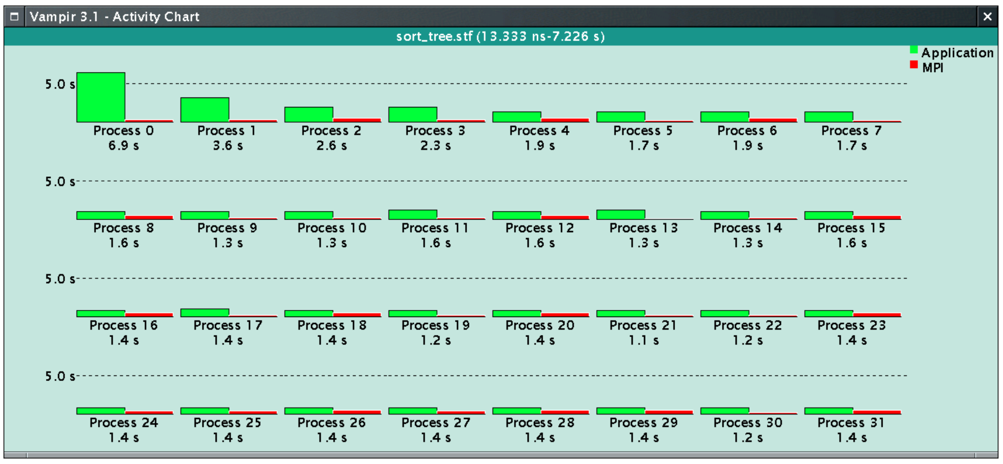
2.3.1 Activity Statistics

Another statistical display in Vampir is the ACTIVITY CHART. It is especially useful in looking at distribution of time between MPI and user code for all processes at a glance. This display shows a pie chart for each MPI process with proportionate division by application, MPI, and Vampirtrace (VT_API) based on the total time utilization. VT_API is listed in addition to MPI and Application because there was one or more calls to a Vampirtrace API function. However, such calls typically take negligible time and thus the display helps to understand the uniformity of the load distribution by process at-a-glance. To demonstrate the usefulness of the ACTIVITY CHART display the sort cases are used. The corresponding Activity Chart displays are shown in 2.8(a) and 2.8(b) for the sort cases of `sort_lin` and `sort_tree`, respectively.

It is no surprise that in the serial merge version of the sort example, `sort_lin`, the processes with rank 0 does all of the merge and thus more work than the others, while all other processes are blocked in MPI, waiting for process 0 to communicate with them. Because process 0 deals with one process after the other, the processes with higher ranks are blocked longer. Whereas, the optimized sort code, `sort_opttree`, is a textbook example that uses the tree-based approach and results in a more even distribution of the work load, which in turn shortens the total runtime from 52.684 to 7.226 seconds.



(a) sort_lin

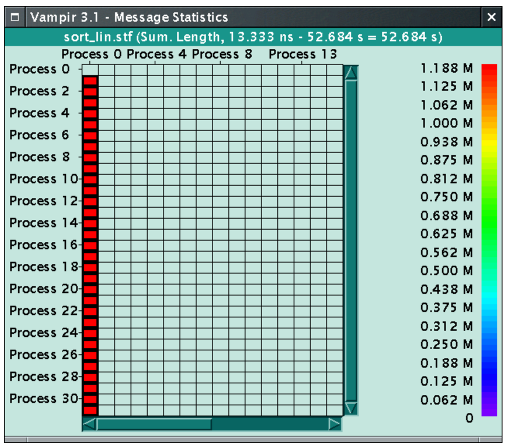


(b) sort_tree

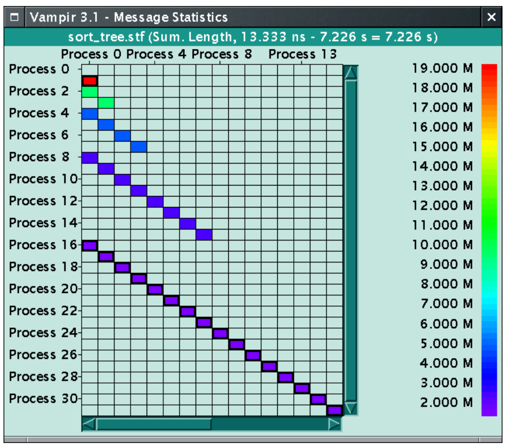
Figure 2.8: ACTIVITY CHART display

2.3.2 Message Statistics

Another important indicator of possible MPI performance problems is in the details of messages passed between processes. For this Vampir provides a feature-rich MESSAGE STATISTICS display for message counts, or the minimum, maximum, or average of message length, rate, or duration. The default is to display the sum of all message lengths and these displays are shown in 2.9(a) and 2.9(b), for the cases of `sort_lin` and `sort_opttree`, respectively. The differences in the two algorithms are well illustrated. In the former case, (`sort_lin`), messages are received only by the process with rank 0 (from all other processes with rank ≥ 0). They are all the same size. In the latter case, (`sort_tree`), interprocess communication occurs for processes with rank ≥ 0 , dependent on the process location in the sort tree. The message size doubles at each level in the sort tree.



(a) sort_lin



(b) sort_tree

Figure 2.9: MESSAGE STATISTICS display (zoomed to exclude empty right half)

2.4 Summary

This purpose of this brief introduction was to introduce the power of Vampir in revealing performance details for parallel algorithms and MPI implementations. Two examples were used: a Poisson solver and a sort algorithm and each was implemented in two MPI versions. It may seem obvious that performance differences are to be expected. However, it has been demonstrated that only with Vampir can the details of performance issues be revealed and quantitative information analysed. Furthermore, the quantitative information that Vampir provides is instrumental in determining where the performance bottlenecks are and suggesting what can be done to relieve them.

In the case of the Poisson solver a Vampir summary chart display gives a snapshot of application versus MPI time utilization. Diving into this display type reveals details of individual MPI procedure calls. A timeline display, especially with the zoom feature, reveals computation-to-communication patterns as a time series. The summary chart display reveals the extent of concurrency in this pattern. Taken together, the Vampir summary chart, timeline display, and summary timeline, give clear, quantitative measures of the superiority of the choice of MPI algorithm in `redblack_icomm` over `redblack_sndrcv`.

Vampir's statistics displays may also be used for load-balance information. The activity chart enables a quick and easy decision between MPI usage choices such as `sort_opttree` versus `sort_lin`. Another statistics display is the message statistics display which focuses exclusive attention on the MPI communication as it appears over the whole execution time. In the message statistics display communication bottlenecks are easily detected by choosing one of several display options.

In both the `redblack_sndrcv` versus `redblack_icomm`, and the `sort_lin` versus `sort_opttree`, the preferred algorithm has the shorter wall clock time. However, only with Vampir is the detail of the reasons for the improved efficiency clear:

- for redblack—removing rank-ordered time dependence effects due to the way `MPI_Sendrecv` and `MPI_Allreduce` are used, and
- for sort—replacing linear message passing with tree communication patterns.

Chapter 3

Getting Started with Vampir

3.1 Setup Procedure

To install Vampir follow these steps:

1. Download the package for the appropriate platform from the Pallas web site (http://www.pallas.com/e/products/vampir).
2. Create a directory for Vampir as a root for all Vampir files and directories.
3. cd into this directory and uncompress the Vampir package.
4. Run the install-Vampir script to update the file and directory permissions.

If `$PAL_ROOT` is the root directory of the Vampir installation, e.g. `/usr/local/vampir`, then the Vampir executable image resides in the directory `$PAL_ROOT/bin`.

3.1.1 License File

In order to run Vampir on a system, a license key must first be obtained from Pallas. The license keys are stored in a plain ASCII file, the pathname of which must be made known to Vampir by setting one of two environment variables:

- `PAL_LICENSEFILE` specifies the complete pathname of the license key file.
- `PAL_ROOT` points to the root of the Vampir installation.

The pathname of the license file is assumed to be `$PAL_ROOT/etc/license.dat`.

If called without a valid license, or with invalid settings of the above environment variables, Vampir aborts with an error message, e.g.:

```
VAMPIR: could not access license file
VAMPIR: need license for product VA30, arch SPRC-SOL, hostid 2155491417, uid E0
VAMPIR: network address 192.168.3.4
VAMPIR: environment var PAL_ROOT not set
VAMPIR: environment var PAL_LICENSEFILE not set
```

In this case make sure that the correct license key file is specified with the above mentioned environment variables. If they are set, ensure that the files they point to are readable and contain a valid license.

3.1.2 Support Tools

Vampir accesses several external tools to perform some of its tasks. These tools are described below and they are required only if the features they support are needed. Otherwise Vampir can be run without the corresponding tools. If they are installed these tools must be located in a directory included in the shell's executable search path (path shell variable or `PATH` environment variable). The tools and the features they support include:

- A print command for lists and window snapshots (like lp or lpr). Lists are printed as plain ASCII files and window snapshots as PostScript.
- Import, a screen snapshot utility from the ImageMagick package to export or print window contents. It is included in the Vampir package and installed in the same directory as the Vampir executable. ImageMagick can also be downloaded from several ftp servers, including:
  - `ftp.x.org`, directory `/contrib/applications/ImageMagick`, or from
  - `ftp.zam.kfa-juelich.de`, directory `pub/graphics/ImageMagick`.
- If trace files are compressed with gzip or compress, Vampir can extract them automatically if their counterparts gunzip or uncompress, respectively, are available. The GNU `gzip` package can be downloaded from
  - `prep.ai.mit.edu`, directory `pub/gnu`, file `gzip-1.2.4.tar`, or
  - other GNU software ftp servers.

3.1.3 Tracefiles

Sample trace files generated with Vampirtrace are available in the directory `$PAL_ROOT/examples/` for use with Vampir. Please refer to the `README` file there for further information. Some of the trace files found there were used in the previous chapter to demonstrate how Vampir helps in understanding performance. In this chapter the tracefile `redblack_sndrcv` is used to introduce Vampir features.

3.2 Vampir Basics

Vampir has many displays and features, and their interaction can be quite complex. Therefore it is important to keep in mind that there are four basic display window styles in Vampir:

- Timeline displays that show changes in the program over a certain time interval
- Statistic displays that show values derived from the trace data of a time interval selected in one of the timeline displays (often in graphical and/or textual format).
- Global displays independent of any time interval (source listings, call tree).

- Configuration dialogs.

A complete description of all possibilities in the style of a *reference guide* would not be a good starting point for the novice user. Therefore the following is intended as a *quick start guide* to some essential functions and Vampir displays that represent a basic subset. It is hoped that this approach will inspire the reader to explore the full potential of Vampir after familiarisation with the basics.

3.2.1 Vampir Activities and Symbols

Visualisation in Vampir is based on the concept of *Activities* and associated *Symbols*. The combination of Activity and Symbol is called a *State*. This is not quite the same as a Symbol, because the same symbol name may appear in different activities while a state is a unique entity. For MPI programs the major activity is named MPI and all MPI functions are defined as symbols inside this activity. With the help of these elements it is possible to show the different states in a parallel program. In general, Activities may be thought of as classes of Symbols. Symbols are detailed descriptions of Activities. In detail, the Activity MPI has all MPI functions, such as `MPI_Bcast`, defined as associated Symbols. With this hierarchical dependence it is possible to reduce the amount of information initially presented by statistical displays, because every statistic display shows only the different Activities of the traced program in the first display it opens.

To distinguish one Activity from another it is useful to assign an individual color to each Activity. Symbols inherit the Activity color in Timeline Displays and in most Statistic Displays. It is also possible to rearrange the assignment of Activities and Symbols with a dialog, which can be reached from the main menu.

Vampirtrace also allows for user-defined (source code level) Activities and Symbols and these will also appear alongside predefined Activities (such as MPI), in Vampir graphical displays with colors definable by the user as described later.

3.2.2 How does Vampir work?

Vampir supports three different approaches to performance analysis that can be used to progress from a rough overview to a very detailed view of single events:

3.2.2.1 Statistical Analysis

The STATISTICS VIEWER visualizes statistical information about the whole program run and covers:

- function calls
- sent messages
- collective operations
- OpenMP regions (if contained in the trace)

This analysis does not require collection of any trace events, therefore it is a good starting point for trying to understand an unknown application that might produce an unmanageable trace. How to do this is explained in more detail in Section 3.3.7.

3.2.2.2 Thumbnail Previews

If Vampirtrace has collected trace data and generated frames for this data, then attached to each frame there are so called *thumbnails*: they provide statistical information similar to the one described in the previous section, but in contrast to that statistics data thumbnails are restricted to the data covered in their frame. In other words, they provide a preview of the data that Vampir would find when loading just a specific frame.

As soon as one has identified a problem in these previews, one can proceed to load trace data and continue with *Event-based Analysis* as described in the next section. Under some circumstances it may be sufficient to just look at the previews without ever loading the real data: for that purpose Vampirtrace supports the DISCARD configuration option which collects events and generates thumbnails, but never writes the event data into the trace.

3.2.2.3 Event-based Analysis

Vampir loads a trace file filled with trace data that describes exactly when and in which process certain events occurred. An event includes a time stamp and a process number, plus additional event-specific information. For example, a function would generate one event for entering the function and one event for leaving it and the additional information for these events is the number of a state (as defined in the previous section). In other words, a state referenced by certain events, but is not an event itself. Other events are used to represent messages and collective operations.

A trace file is generally extensive and after the trace file has been loaded (opened in Vampir) subsets of this data are viewed. Which part of the trace data is viewed depends on the Vampir display chosen. A variety of different displays are available in Vampir to show time series, statistical data, etc. Because of the volume of information in the trace file, data viewed in a Vampir display is generally defined by filters. Some filters depend on the display chosen, while others may be set by menu choices. As, examples, filters may be on processes, on time, by Activity, or by Symbol, depending on the display and menu options chosen. Filters are introduced in Section 3.5.

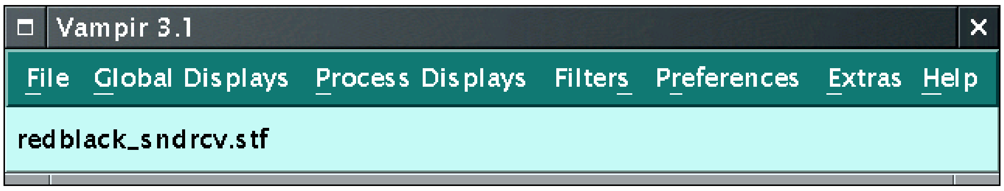
If a trace is too large to be loaded completely, the new Structured Trace Format (STF) and the FRAME DISPLAY support loading a subset of the data. The analysis described above then only takes the loaded data into account.

3.2.3 Visible Time Interval

Crucial to understanding the event trace file and its displays in Vampir is the concept of time interval and displays that are time series defined on it. The time interval is by default the wall clock time for the whole execution and different displays may show results specific to it. However, the time interval displayed may be changed interactively, and a zoom feature in Vampir allows for the time interval in a time series display to be magnified (see Section 3.2.5 below for a description of how zooming is performed). An example of a zoomed timeline display in Vampir was shown in 2.3(b) for the trace file `redblack_sndrcv`. By default other displays that are open will respond to the zoom (or magnification) in the time series, in a context sensitive way, and also display statistics, or results, for the magnified time interval. However, this default behavior may be disabled with a toggle button for a **use timeline portion** switch and in this way other open displays will have a frozen time interval not correlated with further zooming in the timeline display. Further discussion of this behavior is given below.

3.2.4 Vampir GUI Common Features

The following sections describe in more detail the types of graphical displays offered by Vampir and how to manipulate them. The description in this chapter is only a survey of some specific, often-used, features. However, before details on each feature are presented a few characteristics common to most Vampir displays are described.


Vampir 3.1
File Global Displays Process Displays Filters Preferences Extras Help

redblack_sndrcv.stf

Figure 3.1: Vampir main window panel.

Vampir opens with a simple Main Window displaying a panel of main menu buttons, as shown in Figure 3.1, and each is described in more detail in the following Sections. The main menu buttons each have drop-down menus used to select different features and viewing windows. Each main menu may be detached ("torn-off") from the main window panel and moved elsewhere on the desktop. This, "tear-off" feature makes it easier to repeat menu selections without having to pass through the drop-down selection each time. Usually when a user selects a drop-down menu, it is only displayed until a selection is made, after which it is removed. However, a torn-off menu is not removed after a selection and the user may make repeated selections from it. When tear-off functionality is enabled in Vampir, the first item in the drop down menu is a dashed line. By selecting this item with the left mouse button the menu is placed in a separate window with limited window manager decorations. The menu remains torn off until the user cancels the menu by pressing the ESCAPE key within the window or selecting **Close** from the window manager menu.

