



pallas

Competence in High Performance Computing

Software Development Tools to Getting the Most from your Processor

Werner Krotz-Vogel <krotz@pallas.com>

UKHEK 2002, Daresbury
2002-12-10

Pallas GmbH
Hermülheimer Straße 10
D-50321 Brühl, Germany

info@pallas.com
www.pallas.com



- **System-Level- Problems**
 - Increasing performance gap between CPU, memory and interconnect
 - Problems with incorporate CPU pipelining and other parallel techniques (e.g. EPIC, Hyperthreading)
 - Cache and shared memory inconsistency effects
- **User-Level Problems**
 - Abstract programming concepts make optimization difficult
 - Optimization relies more and more on complex models like IPA(IPO), PFA(PGO) and other to achieve desired performance



- ILP Instruction-Level Parallelism
 - SSE, Software Pipelining, Scheduling, Vectorization
- OpenMP Thread-Level Parallelism
 - Multiple Share-Memory SMP Processors
 - OpenMP and Automatic Thread-level Parallelization
- MPI Process-Level Parallelism (1 CPU per Node)
 - Message Passing (MPI, PVM, shmem, GA, ARMCI)
 - High Performance Fortran (HPF), UPC, Co-array FTN
- Hybrid OpenMP + MPI Parallelism
 - Multi-CPU's per Node (2+ CPU's per Node)
 - MPI + OpenMP or MPI + Automatic

Solving the Performance Problem



- High Performance tools:
 - Help to diagnose **system-level** performance problems
 - Help to identify **user-level** performance bottlenecks
 - Assist the user in improving the applications
 - Help to get a better understanding of the involved problems

The more details that are known about the programming framework, the better the performance analysis can be.

Dr. Brian T. Smith - AHPCC



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

- PGI 4.0 x86 Compilers, Cluster Development Kit (CDK)
- PBS Pro 5.3 - flexible workload management system
- UNICORE, seamless GRID solution
- Intel 7.0 Software Development Tools (Libs, Compilers, VTune)
- Etnus TotalView 6.0, debugging complex code
- Vampir-3.0, Vampirtrace-3.0



Compilers & Tools ...



- PGI 4.0 x86 compilers , C, C++, F77, F90, HPF, pgprof, pgdbg
- SMP/OpenMP 1.2 support for C, C++, F77, F90
- Optimizing for AMD CPUs and Intel P3/P4
- ... plus convenient add-on's:
- parallel ScaLAPACK
- optimized BLAS, LAPACK
- MPI/mpich, PVM
- Open PBS
- Tutorial, examples
- Cluster management utilities



New Features



- Improved performance on P3, P4, and AMD CPUs
- Profile feedback code optimization
- new desitributed and shared memory debugger (PGBDG) and profiler (PGPROF)

Future support for Hammer64 architecture.



- Flexible workload management system
- Unified interface to all computing resources
 - POSIX batch standard, all major UNIXs supported, heterogeneous environment, SMPs & clusters, parallel jobs (MPI & Globus & UNICORE)
 - Single interface handles both interactive and batch processing
 - GUI tools for user and administrator
 - Fully configurable scheduler module -- any site policy

The screenshot displays the PBS Pro GUI with several sections:

- HOSTS:** A table listing servers and their resource usage.

Server	Max	Tot	Que	Run	Hld	Mat	Trn	Ext	Status
sp2003.nas.nasa.gov	0	15	7	6	2	0	0	0	Active
poseidon2.larc.nasa.gov	0	8	4	4	0	0	0	0	Active
davinci.nas.nasa.gov	0	13	11	2	0	0	0	0	Active
- QUEUES:** A table listing queues and their resource usage.

Queue	Max	Tot	Ena	Str	Que	Run	Hld	Mat	Trn	Ext	Type	Server
dedtime	0	0	yes	yes	0	0	0	0	0	0	Execution	sp2003.nas.nasa.gov
pbs	0	2	yes	yes	0	0	2	0	0	0	Route	sp2003.nas.nasa.gov
pending	0	13	yes	yes	7	6	0	0	0	0	Execution	sp2003.nas.nasa.gov
pending	0	7	yes	yes	4	3	0	0	0	0	Execution	poseidon2.larc.nasa.gov
pbs	0	0	yes	yes	0	0	0	0	0	0	Route	poseidon2.larc.nasa.gov
special	0	1	yes	yes	0	1	0	0	0	0	Execution	poseidon2.larc.nasa.gov
- JOBS:** A table listing individual jobs with their status and location.

Job id	Name	User	Time	Use	S	Queue	Server
71349.sp2003.nas	case24f	blaisdel	0	R	pending	sp2003.nas.nasa.gov	
71351.sp2003.nas	jein5-128	deese	0	Q	pending	sp2003.nas.nasa.gov	
71398.sp2003.nas	test.q	nance	0	R	pending	sp2003.nas.nasa.gov	
71402.sp2003.nas	r64	dbader	0	Q	pending	sp2003.nas.nasa.gov	
71408.sp2003.nas	57proc.job	tasos	0	Q	pending	sp2003.nas.nasa.gov	
71409.sp2003.nas	1proc_bignem.job	tasos	0	R	pending	sp2003.nas.nasa.gov	
71410.sp2003.nas	STDIH	worley	0	R	pending	sp2003.nas.nasa.gov	
23290.poseidon2.larc	case26b	blaisdel	0	Q	pending	poseidon2.larc.nasa.gov	
23327.poseidon2.larc	run	cheung	0	Q	pending	poseidon2.larc.nasa.gov	
23424.poseidon2.larc	a.out	wright	0	R	pending	poseidon2.larc.nasa.gov	
23429.poseidon2.larc	long.q	rpnance	0	R	pending	poseidon2.larc.nasa.gov	

