Profiling MPI Applications with VAMPIR: An Introduction

Claudia Schmidt
Technische Universität Dresden, Zentrum für Hochleistungsrechnen (ZHR)
schmidt@zhr.tu-dresden.de

Isabel Loebich, Rolf Rabenseifner
Höchstleistungsrechenzentrum Stuttgart (HLRS)
[loebich, rabenseifner]@hlrs.de

Bernd Mohr
Forschungszentrum Jülich, Zentralinstitut für angewandte Mathematik (ZAM)
b.mohr@fz-juelich.de

Event Tracing: Instrumentation , Monitoring , Trace

CPU A:

```
void master {
  trace(ENTER, 1);
  ...
  trace(SEND, B);
  send(B, tag, buf);
  ...
  trace(EXIT, 1);
}
```

CPU B:

```
void slave {
  trace(ENTER, 2);
  ...
  recv(A, tag, buf);
  trace(RECV, A);
  ...
  trace(EXIT, 2);
}
```

MONITOR

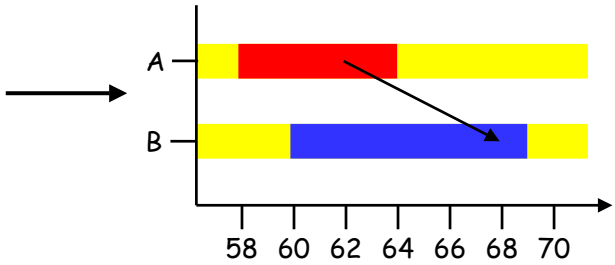
1	master
2	slave
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			

Event Tracing: “Timeline” Visualization

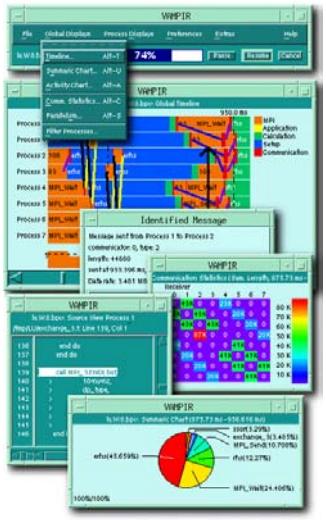
1	master
2	slave
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			



VAMPIR

- Visualization and Analysis of MPI Programs
- Originally developed by Forschungszentrum Jülich
- Now main development by Technical University Dresden
- Commercially distributed by PALLAS, Germany
- `http://www.pallas.de/pages/vampir.htm`

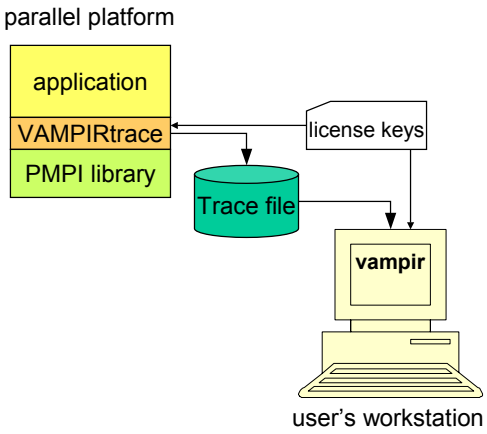


VAMPIR: Visualization and Analysis of MPI Resources

- performance analysis of MPI programs
- event tracing of MPI applications (message passing)
- based on post-mortem trace files
- x Window based system
- available for most Unix platforms

VAMPIRtrace and VAMPIR

- VAMPIRtrace must be linked with the user's MPI application
- VAMPIRtrace writes an trace file
- VAMPIR is the graphical tool to analyze the trace file
- VAMPIR may run
  - locally, or
  - via X window