Inside Vampir displays all three mouse buttons have functions assigned to them. Generally a single click functions as follows:

- The left mouse button is used to select or identify something.
- The middle mouse button is used to deselect all prior selected processes (if process selection is applicable).
- The right mouse button is used, inside a display, to pop up a context menu with display-specific options and functions.

Selecting a function with the right mouse button from the display's context menu may change the meaning of a left mouse button click. To indicate this the mouse cursor shape inside the corresponding display will be changed to a special pixel map such as cross-hairs or a target symbol. This will indicate that a function from a context menu is activated but needs further input, such as an additional mouse click, or selection of an area to complete the previously selected function.

3.2.5 Zooming

The left mouse button also has another special purpose. Moving the mouse with the left button depressed is used to zoom into a display, or if zooming is not applicable, to mark an area which is used for multiple process selection. In most Vampir graphic windows zooming is possible. The only exception is the pie chart mode of the statistic displays. To zoom in (i.e. magnify) a part of a display, press the left mouse button at the start of the region to be magnified. Then, while holding the left mouse button down, drag the mouse to the end of the desired region. In this process a rectangle will be drawn to define the area of the display to be magnified. Once the desired endpoint is reached release the mouse button. After this the display window will be redrawn to show only the magnified portion and the scales are adjusted accordingly.

3.2.6 Filtering

In configuration dialogs one may customize the behaviour of Vampir and filter data to be viewed in other displays. More detail on filtering is given in Section 3.5. Most of the filter dialogs present the current configuration, which may then be modified, and activated when the *Apply* button is pressed. This last action does not close the dialog so that it may be repeatedly used to try out various configurations in rapid succession. Often an *Undo* button is provided to restore the initial configuration. If configuration settings are changed and the dialog is closed without activation of the *Apply* button all changes are discarded. The other type of configuration dialog has no direct influence on the appearance of other displays, so it has only *Okay* and *Cancel* buttons that close the display in addition to accepting (or rejecting) the new configuration.

3.2.7 Context Menus

Usually Vampir displays have context menus available that are accessible through a single click with the right mouse button inside the corresponding display (as discussed in Section 3.2.4). All display-specific functions and settings (or options) are available through such context menus. These context menus may appear to be similar, but differ depending on the display they originate from. Only some context menus and some common options they offer are described in the following sections.

Some common features are listed here. Most selections from a context menu have a primary menu and a submenu. Examples of common selections available from the primary menu are items described in the following table. Secondary items from submenus can be numerous and some are introduced as needed in the following Sections. They control the form of the display and their inter-relationships.

Item	Function
Components	Add/remove certain parts of a display (such as legends, scales,etc.).
Display	Show a certain attribute (such as message length, count, etc).
Hiding	In statistical displays: remove entries from the visible list.
Mode	Switches the visual appearance of the display (e.g. bar vs. pie chart)
Options	Modifies how information is calculated or triggers certain operations.
Print	Bring up the print dialog (see Section 3.6.4).
Ruler	Bring up a message box with the time for a mouse-defined region.
Select	Chooses the entities that are displayed.
Sort (or Sort by)	Sets the order in which entries are displayed.
Timeline	Determines which time interval is used: **Global** = the shared time interval **Local (Freeze)** = the current time interval remains visible **All** = the whole trace
Sync Timeline	show same time interval as elsewhere if enabled, otherwise use a separate time interval for the display

Use Timeline Portion	show same time interval as elsewhere if enabled, otherwise the whole trace
Zoom	Perform a zoom operation.
Undo zoom	Undo zoom operation(s).

3.3 File Handling

3.3.1 Structured Trace Format (STF)

The conventional approach of handling trace data in a single trace file is not suitable for large applications or systems, where the trace file can quickly grow into the tens of gigabytes range. On the display side, such huge amounts of data cannot be squeezed into one display at once. Mechanisms must be provided to start at a more general level of display and then resolve the display into more detailed information. A more general view of the data will be represented by frames, which are simply parts of the trace tied together. Several frames may exist in one trace and the user will be able to navigate through the frames and select one or more to request additional detailed information. The subdivision of a trace into frames can occur along three principal dimensions:

1. along the time axis different frames represent different time intervals
2. along the task/thread axis different frames represent different threads, tasks, or groups
3. along the kind of trace data a frame can contain any combination of the following categories of data: state changes, collective operations, point-to-point messages, HPM counter values, OpenMP region data, and finally file I/O data (for MPI/O)

For an application, the subdivision of trace data into frames can be defined at compile time with calls to the frame definition routines in the Vampirtrace API, or at runtime by specifying the configuration options discussed in Section 3.3.1.3.

These design goals require a more powerful data organization than the traditional Vampir trace file format can provide. In response to this, the Structured Trace File Format (STF) has been developed. The aim of the STF organization is to provide a file format which:

1. can arbitrarily be partitioned into several files, each one representing one or several frames
2. allows fast random access and easy extraction of data
3. is extensible, portable, and upward compatible
4. is clearly defined and structured
5. can efficiently exploit parallelism for reading and writing
6. is as compact as possible

The traditional trace file format is only suitable for small applications, and cannot efficiently be written in parallel. Also, it was designed for reading the entire file at once, rather than for extracting arbitrary data. The structured trace file implements these new requirements, with the ability to store large amounts of data in a more compact form.

Fixed size overview data (such as summary routine profiles, OpenMP statistics) will be shown using the existing Summary Chart display techniques, which can show average, minimum, and maximum statistics, among other scalars that are tracked. To accommodate a large number of different components, such as states, registers, OpenMP regions, and communication operations, the displays will be scrollable and can be sorted by name or value.

The current release of Vampir and Vampirtrace still support the traditional Vampir file format. See Vampirtrace's configuration option LOGFILE-FORMAT for information on how to generate the older formats.

3.3.1.1 Components of a Structured Trace File

A structured trace file actually consists of a number of files. Depending on the number of processes and their distribution to actual files, the following component files will be written, with ¡trace¿ being the basename of the file:

1. one index file with the name <trace>.stf
2. one declaration file with the name <trace>.stf.dcl
3. one frame file with the name <trace>.stf.frm
4. one statistics file with the name <trace>.stf.sts
5. one message file with the name <trace>.stf.msg
6. one global operation file with the name <trace>.stf.gop
7. one or several process files with the name <trace>.stf.pr.<index>
8. for the above three kinds of files, one anchor file each with the added extension .anc

The records for routine entry/exit, counters and OpenMP regions are contained in the process files. The anchor files are used by Vampir to fast-forward within the record files. They can be deleted, but that will result in slower operation of Vampir.

Please make sure that you use different names for traces from different runs; otherwise, you will experience difficulties in identifying which process files belong to an index file, and which ones are left over from a previous run. To catch all component files, use shell wildcards like <trace>.stf.*, and put the files into a tar-archive for transmission, or convert to the single-STF format with the stftool (see next section).

The number of actual process files will depend on the setting of the STF-USE-HWSTRUCTURE and STF-PROCS-PER-FILE configuration options described below.

3.3.1.2 Single-file STF

Because of the many components of a normal STF file, file handling can become difficult and for transmission these files must be packed together. The single-file STF format addresses these problems by putting all the components into one file. This can be done by Vampirtrace, albeit at the cost of reduced performance when writing the trace and some wasted file space. The suggested file suffix is .stf.single.

The other method is to convert a normal STF into single-file STF with the stftool. This can be combined with e.g. gzip to to produce a compressed .stf.single.gz file in one step as described in Section 3.3.1.4. Vampir is able to read single-STF files without any performance hit, but it needs to create a decompressed temporary file if compression was used.

3.3.1.3 Tracefile Configuration Options

Vampirtrace configuration options control how a tracefile is written. These options and how they are used is explained in more detail in VT.pdf and the `man` page for `VT_CONFIG` (the environment variable) that sets a Vampirtrace configuration.

The configuration option LOGFILE-FORMAT switches between the traditional Vampirtrace formats:

STF the default Structured Trace Format

STFSINGLE single-file STF

BINARY the traditional Vampir format

ASCII the traditional human-readable Vampir format

The two main aspects of the STF behavior that can be configured are:

- frame definition: frames can be defined by a regular subdivision of the process and execution time space, and also depend on the hardware structure of the machine (where all of the processes are running on the same SMP node in one frame).
- mapping to files: frames are just a logical concept, and need not coincide with the set of files actually written. Vampirtrace allows the event data to be partitioned in the process files by blocking, or coinciding with the hardware structure, such that events from processes running on the same SMP node end up in one file.

The most important mechanisms for defining frames supported in the current release are:

- FRAME-USE-HW-STRUCTURE is enabled by default unless each process has its own node; it combines all processes running on the same SMP node into the same frame. It can be combined with the next two configuration options.
- DATA-PER-FRAME <number of bytes> starts a new frame as soon as the raw trace data in memory of all processes exceeds the given threshold. This results in frames that cover equal amounts of trace data and allow the user to split the trace into chunks that can be loaded well by Vampir.

To determine the file layout, the following options can be used:

- STF-USE-HW-STRUCTURE is enabled by default. It will save the local events for all processes running on the same SMP node into one process file. It can be combined with the STF-PROCS-PER-FILE directive.
- STF-PROCS-PER-FILE <number> limits the number of processes whose events can be written in a single process file. If STF-USE-HW-STRUCTURE is not enabled, this will result in a regular subdivision of the process space in <number>-sized blocks depending on the rank in `MPI_COMM_WORLD`, the local events for all the processes from one block ending up in a single process file. If STF-USE-HW-STRUCTURE is set, this can produce more than one process file per SMP node. The default value is 16.
- STF-CHUNKSIZE <bytes> determines at which intervals the anchors are set. The default value is 64 kilobytes. Setting a smaller value will increase the performance of Vampir, but it will use up more disk space. Increasing the number will likewise save disk space, but may result in a slower Vampir operation.

3.3.1.4 Handling STF files with stftool

To translate between the new STF format to the traditional BVT or AVT formats, the stftool command line converter is available in the Vampirtrace package. To just convert between STF and binary format, simply type:

```
stftool <input file> --logfile-name <output file> --logfile-format BINARY
```

In the line above `<input file>` is the STF file, and `<output file>` is the BVT (or AVT) file. To convert to a human-readable traditional format, specify ASCII instead of BINARY.

The stftool tool can do much more: it checks the integrity of a structured trace file, and can extract all the various bits and pieces of information contained in it. A brief description of its usage can be displayed by simply running it without any arguments, and detailed information about the many options is available in the stftool.1 man page. Five command-line options are described here for stftool, because the reader may find them useful in regard to stftool and also the utility xstftool, which is documented below.

--convert <filename>

Converts the entire file into the file format specified with –logfile-format or the filename suffix. Options that normally select a subset of the trace data are ignored when this low-level conversion is done. Without this flag writing is restricted to ASCII and BINARY format, while this flag can also be used to copy any kind of STF trace.

Converting from normal stf to compressed single-file STF is done like this:

```
  stftool <trace>.stf --logfile-format STFSINGLE --convert - |
  gzip -c > <trace>.stf.single.gz
```

--logfile-format [ASCII—BINARY—STF—STFSINGLE]

Specifies the format of the trace file. ASCII and BINARY are the traditional Vampir file formats where all trace data is written into one file. ASCII is human-readable, whereas BINARY is a more compact machine-readable format.

--matched-vtf When converting from STF to ASCII-VTF communication records are usually split up into conventional VTF (Vampir Trace Format) records. If this option is enabled, an extended format is written, which puts all information about the communication into a single line, but this is an extension of the format and therefore not understood by Vampir.

--move <file/dirname>

Moves the given file without doing other changes to it. The target can be a directory.

--request '<type>', <thread triplets>, <categories>,<duration>, <window>

This option has the same arguments as the --frame option below, but in contrast to defining a new frame, it restricts the data that is written into the new trace to that which matches the arguments. This option can be used more than once and then data matching any request is written.

--print-statistics/frames/thumbnails All of the --print options extract some of the meta information and print it in a human-readable form.

3.3.1.5 The xstftool Utility

There may be occasions when a user may wish to organize the trace data from Vampirtrace into an ASCII format for possible viewing with a spreadsheet browser or analyze the data in a script. The utility called xstftool has been developed for such purposes. The tool will do a translation from STF format into ASCII text that extends the stftool's conversion to Vampir ASCII format. The xstftool is part of the Vampirtrace package and more detailed documentation is found there.

3.3.2 File Menu

The file menu in Vampir is used to open a tracefile and select a configuration file and the following Sections describe these features. For this introduction the important elements of Vampir are introduced by using the trace file `redblack_sndrcv.stf` from the Vampir distribution. This trace file is found in:

```
$PAL_ROOT/examples/redblack_sndrcv.stf
```

3.3.3 Opening a File

There are several methods by which a trace file may be opened in Vampir. The first method is used if the trace file name is known and is entered in command line mode as in

```
vampir redblack_sndrcv.stf &
```

The suffix .stf stands for the new Structured Trace Format. The traditional Vampir formats use the suffix .bvt for a binary trace file, or .avt for an ASCII trace file. It is recommended that these extensions be used to identify file types when Vampir is used.

The second method of opening a trace file is to enter Vampir in the command line without any parameters as in

```
vampir &
```

After Vampir has completed start up the main window shown in 3.1 is displayed. If it is not already open, the trace file is opened by selecting the menu option **File** and pressing the **Open Tracefile** button shown in the **File** menu.

A Motif-style file selector drops down and, from this selector, navigate to the trace file and press the *OK* button. Alternatively, to re-open a tracefile from an earlier session, it often is easier to open the RECENT TRACEFILE dialog reachable via **File/Recent Tracefiles**. This last choice will list all trace files examined by Vampir in previous sessions. Selecting the file `redblack_sndrcv.stf` from the list and pressing the *OK* button will open the file for visualization.

Compressed tracefiles are identified by their suffix and are uncompressed transparently. The required external uncompress programs have to be registered as external converters with the dialog reachable via **Preferences/Tracefile/External Converters**. The suffixes `.Z` and `.bz2` are registered by default and decompression of `.gz` files is already built into Vampir.

3.3.4 Loading Trace Data

As explained before, Vampir needs to load the event data before it can visualize it. For the traditional Vampir .avt and .bvt files, opening the file immediately starts loading it. The progress of loading the data is visualized in the status line below the menu bar of the main window where a progress bar shows the percentage value of analysed events as well as the absolute number of scanned events. If the trace file is small the progress bar may appear only briefly, whereas if the file is large more time is needed for completion of loading. The load operation may be interrupted at any point by pressing the pause button on the progress panel and Vampir opens a display for the portion of the trace file that has been processed up to that point.

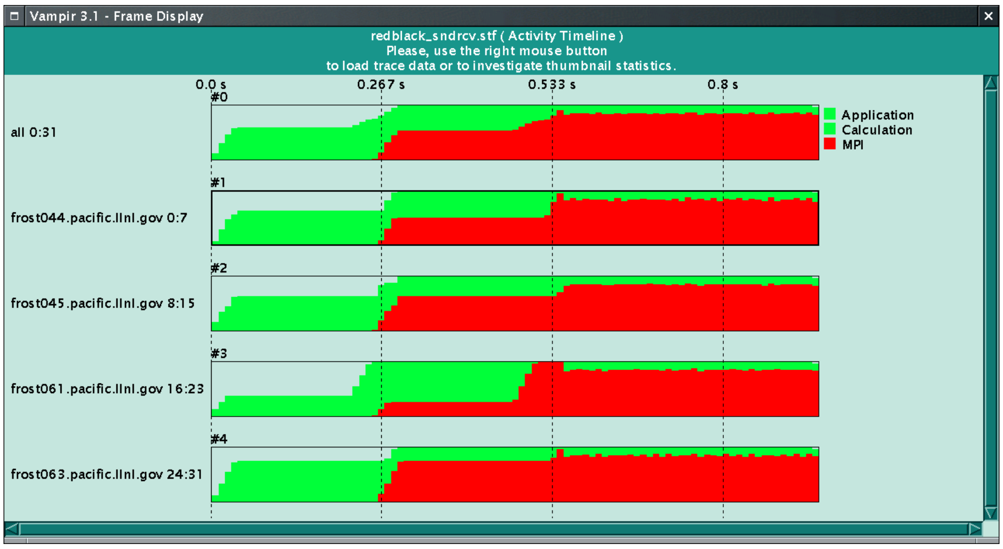
Loading trace data from STF files is done in the FRAME DISPLAY and therefore described below in Section 3.3.5.1.

3.3.5 FRAME DISPLAY

For STF, loading of data is only triggered automatically if no frame information is available. Otherwise the FRAME DISPLAY is opened and the real event data can be loaded from there by opening the context menu with a right mouse click. There are options to load the whole trace, just the selected frames, and a certain time interval that is selected interactively in the FRAME DISPLAY.

As soon as loading starts the FRAME DISPLAY window is closed automatically because it is usually no longer needed. It can always be reopened with the **File/Frame Display** menu item in the main Vampir panel to load a different part of the trace.