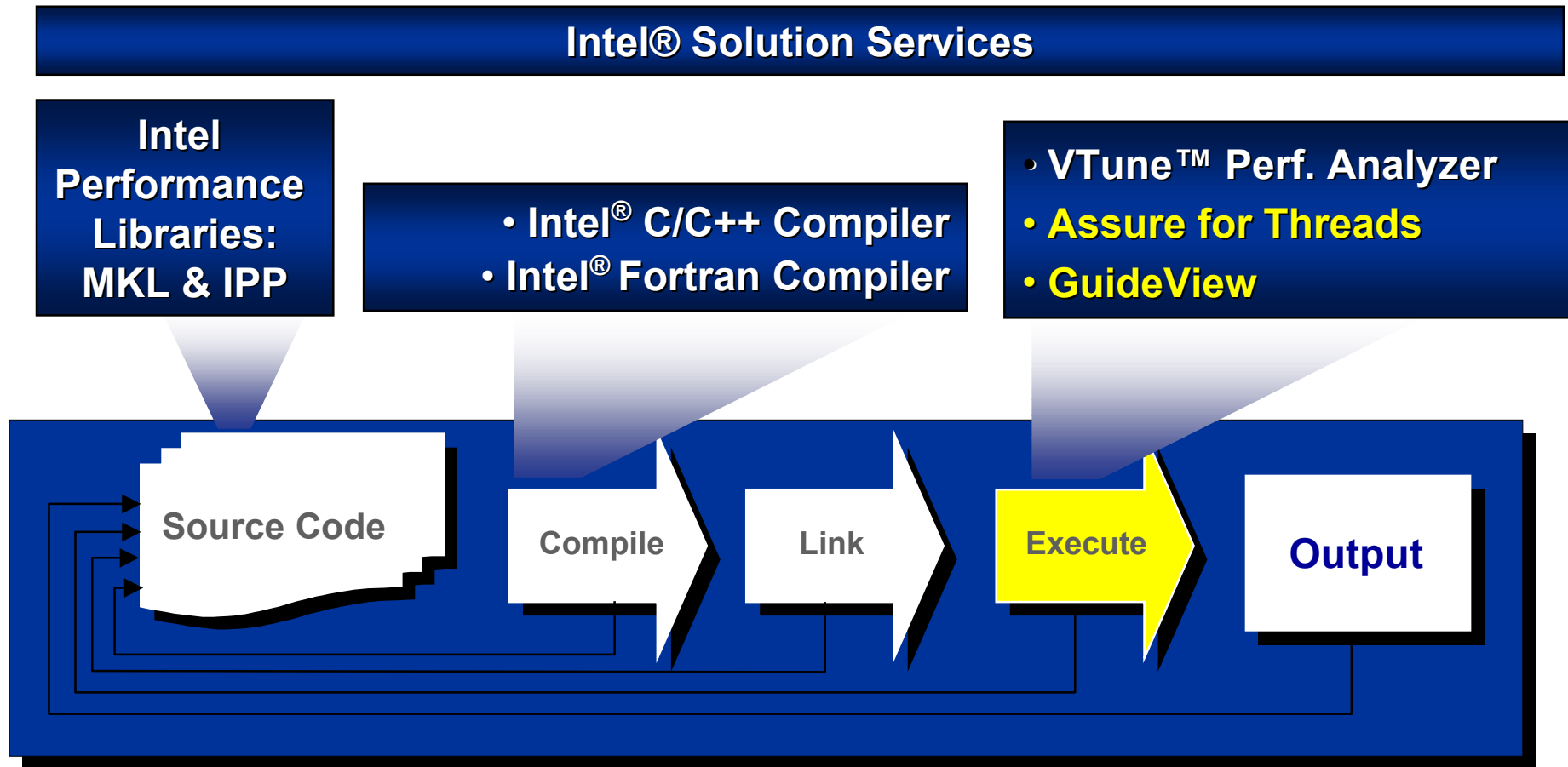


- Enhanced fault tolerance and reliability
 - new and optimized protocols for inter-daemon communication
- Advance reservations & global queuing capabilities
- Architecture specific features:
 - SGI Origin “cpusets” (new 6.5.8+ interface)
 - Checkpoint/Restart for Unicos and IRIX
 - IBM SP SMP running AIX 4.3.x & PSSP 3.2.x, Cray SV1 running Unicos 10, Solaris 2.7 & 2.8, FreeBSD 4.0
- Scheduling
 - Backfilling in “standard” FIFO scheduler
 - Suspend/resume on all systems
 - New modules for: SGI Origin 2000, Cray C90/J90/T3E