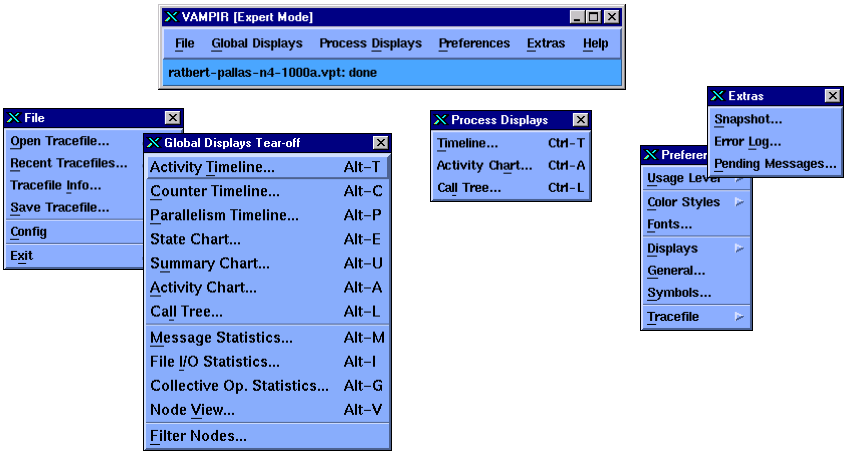
VAMPIR: Tracing

- automatic tracing of MPI activities
  (requires linking against trace library)
- source instrumentation necessary for subroutine event tracing
- each subroutine can be traced by means of the VAMPIR trace library
  - Fortran instrumentation is supported by an automatic source code instrumenter VAMPIRinst
  - C/C++ needs to be instrumented by hand
- tracing can be enabled / disabled for executables via VAMPIRprep

VAMPIR: Views

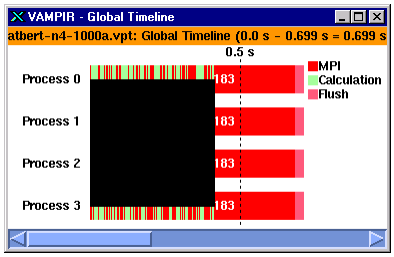
- process displays
  - Activity Timeline
  - Counter Timeline
  - Parallelism Timeline
  - State Chart
  - Summary Chart
  - Activity Chart
  - Call Tree
- Statistics
  - Messages
  - File I/O
  - Global Operations
- trace comparison
  - Summary Chart and local Activity Chart
- zooming and filtering
  - arbitrary time intervals
  - mouse-based zooming
  - arbitrary process sets
  - selection of messages
  - arbitrary selection and grouping of subroutines

VAMPIR - Main Pull Downs



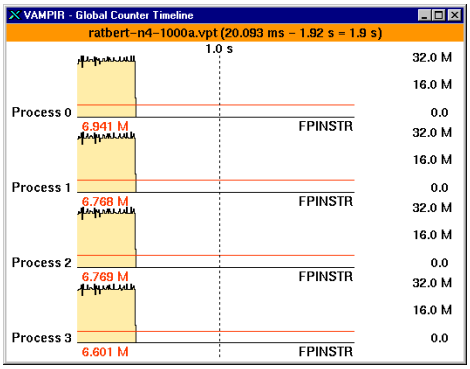
Global Timeline

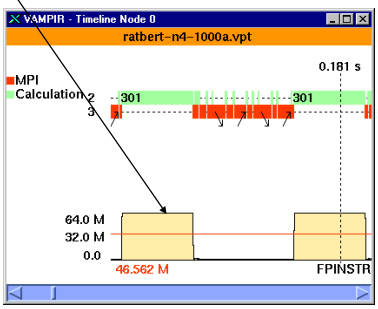
- acts as master display for statistic displays
- one timeline per process
- colored subroutine classes
- subroutine name or number depending on screen space
- message lines connecting sender and receiver
- additional message information on request (simple mouse click on a message)