In order to identify an interesting part of the trace for loading, the FRAME DISPLAY shows frames in a two-dimensional grid aligned along the execution time (horizontal axis) and type (vertical axis). The type string of the frames generated automatically by Vampirtrace usually includes the node name and a triplet representing the process numbers that ran on this node, as seen in Figure 3.2.


Vampir 3.1 - Frame Display

redblack_sndrcv.stf (Activity Timeline)
Please, use the right mouse button
to load trace data or to investigate thumbnail statistics.

0.0 s
0.267 s
0.533 s
0.8 s
#0
all 0:31
#1
frost044.pacific.llnl.gov 0:7
#2
frost045.pacific.llnl.gov 8:15
#3
frost061.pacific.llnl.gov 16:23
#4
frost063.pacific.llnl.gov 24:31
Application
Calculation
MPI

Figure 3.2: FRAME DISPLAY

Frames can be selected by clicking on them, or by using the **Select All Frames** option from the context menu. Selected frames are indicated by a bold outline (see frame #1 in 3.2). The selection status of all frames can be cleared by the **Unselect All Frames** option of the context menu. The **Select/Deselect Frames** option requires you to draw a rectangle around some frames, and then toggles the selection status of the contained frames.

It is possible to select frames that are not adjacent in time and then load their data. However, this will lead to missing data in the middle and these gaps can have a negative influence on the analysis of the data. F.i. it is technically not feasible to accurately identify messages that occur in both parts of a trace, so Vampir might count the same message twice.

In each frame, a minuscule rendition of a preview thumbnail is shown. Currently, the following thumbnails are available:

Activity Timeline showing the distribution of execution time between activities, usually MPI versus user code.

Symbol Statistics showing min/average/max execution times for the routines contained in the frame.

Region Statistics showing min/average/max execution times for the OpenMP regions contained in the frame.

Sent Message Statistics showing a message matrix for messages sent within the frame.

The selection of a thumbnail is made via the **Display** entry of the context menu. However, only the default **Activity Timeline** is really suitable for usage in the FRAME DISPLAY. The others are better suited for detailed analysis of data in one frame with the THUMBNAIL DISPLAY.

3.3.5.1 Loading Data from the FRAME DISPLAY

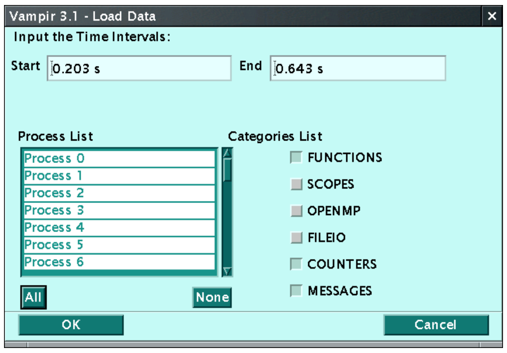
The **Load** context menu contains three different entries:

Whole Trace loads all data found in the trace file.

Selected Frames loads just the data covered by the currently selected frames, or the whole trace if no frames are selected.

Time Interval is used to select data interactively, independent of predefined frames.

After selecting **Time Interval** the cursor changes into a cross-hair and one can select a time interval by dragging the mouse over the FRAME DISPLAY while the left mouse button is pressed. When the button is released, the dialog from Figure 3.3 pops up. Its values default to the time interval that was just selected and all processes and data categories in the trace file enabled, but this can be changed. Sometimes it is useful to e.g. deselect *COLLOPS* (aka collective operations), because that can speed up loading and redrawing of timeline displays considerably.


Vampir 3.1 - Load Data

Input the Time Intervals:

Start: 0.203 s
End: 0.643 s

Process List:
- Process 0
- Process 1
- Process 2
- Process 3
- Process 4
- Process 5
- Process 6

Categories List:
- FUNCTIONS
- SCOPES
- OPENMP
- FILEIO
- COUNTERS
- MESSAGES

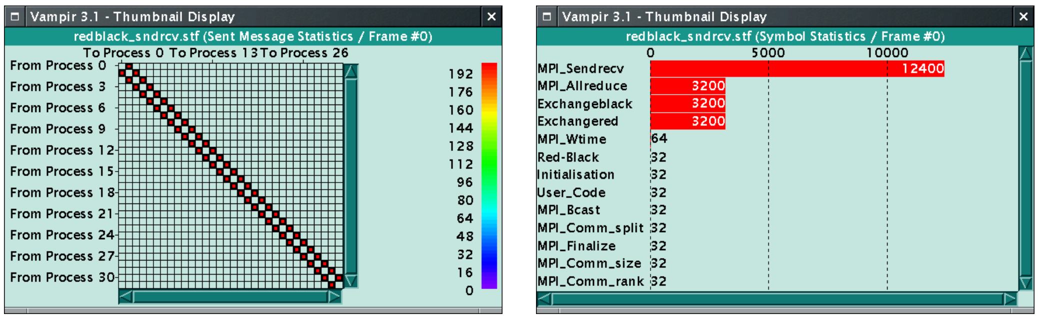
Figure 3.3: loading arbitrary data from the FRAME DISPLAY

As soon as *OK* is pressed, loading of the selected data starts. When loading of trace data commences (regardless of how it was started), the FRAME DISPLAY is closed automatically because usually it is no longer needed after switching to the more detailed analysis of the original trace data. It can be reopened e.g. to load a different subset of the trace with the **File/Frame Display** menu entry of the main panel.

3.3.5.2 Using the THUMBNAIL DISPLAY

The minuscule thumbnail renditions in the frame display are only useful to select the frames that warrant further inspection. For each frame, a large thumbnail display can be opened that shows the data for the frame, and which can be switched to show any kind of thumbnail information present. Generally, the thumbnail displays have been derived from existing Vampir statistics displays, with the exception of the combined min/average/max statistics for symbols and regions, which do not yet appear in other Vampir displays.

To open a full size thumbnail display, use the **Open Thumbnail** entry of the context menu, move the crosshair cursor over the frame to be analyzed, and click on the left mouse button. In the example, opening up Thumbnail Displays for frame #0 will produce displays like those shown in Figure 3.4.



(a) message count (b) symbol count

Figure 3.4: THUMBNAIL DISPLAY

Within each of the THUMBNAIL DISPLAYS, use the **Display** entry of the context menu to select different kinds of thumbnail data. The default is to show the same kind as in the FRAME DISPLAY.

3.3.6 TRACEFILE INFO Dialog

The TRACEFILE INFO dialog displays the tracefile info record(s) of the loaded trace file and is invoked with the **File/Tracefile** Info menu entry. The information shown includes the event count, the version number of the tracefile format, and the creator record(s).

3.3.7 STATISTICS VIEWER

This Section will discuss the *Statistical Analysis* methodology for performance analysis with Vampir and Vampirtrace. It is based on doing a cursory analysis of application performance using the statistics capabilities of Vampirtrace, and then using the statistical data to direct what additional types of instrumentation analysis should be done to achieve improved MPI and/or OpenMP execution performance.

In reference to the trace output file, MPI and/or OpenMP applications can generate so much performance data that the trace output can grow prohibitively large. Therefore, this section will focus on the use of statistic data to help guide the instrumentation process so that the user may focus on execution bottlenecks in terms of providing performance tuning for an application.

3.3.7.1 Controlling the Gathering of Statistics and Tracing Information

The Vampirtrace documentation describes how to set configuration options before running an application. To enable statistics gathering, this option must be enabled in a configuration file:

```
STATISTICS ON
```

Also, to disable the recording of an event trace, include another directive:

```
PROCESS 0:N OFF
```

Both options can also be set as environment variables; in Bourne shell syntax:

```
VT_STATISTICS=ON
VT_PROCESS="0:N OFF"
export VT_STATISTICS VT_PROCESS
```

A tracefile will be generated, but it will only contain definition and environment records and will be very short. To use the STATISTICS VIEWER within Vampir, do the following:

1. Start Vampir without specifying a trace file.
2. Select the menu item **File/Statistics Viewer** from the main panel.
3. Press **Load Statistics** from the context menu.
4. Select the tracefile that has the suffix .stf or .stf.single.

Alternatively, one can start Vampir and tell it to go to the statistics viewer for a given trace directly:

```
vampir --statistics-viewer <STF trace>
```

If you are already using a structured trace file in Vampir, this trace file is automatically chosen if you select **File/Statistics Viewer** in the main menu.

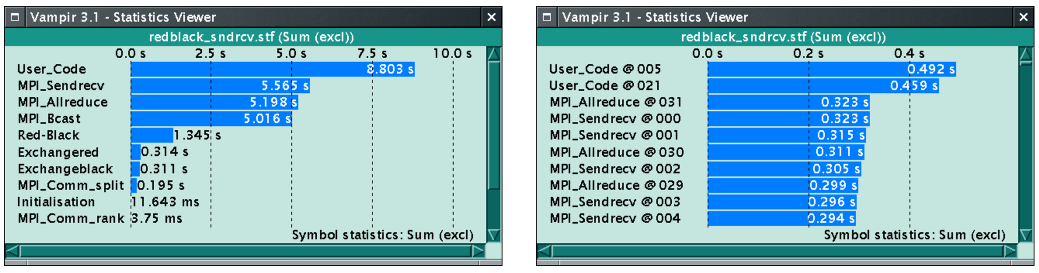
The STATISTICS VIEWER can view symbol statistics, region statistics, and message statistics. With these categories of statistics, users can ascertain a cursory analysis of how well an executing application is performing.

3.3.7.2 Symbol Statistics

After the STF file has been loaded, the statistics display will show a histogram of symbol (aka function call) statistics as depicted in Figure 3.5(a). For each symbol occurring in the tracefile, an entry is associated with the value of the chosen execution metric, and with a histogram bar visualizing that value. With the **Display** entry of the context menu, one of the following metrics can be selected:

- Number of calls (Occurrences)
- Minimum, maximum and aggregated execution time (excluding time spent in called routines)
- Minimum, maximum and aggregated execution time (including time spent in called routines)
- Average execution time, both including and excluding time spent in called routines

The statistics viewer indicates the chosen metric in the lower right corner. Using the **Select** entry of the context menu, it is possible to specify if the display should show all entries in the trace file separately (that means a single histogram bar for each process) or cumulated values for each symbol (which is the default selection) or for each activity. If **Select/All Entries** was chosen


Vampir 3.1 - Statistics Viewer
redblack_sndrcv.stf (Sum (excl))
0.0 s 2.5 s 5.0 s 7.5 s 10.0 s
User_Code 8.803 s
MPI_Sendrecv 5.565 s
MPI_Allreduce 5.198 s
MPI_Bcast 5.016 s
Red-Black 1.345 s
Exchangered 0.314 s
Exchangeblack 0.311 s
MPI_Comm_split 0.195 s
Initialisation 11.643 ms
MPI_Comm_rank 3.75 ms
Symbol statistics: Sum (excl)
Vampir 3.1 - Statistics Viewer
redblack_sndrcv.stf (Sum (excl))
0.0 s 0.2 s 0.4 s
User_Code @ 005 0.492 s
User_Code @ 021 0.459 s
MPI_Allreduce @ 031 0.323 s
MPI_Sendrecv @ 000 0.323 s
MPI_Sendrecv @ 001 0.315 s
MPI_Allreduce @ 030 0.311 s
MPI_Sendrecv @ 002 0.305 s
MPI_Allreduce @ 029 0.299 s
MPI_Sendrecv @ 003 0.296 s
MPI_Sendrecv @ 004 0.294 s
Symbol statistics: Sum (excl)

(a) aggregated by symbol (default) (b) for individual processes

Figure 3.5: STATISTICS VIEWER for symbol statistics

the entries have the form `<symbol name> @ <process rank>` indicating the process they belong to. In Figure 3.5(b), it is easy enough to compare the time spent in the `MPI_Recv` routine across all processes. Using the **Sort/Value Up** or **Sort/Value Down** entries of the context menu, the display will show symbol statistics sorted by value, like in Figure 3.5(b). Here, it becomes immediately clear that `MPI_Allreduce` consumes a lot of the execution time in processes with high rank numbers, while `MPI_Sendrcv` dominates in those with small ranks. Zooming in both the horizontal and vertical directions can be performed in the usual way by clicking the left mouse button and drawing a rectangle. Zoom undo also works in the usual way with the **Undo** or **Reset Zoom** entries of the context menu.

Use the **Mode** entry of the context menu to switch to a list-based display of one metric, or to a table that shows all available metrics. Besides the symbol statistics, execution metrics for OpenMP regions (if available in the trace) and MPI messages can be displayed. Use the **Category** entry of the context menu to switch between these three kinds of statistics.

3.3.7.3 Region Statistics

Figure 3.6 shows an OpenMP region statistics histogram for the sweep3d example trace file, as found in the ASCI 3.2.0.0 example archive. Only one OpenMP region has been recorded (r002), and the histogram clearly shows a severe load imbalance across processes for this region. Other metrics available via the **Display** context menu entry are:

- number of executions of a region (occurrences)
- minimum, maximum and aggregated execution time
- average execution time

3.3.7.4 Message Statistics

Figure 3.7 shows a message statistics matrix: for each pair of sending (on the Y axis) and receiving (on the X axis) processes, one of these metrics is displayed:

- number of messages
- minimum, maximum, or aggregated message length

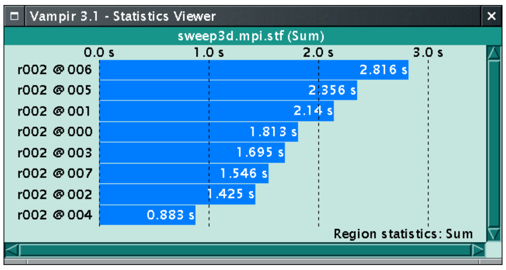


Figure 3.6: STATISTICS VIEWER for sweep3d region statistics

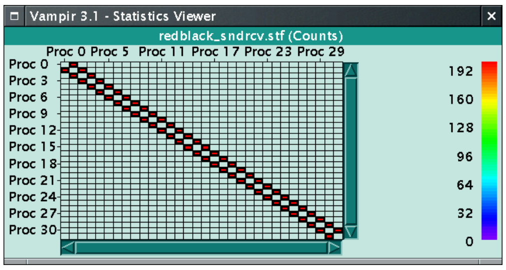


Figure 3.7: STATISTICS VIEWER for message statistics

3.3.7.5 Statistics in ASCII Format

The protocol file that is written together with each trace by Vampirtrace also contains the statistics mentioned above, but in a format that can be parsed by scripts. In addition to that, using `stftool <STF file> --print-statistics` on the command line will also print the statistics in the same format as in the protocol file, which can be useful if the protocol file got lost. The Vampirtrace documentation explains this ASCII format and how it can be parsed.

3.3.8 Configuration

Starting Vampir for the first time creates a configuration file called `VAMPIR3.cnf` resp. `VAMPIR2.cnf` in the directory `$HOME/.VAMPIR_defaults`, where `$HOME` references the home directory of the current user. This file contains all configuration settings for Vampir and enables Vampir to preserve settings from one session to the next. In order to restore the default settings, simply ensure that Vampir is not running, delete the `VAMPIR3.cnf` file, and restart Vampir.

Many settings (e.g. colouring) are file-specific, so it can be useful to associate them with a trace file instead of saving them as global defaults. This is done with so-called *local configs* that are created with the **Save Local** menu item. The latter action saves the current configuration together with the trace file, using the base name of the trace file and the suffix `.cfg`. The next time Vampir opens this trace file it will automatically use this local configuration and not the global default.

3.4 Global and Process Displays Menus

This section reviews some Vampir graphical displays. Displays are reached by drop down menus that open when the corresponding button on the main window panel (3.1) is pressed. All global displays are able to show data for more than one process, thus supporting either analysis of the whole program (e.g. SUMMARY CHART) or quick comparison of different processes (e.g. ACTIVITY CHART).

