



pallas

Competence in High Performance Computing

Linux tools for debugging and profiling MPI codes

Werner Krotz-Vogel, Pallas GmbH

<krotz@pallas.com>

MRCCS September 02000

Pallas GmbH
Hermülheimer Straße 10
D-50321 Brühl, Germany

info@pallas.com
www.pallas.com



State-of-the-art program development tools ...

... in detail:

- Vampir-2.5 (online-Demo), Vampirtrace-2.0, Dimemas
- Etnus TotalView 4.0 Multi-process Debugger

... briefly:

- KAP/Pro Toolset 3.8, OpenMP
- KAI C++ 3.4, ISO standard
- PGI 3.1 x86 Compilers, Cluster Development Kit (CDK)
- FORESYS - Fortran Restructuring Tool

... free open source:

- PMB - Pallas MPI Benchmark Suite (incl. “effective Bandwidth”)

KAP/Pro Toolset - Assure Example



The screenshot displays the KAP/Pro Toolset interface. The main window is titled "Project: parbugs Data File: parbugs". The menu bar includes File, View, Search, Print, Preferences, Reorder, Windows, and Help. The toolbar contains icons for printing, navigating back, and navigating forward.

The left pane shows a tree view of error analysis for "parbugs.f". The errors listed include:

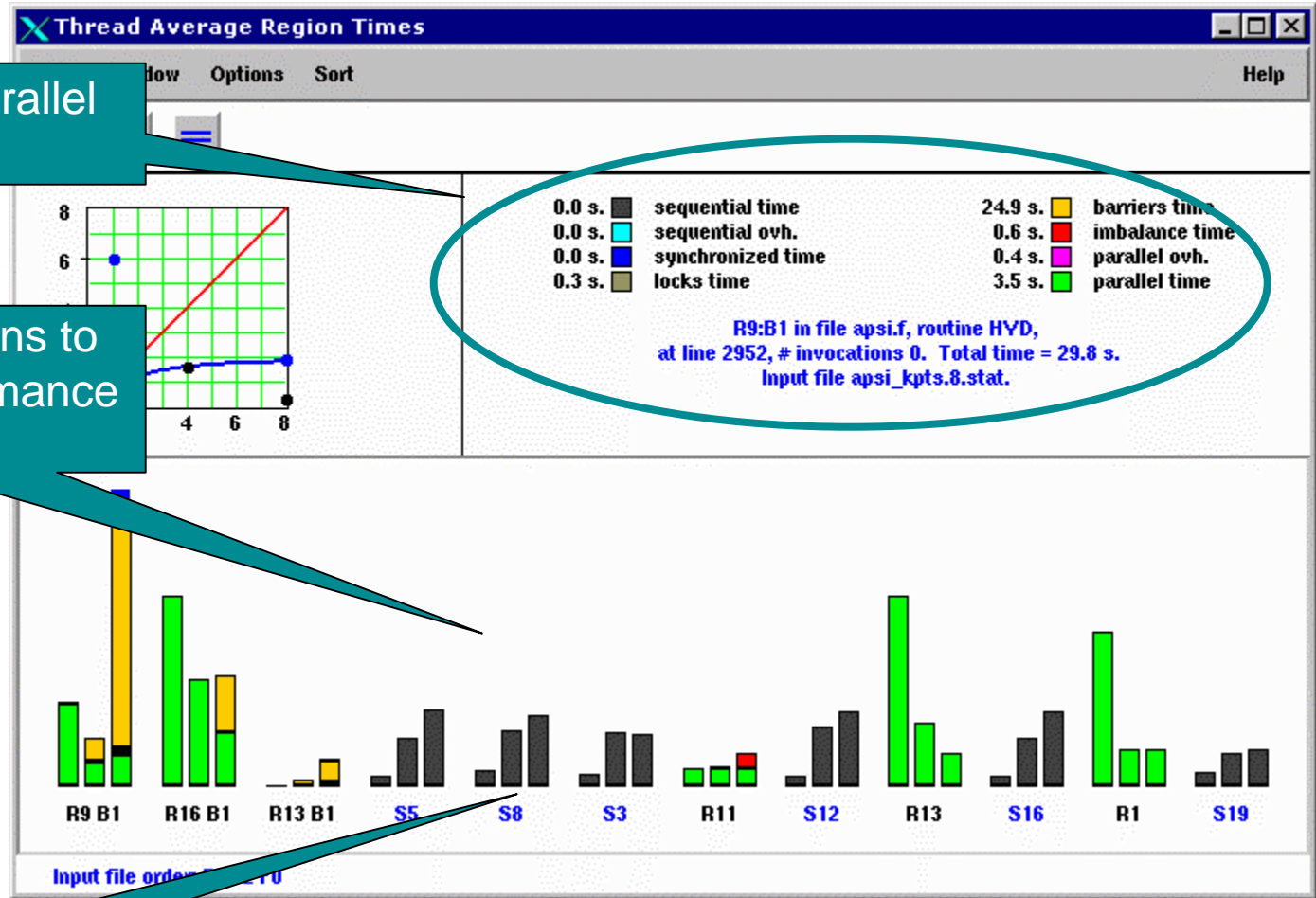
- Write -> Write II in PARBUGS
- 2 Errors in PARALLEL region: PARBUGS@46-51
- Write -> Read ISQMAX in PARBUGS -> ISQMAX
- Parallel I/O incorrectly synchronized in PARBUGS@47-51
- 2 Errors in PDO: PARBUGS@47-51
- Write -> Read ISQMAX in PARBUGS (highlighted)
- Write -> Write ISQMAX in PARBUGS
- 1 Error in PARALLEL DO: PARBUGS@62-68
- Uninitialized read in PARBUGS of PRIVATE
- 1 Construct which was not executed
- PARALLEL DO: COMPUTE@80-83
- Copyright © 1997 - 1999 by Kuck and Associates

The bottom left pane shows a "Program Wide Errors per Construct" bar chart. A legend indicates: Errors (red), Cautions (orange), Warnings (yellow), OK (green), and Not Run (blue). The chart shows a significant number of errors.

The right pane, titled "Source & Sink: parbugs.f", shows the source code with annotations. A red box highlights line 49: `isqmax = max(isquared(i), isqmax)`. A blue arrow points from this line to the "Source location" label in the bottom left. The code includes comments explaining synchronization issues and provides fixes using `!$omp single` and `!$omp end single` constructs.

Source location

KAP/Pro Toolset - GuideView Example



Analyse each parallel region

Sort or filter regions to navigate to performance hotspots

Identify serial regions that hurt scalability



The most modern, best performing, platform independent C++

- ISO C++ standard syntax, including exceptions and member templates
- ISO C++ standard class library
- multi-platform support
- meet C performance requirements
- thread safety (on most platforms)



Compilers & Tools ...

- PGI 3.1 x86 compilers , C, C++, F77, F90, HPF, pgrof, pgdbg
- SMP/OpenMP support for C, C++, F77, F90

... plus convenient add-on's:

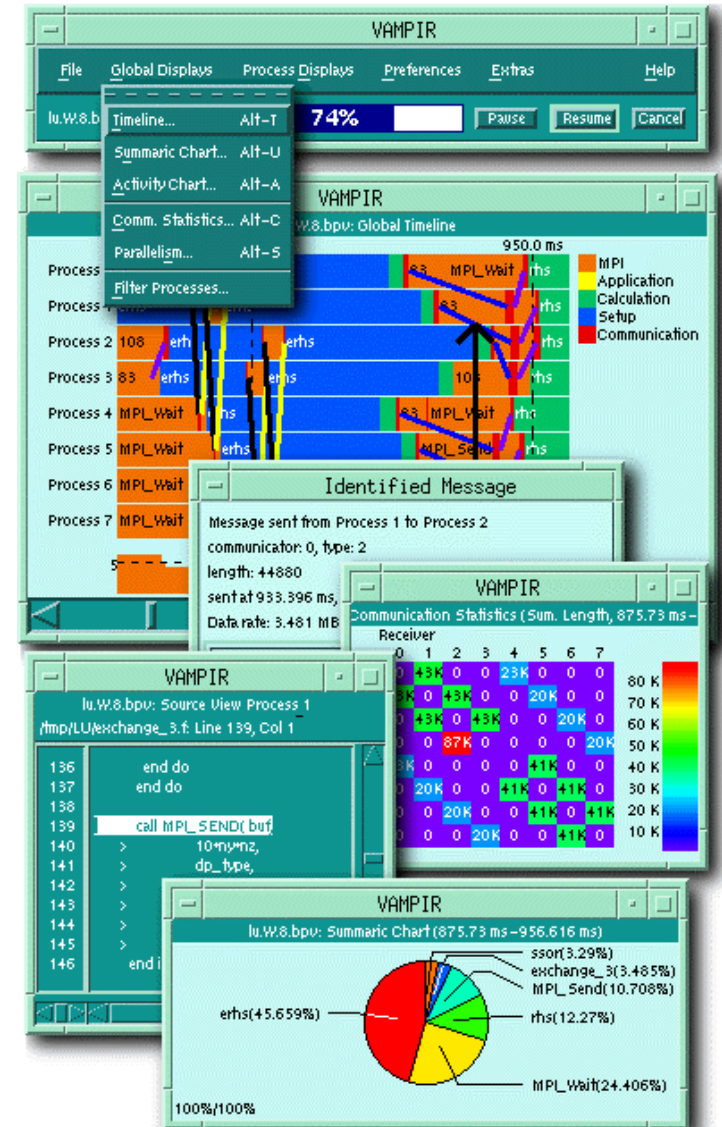
- parallel ScaLAPACK
- optimized BLAS, LAPACK
- MPI/mpich
- PVM
- PBS - Portable Batch System
- Tutorial, examples
- Cluster management utilities



- Translates FORTRAN code (F77 - F95) into abstract syntax tree (ForLib)
- FORTRAN code consistency checks (definitions of functions, common blocks etc.)
- Interactive visualization & analysis of inconsistencies
- Upgrading from FORTRAN 77 to FORTRAN 90
- Interactive/batch analysis of parallelization possibilities
- Automatic code quality analysis/improvements



