Mitglied der Helmholtz-Gemeinschaft

JÜLICH FORSCHUNGSZENTRUM

Principles and Practice of Application Performance Measurement and Analysis on Parallel Systems

Lecture 2: Practical Performance Analysis and Tuning

1. July 2011 |

Bernd Mohr

Institute for Advanced Simulation (IAS)
Jülich Supercomputing Centre (JSC)

JÜLICH FORSCHUNGSZENTRUM

CONTENT

- Fall-back
  - Simple timers
  - Hardware counter measurements
- Overview of some performance tools
  - mpiP, TAU, Vampir + Vampirtrace
- Practical Performance Analysis and Tuning
- Challenges and open problems in performance optimization
  - Heterogeneous systems
  - Automatic performance analysis: KOJAK/Scalasca
  - Beyond execution time and flops
  - Extremely large HPC systems

II-2

Fall-back: Home-grown Performance Tools

- If performance analysis and tuning tools are
  - not available
  - too complex and complicated

  it is still possible to do some simple measurements
- Time Measurements
  - gettimeofday()
  - clock_gettime()
  - ...
- Hardware Performance Counter Measurements
  - PAPI

II-3

Timer: gettimeofday()

- UNIX function
- Returns wall-clock time in seconds and microseconds
- Actual resolution is hardware-dependent
- Base value is 00:00 UTC, January 1, 1970
- Some implementations also return the timezone

```
#include <sys/time.h>

struct timeval tv;
double walltime;   /* seconds */

gettimeofday(&tv, NULL);
walltime = tv.tv_sec + tv.tv_usec * 1.0e-6;
```

II-4

Timer: clock_gettime()

- POSIX function
- For clock_id CLOCK_REALTIME
  returns wall-clock time in seconds and nanoseconds
- More clocks may be implemented but are not standardized
- Actual resolution is hardware-dependent

```
#include <time.h>

struct timespec tv;
double walltime;   /* seconds */

Clock_gettime(CLOCK_REALTIME, &tv);
walltime = tv.tv_sec + tv.tv_nsec * 1.0e-9;
```

II-5

Timer: getrusage()

- UNIX function
- Provides a variety of different information
  - Including user time, system time, memory usage, page faults, etc.
  - Information provided system-dependent!

```
#include <sys/resource.h>

struct rusage ru;
double usrtime;   /* seconds */
int memused;

getrusage(RUSAGE_SELF, &ru);
usrtime = ru.ru_utime.tv_sec +
          ru.ru_utime.tv_usec * 1.0e-6;
memused = ru.ru_maxrss;
```

II-6

Timer: Others

- MPI provides portable MPI wall-clock timer

```
#include <mpi.h>
double walltime;   /* seconds */

walltime = MPI_Wtime();
```

  - Not required to be consistent/synchronized across ranks!
- Same for OpenMP 2.0 (!) programming

```
#include <omp.h>
double walltime;   /* seconds */

walltime = omp_get_wtime();
```

- Hybrid MPI/OpenMP programming?
  - Interactions between both standards (yet) undefined

Timer: Others

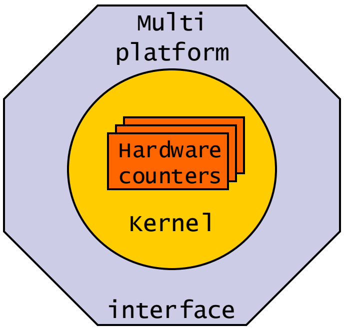
- Fortran 90 intrinsic subroutines
  - cpu_time()
  - system_clock()
- Hardware Counter Libraries
  - Vendor APIs (PMAPI, HWPC, libhpm, libpfm, libperf, ...)
  - PAPI

What Are Performance Counters

- Extra logic inserted in the processor to count specific events
- Updated at every cycle
- Strengths
  - Non-intrusive
  - Very accurate
  - Low overhead
- Weaknesses
  - Provides only hard counts
  - Specific for each processor
  - Access is not appropriate for the end user nor well documented
  - Lack of standard on what is counted

II-9

Hardware Counters Interface Issues



- Kernel level issues
  - Handling of overflows
  - Thread accumulation
  - Thread migration
  - State inheritance
  - Multiplexing
  - Overhead
  - Atomicity
- Multi-platform interfaces
  - The Performance API - PAPI
    - University of Tennessee, USA
  - LIKWID
    - University of Erlangen, Germany