Intel SW Development Tools - Overview



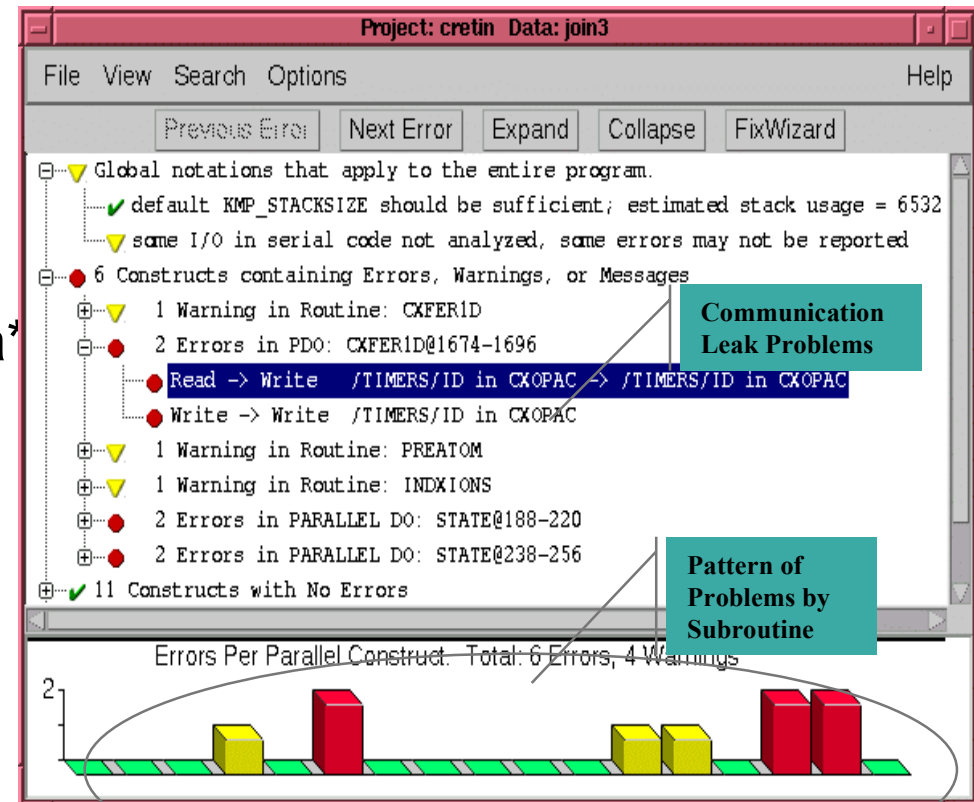
Intel® Software Development Products
for the Full Development Cycle





Assure Threading Verification

- Automatic debugger for multithreaded programs
 - Assure Parallel correctness
 - Simulates parallel execution
 - OpenMP, POSIX & NT, Java[™]
- Graphical interface
- Automatically locates both common and subtle coding errors
 - Boosts your productivity
 - Increases software quality





Deadlock Detection

The screenshot displays the Intel SW Development Tools interface. The top window, titled "Project: demo Data File: dead", shows the "Assure" tool's output. The output indicates a deadlock in the main thread, with the following details:

- Global notations that apply to the entire program.
- Deadlock in main was detected.
- Thread in work_1 wants lock on object held in work_2.
- Thread in work_2 wants lock on object held in work_1.
- Execution data for Thread main.

Below the output, two source code windows are shown. The left window, "Source: dead.c", displays the code for the `work_1` function. The right window, "Sink: dead.c", displays the code for the `work_2` function. Red circles and lines highlight the specific lines of code that caused the deadlock:

- In `work_1`, line 21 (`pthread_mutex_lock(& lock_2);`) is highlighted with a red circle and line.
- In `work_2`, line 36 (`pthread_mutex_lock(& lock_2);`) is highlighted with a red circle and line.

```
15
16 void *work_1( void *arg )
17 {
18     int i;
19
20     pthread_mutex_lock( & lock_1 );
21     pthread_mutex_lock( & lock_2 );
22
23     for ( i = 0; i < ITERS; ++i)
24         sum = sum + 1;
25
26     pthread_mutex_unlock( & lock_2 );
27     pthread_mutex_unlock( & lock_1 );
28
29     return 0;
30 }
31
32 void *work_2( void *arg )
33 {
34     int i;
35
36     pthread_mutex_lock( & lock_2 );
37     pthread_mutex_lock( & lock_1 );
38
39     for ( i = 0; i < ITERS; ++i)
40         sum = sum + 1;
41
42     pthread_mutex_unlock( & lock_1 );
43     pthread_mutex_unlock( & lock_2 );
44
45     return 0;
46 }
```



GuideView

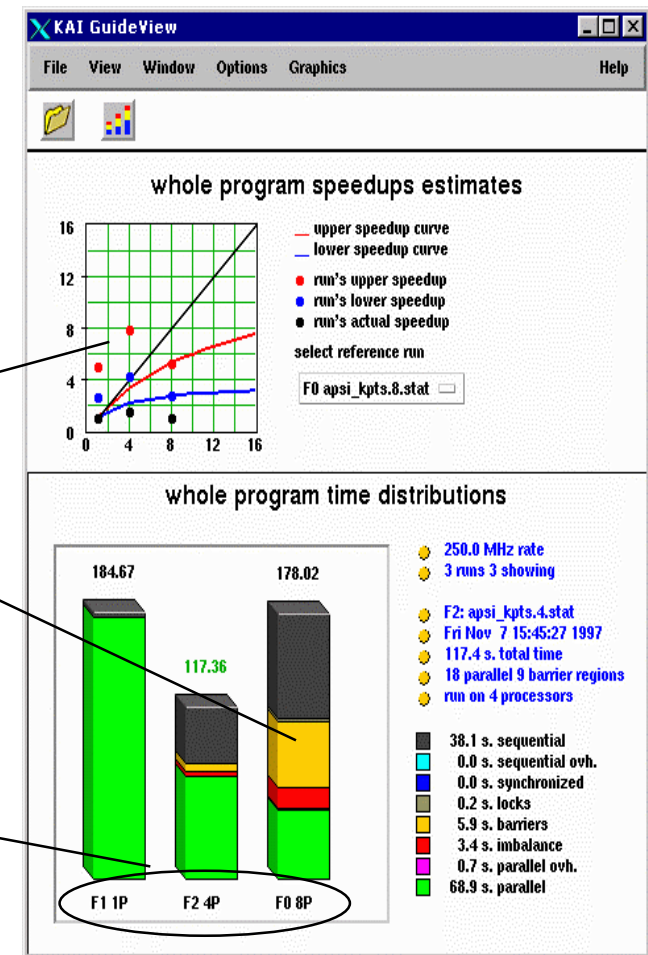
- Threading Tuning Tool
 - Tuning for optimal parallel (SMP) performance
 - Analyzes actual parallel execution
- Graphical User Interface
 - Visualization of collected and analyzed parallel execution data
 - Improves programmer productivity
- Being extended to include clusters