Visualization and Analysis of MPI Programs





- Current version: Vampir 2.5

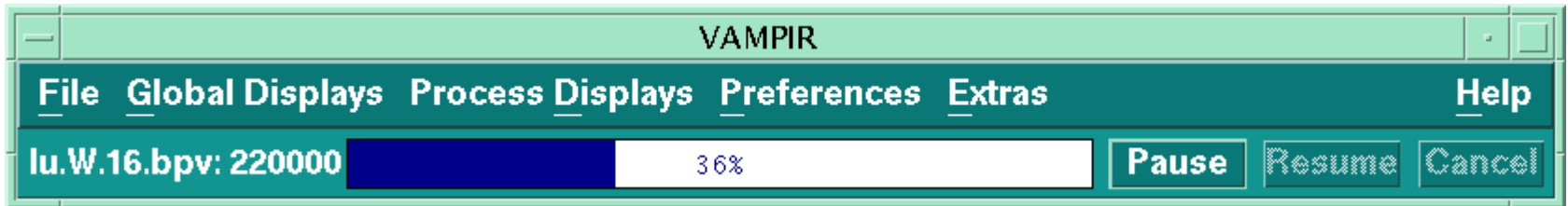
- Significant new features
 - support for collective MPI operations
 - trace comparison
 - tracefile re-write
 - message-length histogram
 - local and global calling trees
 - source-code reference
 - support for MPI-2 I/O operations



- Offline trace analysis for MPI (and others ...)
- Traces generated by **Vampirtrace** tool (``ld ... -IVT -lpmpi -lmpi``)
- Convenient user–interface
- **Scalability** in time and processor–space
- Excellent **zooming** and **filtering**
- High–performance graphics
- Display and analysis of **MPI** and **application** events:
 - execution of **MPI** routines
 - point–to–point and collective communication
 - MPI–2 I/O operations
 - execution of application subroutines (optional)
- Easy customization



Vampir 2.5 main window



- Tracefile loading can be interrupted at any time
- Tracefile loading can be resumed
- Tracefile can be loaded starting at a specified time offset
- Tracefile can be re-written (re-grouped symbols)

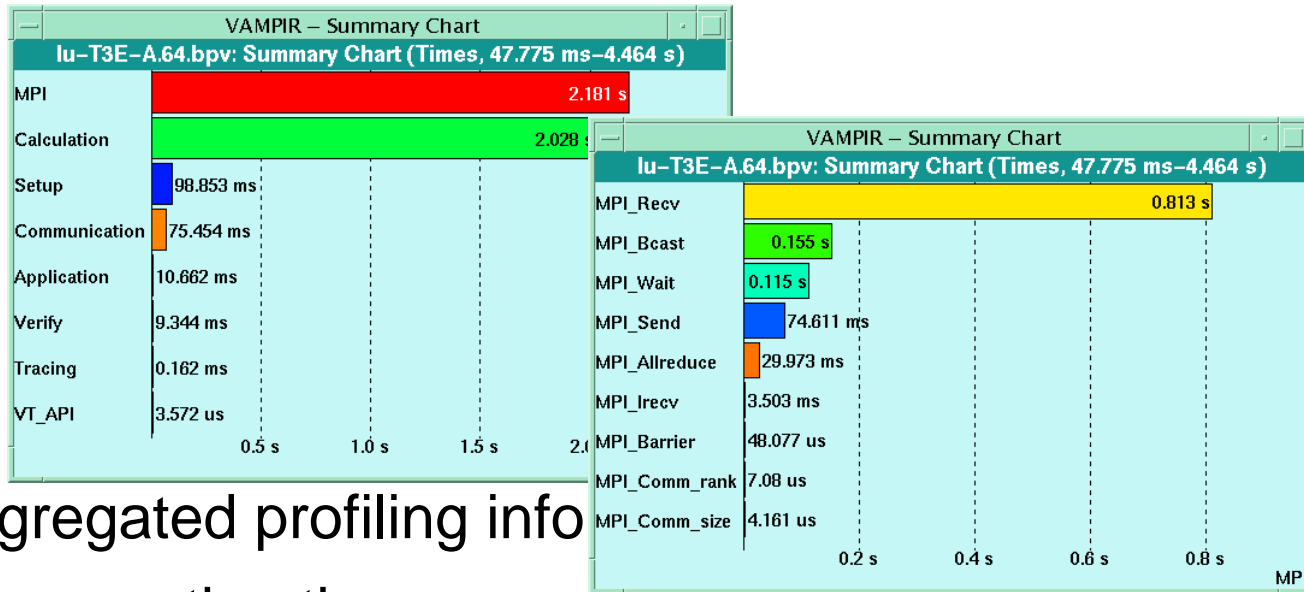


- **Global displays** show all selected processes
 - **Summary Chart:** aggregated profiling information
 - **Activity Chart:** presents per-process profiling information
 - **Timeline:** detailed application execution over time axis
 - **Communication statistics:** message statistics for each process pair
 - **Global Comm. Statistics:** collective operations statistics
 - **I/O Statistics:** MPI I/O operation statistics
 - **Calling Tree:** draws global or local dynamic calling trees

- **Process displays** show a single process per window
 - **Activity Chart**
 - **Timeline**
 - **Calling Tree**



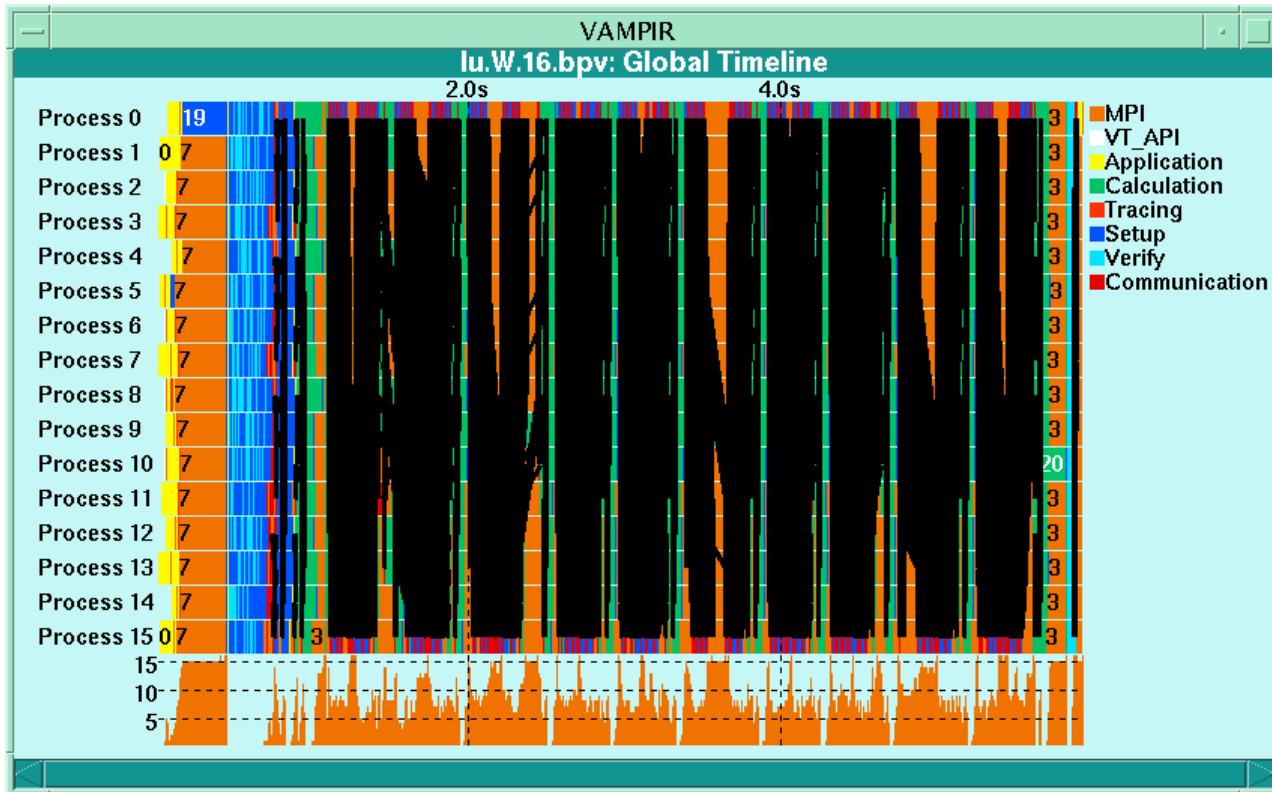
Summary Chart



- Aggregated profiling info
 - execution time
 - number of calls
- Inclusive or exclusive of called routines
- Look at all/any category or all states
- Values can be exported/imported
- Tracefiles can be compared



Timeline Display



- Now displays MPI collective and I/O operations
- To zoom, draw rectangle with the mouse
- Also used to select sub-intervals for statistics