II-10

Hardware Measurement

- Typical measured events account for:
  - Functional units status
    - Float point operations
    - Fixed point operations
    - Load/stores
  - Access to memory hierarchy
  - Cache coherence protocol events
  - Cycles and instructions counts
  - Speculative execution information
    - Instructions dispatched
    - Branches mispredicted

II-11

Hardware Metrics

- Typical Hardware Counter
  - Cycles / Instructions
  - Floating point instructions
  - Integer instructions
  - Load/stores
  - Cache misses
  - TLB misses
- Useful derived metrics
  - IPC - instructions per cycle
  - Float point rate
  - Computation intensity
  - Instructions per load/store
  - Load/stores per cache miss
  - Cache hit rate
  - Loads per load miss
  - Loads per TLB miss
- Derived metrics allow users to correlate the behavior of the application to one or more of the hardware components
- One can define threshold values acceptable for metrics and take actions regarding program optimization when values are below/above the threshold

II-12

Accuracy Issues

- Granularity of the measured code
  - If not sufficiently large enough, overhead of the counter interfaces may dominate
- Pay attention to what is not measured:
  - Out-of-order processors
  - Sometimes speculation is included
  - Lack of standard on what is counted
    - Microbenchmarks can help determine accuracy of the hardware counters

II-13

Hardware Counters Access on Linux

- Linux had not defined an out-of-the-box interface to access the hardware counters!
  - Linux Performance Monitoring Counters Driver (PerfCtr) by Mikael Pettersson from Uppsala X86 + X86-64
    - Needs kernel patching!
      - http://user.it.uu.se/~mikpe/linux/perfctr/
  - Perfmon by Stephane Eranian from HP – IA64
    - It was being evaluated to be added to Linux
      - http://www.hpl.hp.com/research/linux/perfmon/
- **Linux 2.6.31**
  - Performance Counter subsystem provides an abstraction of special performance counter hardware registers

II-14

Utilities to Count Hardware Events

- There are utilities that start a program and at the end of the execution provide **overall** event counts
  - hpmcount (IBM)
  - CrayPat (Cray)
  - pfmon from HP (part of Perfmon for AI64)
  - psrun (NCSA)
  - cputrack, har (Sun)
  - perfex, ssrun (SGI)
  - perf (Linux 2.6.31)

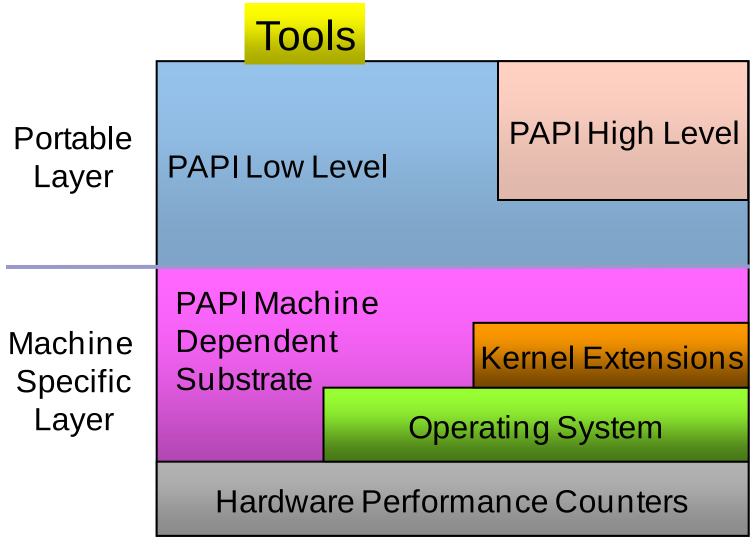
II-15

Hardware Counters: PAPI

- Parallel Tools Consortium (PTools) sponsored project
- **P**erformance **A**pplication **P**rogramming **I**nterface
- Two interfaces to the underlying counter hardware:
  - The high-level interface simply provides the ability to start, stop and read the counters for a specified list of events
  - The low-level interface manages hardware events in user defined groups called EventSets
- Timers and system information
- C and Fortran bindings
- Experimental PAPI interface to performance counters support in the linux 2.6.31 kernel
- http://icl.cs.utk.edu/papi/