VAMPIR: Counter Timeline

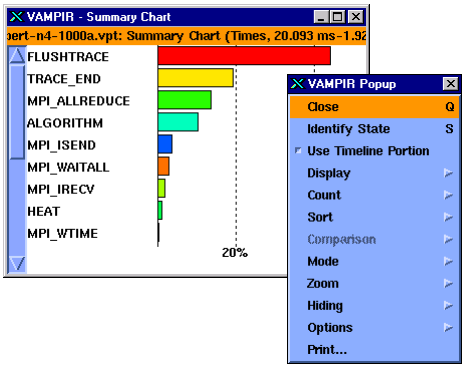
- performance / event counters for processes (MFLOPS, instructions, etc.)
- shows information for arbitrary time interval



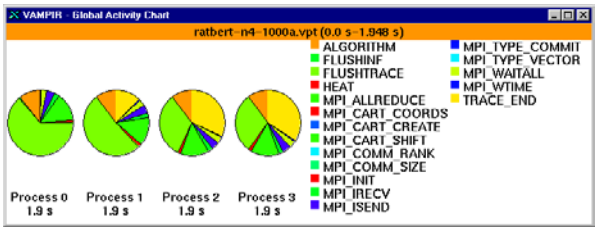


VAMPIR: Summary Chart

- Accumulated time consumed by subroutines / classes
- displays times or occurences for overall runtime or for a selected time interval
- gives information about global activities (MPI and Calculation) or for each traced subroutine and MPI activity



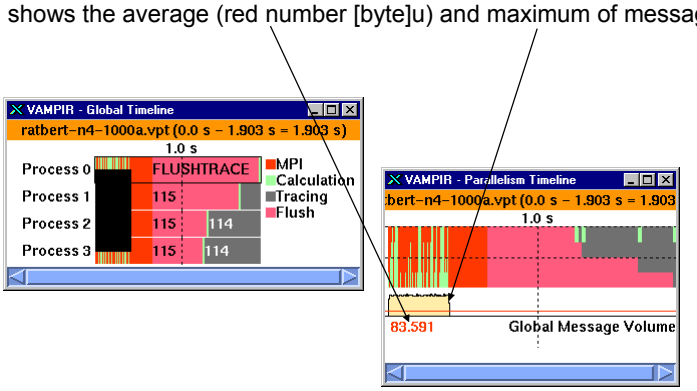
VAMPIR: Activity Charts



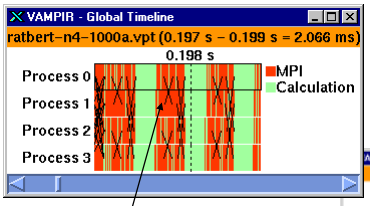
- for all or single processes
- pie chart, histogram or table
- linear or logarithmic scale
- hide dominating parts
- number of calls or execution time (absolute / relative)
- selection of arbitrary time interval using a timeline diagram

VAMPIR: Parallelism Timeline

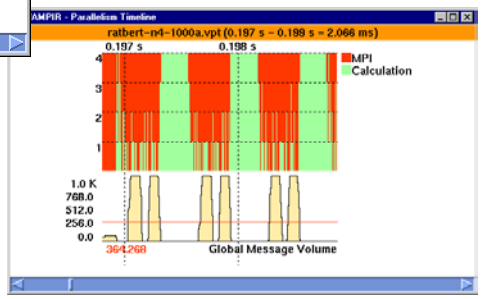
- parallelism view for the complete timeline and for each special time interval
- shows the average (red number [byte]u) and maximum of message volume

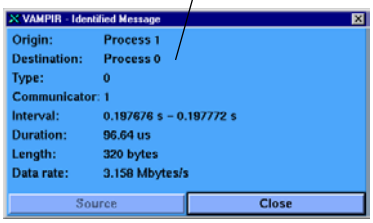


VAMPIR: Parallelism Timeline for an interval



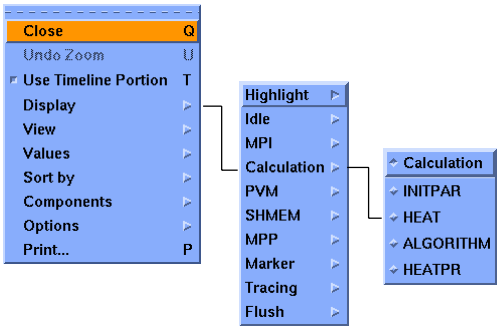
- detailed overview messages and parallelism of calculation