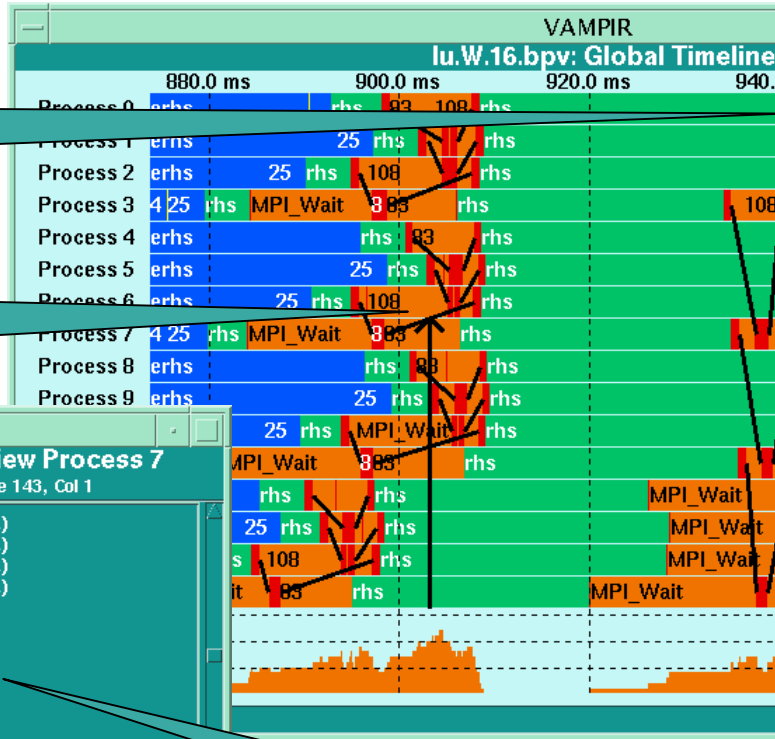


Timeline Display (Message Info)



See message details

Click on message line



Identified Message

Message sent from Process 7 to Process 6
communicator: 0, type: 1
length: 21120
sent at 898.439 ms, received at 907.921 ms
Data rate: 2.227 MBytes/sec

Close

VAMPIR

lu.W.16.bpv: Source View Process 7

/tmp/traces/NPB-LU/exchange_3.f: Line 143, Col 1

```

136   buf(2,ipos2) = g(2,1,j,k)
137   buf(3,ipos2) = g(3,1,j,k)
138   buf(4,ipos2) = g(4,1,j,k)
139   buf(5,ipos2) = g(5,1,j,k)
140   end do
141   end do
142
143   call MPI_SEND( buf,
144 >    10*ny*nz,
145 >    dp_type,
146 >    north,
147 >    from_s,
148 >    MPI_COMM_WORLD,
149 >    IERROR )
150   end if

```

VAMPIR

lu.W.16.bpv: Source View Process 6

/tmp/traces/NPB-LU/exchange_3.f: Line 156, Col 1

```

149   >    IERROR )
150
151   end if
152 c-----
153 c receive from south
154 c-----
155   if (south.ne.-1) then
156     call MPI_WAIT( mid, STATUS, IERROR )
157
158     do k = 1,nz
159       do j = 1,ny
160         ipos1 = (k-1)*ny + j
161         ipos2 = ipos1 + ny*nz
162         buf(2,j,k) = buf1(1,ipos1)
163         buf(2,j,k) = buf1(2,ipos1)

```

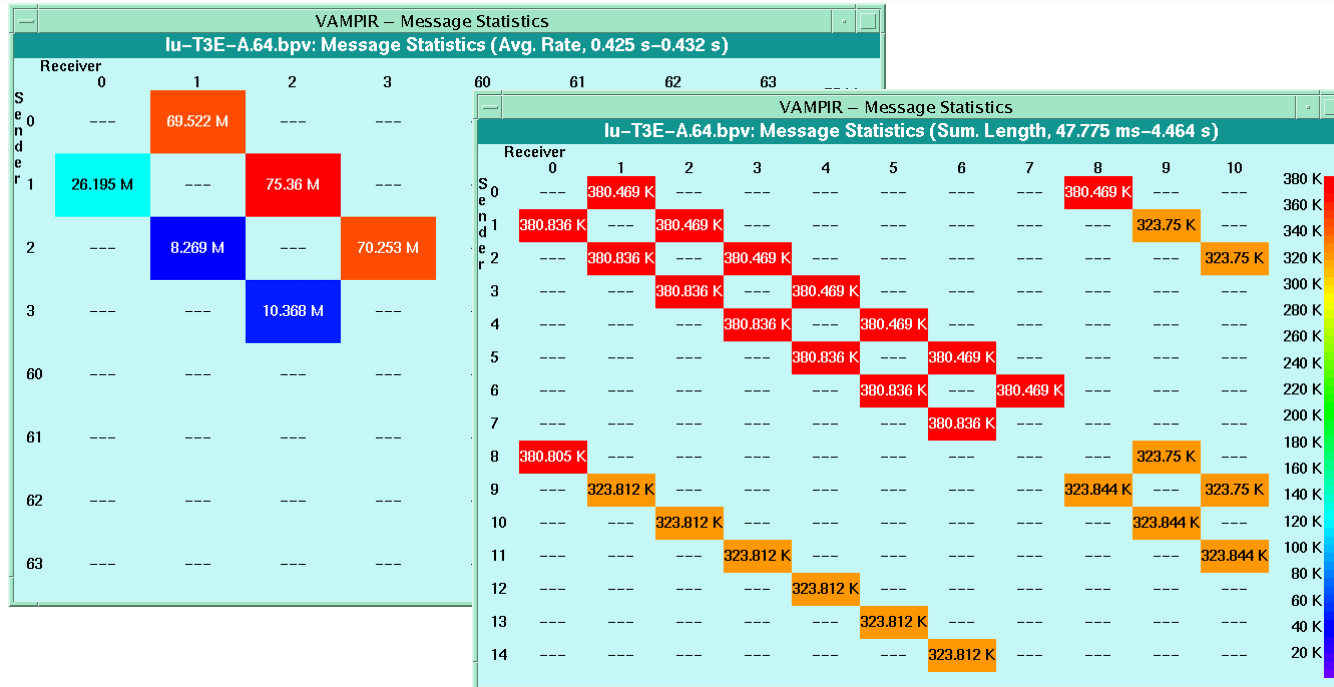
Message send op

Message receive op

- Source-code references are displayed if recorded by Vampirtrace



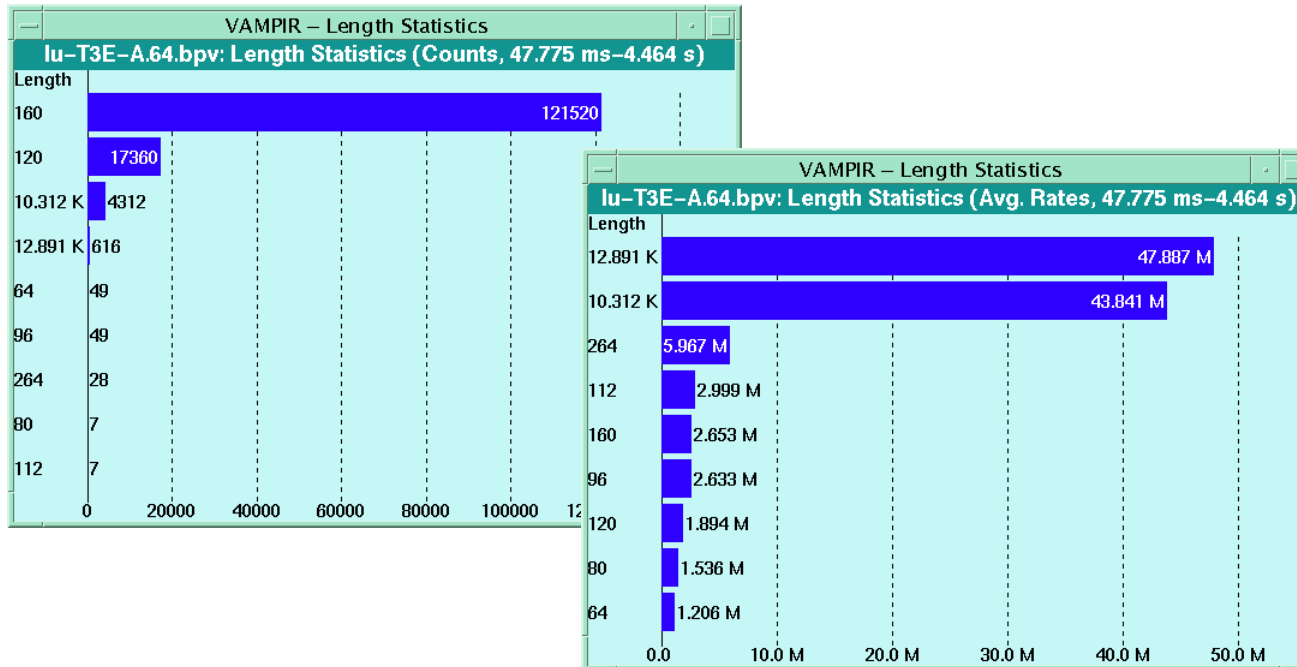
Communication Statistics



- Message statistics for each process pair:
 - Byte and message count
 - min/max/avg message length
 - min/max/avg bandwidth
- Filter for message tags or communicators



Message Histograms



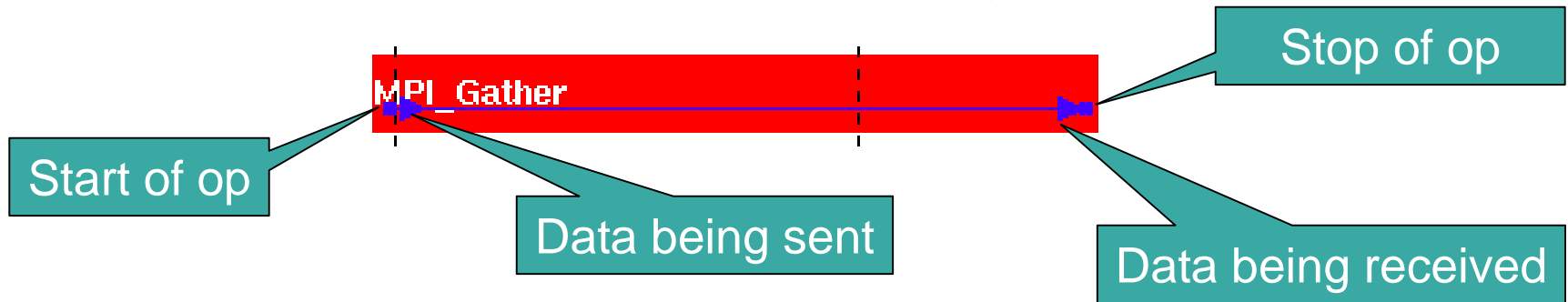
- Message statistics by length, tag or communicator
 - Byte and message count
 - min/max/avg bandwidth
- Filter for message tags or communicators