II-16

PAPI Architecture



II-17

PAPI Predefined Events

- Common set of events deemed relevant and useful for application performance tuning (wish list)
  - papiStdEventDefs.h
  - Accesses to the memory hierarchy, cache coherence protocol events, cycle and instruction counts, functional unit and pipeline status
  - Run PAPI papi_avail utility to determine which predefined events are available on a given platform
  - Semantics may differ on different platforms!
- PAPI also provides access to native events on all supported platforms through the low-level interface
  - Run PAPI papi_native_avail utility to determine which predefined events are available on a given platform

II-18

PAPI avail Utility

```
% papi_avail -h
This is the PAPI avail program.
It provides availability and detail information
for PAPI preset and native events.  Usage:

    papi_avail [options] [event name]
    papi_avail TESTS_QUIET

Options:

  -a            display only available PAPI preset events
  -d            display PAPI preset event info in detailed format
  -e EVENTNAME  display full detail for named preset or native event
  -h            print this help message
  -t            display PAPI preset event info in tabular format (default)
```

II-19

PAPI Preset Listing

```
(derose@jaguar1) 184% papi_avail
LibLustre: NAL NID: 0005dc02 (2)
Lustre: OBD class driver Build Version: 1, info@clusterfs.com
Test case avail.c: Available events and hardware information.
-------------------------------------------------------------------------
Vendor string and code    : AuthenticAMD (2)
Model string and code     : AMD K8 (13)
CPU Revision              : 1.000000
CPU Megahertz             : 2400.000000
CPU's in this Node        : 1
Nodes in this System      : 1
Total CPU's               : 1
Number Hardware Counters : 4
Max Multiplex Counters    : 32
-------------------------------------------------------------------------
Name          Code          Avail   Deriv   Description (Note)
PAPI_L1_DCM   0x80000000    Yes     Yes     Level 1 data cache misses ()
PAPI_L1_ICM   0x80000001    Yes     Yes     Level 1 instruction cache misses ()
PAPI_L2_DCM   0x80000002    Yes     No      Level 2 data cache misses ()
PAPI_L2_ICM   0x80000003    Yes     No      Level 2 instruction cache misses ()
PAPI_L3_DCM   0x80000004    No      No      Level 3 data cache misses ()
PAPI_L3_ICM   0x80000005    No      No      Level 3 instruction cache misses ()
PAPI_L1_TCM   0x80000006    Yes     Yes     Level 1 cache misses ()
PAPI_L2_TCM   0x80000007    Yes     Yes     Level 2 cache misses ()
PAPI_L3_TCM   0x80000008    No      No      Level 3 cache misses ()
. . .
```

II-20

Example: papi_avail –e PAPI_L1_TCM (AMD Opteron)

```
Event name:                      PAPI_L1_TCM
Event Code:                      0x80000006
Number of Native Events:         4
Short Description:               |L1 cache misses|
Long Description:                |Level 1 cache misses|
Developer's Notes:               ||
Derived Type:                    |DERIVED_ADD|
Postfix Processing String:       ||
 |Native Code[0]: 0x40001e1c  DC_SYS_REFILL_MOES|
 |Number of Register Values: 2|
 |Register[0]: 0x20f          P3 Ctr Mask|
 |Register[1]: 0x1e43         P3 Ctr Code|
 |Native Event Description: |Refill from system. Cache bits: Modified Owner Exclusive Shared|

 |Native Code[1]: 0x40000037  IC_SYS_REFILL|
 |Number of Register Values: 2|
 |Register[0]: 0xf            P3 Ctr Mask|
 |Register[1]: 0x83           P3 Ctr Code|
 |Native Event Description: |Refill from system|

 |Native Code[2]: 0x40000036  IC_L2_REFILL|
 |Number of Register Values: 2|
 |Register[0]: 0xf            P3 Ctr Mask|
 |Register[1]: 0x82           P3 Ctr Code|
 |Native Event Description: |Refill from L2|

 |Native Code[3]: 0x40001e1b  DC_L2_REFILL_MOES|
 |Number of Register Values: 2|
 |Register[0]: 0x20f          P3 Ctr Mask|
 |Register[1]: 0x1e42         P3 Ctr Code|
 |Native Event Description: |Refill from L2. Cache bits: Modified Owner Exclusive Shared|
```