GuideView

Compare achieved to ideal Performance

Identify parallel bottlenecks such as Barriers, Locks, and Sequential time

Compare multiple runs



TotalView 6.0

- 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 (Motif)
- Platforms;
 - Linux86, LinuxAlpha, Tru64Alpha, SGI, Sun SPARC, HP, IBM RS6000, IBM SP2, Fujitsu VPP, Cray, NEC SX, NEC Cenju, Hitachi SR, Quadrics CS, Lynx Real-Time OS, CSPI (vxworks)
- Linux Compilers:
 - Apogee, gnu, Intel, KAI, Lahey/Fujitsu, PGI
- *TotalView is the undisputed market leader in multi-processor debugging*



- Attach to running processes

```
% totalview
```

(then select processes from Root Window)

- Start your program under the control of TotalView

```
% totalview myprog -a arguments to myprog
```

- Start TotalView with a core file

```
% totalview myprog core
```

- To debug e.g. MPI programs

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

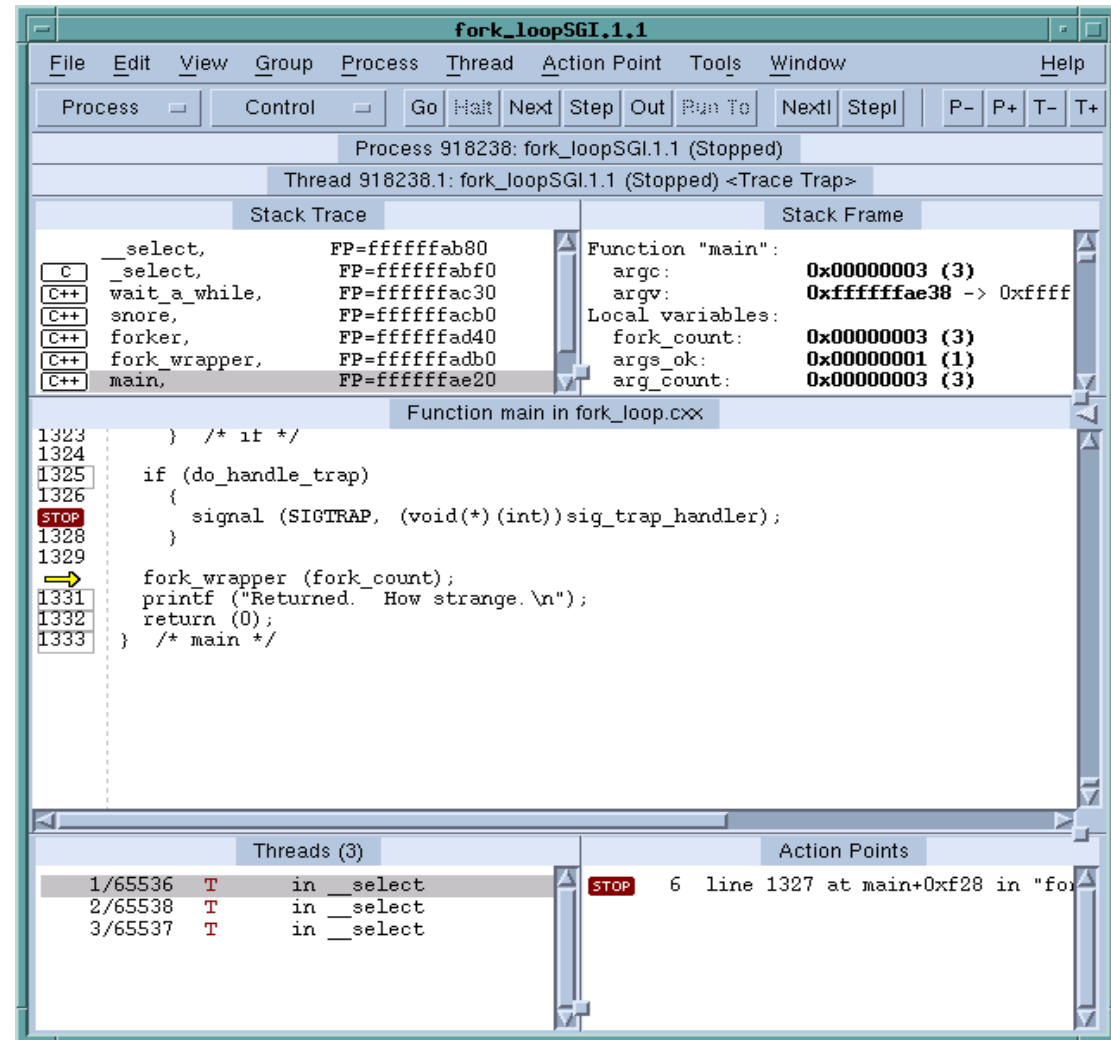
```
% tvmpimon foo -- localhost 3
```

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

TotalView 6 Graphical Interface



- Feature-rich menus
- Toolbar & keyboard control
- *Dive* on variables or functions to drill deeper (right mouse click or double left click)