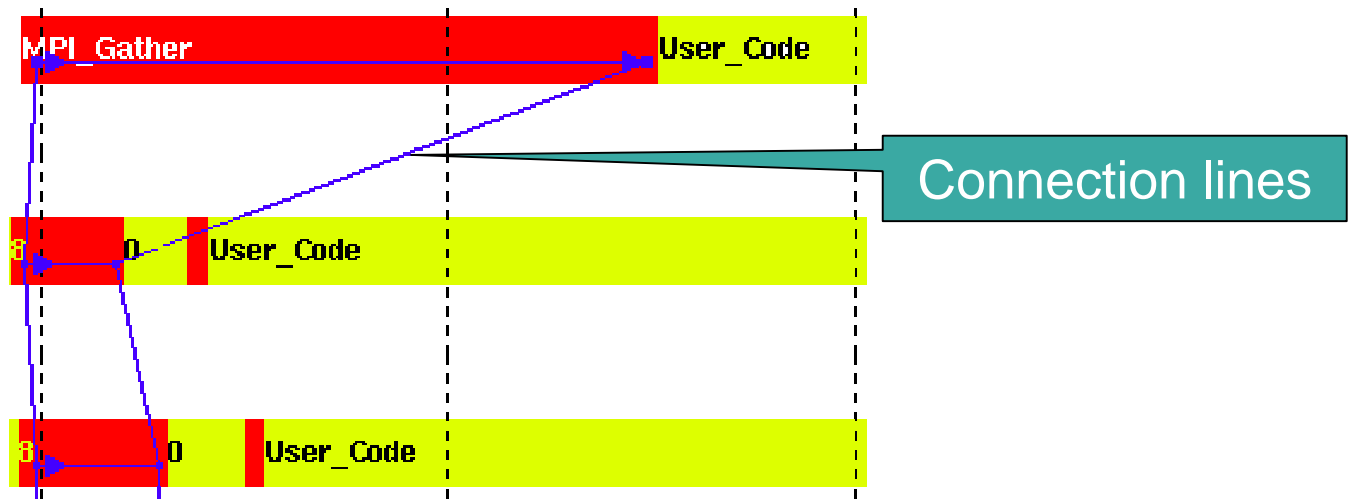
Collective Operations



- For each process: mark operation locally



- Connect start/stop points by lines

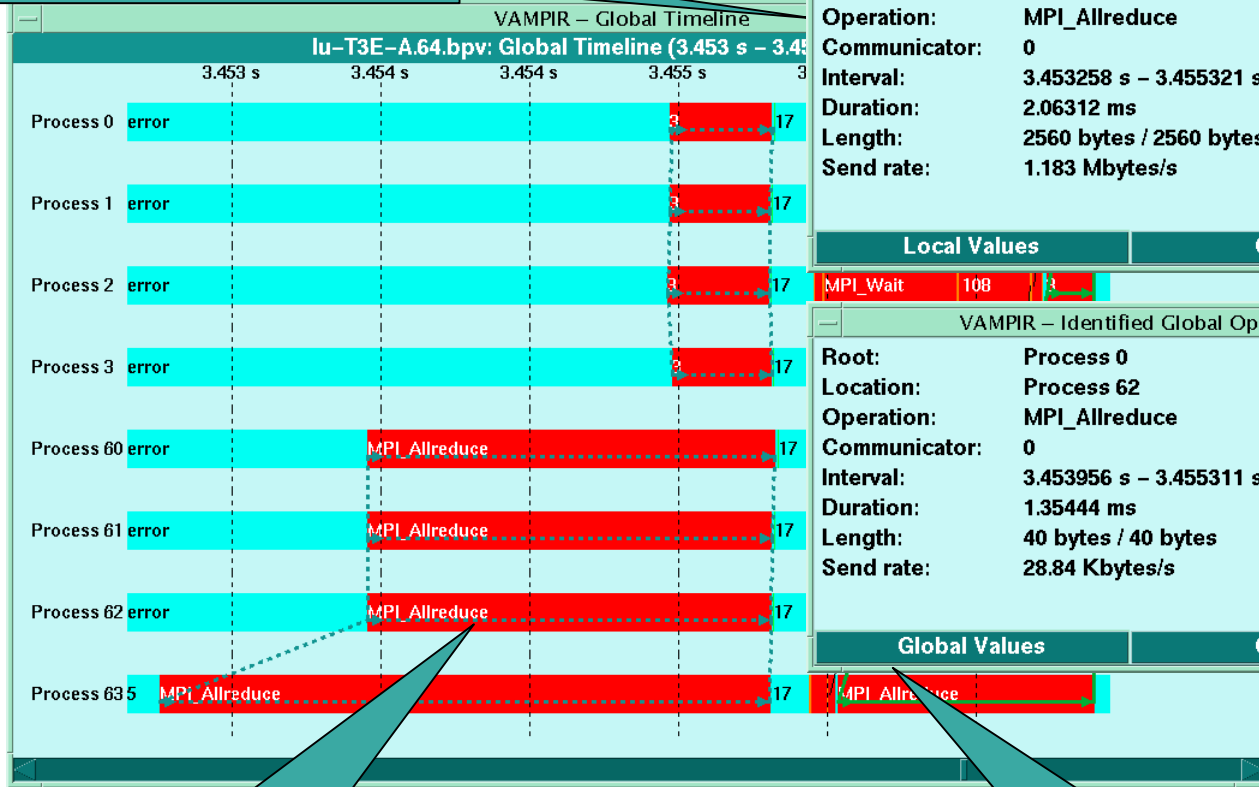




Collective Operations



See global timing info



Click on collective operation display

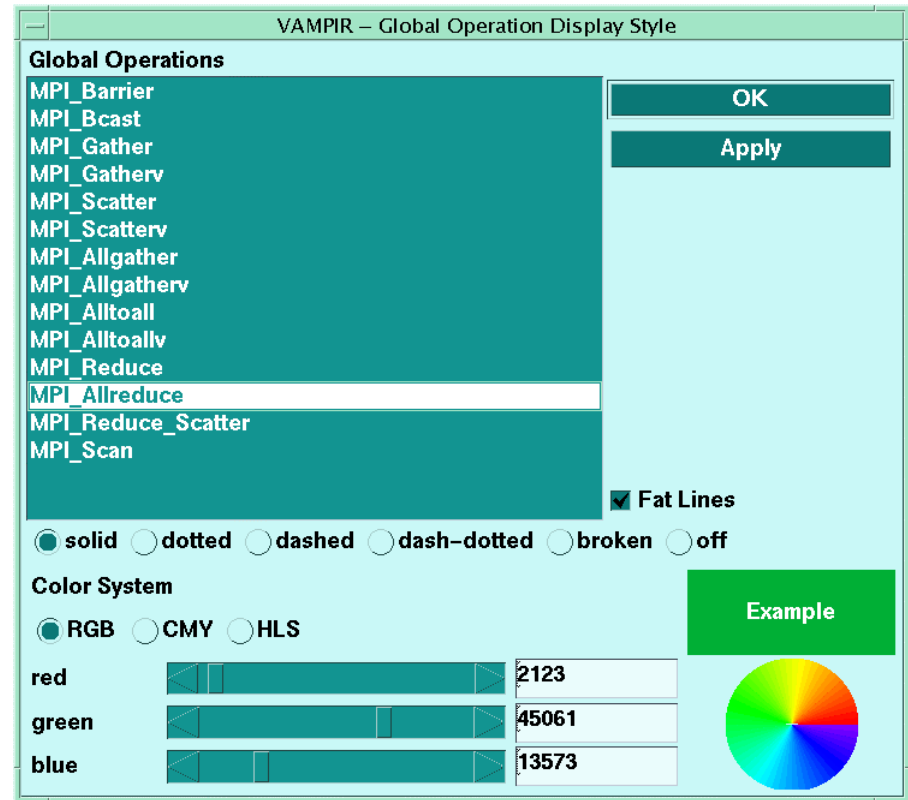
See local timing info



Collective Operations



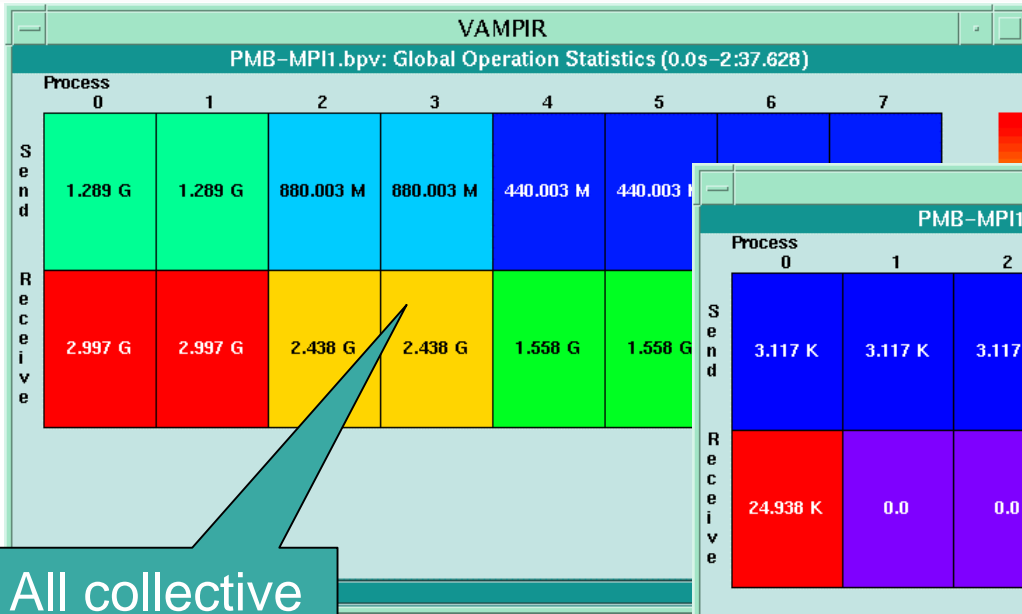
- Collective operations can be filtered



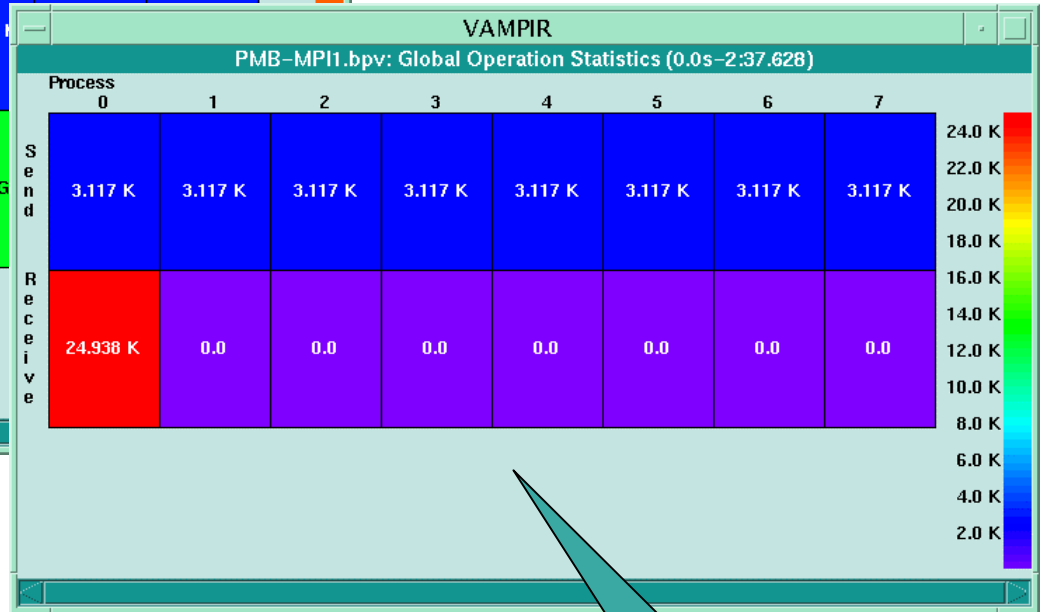
- The display style can be adapted for each collective operation



Global Communication Statistics