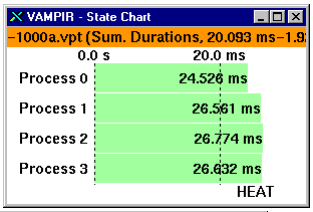


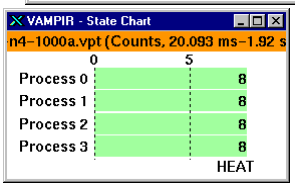


VAMPIR: State Chart

- information about selected activities (per chart only one activity)
  - number of counts
  - sum, average, minimum/maximum of duration

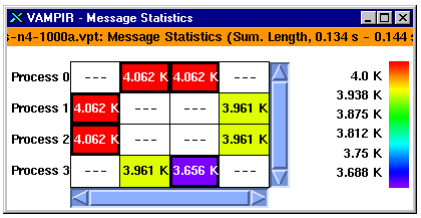


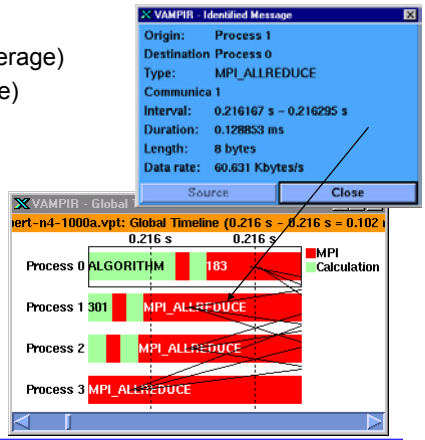




VAMPIR: Message Statistics

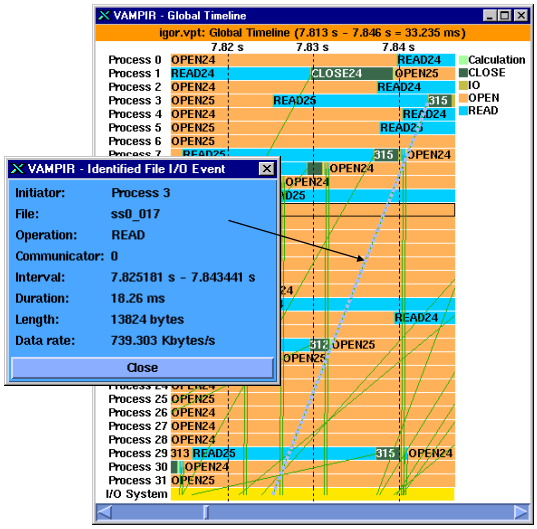
- global statistics for messages (length, time)
- for each sender / receiver pair:
  - message length (min, max, sum, average)
  - transmission rate (min, max, average)
- selection of arbitrary time interval using a timeline diagram





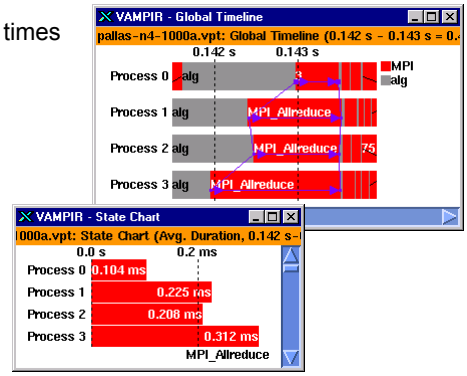
VAMPIR: File I/O Statistics

- special colored lines mark the file I/O in timeline (yellow line is the disk)
- gives an insight into different length of read, writes, open and close activities
- possibility to see, if the activity is cache or disk I/O