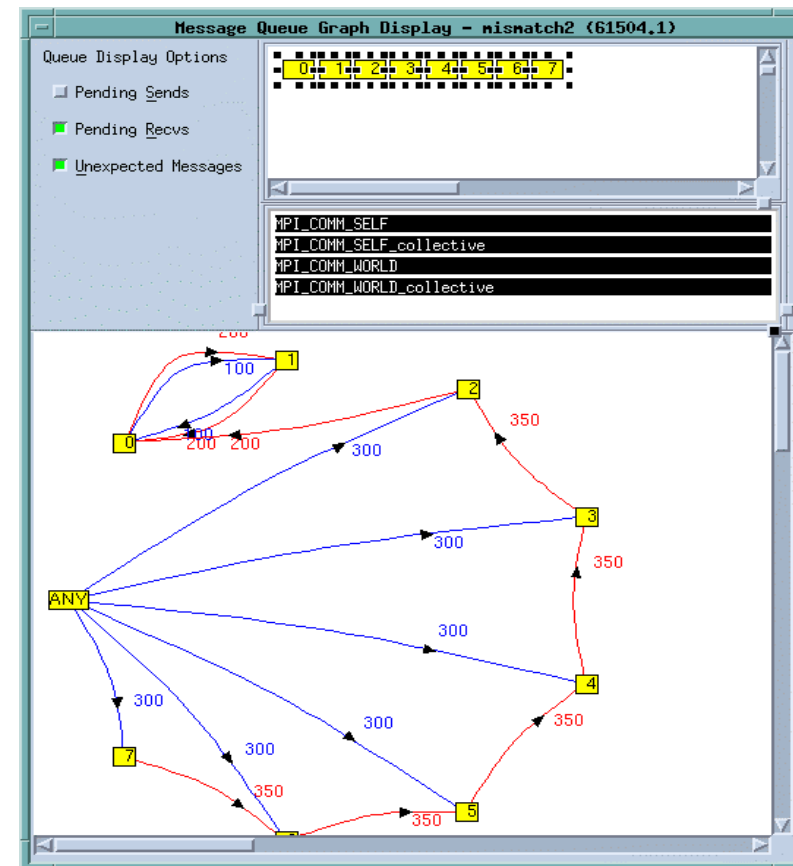


- Action point types:
 - Breakpoint
 - Barrier breakpoint
 - Conditional breakpoint
 - Evaluation point
 - Watchpoint
- Data Management
 - Diving... show me more info
 - Slicing... show me a dimension
 - Filtering... show me the elements I want
 - Sorting... in memory only
 - Statistics... about my array
 - Visualization... a picture is worth a thousand words

Message Queue Graph



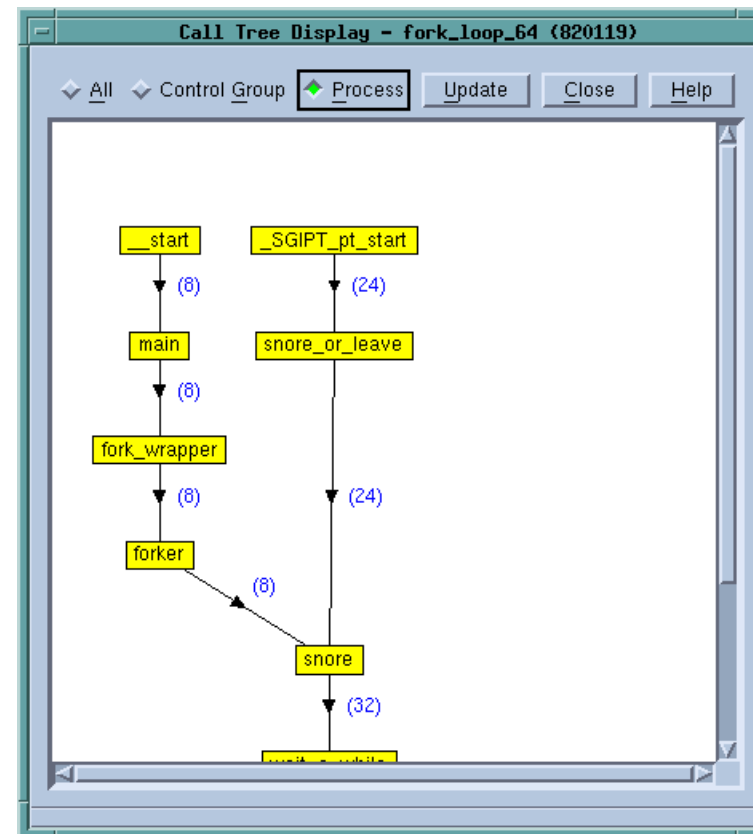
- Visualization of message queues
- Depicts inter-process communication
- Immediately spot the source of deadlock or other inter-process communication problems



Call Tree Graph



- Displays an amalgamated snapshot of your program's execution
- See potential bottlenecks or conflicts
- Review flow of execution