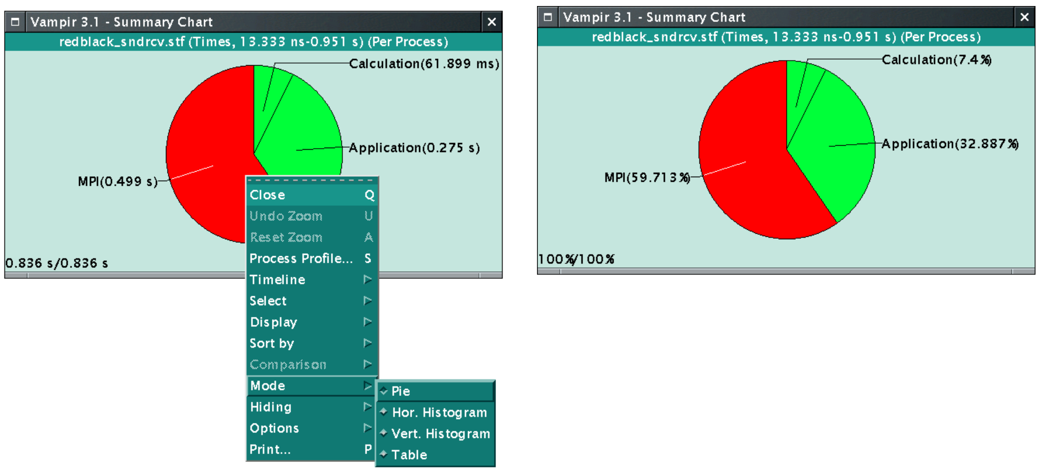
When the trace file contains many processes, the display of individual processes in a global display can become very dense and uninformative, or is aggregated in the first place. To help in this situation, Vampir can be used to open additional windows containing information for only one process at a time:

Vampir maintains a list of currently selected processes. If a process is selected, then it is drawn differently in the global displays. Its selection state can be modified there (and only there), as described below. As soon as (at least) one process is selected, then the menu items in the **Process Displays** drop-down menu shown in can be chosen. This action brings up process displays for all selected processes. Vampir keeps track of which displays have been opened already and does not open any of them twice.

3.4.1 Summary Chart Display

Vampir opens with the SUMMARY CHART display as the default. This display shows the sum of the time consumed by all instrumented activities over all selected processes and is analogous to the information displayed by conventional profilers. This display may also be opened by selecting **Global Displays/Summary Chart** from the main menu. An example of this default display for the trace file `redblack_sndrcv` was shown in 2.1(a).

By default, the SUMMARY CHART display shows a horizontal bar chart of those activities that occur in the time interval displayed in the window's top panel. However this may be changed by use of the context menu. This opens with a right mouse click inside the summary chart display window. The **Mode** option offers a pie chart (as seen in 3.8(a)), vertical bar chart, or tabular representations of the same information, in addition to the horizontal bar chart default shown in 2.1(a).



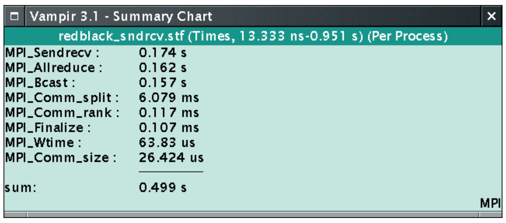
(a) absolute times

(b) relative times

Figure 3.8: SUMMARY CHART in pie mode

Various other options accessible from the context menu provide information on the symbols of any (or all) activities, a switch from exclusive to inclusive profiling, customisation of the display style etc. The **Comparison** entry of the context menu allows for a comparison of the summary profiling information from different trace files as is described in Section 3.7. The **Options/Per Process** option in the context menu changes the display to show the average execution time per process (which is also the default). One way to use the context menu here is to change the appended values in 3.8(a) from absolute times to percentages. To show the relative times (in percent), as in 3.8(b), choose **Options/Absolute Scale** from the context menu and deselect absolute values.

The displays in 2.1 show the aggregate results for MPI and the Application. Details for MPI procedures may be seen by choosing **Select/MPI** from the context menu. If a tabular presentation is preferred then also choosing **Mode/Table** from the context menu shows the display in 3.9.



Vampir 3.1 - Summary Chart

redblack_sndrcv.stf (Times, 13.333 ns-0.951 s) (Per Process)

MPI_Sendrecv :	0.174 s
MPI_Allreduce :	0.162 s
MPI_Bcast :	0.157 s
MPI_Comm_split :	6.079 ms
MPI_Comm_rank :	0.117 ms
MPI_Finalize :	0.107 ms
MPI_Wtime :	63.83 us
MPI_Comm_size :	26.424 us
sum:	0.499 s

MPI

Figure 3.9: SUMMARY CHART display for MPI procedures only in tabular form.

3.4.2 Timeline Displays

This is the timeline display available by selecting **Global Displays/Timeline** and shows all analysed state changes for each process over the complete period of time in one display as in 3.10.

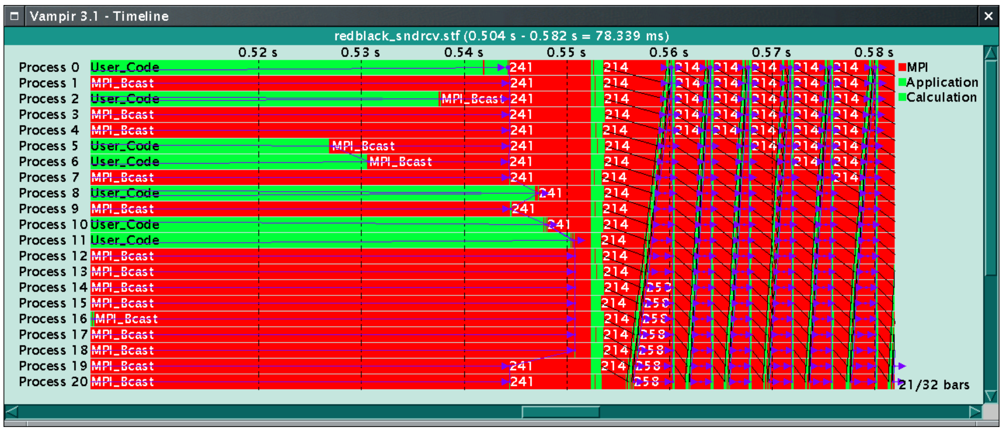


Figure 3.10: GLOBAL TIMELINE display, zoomed

In this timeline display predefined activities show up as different colors by default: red is used for MPI, green for user application subroutine events, and purple is used for MPI collective operations. However, the default colors may be changed as described in Section 3.6.1. The purple lines tend to dominate displays and may be turned off by deselecting **Components/Collective Operations** from the context menu[1].

[1]Because loading their data can cost quite some time, it might make sense to load the trace with the FRAME DISPLAY's **Load/Time Interval** menu: in the dialog that pops up after selecting a time interval one can suppress loading of certain data categories, among them *Collective Operations*.

In the example of 3.10 the process label and numbering is shown on a vertical scale on the left side. For the time series the tick marks are shown horizontally at the top of the drawing area and, on the right side, the legend for colors and activities appears. Messages between processes are shown as black lines that often appear as solid black message passing clusters in the display.

The default timeline display for the whole execution interval is often of too coarse a granularity for the runtime behaviour of the traced program. To see a finer granularity in the timeline display use the zooming feature. The scrollbar at the bottom and the cursor keys allow for ease of panning in the horizontal direction of the timeline.

This zooming method can be repeated to an unlimited depth and will show the names of symbols or at least their number when a sufficient depth is reached. The reverse action, undo zoom, is invoked by the timeline context menu choice **Undo Zoom**. To return to the original timeline trace from any level of zooming, use the **Window/Adapt Zoom** entry of the context menu.

The timeline display exists in either a global (all processes shown as above), or an individual version. To see the display for the individual version of a process use the left mouse button to select it by clicking on its label. Several processes can be selected or deselected by dragging the mouse over their labels. Then, navigate to the main menu selection **Process Displays/Timeline** to open a display similar to that shown in 3.11. Note that as in this example the initial time interval of the PROCESS TIMELINE is the same as the current one in the GLOBAL TIMELINE, but apart from that it is completely independent of the global displays.

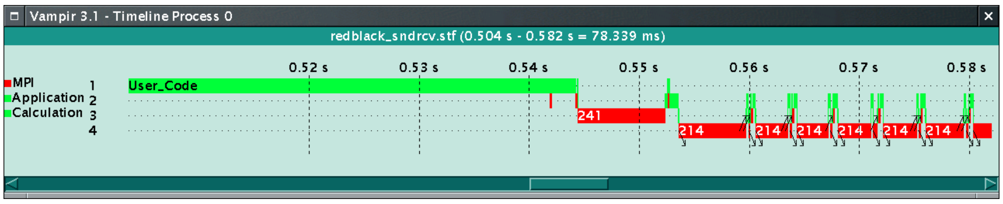


Figure 3.11: TIMELINE display for a single process.

In 3.11 the vertical window scale is used to display different states at different heights. One way to envision this is to think of this display as a function that has time as its input parameter and the system state as the function result. Separating the state display in the vertical direction has another advantage: short time durations of one state cannot be masked by long time periods of another state. For example, if a state existed for a few nanoseconds (less than one pixel on the horizontal scale), it would still be visible in this display.

The ruler function is available from the context menu (reachable with the right mouse button) and may be used to measure the exact lengths of time periods. Select **Ruler** from the context menu and a cross-hair appears for the mouse cursor. Then press the left mouse button and drag the cursor symbol over the desired time period and release the mouse button. The measured range is then shown in a message box.

3.4.3 SUMMARY TIMELINE Display

In addition to the timeline display described above another timeline is available by selecting **Global Displays/Summary Timeline** from the main panel to show a display similar to the one shown in 3.12.

The summary timeline shows all analysed state changes for each process (over the complete time interval in one display), as a histogram over time bins. Within each time bin the number of processors in either an MPI (red) or Application (green) state is shown. Just as in the case of

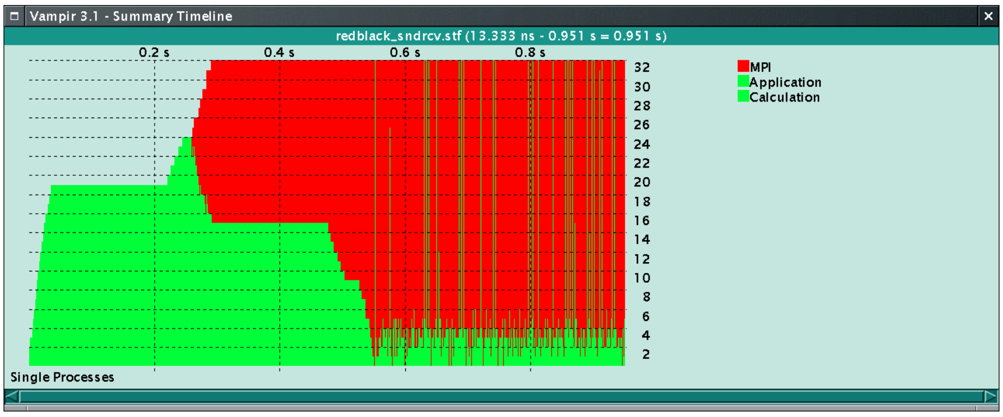


Figure 3.12: SUMMARY TIMELINE display

the global timeline, there is a context menu and zooming is possible to show magnified displays within subintervals. It is important to note that the rows in the display do not correlate to specific processes, as in the global timeline.

This display is also available (in a vertically condensed version) as an addendum along the bottom of the global timeline by choosing from its context menu **Components/Parallelism Display**. These displays are particularly useful in the study of communication overhead issues as a function of time since wider and higher green regions indicate a greater degree of parallelism.

3.4.4 Displaying Counter Data

If counter sample data is stored in the trace, then Vampir can display it both in the global COUNTER TIMELINE and the PROCESS TIMELINE. The difference is that the COUNTER TIMELINE as seen in 3.13 displays just one counter, but for all selected processes at once. By synchronizing the visible time interval with other displays the counter values are correlated with other events.

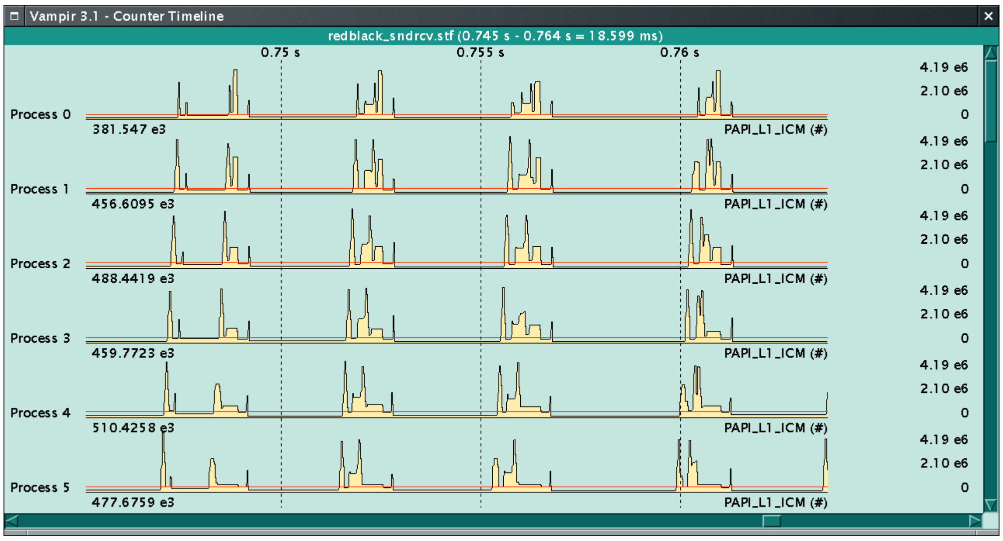


Figure 3.13: COUNTER TIMELINE display

The PROCESS TIMELINE on the other hand can display more than one counter and relates it directly with the function calls.

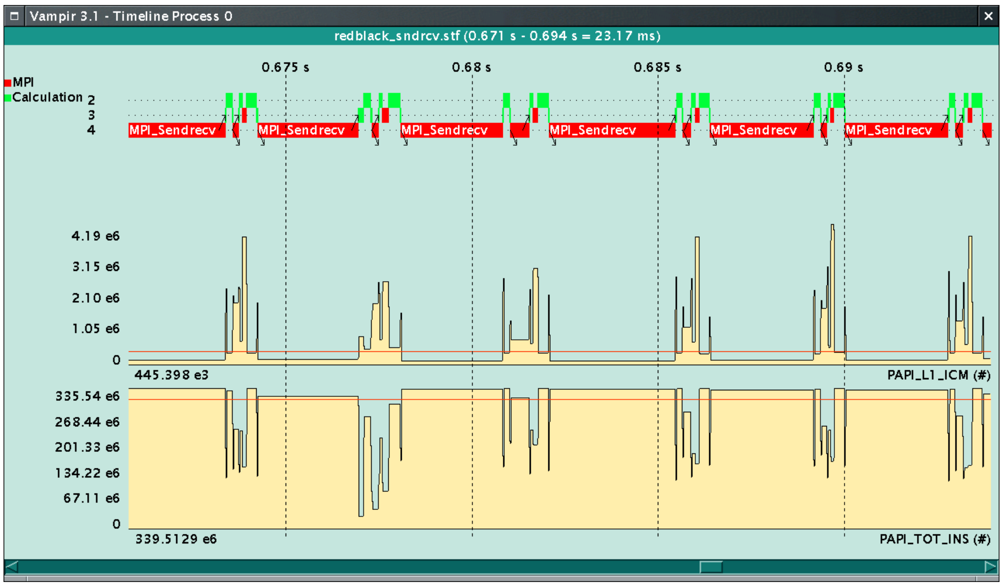


Figure 3.14: PROCESS TIMELINE display with counters

Both kinds of displays share the same kind of context menu options to control which and how counter data is displayed: the **Counters** context menu contains one submenu for each class of counters and inside these submenus one can select counters—exactly one in the COUNTER TIMELINE, more than one in the PROCESS TIMELINE.

Counter values can be used to store not just hardware performance counters, but arbitrary sample values. Vampirtrace uses them to provide information about its internal memory management. By enabling Vampirtrace's internal counters in a Vampirtrace VT_CONFIG file (see Vampirtrace documentation for details on how to use a configuration file), one can see how many bytes of trace data were kept in main memory (`data_in_ram`), in a flush file (`data_in_file`) and when flushing was active to transfer data from main memory into the flush file (`flush_active`). This is demonstrated in 3.15.

Depending on the definition of a counter, Vampir will interpret its samples differently. In 3.16 the values from 0 to 15 are logged at the same points in four counters with different definitions: for the first counter `before` a sample value is valid (and thus displayed) before the point in time where it was logged, for the second one (`after`) it is valid afterwards. In both cases the value does not change unless a new value is logged. In the third case, the value is only displayed exactly at the `point` in time where it is logged as a peak. Finally, values can be treated as samples of a contiguous curve that Vampir tries to approximate with linear interpolation of sample values.

Examples of how an programmer could use the Vampirtrace API to log information about his program run as counter data include logging values associated with events (like receiving messages of variable length, as the `msg` counter in 3.17) or the epsilon in an iterative approximation of the final result (`epsilon`). Note that the artificial message sizes were chosen so that they follow a sine curve with amplitude 1024 and period 0.1 seconds. They even become negative, which Vampir indicates by drawing them in red—not quite realistic for a message size, but aesthetically pleasing. . .