All collective operations

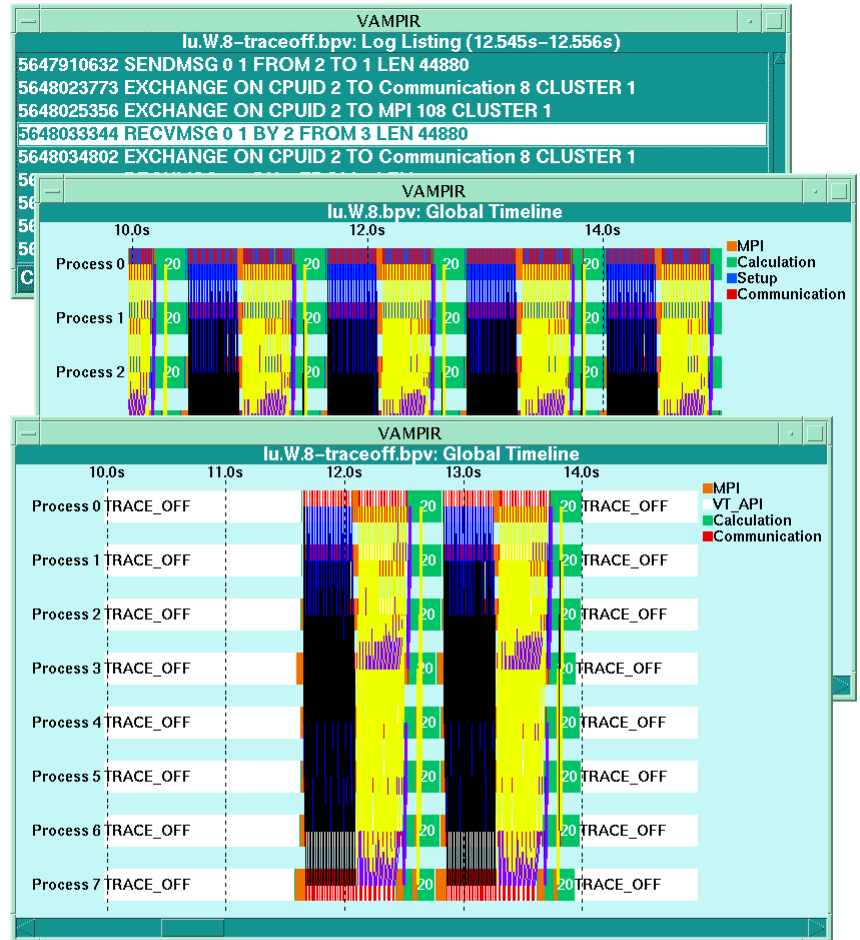


MPI_Gather only

- Statistics for collective operations:
 - operation counts, Bytes sent/received
 - transmission rates
- Filter for collective operation

Vampirtrace

Tracing of MPI and Application Events





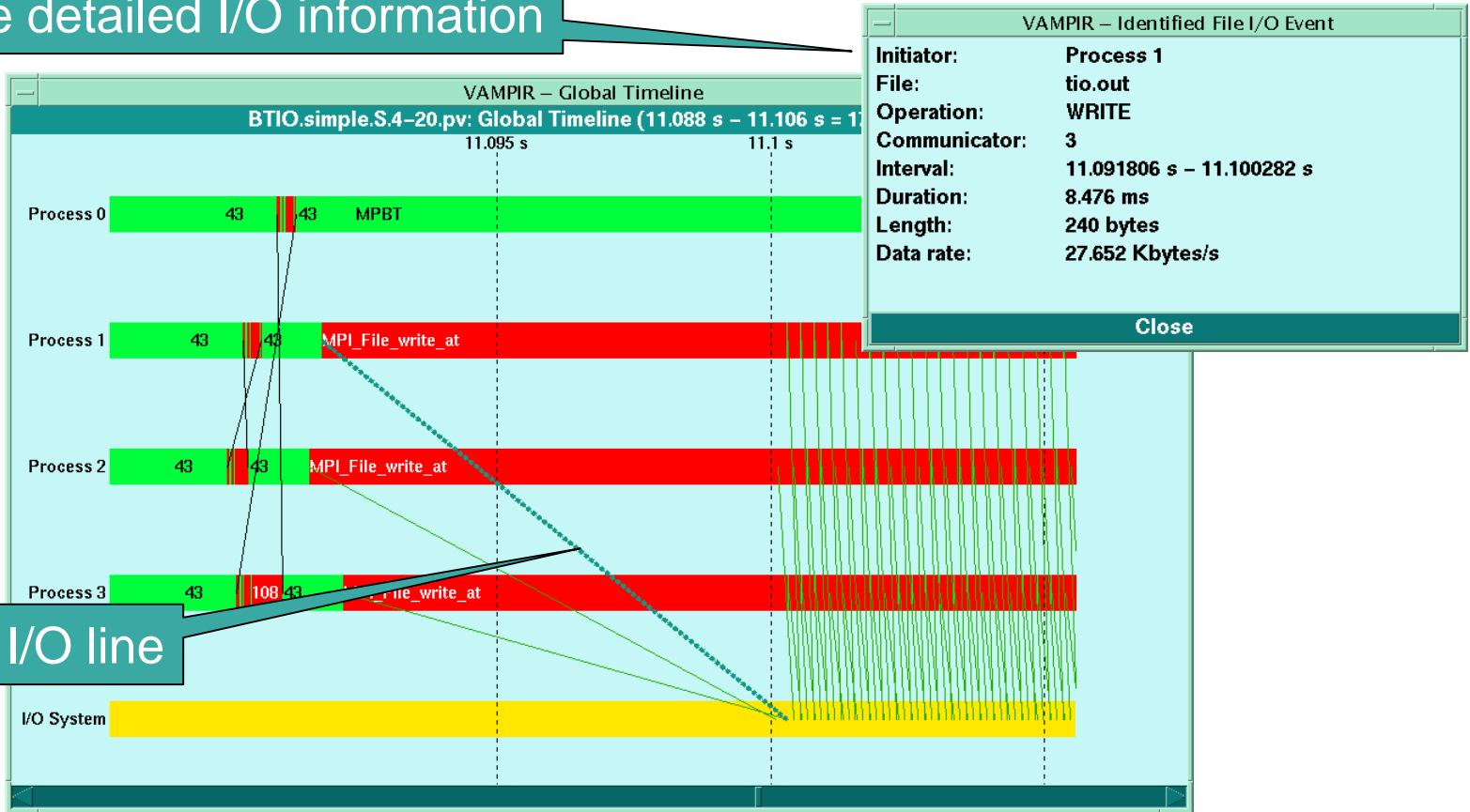
- New version: Vampirtrace 2.0
- Significant new features:
 - records collective communication
 - enhanced filter functions
 - extended API
 - records source–code information (selected platforms)
 - support for shmemp (Cray T3E)
 - records MPI–2 I/O operations
- Available for all major MPI platforms