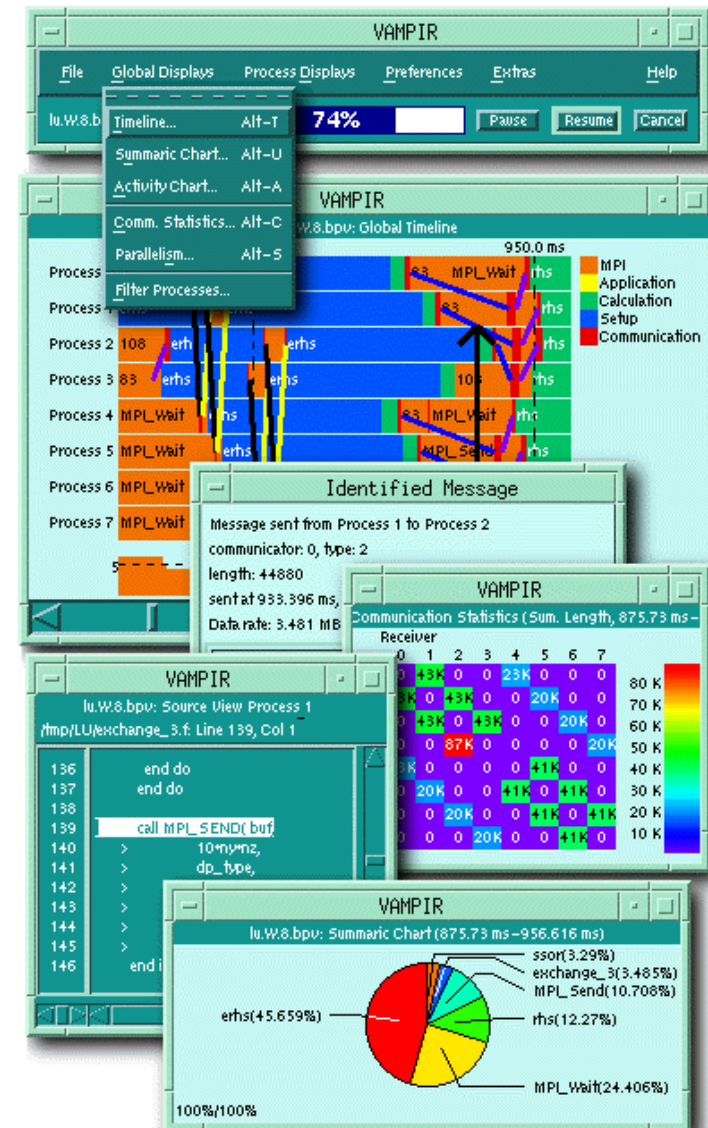




- dynamic process attachment
- examine complex data
- debugging parallel



Visualization and Analysis of MPI Programs





Vampir Features



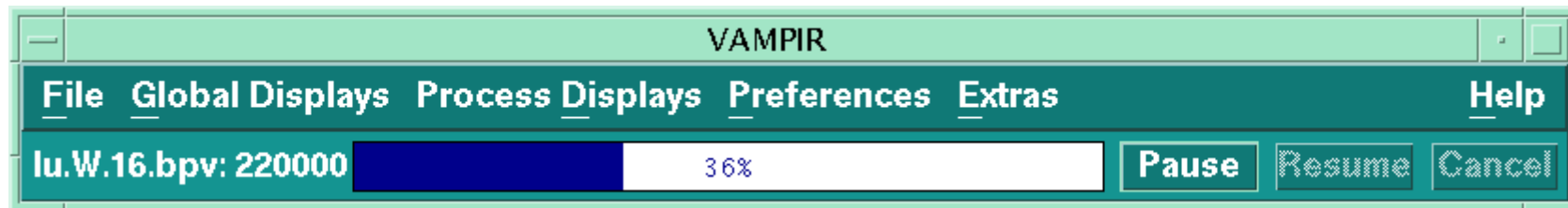
- Offline trace analysis for MPI (and others ...)
- Traces generated by **Vampirtrace** tool (``ld ... -IVT -lpmpi -lmpi``)
- Convenient user–interface
- **Scalability** in time and processor–space (STF)
- Excellent **zooming** and **filtering**
- High–performance graphics
- Display and analysis of **MPI**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



Vampir Main Window



Vampir 3.0 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)

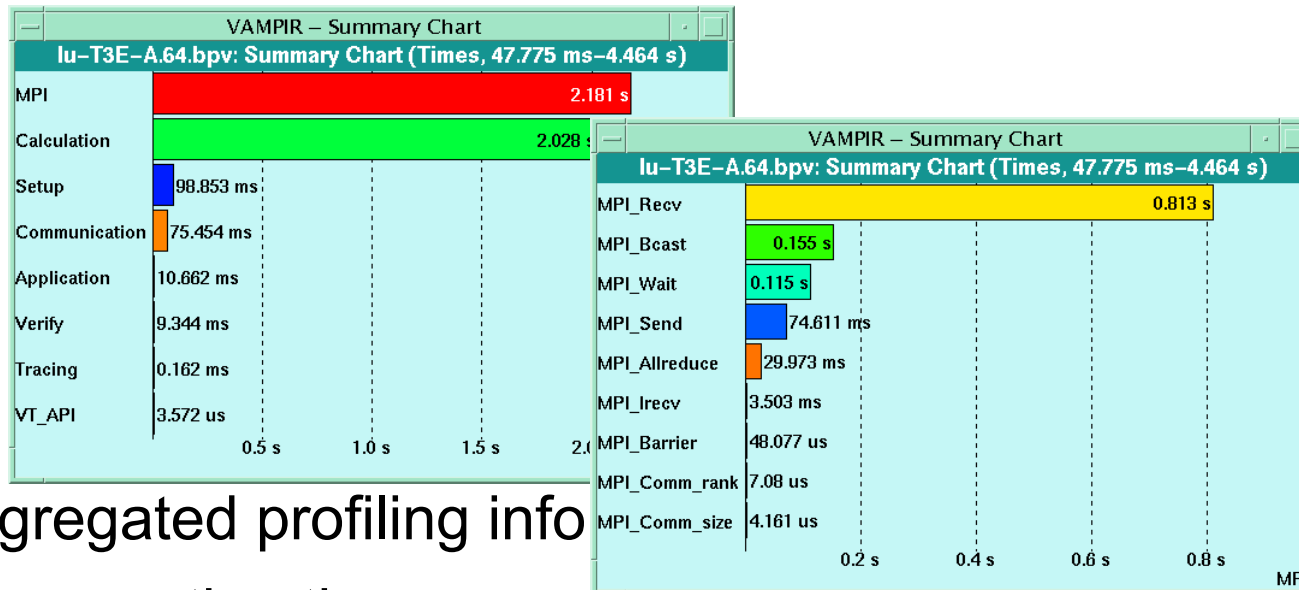


- Global displays show all selected processes
 - Summary Chart: aggregated profiling information
 - Activity Chart: presents per-process profiling information
 - Timeline: detailed application execution over time axis
 - Communication statistics: message statistics for each process pair
 - Global Comm. 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