These sine values are also logged as a sampled curve (`sine` in 3.18(a)). Vampir is able to display the absolute sample values of a counter or the first derivative (aka *rate*), i.e. the difference

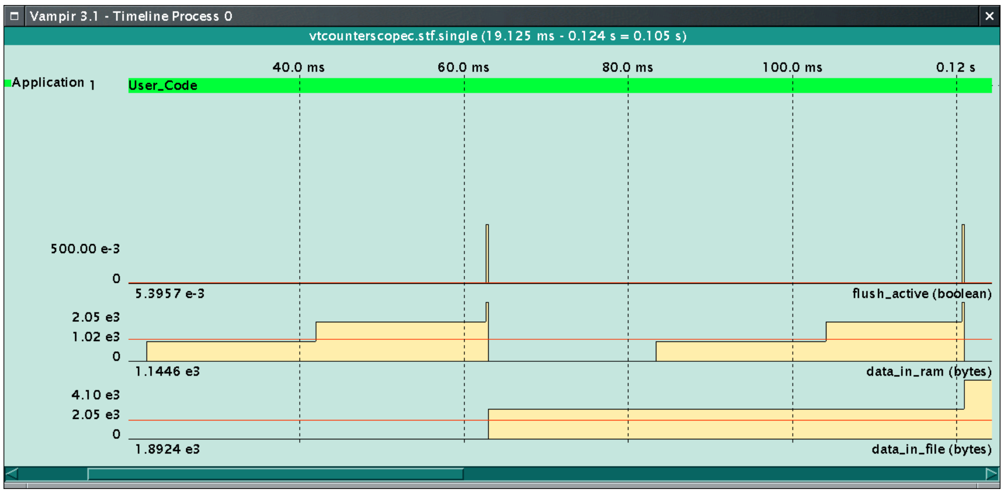


Figure 3.15: PROCESS TIMELINE display with Vampirtrace memory management counters

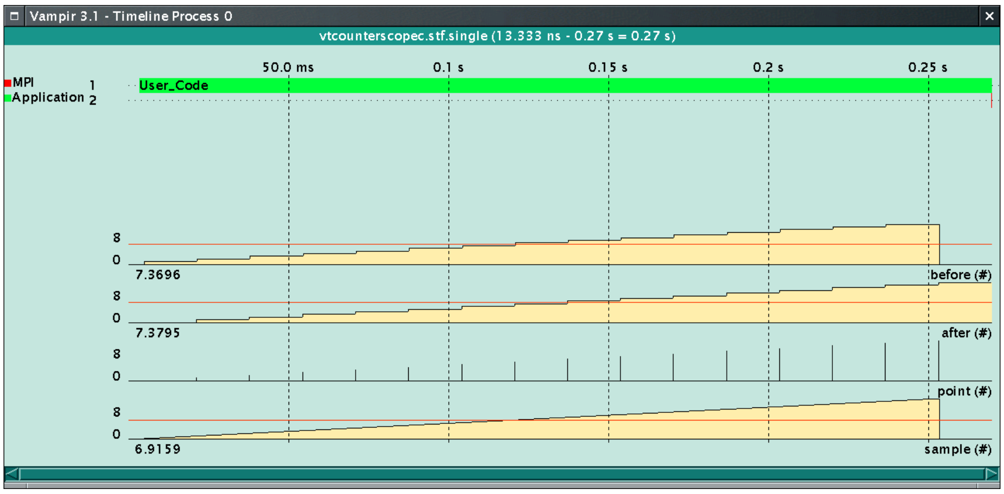


Figure 3.16: PROCESS TIMELINE display with counters of different scope

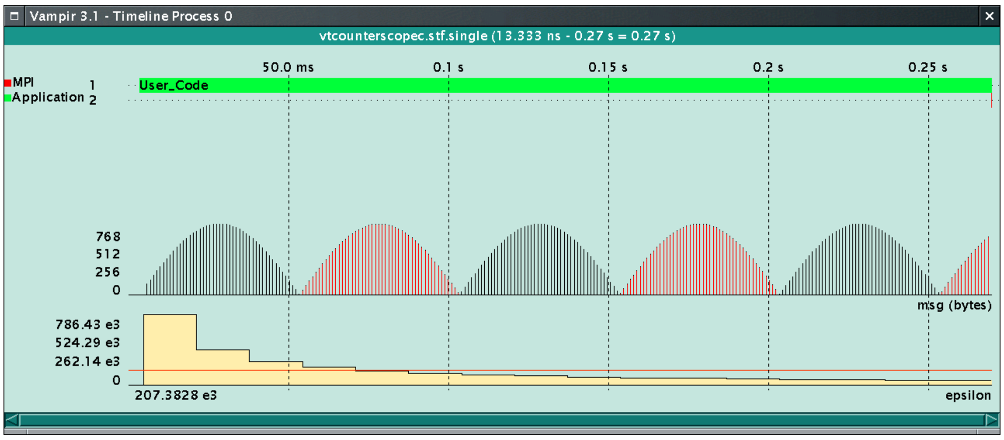
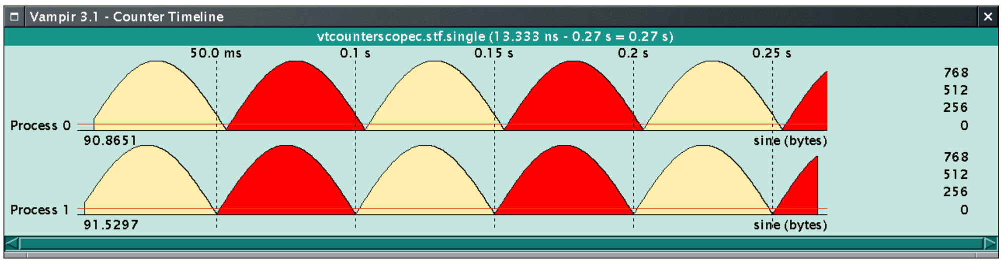
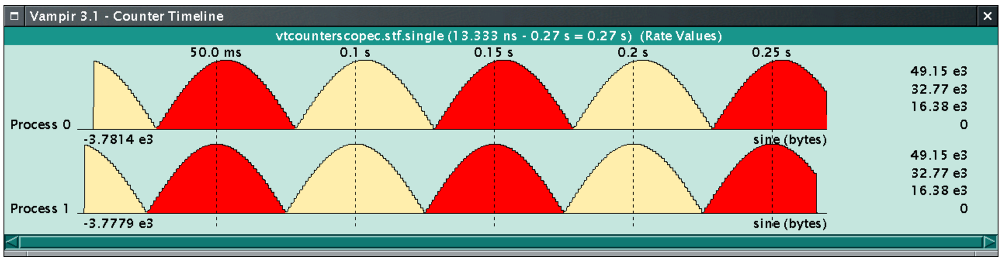


Figure 3.17: PROCESS TIMELINE display with (artificial) application counters

between adjacent sample values divided by the time delta between them. The first derivative is always displayed as a constant value between the original sample values. The default display mode depends on how the counter was defined. For hardware counters the default is to display the first derivative, because otherwise one just gets a continuously increasing curve with very little variation which is not very useful. Vampir is able to override the default presentation mode in the **Options/Differentiate Values** context menu. Not surprisingly, the first derivative of the sine counter is a cosine with the same period, but a different phase (3.18(b)).



(a) sine counter



(b) first derivative = cosine

Figure 3.18: COUNTER TIMELINE display

In addition to these sampled counter values in a trace file, Vampir can also calculate the amount of data currently in transit in the network at each point in time and makes this available as the counter *Global Message Volume*. In the SUMMARY TIMELINE it can also display the data in transit inside the currently selected groups (see 3.5.2 for details), as shown in 3.19. This counter is called *Cluster Message Volume* because usually it is used to analyse the behaviour of clustered nodes, as in this example.

3.4.5 MESSAGE STATISTICS Display

Message statistics may be displayed by selecting **Message Statistics** from the **Global Displays** drop down menu in the Vampir main panel shown in 3.1. The default display is shown in 3.20 for the cumulative length of all messages in pair-wise communications between processors. The left hand side scale gives the rank of the sending process and the top scale gives the rank of the receiving process. The right hand legend gives the color map for the total length of all messages. Note that zooming is allowed in this display and for a description of how to apply it see Section 3.2.5.

Information shown in this display is changed by selecting the **Display** button from the context menu. Then the quantity of interest is chosen from a list that includes: counts, length, rate, or

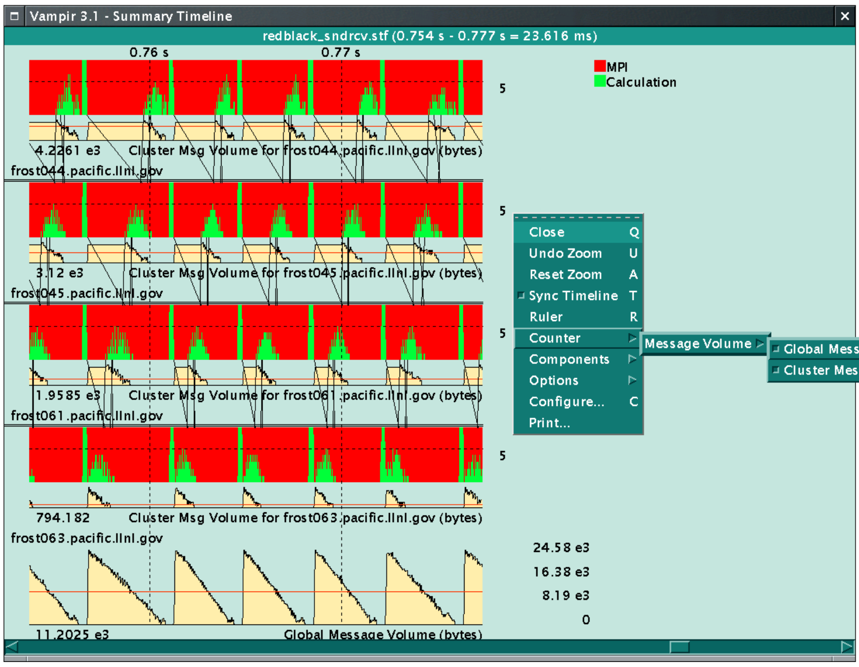
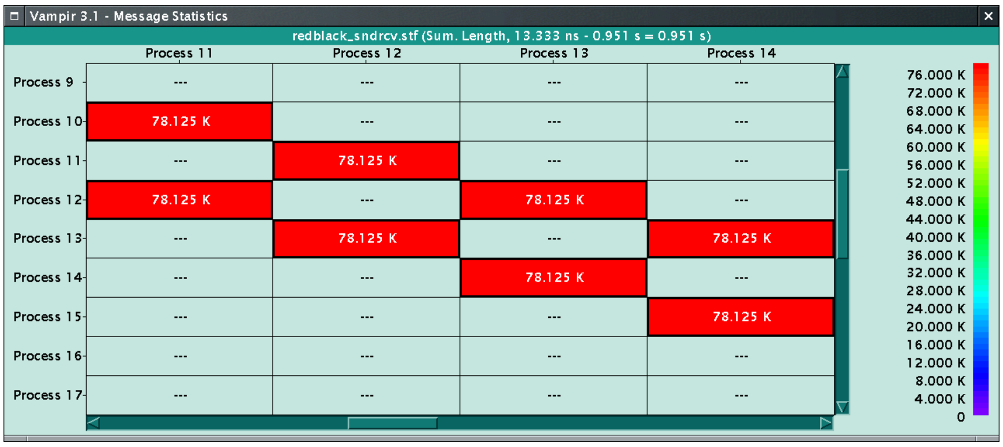

Vampir 3.1 - Summary Timeline
redblack_sndrcv.stf (0.754 s - 0.777 s = 23.616 ms)
0.76 s
0.77 s
MPI
Calculation
4.2261 e3
Cluster Msg Volume for frost044.pacific.llnl.gov (bytes)
frost044.pacific.llnl.gov
3.12 e3
Cluster Msg Volume for frost045.pacific.llnl.gov (bytes)
frost045.pacific.llnl.gov
1.9585 e3
Cluster Msg Volume for frost061.pacific.llnl.gov (bytes)
frost061.pacific.llnl.gov
794.182
Cluster Msg Volume for frost063.pacific.llnl.gov (bytes)
frost063.pacific.llnl.gov
11.2025 e3
Global Message Volume (bytes)
24.58 e3
16.38 e3
8.19 e3
0
Close Q
Undo Zoom U
Reset Zoom A
Sync Timeline T
Ruler R
Counter
Components
Options
Configure... C
Print...
Message Volume
Global Message Volume
Cluster Message Volume

Figure 3.19: SUMMARY TIMELINE and message volume


Vampir 3.1 - Message Statistics

redblack_sndrcv.stf (Sum. Length, 13.333 ns - 0.951 s = 0.951 s)

	Process 11	Process 12	Process 13	Process 14
Process 9	---	---	---	---
Process 10	78.125 K	---	---	---
Process 11	---	78.125 K	---	---
Process 12	78.125 K	---	78.125 K	---
Process 13	---	78.125 K	---	78.125 K
Process 14	---	---	78.125 K	---
Process 15	---	---	---	78.125 K
Process 16	---	---	---	---
Process 17	---	---	---	---

Figure 3.20: MESSAGE STATISTICS display, summarized length, zoomed

duration, for minimum, maximum, or average values. For example, 3.21 shows the results of the context menu selection **Display/Max. Duration**. In contrast to the message lengths which are the same for each pair of processes in this example, the message duration varies considerably. The minimum and maximum value are marked with a thicker border around the square—f.i. the highest duration is found for sending a message from process 0 to process 1.

In this example this is not caused by a slow communication path between these two processes, but rather due to the way the message duration is calculated: it is the time between starting transmission and successful reception of the message. If the recipient delays reception for some reason, then the duration is increased by this delay and the message rate, which is just the size of the message divided by the duration, decreases accordingly. As demonstrated in section 2.1, in `redblack_sndrcv` such a delay is caused by the communication pattern.

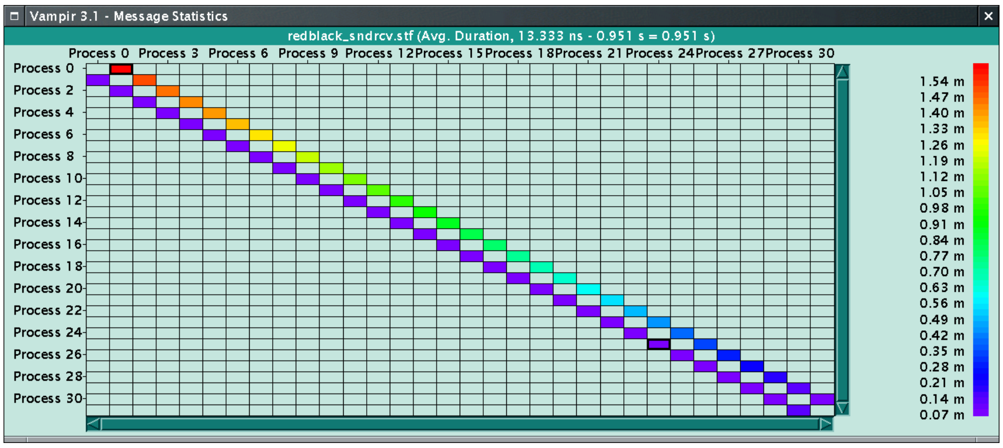
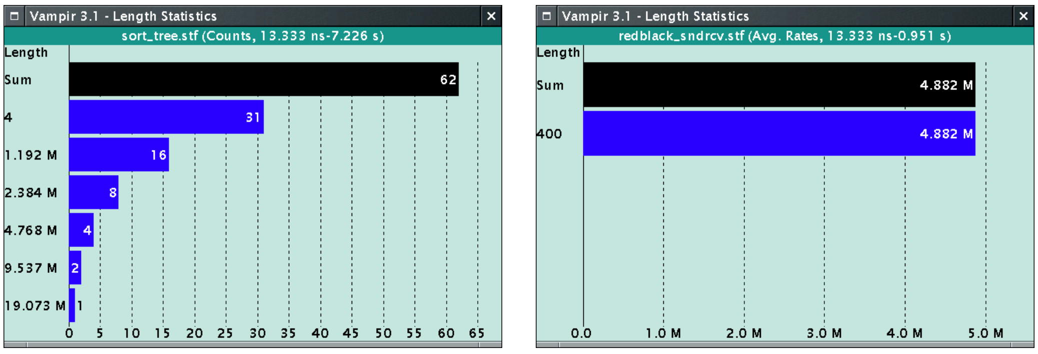


Figure 3.21: MESSAGE STATISTICS display, maximum duration

The LENGTH STATISTICS display is invoked by clicking on the **Length Statistic** entry of the context menu to show a LENGTH STATISTICS window, similar to those shown in 3.22, with a histogram of the distribution of message lengths. The **Length Statistics** context menu also allows the display of counts, length, or rates. For `redblack_sndrcv` all messages have the same length, so in addition to the summary there is only one entry. For `sort_tree` the message length increases by a factor of two at each level in the merge tree, while the number decreases by the same factor.