MPI-I/O Operations



See detailed I/O information

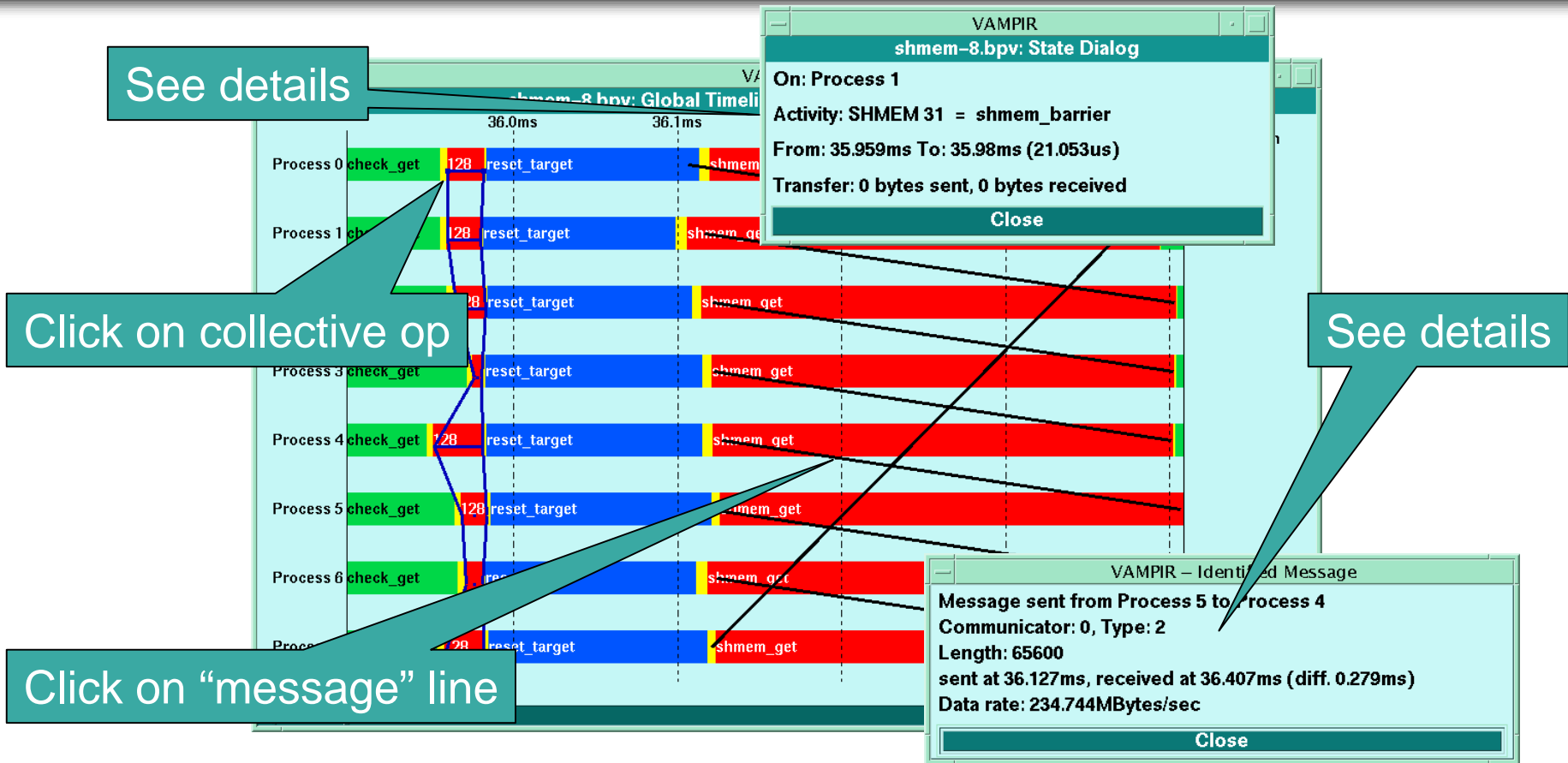


Click on I/O line

- I/O transfers are shown as lines



shmem Operations



- Display one-sided transfers as messages
- Display shmem global operations



- Reference customers: ARL, ARSC, CEWES, LANL, LLNL, MHPCC, NASA, NERSC, NSA, Cornell TC, Oregon Univ., CEA, DWD, ECMWF, GMD, HLRS, LRZ, PC², RUKA, ...

- URLs:
 - www.tc.cornell.edu/Edu/Tutor/Vampir
 - www.llnl.gov/sccd/lc/DEG/vampir/vampir.html
 - www.uni-karlsruhe.de/~Vampir
 - www.lrz-muenchen.de/services/software/parallel/vampir
 - www.hlrs.de/structure/support/parallel_computing/tools/performance/vampir.html



- Start simple: only single thread per process calls MPI
- move MPI call outside parallel region or remove OMP barrier, if possible, for optimization

...

! We are inside an MPI process

!\$OMP PARALLEL

....

!\$OMP BARRIER ! to ensure consistent memory

!\$OMP MASTER ! cleanly separate OpenMP parallelism from MPI ...

CALL MPI_<some_action>(...)

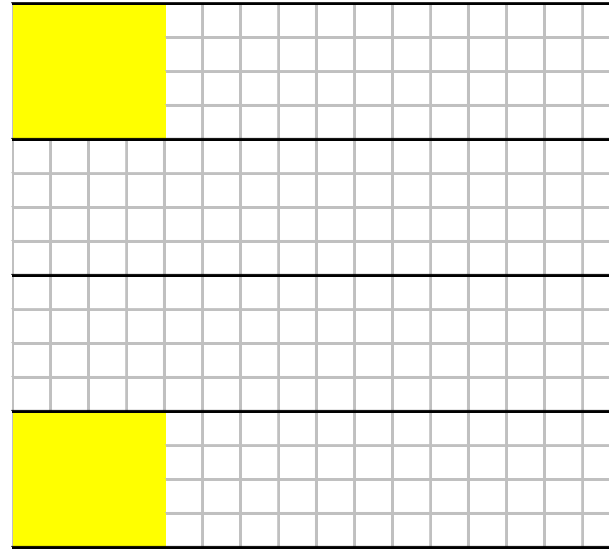
!\$OMP END MASTER

!\$OMP BARRIER ! to ensure consistent memory

....



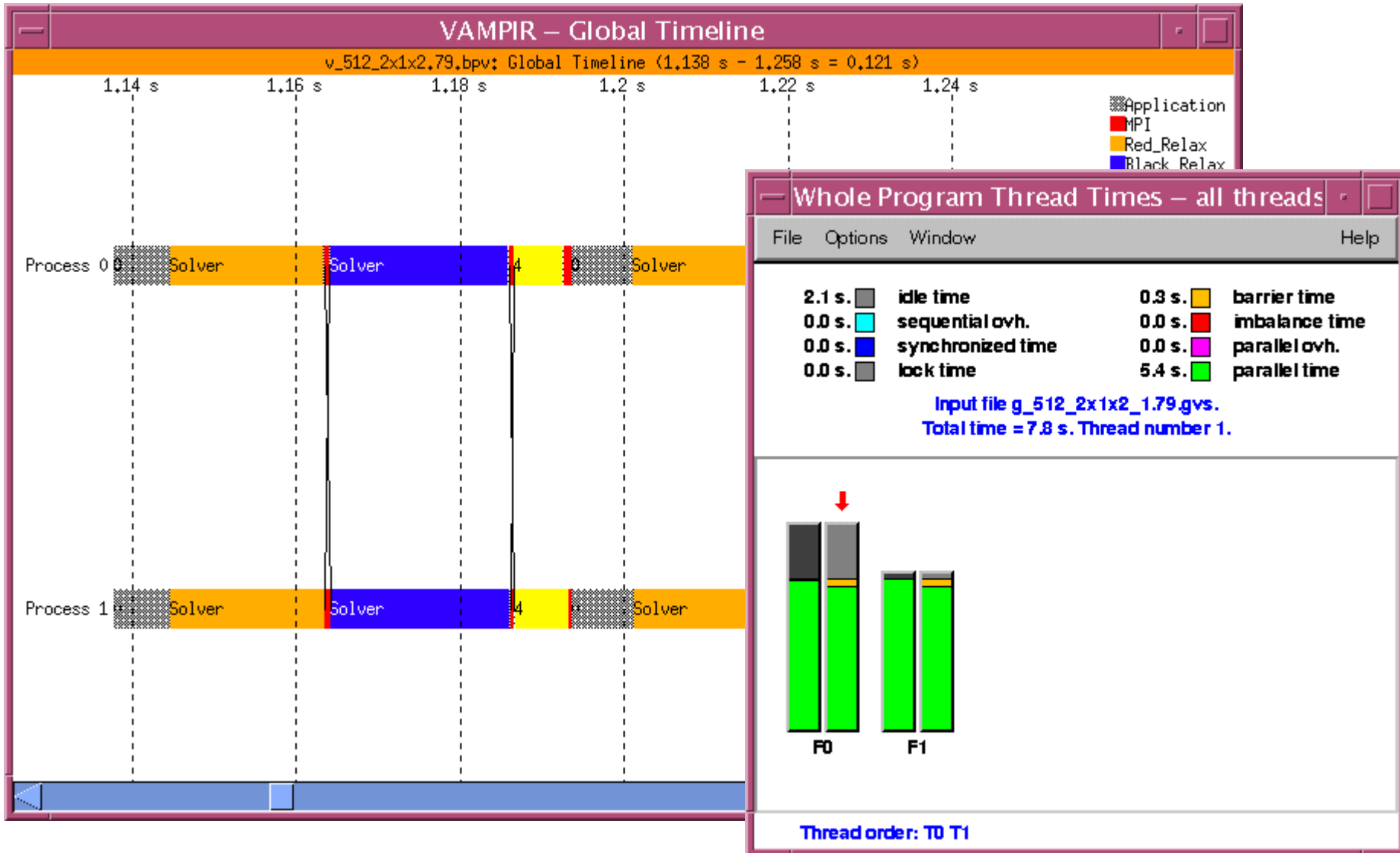
- Parallel Poisson solver in MPI and OpenMP
- certain sections of the grid with more work load
- MPI split not easy, except 2x1 horizontal
- 4 MPI processes run not balanced