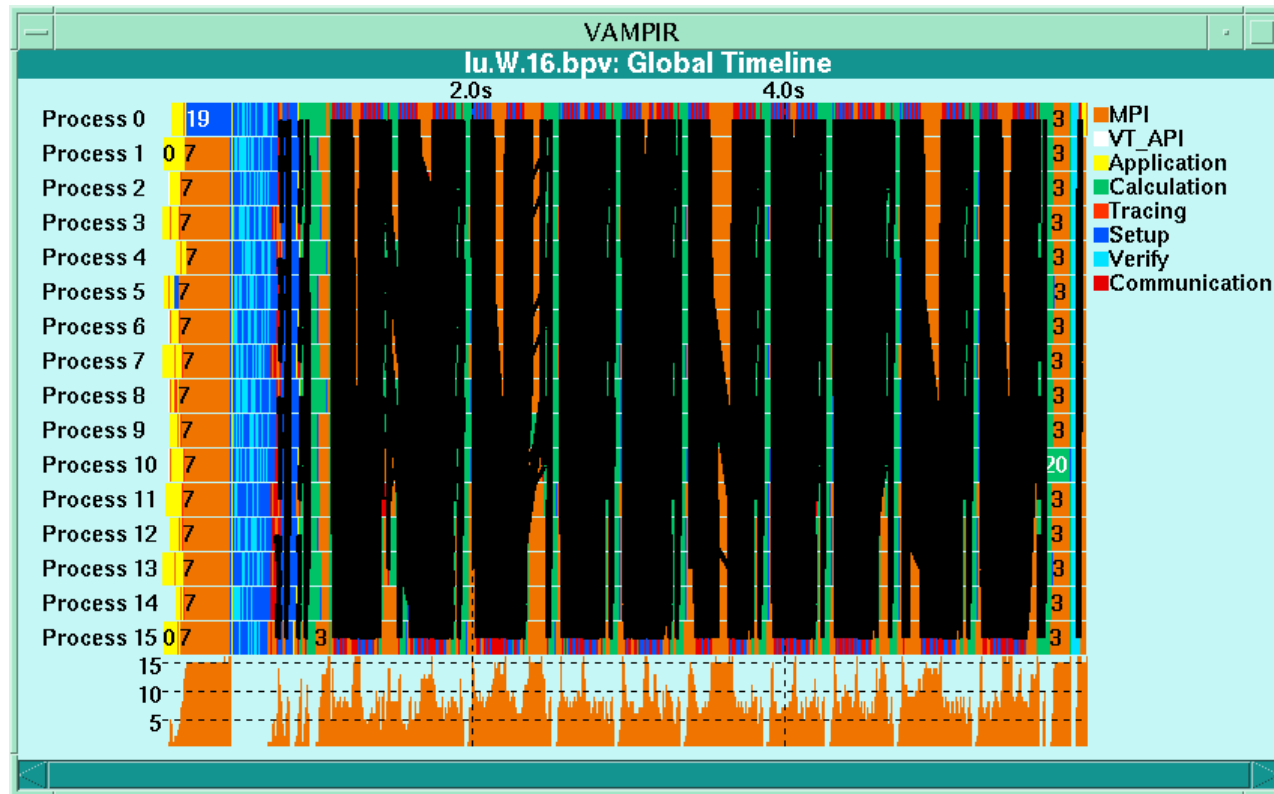
Summary Chart



- 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

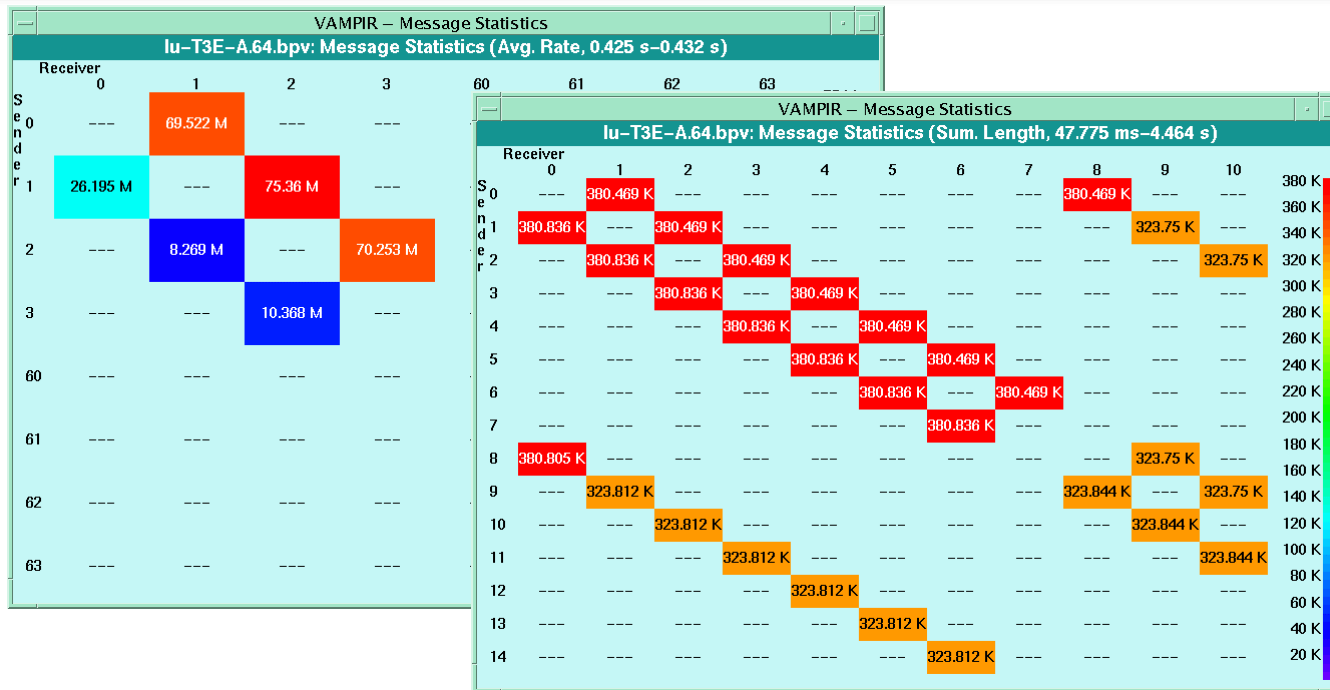
Click on message line

The screenshot displays the Vampir MPI timeline for 'lu.W.16.bpv: Global Timeline'. The timeline shows processes 0 through 9 with various activities like 'erhs', 'rhs', and 'MPI_Wait'. A callout box titled 'Identified Message' provides details: 'Message sent from Process 7 to Process 6', 'communicator: 0, type: 1', 'length: 21120', 'sent at 898.439 ms, received at 907.921 ms', and 'Data rate: 2.227 MBytes/sec'. Two source code windows are shown: 'lu.W.16.bpv: Source View Process 7' and 'lu.W.16.bpv: Source View Process 6'. The Process 7 window shows a call to 'MPI_SEND' at line 143. The Process 6 window shows a call to 'MPI_WAIT' at line 156. Arrows indicate the flow of information from the timeline to the source code and from the source code to the message details.

- Source-code references are displayed if recorded by Vampirtrace