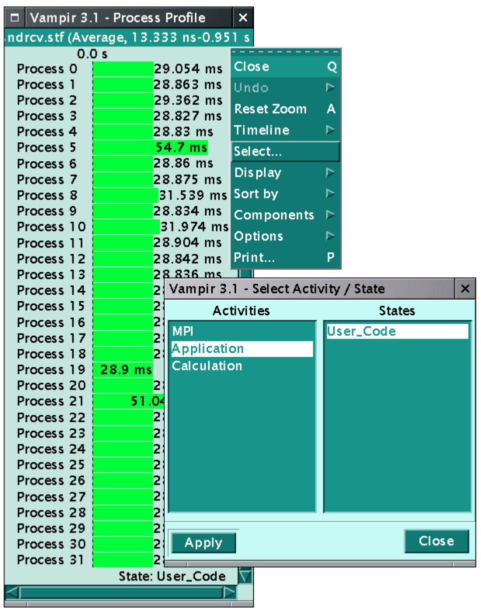

Vampir 3.1 - Length Statistics
sort_tree.stf (Counts, 13.333 ns-7.226 s)
Length
Sum
4
1.192 M
2.384 M
4.768 M
9.537 M
19.073 M
62
31
16
8
4
2
1
0
5
10
15
20
25
30
35
40
45
50
55
60
65
Vampir 3.1 - Length Statistics
redblack_sndrcv.stf (Avg. Rates, 13.333 ns-0.951 s)
Length
Sum
400
4.882 M
4.882 M
0.0
1.0 M
2.0 M
3.0 M
4.0 M
5.0 M

(a) sort_tree, message counts (b) redblack_sndrcv, average rates

Figure 3.22: LENGTH STATISTICS dialog

3.4.6 PROCESS PROFILE Display

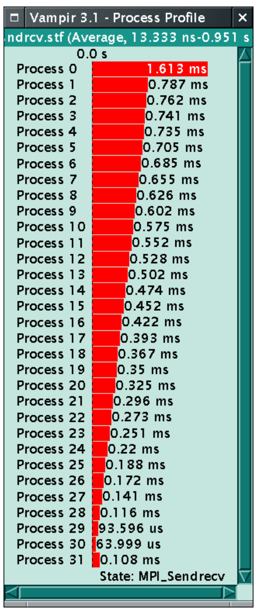
The selection of **Global Displays/Process Profile** from the main panel shows a display similar to the ones in 3.23. This display is similar to the ACTIVITY CHART in the sense that it compares values between different processes and could be used to detect imbalances. But in contrast to the ACTIVITY CHART it is possible to concentrate on a specific activity or state by choosing **Select** from the context menu to open a SELECT ACTIVITY/STATE dialog box, as seen in 3.23(a). From this dialog individual activities (typically MPI or Application) may be selected followed by depression of the *Apply* button. 3.23(a) shows the initial state, which is the *Activity User Application* and the *Symbol User Code*. Selecting `MPI_Sendrecv` resp. `MPI_Allreduce` in 3.23(b) and 3.23(c) confirms the observation that there is a severe imbalance. Not shown here, but also possible in the SELECT ACTIVITY/STATE dialog is the selection of just an activity like `MPI` instead of a specific state like `MPI:MPI_Sendrecv`.



```
Vampir 3.1 - Process Profile
ndrcv.stf (Average, 13.333 ns-0.951 s
          0.0 s
Process 0    29.054 ms
Process 1    28.863 ms
Process 2    29.362 ms
Process 3    28.827 ms
Process 4    28.83 ms
Process 5       54.7 ms
Process 6    28.86 ms
Process 7    28.875 ms
Process 8     31.539 ms
Process 9    28.834 ms
Process 10    31.974 ms
Process 11   28.904 ms
Process 12   28.842 ms
Process 13   28.836 ms
Process 14   2
Process 15   2
Process 16   2
Process 17   2
Process 18   2
Process 19  28.9 ms
Process 20   2
Process 21       51.04
Process 22   2
Process 23   2
Process 24   2
Process 25   2
Process 26   2
Process 27   2
Process 28   2
Process 29   2
Process 30   2
Process 31   2
                 State: User_Code
```

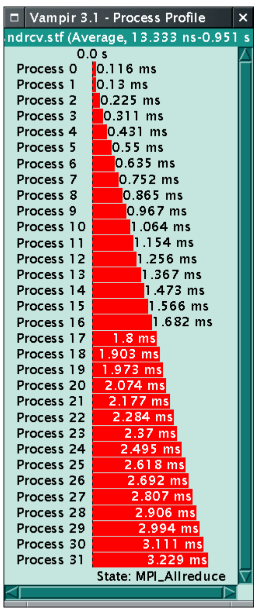
```
Vampir 3.1 - Select Activity / State
        Activities              States
  MPI                     User_Code
  Application
  Calculation
```

(a) default state and state selection



```
Vampir 3.1 - Process Profile
ndrcv.stf (Average, 13.333 ns-0.951 s
          0.0 s
Process 0          1.613 ms
Process 1    0.787 ms
Process 2    0.762 ms
Process 3    0.741 ms
Process 4    0.735 ms
Process 5    0.705 ms
Process 6    0.685 ms
Process 7    0.655 ms
Process 8    0.626 ms
Process 9    0.602 ms
Process 10   0.575 ms
Process 11   0.552 ms
Process 12   0.528 ms
Process 13   0.502 ms
Process 14   0.474 ms
Process 15   0.452 ms
Process 16   0.422 ms
Process 17   0.393 ms
Process 18   0.367 ms
Process 19   0.35 ms
Process 20   0.325 ms
Process 21   0.296 ms
Process 22   0.273 ms
Process 23   0.251 ms
Process 24   0.22 ms
Process 25   0.188 ms
Process 26   0.172 ms
Process 27   0.141 ms
Process 28   0.116 ms
Process 29   93.596 us
Process 30   63.999 us
Process 31   0.108 ms
               State: MPI_Sendrecv
```

(b) MPI_Sendrecv()



```
Vampir 3.1 - Process Profile
ndrcv.stf (Average, 13.333 ns-0.951 s
          0.0 s
Process 0    0.116 ms
Process 1    0.13 ms
Process 2    0.225 ms
Process 3    0.311 ms
Process 4    0.431 ms
Process 5    0.55 ms
Process 6    0.635 ms
Process 7    0.752 ms
Process 8    0.865 ms
Process 9    0.967 ms
Process 10   1.064 ms
Process 11   1.154 ms
Process 12   1.256 ms
Process 13   1.367 ms
Process 14   1.473 ms
Process 15   1.566 ms
Process 16   1.682 ms
Process 17     1.8 ms
Process 18   1.903 ms
Process 19    1.973 ms
Process 20    2.074 ms
Process 21     2.177 ms
Process 22     2.284 ms
Process 23       2.37 ms
Process 24      2.495 ms
Process 25       2.618 ms
Process 26       2.692 ms
Process 27        2.807 ms
Process 28        2.906 ms
Process 29         2.994 ms
Process 30          3.111 ms
Process 31           3.229 ms
              State: MPI_Allreduce
```

(c) MPI_Allreduce()

Figure 3.23: PROCESS PROFILE dialog

3.4.7 Statistic Displays

In addition to the small statistics field contained in the process state display described above, other Vampir statistical display types include:

- Global activity, and
- Individual activity.

The first of these can be reached by selecting the **Global Displays/Activity Chart** menu entry of the main panel, and a window will appear similar to the one shown in 3.24. This depicts the statistics of the complete trace file in a chart for each process subdivided by activity (e.g. MPI versus Application). This display is very useful for determining load balance issues for different processes at-a-glance. Different presentation modes can be selected in the context menu. There is a setting that determines the initial setting for this mode and other context menu options: **Preferences/Displays/Activity Charts**. Previous versions of Vampir ignored these default settings and chose the pie mode by default, now the global default of a bar chart is used.

To select processes for which individual statistics are desired, simply click on each, in turn, on the global display, with the left mouse button, and the corresponding process caption will change color. Then, from the main menu, select **Process Displays/Activity Chart** to show the activity

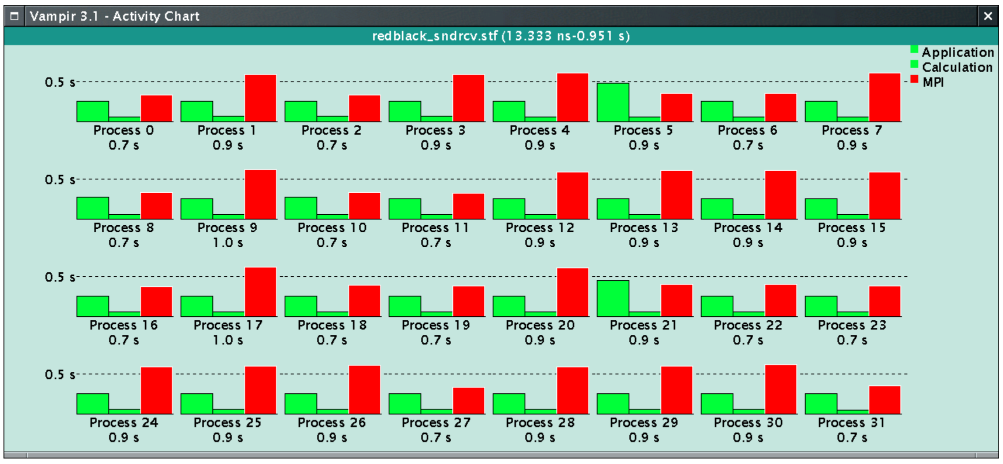
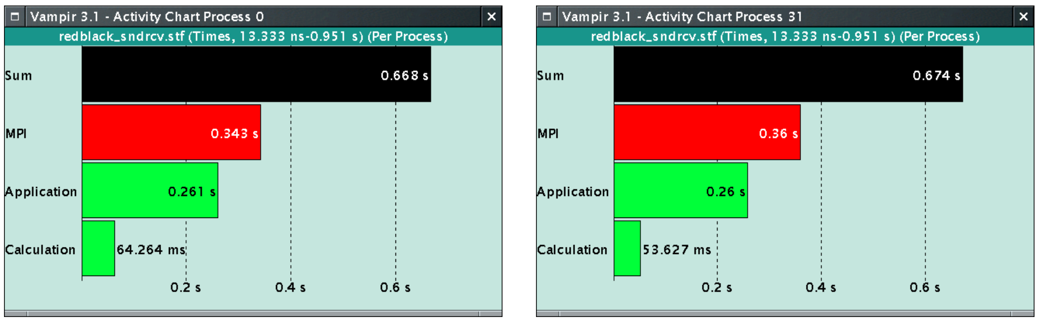


Figure 3.24: GLOBAL ACTIVITY CHART

charts for the selected local processes. The case of processes 0 and 31 are shown in 3.25. For a description of how this may also be done in the global display by filtering methods see Section 3.5.



(a) first process (b) last process

Figure 3.25: PROCESS ACTIVITY CHART dialog

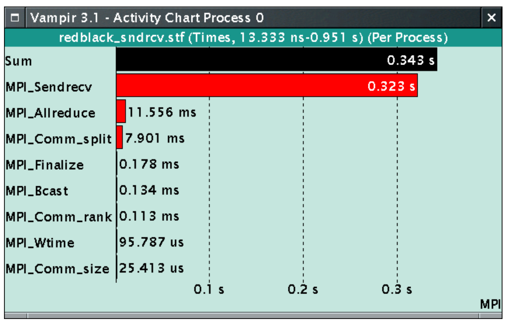
It is also possible to press the left mouse button and draw a rectangle over a group of processes to select them with one action, or, press the middle mouse button to deselect all processes. Then, to generate statistics window(s), select Process Displays/Activity Chart from the opening main menu panel. Vampir keeps track of the processes for which statistics windows are opened or closed and thereby avoids duplicate windows.

The usage and principles of operation for the global statistics display and the local statistics windows are similar, so only the local windows are described in the following paragraphs. By analogy, everything said about an local window is also valid for the corresponding statistics display window.

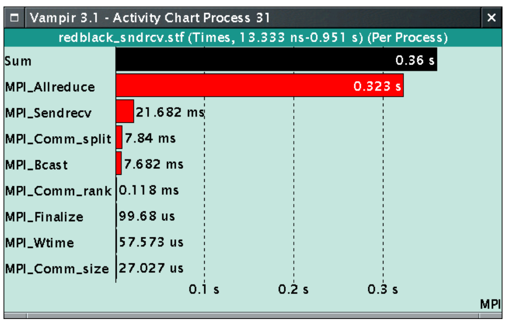
The local statistics look much like the global statistics with the difference that the activity names are directly displayed on the corresponding pie sectors (however, only for portions that are not too small). Color is used to indicate the activity of a symbol. For the global display this provides a better overview, but exact identification of entries in individual processes is only possible with the **Identify State** context menu entry. Selecting it changes the cursor into a cross-hair and then clicking on an entry pops up a dialog with more detailed information. The behaviour of previous

Vampir versions which assigned random colors to each entry of the global ACTIVITY CHART is still available by selecting its context menu entry **Options/Use Random Colors**, but it is no longer the default because the colors often just make the display confusing.

To show statistics for all MPI events on a process bring up the context menu and then choose **Select/MPI** from it. The results shown are similar to 3.26. Due to the imbalance either the `MPI_Sendrecv` or the `MPI_Allreduce` symbol uses the most significant amount of the time in both cases and the remaining symbols are not easy to see in this display because they are relatively small on this scale.



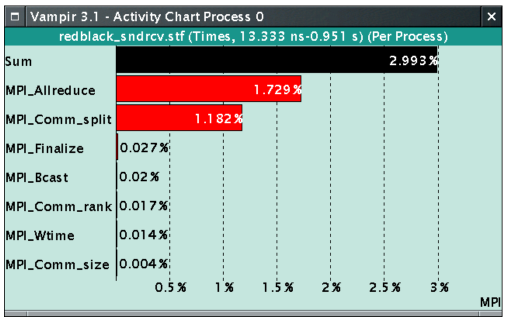
(a) first process



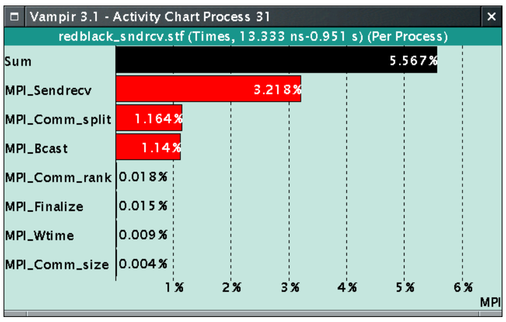
(b) last process

Figure 3.26: PROCESS ACTIVITY CHART dialog for MPI only

However, information on the remaining symbols (other than `MPI_Sendrecv` resp. `MPI_Allreduce`) can be viewed by hiding those that used the largest amount of time. For example, from the context menu (accessible via the right mouse button), selecting the option **Hiding/Hide Max** removes from the statistics display the symbol with the maximal time portion. The result of this operation, in each of the displays in 3.26, is shown in 3.27 after deselecting **Options/Absolute Scale** from the respective context menus. In the global chart display, if use of the **Hide/Max** option is followed by a left mouse click on one process, then the activity representing the largest portion of the time for that process will be removed from all charts. Note that the mouse cursor changes to a small eye symbol to suggest click action is pending for process selection.



(a) first process



(b) last process

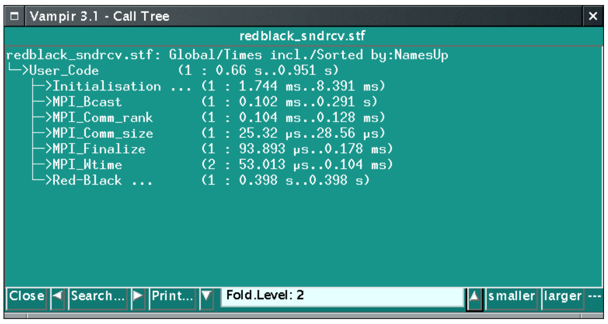
Figure 3.27: PROCESS ACTIVITY CHART dialog after hiding maximum once, with relative scale

The process of removing the symbol with the maximum time portion from the statistics display may be repeated for the next symbol in the time portion hierarchy. If the maximum time portion symbol is already hidden then selecting **Hiding/Hide Max** again, hides the symbol with the next largest time portion, etc. Generally speaking, multiple use of this context menu item will hide more and more symbols until finally the chart is entirely empty. The hiding process may be undone altogether by selecting **Hiding/Reset Hiding** from the context menu to return to the original display. Alternatively, the hiding may be undone incrementally (one step at a time) by selecting **Hiding/Undo Hiding** from the context menu. Another way to view symbols with large and small time portions more easily, side-by-side, is to switch on a logarithmic horizontal scale by selecting **Options/Logarithmic** from the context menu.