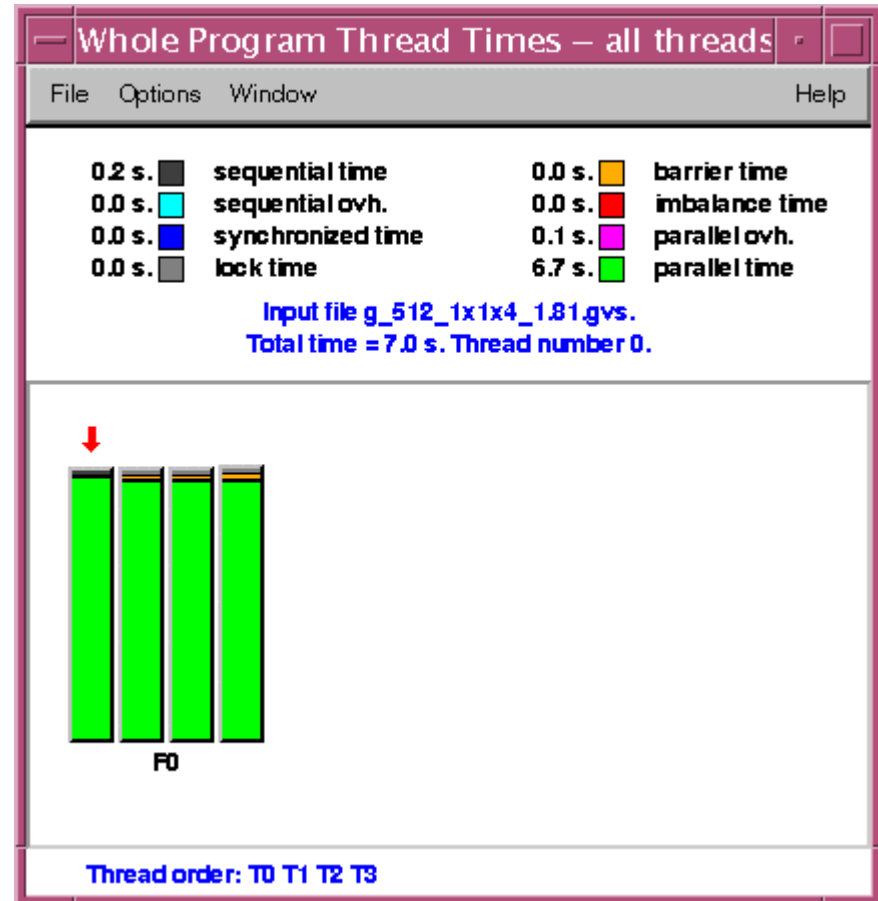


- Judge quality of parallelization by means of tools:
 - Vampir, GuideView --> performance
 - Assure --> correctness
- Yes, the tools work for hybrid approaches. When running, e.g., a 2 by 2 hybrid model (2 MPI processes, 2 threads per process), one gets
 - one Vampir tracefile and,
 - for each (!) MPI process, an own GuideView profile. GuideView can combine these into one single graph. (--> `setenv KMP_STATS_FILE guide_%l`)

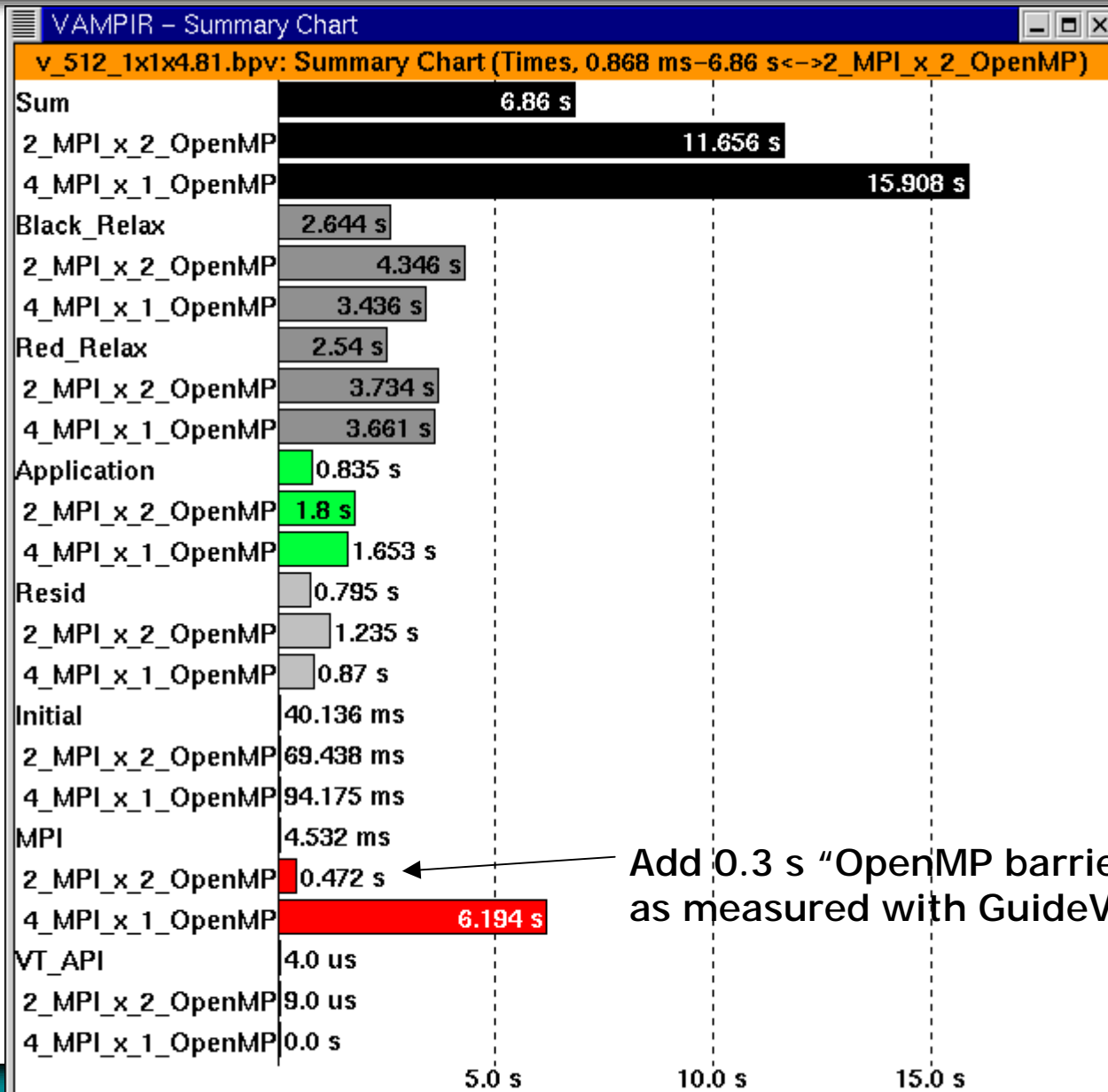
MPI+OpenMP Example - 2 MPI x 2 OpenMP



MPI+OpenMP Example - 1 MPI x 4 OpenMP



MPI+OpenMP Example - Vampir compares the runs



← Add 0.3 s "OpenMP barrier-time"
as measured with GuideView

Future plans - Vampir

- Towards automatic performance analysis
 - improve user guidance in Vampir
 - add “assistant” module for inexperienced users
- Support for clustered shared–memory systems
 - support shared–memory programming models (threads, OpenMP)
 - expose cluster structure
 - aggregate information on SMP nodes
- Support for (very) large systems
 - new structured tracefile format
 - fine–grain interactive control over tracing
 - scalable displays
 - new Vampir structure (can exploit parallelism)

TotalView

- Symbolic debugging for C, C++, Fortran 77, 90, HPF
- Multi-process debugging for MPI, PVM, distributed processes, HPF, OpenMP, threads
- Fast, easy to learn, easy to use GUI
- Platforms;
 - Linux86, LinuxAlpha, Tru64Alpha, SGI, Sun SPARC, IBM RS6000, IBM SP2, Fujitsu VPP, Cray, NEC SX, NEC Cenju, Hitachi SR, Quadrics CS, Lynx Real-Time OS, CSPI (vxworks)
 - HP port in progress
- *TotalView is the undisputed market leader in multi-processor debugging*



- On a new process

```
% totalview foo -a arguments to foo
```

- On a core file

```
% totalview foo core
```

- To attach to a running process

```
% totalview
```

- To debug e.g. MPI/mpich programs

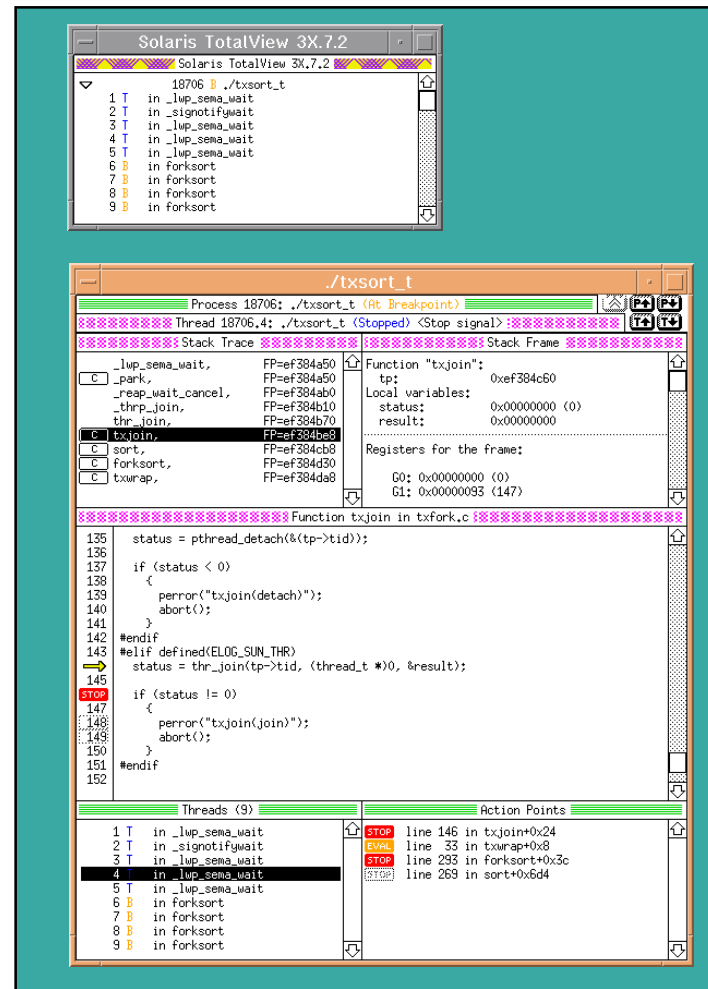
```
% mpirun -np 3 -tv foo
```

Totalview will start up on the first process and acquire all of the others automatically.