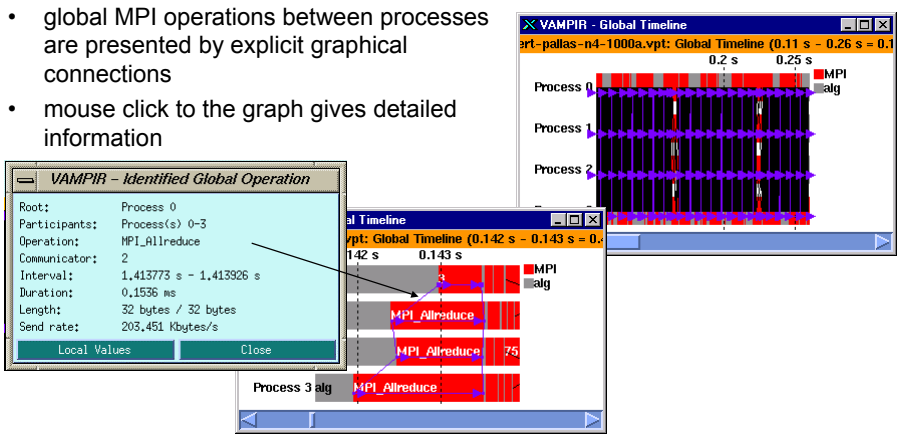
VAMPIR: Global Operation Statistics

- global MPI operations between processes are presented by explicit graphical connections
- a state chart shows the different times for an operation for all or some selected processes
  (so for times, sum, minimum and maximum of duration, counts)



VAMPIR: Global Operation Statistics

- global MPI operations between processes are presented by explicit graphical connections
- mouse click to the graph gives detailed information



Vampirtrace

- Commercial product of Pallas, Germany

- Library for Tracing of MPI and Application Events
  - records point-to-point communication
  - records user subroutines (on request)
  - records collective communication (V2.0)
  - records MPI–2 I/O operations (V2.0)
  - records source–code information (V2.0 some platforms)
  - support for shmem (V2.0 Cray T3E)
- uses the MPI profiling interface

`http://www.pallas.de/pages/vampirt.htm`

Vampirtrace Instrumentation API (C/C++)

- Calls for recording user subroutines

```
#include "VT.h"

VT_symdef(123, "foo", "USER"); // once after MPI_Init!

void foo {
    VT_begin(123);              // 1st executable line
    ...
    VT_end(123);                // at EVERY exit point!
}
```

- Event numbers used must be unique
- Selective tracing through
  - `VT_traceoff();` # Turn tracing off
  - `VT_traceon();` # Turn tracing on

VT++.h - C++ class wrapper for Vampirtrace

```
#ifndef __VT_PLUSPLUS_
#define __VT_PLUSPLUS_
#include "VT.h"
class VT_Trace {
  public:  VT_Trace(int code) {VT_begin(code_ = code);}
          ~VT_Trace()         {VT_end(code_);}
  private: int code_;
};
#endif /* __VT_PLUSPLUS_ */
```

- Same tricks can be used to wrap other tracing APIs for C++ too
- Usage:

```
VT_symdef(123, "foo", "USER"); // symbol definition as before
void foo(void) {               // user subroutine to monitor
  VT_Trace vt(123);            // declare VT_Trace object in 1st line
  ...                          // => automatic tracing by ctor/dtor
}
```


Vampirtrace Instrumentation API (Fortran)

- Calls for recording user subroutines

```
      include 'VT.inc'
      integer ierr
      call VTSYMDEF(123, "foo", "USER", ierr)
C
      SUBROUTINE foo(...)
      include 'VT.inc'
      integer ierr
      call VTBEGIN(123, ierr)
C     ...
      call VTEND(123, ierr);
      END
```

- Selective tracing through
  - `call VTTRACEOFF()` # Turn tracing off
  - `call VTTRACEON()` # Turn tracing on

Vampirtrace: Runtime Configuration

- Trace collection and generation can be controlled by using a configuration file
  - trace file name, location, size
  - flush behavior
  - activation/deactivation of trace recording for specific processes, activities (groups of symbols), and symbols
- Activate a configuration file by setting environment variables