3.4.8 CALL TREE Displays

The global and local CALL TREE displays show the dynamic calling tree of all instrumented routines, either locally (for one selected process), or globally (for all selected processes combined). These displays depend on proper instrumentation of the application, and on system-dependent support for automatic subroutine instrumentation.

In addition to the caller-callee relationship, the displays present the number of calls and the time spent in each particular routine. An example for a global calling tree is shown in 3.28.



```
redblack_sndrcv.stf: Global/Times incl./Sorted by:NamesUp
└─>User_Code           (1 : 0.66 s..0.951 s)
   ├─>Initialisation ... (1 : 1.744 ms..8.391 ms)
   ├─>MPI_Bcast          (1 : 0.102 ms..0.291 s)
   ├─>MPI_Comm_rank      (1 : 0.104 ms..0.128 ms)
   ├─>MPI_Comm_size      (1 : 25.32 µs..28.56 µs)
   ├─>MPI_Finalize       (1 : 93.893 µs..0.178 ms)
   ├─>MPI_Wtime          (2 : 53.013 µs..0.104 ms)
   └─>Red-Black ...      (1 : 0.398 s..0.398 s)
```

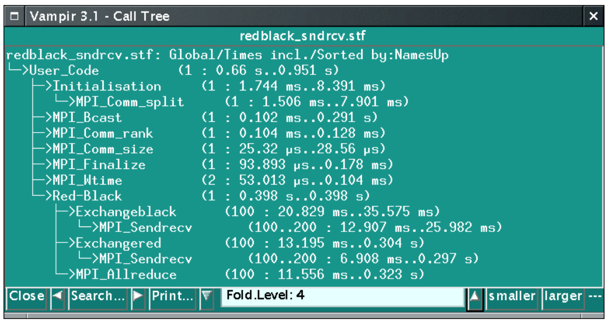
Figure 3.28: GLOBAL CALL TREE display (default)

The global call tree is invoked with the **Global Displays/Call Tree** menu entry. If processes are selected in a TIMELINE or ACTIVITY CHART display, their local call tree displays can be opened with the **Global Displays/Call Tree** menu entry. The levels of the tree display is expanded by pressing the down-arrowhead to the left of the text window in the bottom of the display, to give the display in 3.29. Conversely, the up-arrowhead on the right hand side collapses the calling tree. It is also possible to double-click on an entry to fold or unfold it. The context menu for this display offers several options, such as a sort by name, counts or time with the **Sort** tab.

3.4.9 SOURCE VIEW Display

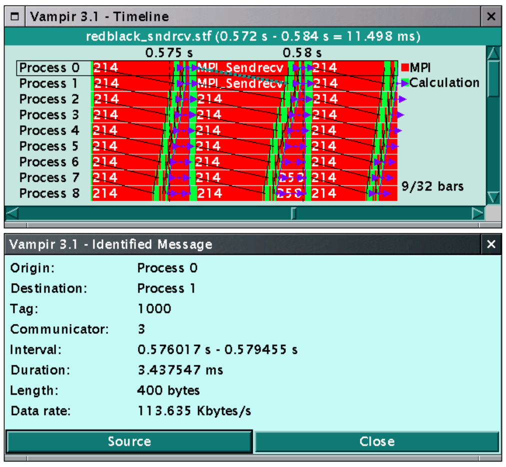
Vampir can relate a trace event or a symbol to a specific location in the source program, and display this source location from the TIMELINE, SUMMARY CHART, ACTIVITY CHART and PROCESS DISPLAY displays. This feature depends on the information present in the tracefile, and ultimately on the instrumentation of the application with Vampirtrace.

If the source location information is present in a tracefile, and if Vampir can access the source code itself, a window like Figure 3.30(b) will be opened when pressing the *Source* button in an IDENTIFY STATE or IDENTIFY MESSAGE dialog in a TIMELINE, SUMMARY CHART, or ACTIVITY CHART display. Vampir's **Preferences/Tracefile/Source** main panel menu item can be used to set a search path for source files which may contain one or more directories.



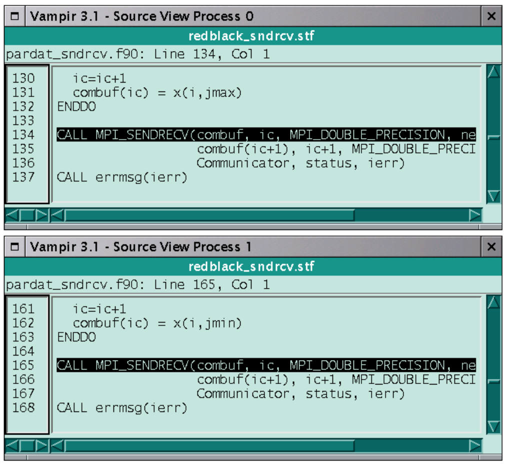
```
Vampir 3.1 - Call Tree
redblack_sndrcv.stf
redblack_sndrcv.stf: Global/Times incl./Sorted by:NamesUp
└─>User_Code            (1 : 0.66 s..0.951 s)
   ├─>Initialisation     (1 : 1.744 ms..8.391 ms)
   │  └─>MPI_Comm_split     (1 : 1.506 ms..7.901 ms)
   ├─>MPI_Bcast          (1 : 0.102 ms..0.291 s)
   ├─>MPI_Comm_rank      (1 : 0.104 ms..0.128 ms)
   ├─>MPI_Comm_size      (1 : 25.32 µs..28.56 µs)
   ├─>MPI_Finalize       (1 : 93.893 µs..0.178 ms)
   ├─>MPI_Wtime          (2 : 53.013 µs..0.104 ms)
   └─>Red-Black          (1 : 0.398 s..0.398 s)
      ├─>Exchangeblack      (100 : 20.829 ms..35.575 ms)
      │  └─>MPI_Sendrecv       (100..200 : 12.907 ms..25.982 ms)
      ├─>Exchangered        (100 : 13.195 ms..0.304 s)
      │  └─>MPI_Sendrecv       (100..200 : 6.908 ms..0.297 s)
      └─>MPI_Allreduce      (100 : 11.556 ms..0.323 s)
Close ◀ Search... ▶ Print... ▽ Fold.Level: 4 △ smaller larger ---
```

Figure 3.29: GLOBAL CALL TREE display, completely unfolded



Origin:	Process 0
Destination:	Process 1
Tag:	1000
Communicator:	3
Interval:	0.576017 s - 0.579455 s
Duration:	3.437547 ms
Length:	400 bytes
Data rate:	113.635 Kbytes/s

(a) TIMELINE with identified message



```
Vampir 3.1 - Source View Process 0
redblack_sndrcv.stf
pardat_sndrcv.f90: Line 134, Col 1
130     ic=ic+1
131     combuf(ic) = x(i,jmax)
132   ENDDO
133
134   CALL MPI_SENDRECV(combuf, ic, MPI_DOUBLE_PRECISION, ne
135                     combuf(ic+1), ic+1, MPI_DOUBLE_PRECI
136                     Communicator, status, ierr)
137   CALL errmsg(ierr)
```

```
Vampir 3.1 - Source View Process 1
redblack_sndrcv.stf
pardat_sndrcv.f90: Line 165, Col 1
161     ic=ic+1
162     combuf(ic) = x(i,jmin)
163   ENDDO
164
165   CALL MPI_SENDRECV(combuf, ic, MPI_DOUBLE_PRECISION, ne
166                     combuf(ic+1), ic+1, MPI_DOUBLE_PRECI
167                     Communicator, status, ierr)
168   CALL errmsg(ierr)
```

(b) SOURCE VIEW for sender and receiver

Figure 3.30: Jumping to source code

If the trace was generated using automatic program counter tracing, then the source location information might cover more than one level. By using the context menu items **Up** resp. **Down** one can jump between different levels.

3.5 Filters

3.5.1 Basics

Vampir allows the simultaneous display of thousands of processes and a time period only limited by main memory. Of course, displaying such an enormous amount of data on one screen would be incomprehensible. Therefore for large trace files Vampir offers several filtering functions to enable the viewing of a subset from the whole tracefile. The **Filters** menu shown is reached from the Vampir main window in 3.1 and allows filtering of certain events in the tracefile: **Processes**, **Messages**, **Collective Operations**, and **I/O Events**.

Filtering in Vampir may be divided into three stages:

1. A filter dialog is opened and options are selected in it.
2. Vampir is directed to *Apply* the new filter rules.
3. All displays redraw themselves, often restricting themselves to an even smaller subset of the trace data (e.g. just one specific activity)

The way in which the selection of interesting objects is done in the filter dialogs depends on the objects type, but can generally be divided into two methods:

- explicitly by the user, or
- automatically by Vampir based on user defined criteria.

Vampir supports filtering in several ways:

in time:	Filtering in time is provided by the zooming facility of timeline displays. As stated earlier, the options to influence the time period visible in a timeline display are extremely versatile and would go beyond the scope of this overview.
by processes:	Filtering by process is the other major dimension either because processes behave similarly, or inspection of a subset already demonstrates a problem. Therefore this feature is described in this brief introduction.
by event:	The choice of event (message, collective operations, or I/O) depends on specifying criteria that the event must fulfill.

3.5.2 PROCESS FILTER and Grouping

Previous releases of Vampirtrace and Vampir had a fixed two-level hierarchy built in: processes (or CPUs) at the lower level, and cluster nodes at the upper level. The so called *Grouping* support extends this concept and makes it configurable during tracing, both with Vampirtrace API calls and configuration options (see `VT_groupdef()` call and `GROUP` configuration option in the Vampirtrace documentation for details).

The default grouping is still based on the hardware hierarchy, starting with the group *All_Nodes* that contains one entry for each node in a cluster. Each node contains the processes that ran on it. If the application was multi-threaded in the sense that it had several independent *threads of control*, then each process entry itself is a group that contains all of its threads. As a process can also be regarded as one thread of control, this section uses the term *thread* instead of mentioning *process* and *threads* all the time. Vampir also presents one event stream for each thread resp. process in single-threaded applications.

More than one hierarchy can be defined. By default, the groups *All_Processes*, *All Master Threads* and *All_Threads* are also defined to provide faster access to threads than through the *All_Nodes* hierarchy, with *All_Threads* only being defined if there are threads. Figure 3.31 shows these groups after *unfolding All_Nodes* and the first node, *frost044.pacific.llnl.gov*. This reveals that the first eight processes ran on this node.

The PROCESS FILTER serves two independent purposes:

- selecting threads
- selecting groups to aggregate threads

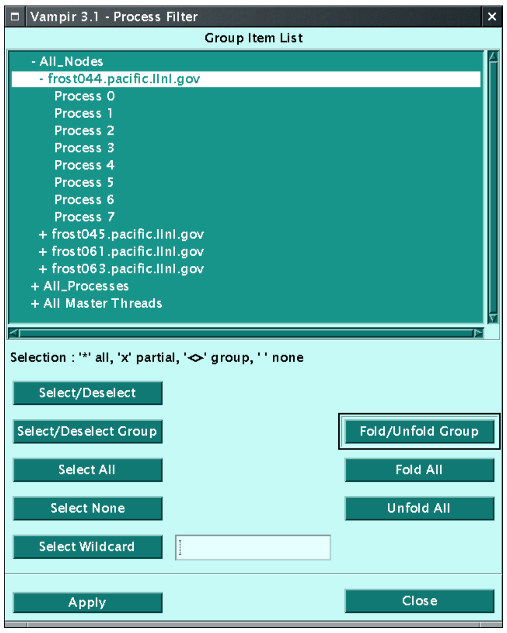


Figure 3.31: PROCESS FILTER and Grouping

Only selected threads (or processes, if the applications is not multi-threaded) are considered by Vampir's displays. This allows one to interactively restrict the analysis to a subset of the application run. Selecting groups has the effect that some displays aggregate the data from all selected threads in a group and present statistics for the groups instead of individual threads. This is explained in more detail below.

The PROCESS FILTER is implemented as a text view, as seen in Figure 3.31. In order to execute one of the available actions like folding/unfolding or selection, one first has to click on the list entry that is to be modified. This highlights the line in the text view with this entry. Then one presses the button for the desired action, selects the corresponding context menu item or uses a keyboard shortcut. The shortcuts are listed in the context menu. When changing several items in a row it is often fastest to keep one finger on e.g. the "s" key and then click with the mouse on an item followed by pressing the button to select it. This can be repeated quickly for several items.

Folded groups are marked with a + sign in front of them and unfolded ones with a - sign. An * indicates that a thread is selected. In front of a group an * is printed if all threads inside this group are selected and an *x* if only some of them are selected.

Because some displays like the GLOBAL TIMELINE only care for which threads are selected and ignore the group selection, the selection state of each thread is kept consistent throughout the process filter, even though it may appear in several groups at once.

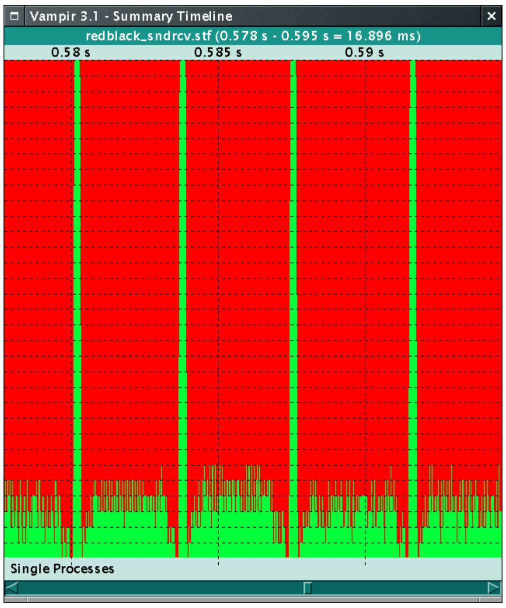
To select threads there are several shortcuts:

- *All* selects all threads.
- *None* deselects all threads.
- Clicking on a group and then *Select/Deselect* selects all members if none or only some were selected. Otherwise it deselects them.

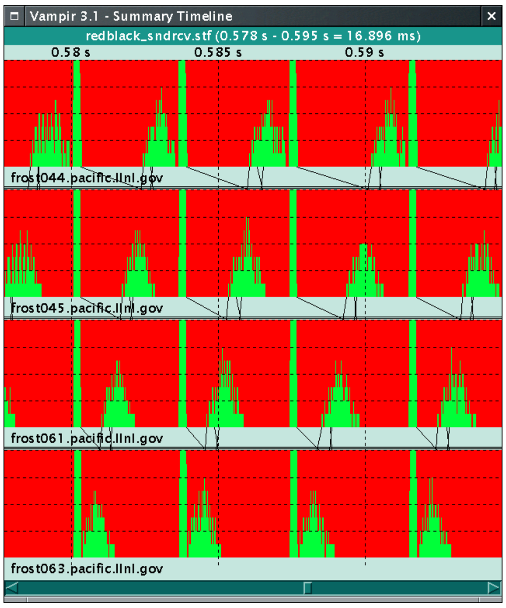
The last action has another effect: it also selects all top-level groups inside the group. For example, highlighting the *All_Nodes* group and then selecting it with *Select/Deselect* will select all threads in the whole application run and all nodes. Several displays use the group selection to display statistics for the selected groups instead of individual threads. This is often more scalable and it allows one to concentrate on e.g. messages sent between nodes in a cluster in the MESSAGE STATISTICS DISPLAY as in 3.33.

Group selections are marked in the text view with angle brackets around the * or *x* as follows: <*> or <*x*>. It is possible to do by-group analysis without having selected all threads inside the group, but as soon as the last thread in a group is deselected no data would contribute to the group and thus the group is automatically deselected. The *Select/Deselect Group* has the opposite effect if applied to a group without any selected threads: all threads are automatically selected in addition to the group itself. In other situations this button just changes the selection state of the highlighted group. This can be used to quickly select all threads and just the threads inside a specific group: simply highlight the group, then hit *Select/Deselect Group* twice.

The two figures in 3.32 show the result of selecting different groups upon the SUMMARY TIMELINE display. In 3.32(a) all processes, but no groups were selected (the default setting). This is indicated in the display with the string *Single Processes*. After pressing *None*, highlighting *All_Nodes* and pressing *Select/Deselect*, all process are selected again, but also each node group itself is selected. This leads to the display in 3.32(b), where one summary timeline is presented for each selected group.