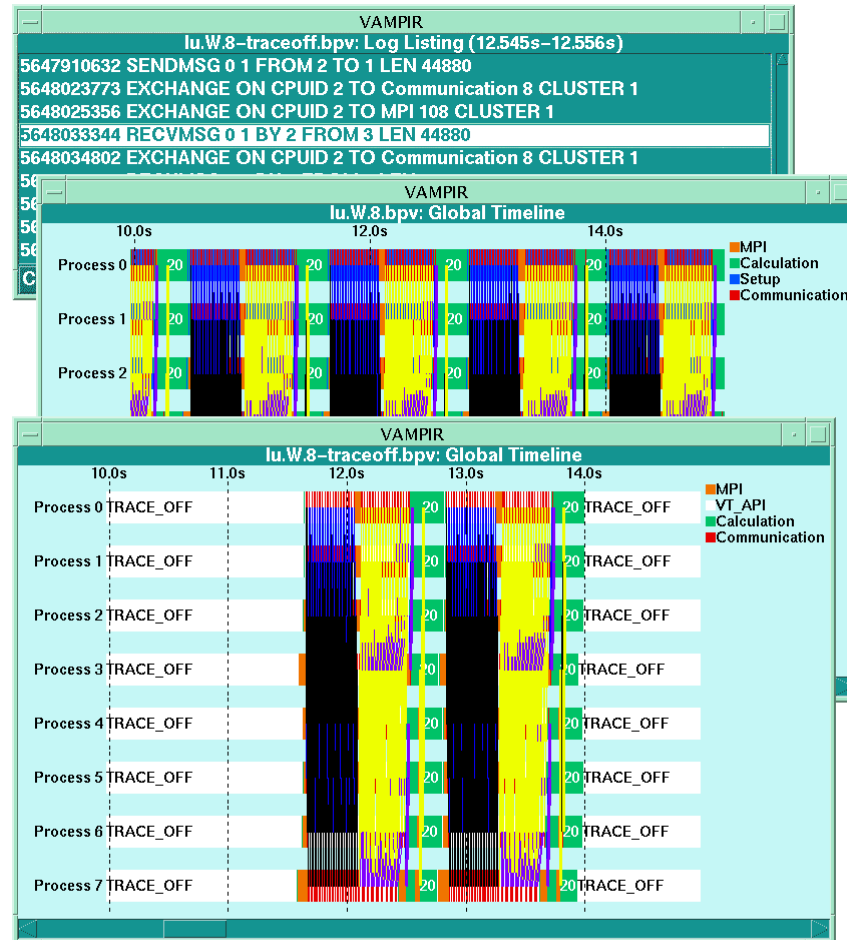
Communication Statistics



- 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

Vampirtrace

Tracing of
MPI and
Application
Events





Vampir - LIVE DEMO



- simple tracing
- spot some bottleneck
- scaling to long runs and many procs



Vampir Track Record



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

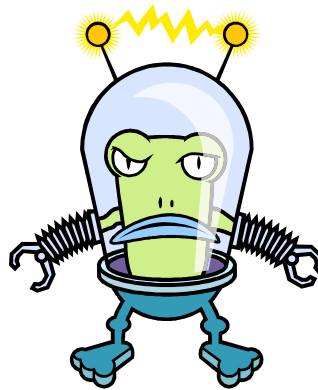
- URLs:
 - www.tc.cornell.edu/Edu/Tutor/Vampir
 - www.llnl.gov/sccd/lc/DEG/vampir/vampir.html
 - www.uni-karlsruhe.de/~Vampir
 - www.lrz-muenchen.de/services/software/parallel/vampir
 - www.hlrs.de/structure/support/parallel_computing/tools/performance/vampir.html

The* Uncertainty-Principle of Computing



By experience, you can compute a result

either **fast** or **correct**,



but never both !

* Werner's

Let the Tools do the work for you ...

Download **free evaluation copies**



<http://www.pallas.com>

Time for questions ... ?



Pallas GmbH
Hermülheimer Straße 10
D-50321 Brühl,
Germany

info@pallas.com
www.pallas.com