TotalView - Basic Display



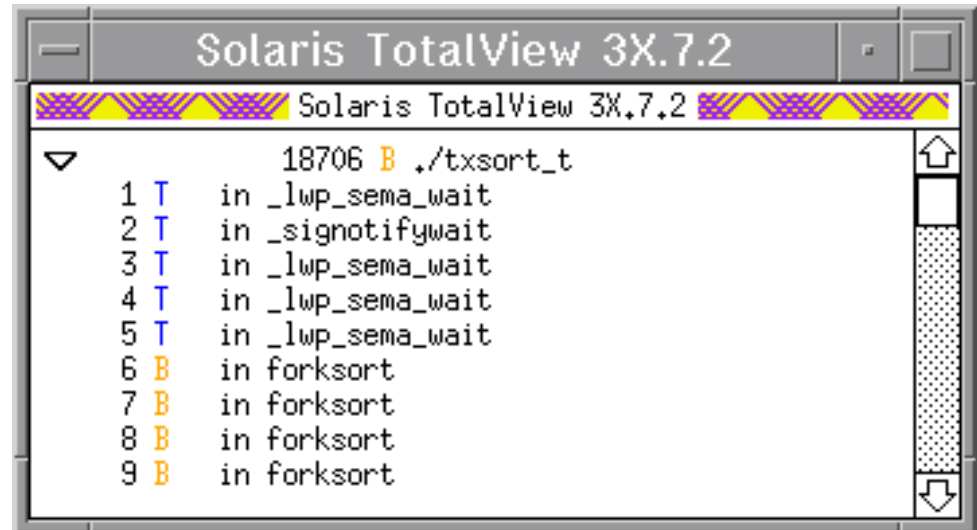
- Root Window shows processes under TotalView's control
- Process Window shows information about a single process
- GUI is fast and very "active"
 - single click breakpoints
 - single click "dive"
 - single character accelerators



TotalView - Root Window



- Shows processes and threads under TotalView's control
- Toggle allows expansion / collapse of the thread list
- "Diving" on a process or thread opens a new process window
- "Selecting" a process or thread changes the focus of an existing process window
- Machine / node name is listed when processes are distributed
- Colored letters indicate state B=breakpoint, R=running, etc.



TotalView - Process Window



The screenshot displays the TotalView Process Window for a process named `./txsort_t`. The main window shows the thread `Thread 18706.4: ./txsort_t (Stopped) <Stop signal>`. Below this, the **Stack Trace** and **Stack Frame** are visible. The stack trace lists several frames, with `txjoin` selected. The stack frame for `txjoin` shows local variables: `status: 0x00000000 (0)` and `result: 0x00000000`. Below the stack frame, the registers for the frame are shown: `G0: 0x00000000 (0)` and `G1: 0x00000093 (147)`.

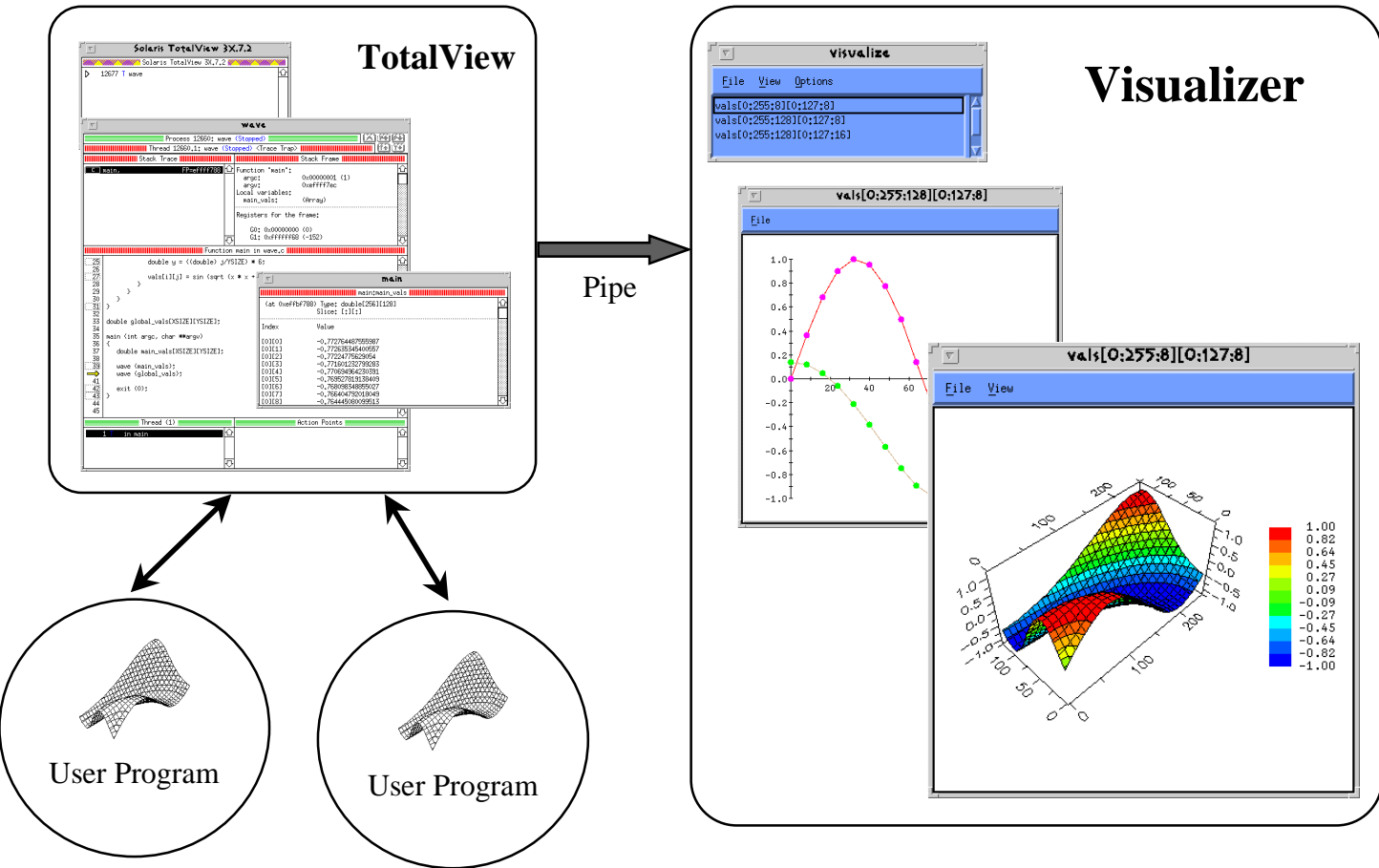
The **Function txjoin in txfork.c** is also visible, showing the following code:

```
135 status = pthread_detach(&(tp->tid));
136
137 if (status < 0)
138 {
139     perror("txjoin(detach)");
140     abort();
141 }
142 #endif
143 #elif defined(ELOG_SUN_THR)
144 → status = thr_join(tp->tid, (thread_t *)0, &result);
145
146 STOP if (status != 0)
147 {
148     perror("txjoin(join)");
149     abort();
150 }
151 #endif
152
```

At the bottom, the **Threads (9)** and **Action Points** panels are visible. The threads panel shows a list of threads, with thread 4 selected. The action points panel shows a list of action points, with a `STOP` point at line 146 in `txjoin+0x24`.

- Shows information about a single process / thread
 - Stack Trace
 - Registers and Locals
 - Source Code
 - Thread list
 - Action Points
- GUI is fast and very “active”
 - single click breakpoints
 - single click “dive”
 - single character accelerators
- Diving is a fast way to explore
 - variables and structures
 - function names
 - stack frames
 - threads
 - action points
- Values of registers and locals can be changed ‘on screen’

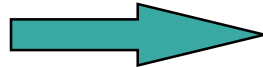
TotalView - Data Visualization



TotalView - MPI Message window



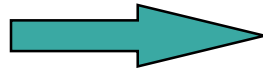
Communicator name and info



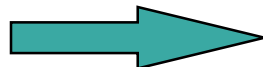
Non-blocking receive operations



Unmatched incoming messages



Non-blocking send operations



Dive on source or target to refocus process window.

Dive on buffer to see message contents

```
testsome.0
Message State for "testsome,0" (1288,1)
MPI_COMM_WORLD
Comm_size      3
Comm_rank      0
Pending receives
[0]
  Status        Pending
  Source        2 (testsome,2)
  Tag           0x00000000 (0)
  User Buffer    0x000605c0 -> 0x00000000 (0)
  Buffer Length  0x00000014 (20)
Unexpected messages
[0]
  Status        Complete
  Source        2 (testsome,2)
  Tag           0x00000002 (2)
  System Buffer  0x00000000
  Buffer Length  0x00000000 (0)
  Received Length 0x00000000 (0)
Non-blocking sends
[0]
  Status        Complete
  Target        2 (testsome,2)
  Tag           0x00000000 (0)
  Buffer         0x000605a0 -> 0x00000001 (1)
  Buffer Length  0x00000014 (20)
-----
MPI_COMM_WORLD_collective
Comm_size      3
Comm_rank      0
```


Access to Pallas Tools

Download **free** evaluation copies

<http://www.pallas.com>

Thanks for your attention!



Pallas GmbH
Hermülheimer Straße 10
D-50321 Brühl,
Germany

info@pallas.com
www.pallas.com