Vampir 3.1 - Summary Timeline
redblack_sndrcv.stf (0.578 s - 0.595 s = 16.896 ms)
0.58 s
0.585 s
0.59 s
Single Processes

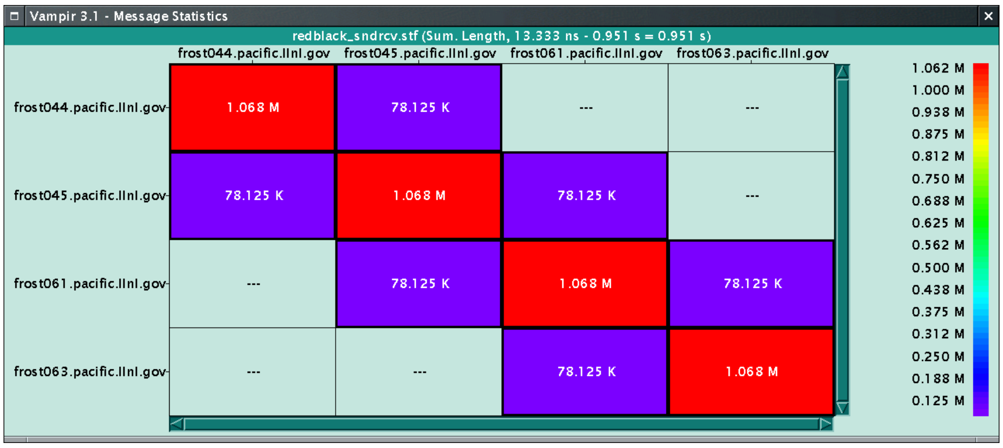
(a) no groups selected


Vampir 3.1 - Summary Timeline
redblack_sndrcv.stf (0.578 s - 0.595 s = 16.896 ms)
0.58 s
0.585 s
0.59 s
frost044.pacific.llnl.gov
frost045.pacific.llnl.gov
frost061.pacific.llnl.gov
frost063.pacific.llnl.gov

(b) all node groups selected

Figure 3.32: SUMMARY TIMELINE and Grouping

The MESSAGE STATISTICS and the COLLECTIVE OPERATION STATISTICS displays adapt in a similar manner. Instead of looking at messages sent between processes, one might learn more from looking at messages sent between and inside nodes: in 3.33 the top-left to bottom-right diagonal represents messages sent inside a node and the other squares represent those messages that were sent over the network. As shown in 3.33 most of the data is sent between processes running on the same node. Due to `redblack_sndrcv`'s communication pattern message duration is not determined by the transfer speed of the hardware at all. In `redblack_icomm` receives from the right neighbor are always completed first, which causes a visible bias in the average message duration, as seen in 3.34.

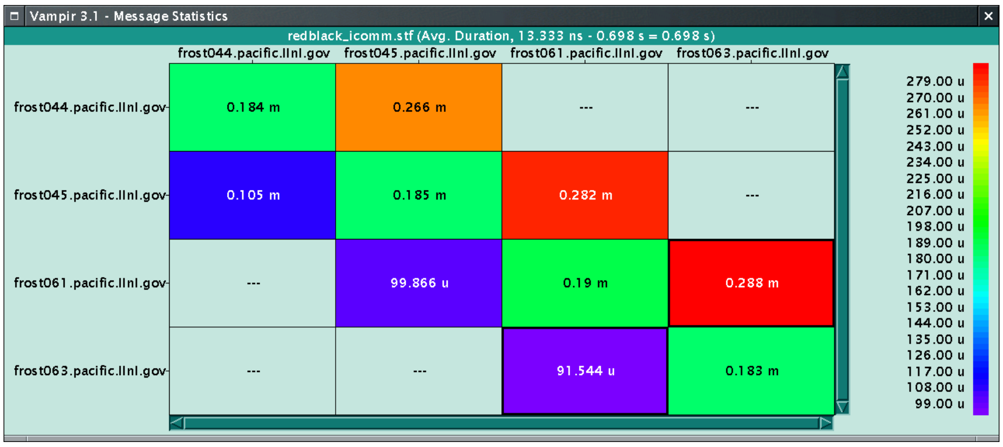


Vampir 3.1 - Message Statistics

redblack_sndrcv.stf (Sum. Length, 13.333 ns - 0.951 s = 0.951 s)

	frost044.pacific.llnl.gov	frost045.pacific.llnl.gov	frost061.pacific.llnl.gov	frost063.pacific.llnl.gov
frost044.pacific.llnl.gov	1.068 M	78.125 K	---	---
frost045.pacific.llnl.gov	78.125 K	1.068 M	78.125 K	---
frost061.pacific.llnl.gov	---	78.125 K	1.068 M	78.125 K
frost063.pacific.llnl.gov	---	---	78.125 K	1.068 M

Figure 3.33: by-node message volume for redblack_sndrcv



Vampir 3.1 - Message Statistics

redblack_icomm.stf (Avg. Duration, 13.333 ns - 0.698 s = 0.698 s)

	frost044.pacific.llnl.gov	frost045.pacific.llnl.gov	frost061.pacific.llnl.gov	frost063.pacific.llnl.gov
frost044.pacific.llnl.gov	0.184 m	0.266 m	---	---
frost045.pacific.llnl.gov	0.105 m	0.185 m	0.282 m	---
frost061.pacific.llnl.gov	---	99.866 u	0.19 m	0.288 m
frost063.pacific.llnl.gov	---	---	91.544 u	0.183 m

Figure 3.34: by-node message transfer rate for redblack_icomm

To actually measure the true hardware performance one has to ensure that receives are posted or active before the sender starts transmitting data. In these application runs the distribution of processes to nodes was chosen by the MPI so that consecutive process ranks ran on the same node, which happened to maximize the number of messages sent inside a node. For mpich the same effect is only achieved when invoking the mpirun command with a machine file. Without this additional parameter, processes would have been distributed in a round-robin fashion, leading to an arrangement with e.g. processes #0 and #1 on different nodes.

3.6 Preferences

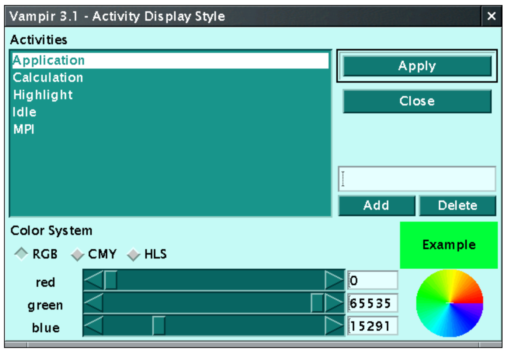
The **Preferences** menu is reached from the Vampir main panel and allows changes in default settings for several aspects of Vampir, among them: **Color Styles**, **Fonts**, **Displays**, **General**, **Symbols**, and **Tracefile**. Some of these choices have submenus or dialogs and these are described in the following.

3.6.1 Color Styles

This menu choice allows the change of color options for the following entities: **Activities**, **Messages**, **Collective Operations**, **I/O Events**. **Desktop** changes the window colors and **Color Mode** allows switching between a color, grey-scale and black-and-white mode.

Selecting the **Color Styles/Activites** menu item shows the dialog window in 3.35. From the *Activities* panel in this window the specific activity may be selected and the *Example* panel shows the default color, e.g. green for *Application*, red for *MPI*, etc. This default may be changed by mouse selection of a color from the circular color palette in the bottom right-hand corner. Alternatively, to mix an arbitrary color, the three scroll bars may be adjusted for red, green, and blue. Activities may be deleted or added in the dialog box above the corresponding buttons with these function names. Also, the color system used may be selected from the following choices:

- *RGB* Color composed from adding three components: red, green, and blue.
- *CMY* The inversion of RGB uses the complimentary colors: cyan, magenta, and yellow.
- *HLS* This is the Hue, Saturation, and Luminance (brightness) scheme.



red 0
green 65535
blue 15291

Figure 3.35: ACTIVITY DISPLAY STYLE dialog

A similar dialog window is available in the other **Color Styles** menu items. These other windows also support selecting different line styles. These settings can be chosen based on properties of the items. Colors for individual states cannot be set because there are too many of them—they inherit the color of their activity.

3.6.2 Displays

The **Preferences/Displays** menu choice allows the set up of how Vampir presents its displays. One can choose the default display that is opened after trace data has been loaded (**Startup**) and default values for several of the context menu items in some displays.

3.6.3 Tracefile

These options are available within the **Preferences/Tracefile** menu. They cover aspects of loading trace data and file handling. **Source** selects a search path for source files. This is used by the SOURCE CODE DISPLAY to find the source files of the program that generated a trace (see 3.4.9).

External Converters chooses programs that are started by Vampir whenever the suffix of a trace file matches the extension given for the converter in the Vampir preferences list. A dollar sign (`$`) in the conversion command is replaced with the file name and the command string may contain further arguments to the program. Once the program is started, it must produce a standard Vampir trace file on stdout. This is meant to be used to read compressed files in formats that Vampir does not support directly.

3.6.4 Extras

Selecting the **Extras** menu item from the Vampir main window allows control of printout and two viewing windows for errors or pending messages. The first choice on the **Extras** menu brings up the dialog box shown in 3.36. Most graphical displays in Vampir feature the **Print** entry in their context menus and its output serves the same purpose as the **Snapshot** function but is of much better quality and can easily be included into documents. Depending on the options selected in the print dialog, the display contents (always excluding the window frame) is converted to PostScript and put into a file or printed. Printing is started by clicking on the *OK* button.

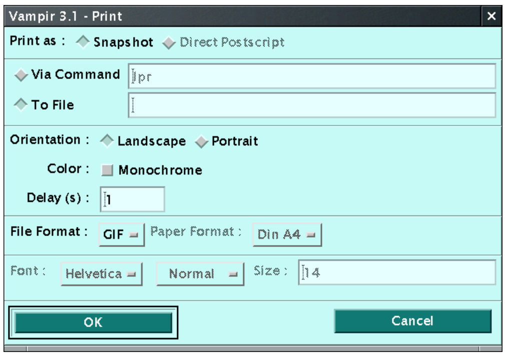

Vampir 3.1 - Print
Print as : Snapshot / Direct Postscript
Via Command: lpr
To File:
Orientation : Landscape / Portrait
Color : Monochrome
Delay (s) : 1
File Format : GIF
Paper Format : Din A4
Font : Helvetica, Normal
Size : 14

Figure 3.36: PRINT dialog

The last two choices bring up the status windows shown in 3.37 and 3.38, respectively. Vampir redirects messages to standard error from the terminal window (where it was started) to the ERROR LOG window (while it is open). If the ERROR LOG window is closed, messages continue to appear in the terminal window. In this example, an external converter was not found.

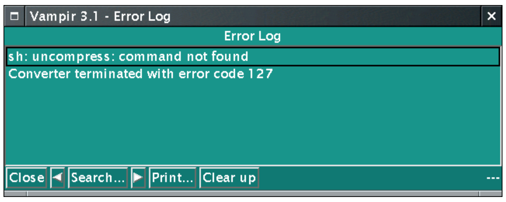
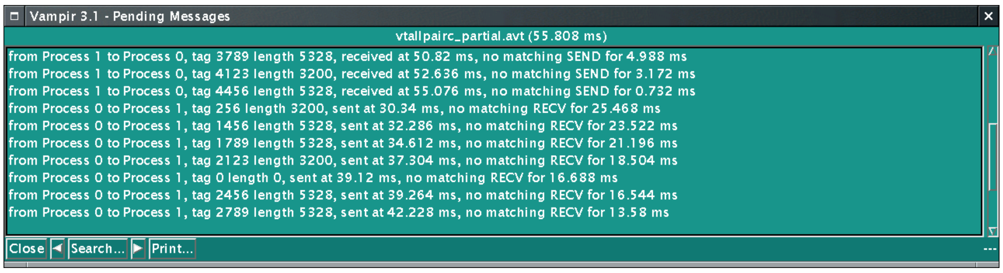

Vampir 3.1 - Error Log
Error Log
```
sh: uncompress: command not found
Converter terminated with error code 127
```

Figure 3.37: ERROR LOG dialog

Messages sent but not received are listed in the Pending Messages window in chronological order (with last message sent at the bottom of the list). Possible reasons for not received messages are either program errors or incomplete tracing. In the case of incomplete tracing the display may also show messages that were only receiving, and no sending was traced.


Vampir 3.1 - Pending Messages
vtallpairc_partial.avt (55.808 ms)
Close
Search...
Print...

```
from Process 1 to Process 0, tag 3789 length 5328, received at 50.82 ms, no matching SEND for 4.988 ms
from Process 1 to Process 0, tag 4123 length 3200, received at 52.636 ms, no matching SEND for 3.172 ms
from Process 1 to Process 0, tag 4456 length 5328, received at 55.076 ms, no matching SEND for 0.732 ms
from Process 0 to Process 1, tag 256 length 3200, sent at 30.34 ms, no matching RECV for 25.468 ms
from Process 0 to Process 1, tag 1456 length 5328, sent at 32.286 ms, no matching RECV for 23.522 ms
from Process 0 to Process 1, tag 1789 length 5328, sent at 34.612 ms, no matching RECV for 21.196 ms
from Process 0 to Process 1, tag 2123 length 3200, sent at 37.304 ms, no matching RECV for 18.504 ms
from Process 0 to Process 1, tag 0 length 0, sent at 39.12 ms, no matching RECV for 16.688 ms
from Process 0 to Process 1, tag 2456 length 5328, sent at 39.264 ms, no matching RECV for 16.544 ms
from Process 0 to Process 1, tag 2789 length 5328, sent at 42.228 ms, no matching RECV for 13.58 ms
```

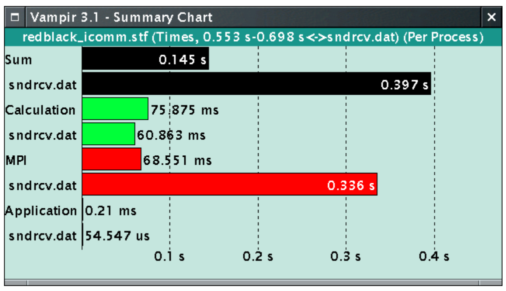
Figure 3.38: PENDING MESSAGES in an incomplete trace

3.7 Using One Vampir Main Window to Compare Trace Files

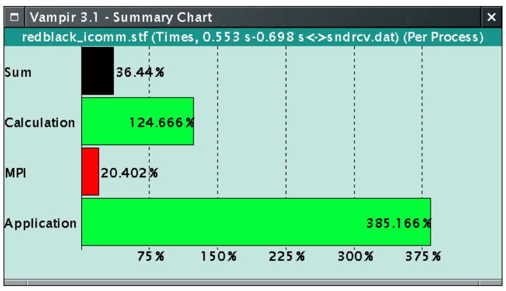
In the following a step-by-step process guides the reader through side-by-side comparisons of two separate trace files by the use of one Vampir main window. This allows comparison of results from two trace files only in some Vampir displays and dialogs.

In Vampir, profiling statistics displays such as SUMMARY CHART and local ACTIVITY CHART allow profiling data from multiple traces to be compared. To demonstrate usage of this comparison procedure follow these steps:

1. Load the trace file `redblack_sndrcv`.
2. Open the global timeline display.
3. Zoom to exclude the MPI start up and finalisation procedures.
4. Open the SUMMARY CHART display.
5. From the context menu ensure that:
   - **Timeline/Global** is enabled,
   - **Options/Include Sum** is enabled, and
   - **Options/Store Values** is selected.
6. In the SELECT OUTPUT FILE dialog box that opens, save the file as `sndrcv.dat`.
7. Load the trace file `redblack_icomm` into Vampir.
8. Open the global timeline display.
9. Zoom to exclude the MPI start up and finalisation procedures.
10. Open the SUMMARY CHART display.
11. From the context menu ensure that:
    - **Timeline/Global** is enabled,
    - **Options/Include Sum** is enabled, and
    - **Options/Load Values** is selected.
12. In the SELECT INPUT FILE dialog box that opens load the previously generated `sndrcv.dat`.



(a) comparing absolute values



(b) quotient

Figure 3.39: SUMMARY CHART comparison of redblack_sndrcv and redblack_icomm

A comparative summary chart display similar to 3.39(a) should appear. This shows the current (open) trace file results in the upper bar of each pair, and the results of the saved (and then loaded) trace file in the lower bar for each pair.

When comparing trace files by this method note the following:

- Keep in mind that only the data in the current display is saved, e.g. if the display is showing activity data only, no symbol-specific data will be present in the saved file.
- When analysing different trace files, change the profiling display to show a horizontal histogram (**Mode/Hor. Histogram**) and use the same options for both displays.
- With the **Comparison** submenu of the context menu, the comparison style can be selected to display both sets of values (**Show**) as in 3.39(a), the difference (**Difference**), or the quotients (**Quotient**, once for current trace divided by the other one as in 3.39(b), once for the inverse). If multiple sets of comparison data have been loaded, the **Comparison/Compare** to selection from the context menu allows switching between them.