`VT_CONFIG`	name of configuration file (use absolute pathname if possible)
`VT_CONFIG_RANK`	MPI rank of process which should process configuration file (default: 0)

- Reduce trace file sizes
  - restrict event collection in a configuration file
  - use selective tracing functions (`VT_traceoff` / `VT_traceon()`)

Vampirtrace: Configuration File Example

```
# collect traces only for MPI ranks 1 to 5
TRACERANKS 1:5:1
# record at most 20000 records per rank
MAX-RECORDS 20000

# do not collect administrative MPI calls
SYMBOL MPI_comm*  off
SYMBOL MPI_cart*  off
SYMBOL MPI_group* off
SYMBOL MPI_type*  off

# do not collect USER events
ACTIVITY USER off
# except routine foo
SYMBOL foo on
```

VAMPIR at ZHR

- preparation for tracing (Fortran: VAMPIRprep; C/C++ per hand)
- at Origin2000
  - set environment: setsoft vampir
  - for registertraces: export VAMPIR_REGISTER_SET=1
  - link to: -L/licsoft/vampir/lib -lmpitrace -lmpi
- at T3E
  - link to: -L/licsoft/vampir/lib -lmpitracewithregs -lpcl \
    -L/opt/ctl/mpt/1.3.0.3/lib -lmpi
- mpirun -np .... a.out
- merge tracefiles: vptmerge .trace.* (creates a tracefile trace.vpt)

- call VAMPIR at Origin2000: vampir trace.vpt&

VAMPIR at HLRS

- for workstations in /client/bin (dfs): vampir
- Documentation:
  - VAMPIR man pages in /client/man
  - Postscript files in /client/doc/vampir-2.0

- for Paragon, Hitachi SR8000, NEC SX-4, and CRAY T3E
- set USE_VAMPIR
- link to:
  -L$VAMPIRpath -lVT -lmpi
  (NEC SX-4)
  -L$VAMPIRpath -lVT -lpmpi
  (CRAY T3E)
  -L$VAMPIRpath -lVT -lpmpi
  (Hitachi SR8000)
- Documentation:
  $VAMPIRdocu/userguide.ps

VAMPIR at NIC

- Preparation (once per session):
  export PAL_ROOT=/usr/local/vt
- if Vampirtrace API is used, add to compile flags:
  -I/usr/local/vt/include
- link to:
  -L/usr/local/vt/lib -lVT -lpmpi -lmpi
- Documentation:
  - /usr/local/vampir/doc/Vampir-userguide.ps.gz
  - /usr/local/vt/doc/VT20-userguide.pdf

First activities with VAMPIR

program analysis with VAMPIR

- vampir&
- open tracefile
- Global Timeline
  - zoom in
  - identify some messages
- Summary Chart
  - use all symbols
  - display times and occurences
- Activity Chart
  - use Timeline Portion
  - display only MPI activities
- Message Statistics
- look for parallelism
- Global Operations
- .....

VAMPIR Example: Heat Conduction Program (1)

2 steps: - trace file generation
- analysis

Parallel system:

- modify program: reduce number of iterations
- add VAMPIRtrace library to makefile:
  at HLRS: -L$(VAMPIRpath) -L/usr/local/mpi/lib -lfmpi -lVT -lpmpi -lmpi
  at ZHR: -L/licsoft/vampir/lib -lmpitrace -lmpi (Origin2000)
- make
- mpirun -n <nodes> <program>
- at HLRS: ftp resulting tracefile to workstation
- at ZHR: merge tracefiles

VAMPIR Example: Heat Conduction Program (2)

program analysis with VAMPIR

- vampir&
- open tracefile
- Global Timeline
  - zoom in
  - identify some messages
- Summary Chart
  - use all symbols
  - display times and occurrences
- Activity Chart
  - use Timeline Portion
  - display only MPI activities
- Message Statistics
- look for parallelism
- Global Operations
- .....