PAPI papi_native_avail Utility (AMD Opteron)

```
(derose@sleet) 187% papi_native_avail |more
Test case NATIVE_AVAIL: Available native events and hardware information.
-------------------------------------------------------------------------
Vendor string and code   : AuthenticAMD (2)
Model string and code    : AMD K8 Revision C (15)
CPU Revision             : 10.000000
CPU Megahertz            : 2193.406982
CPU's in this Node       : 2
Nodes in this System     : 1
Total CPU's              : 2
Number Hardware Counters : 4
Max Multiplex Counters   : 32
-------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.
Symbol  Event Code      Count
 |Short Description|
 |Long Description|
 |Derived|
 |PostFix|
The count field indicates whether it is a) available (count >= 1) and b) derived
 (count > 1)
FP_ADD_PIPE     0x40000000
 |Dispatched FPU ops - Revision B and later revisions - Speculative add pipe ops
 excluding junk ops|
 |Register Value[0]: 0xf            P3 Ctr Mask|
 |Register Value[1]: 0x100          P3 Ctr Code|

FP_MULT_PIPE    0x40000001
 |Dispatched FPU ops - Revision B and later revisions - Speculative multiply pipe
 ops excluding junk ops|
 |Register Value[0]: 0xf            P3 Ctr Mask|
 |Register Value[1]: 0x200          P3 Ctr Code|
. . .
```

High Level API

- Meant for application programmers wanting simple but accurate measurements
- Calls the lower level API
- Allows only PAPI preset events
- Eight functions:
  - `PAPI_num_counters`
  - `PAPI_start_counters`, `PAPI_stop_counters`
  - `PAPI_read_counters`
  - `PAPI_accum_counters`
  - `PAPI_flops`
  - `PAPI_flips`, `PAPI_ipc` (New in Version 3.x)
- Not thread-safe (Version 2.x)

II-23

Example: Quick and Easy Mflop/s

```
      program papiMflops
        parameter (N=1024)
        include "f77papi.h"
        integer*8 fpins
        real*4    realtm, cputime, mflops
        integer   ierr
        real*4    a(N,N)

        call random_number(a)
        call PAPIF_flops(realtm, cputime, fpins, mflops, ierr)
        do j=1,N
          do i=1,N
            a(i,j)=a(i,j)*a(i,j)
          end do
        end do
        call PAPIF_flops(realtm, cputime, fpins, mflops, ierr)
        print *,' realtime: ', realtm, ' cputime: ', cputime
        print *,' papi_flops: ', mflops, ' MFlop/s'
      end

% ./papiMflops
  realtime:     3.640159      cputime:     3.630502
  papi_flops:     29.67809      MFlops
```

II-24

General Events

```
      program papicount
         parameter (N=1024)
         include "f77papi.h"
         integer*8 values(2)
         integer events(2), ierr
         real*4 a(N,N)

         call random_number(a)
         events(1) = PAPI_L1_DCM
         events(2) = PAPI_L1_DCA
         call PAPIF_start_counters(events, 2, ierr)
         do j=1,N
           do i=1,N
             a(i,j)=a(i,j)*a(i,j)
           end do
         end do
         call PAPIF_read_counters(values, 2, ierr)
         print *,' L1 data misses  : ', values(1)
         print *,' L1 data accesses: ', values(2)
      end
% ./papicount
  L1 data misses  :                  13140168
  L1 data accesses:                 500877001
```

II-25

Low Level API

- Increased efficiency and functionality over the high level PAPI interface
- 54 functions
- Access to native events
- Obtain information about the executable, the hardware, and memory
- Set options for multiplexing and overflow handling
- System V style sampling (profil())
- Thread safe

II-26

CONTENT

- Fall-back
  - Simple timers
  - Hardware counter measurements
- Overview of some performance tools
  - mpiP, TAU, Vampir + Vampirtrace
- Practical Performance Analysis and Tuning
- Challenges and open problems in performance optimization
  - Heterogeneous systems
  - Automatic performance analysis: KOJAK/Scalasca
  - Beyond execution time and flops
  - Extremely large HPC systems

II-27

MPI Profiling: mpiP

- Scalable, light-weight MPI profiling library
- Generates detailed text summary of MPI behavior
  - Time spent at each MPI function callsite
  - Bytes sent by each MPI function callsite (where applicable)
  - MPI I/O statistics
  - Configurable traceback depth for function callsites
- Controllable from program using MPI_Pcontrol
  - Allows you to profile just one code module or cycle
  - Allows mpiP profile dumps mid-run
- Uses PMPI interface ⇨ only re-link of application necessary
- http://mpip.sourceforge.net/

II-28

mpiP Text Output Example

```
@ mpiP
@ Version: 3.1.1
// 10 lines of mpiP and experiment configuration options
// 8192 lines of task assignment to BlueGene topology information

@--- MPI Time (seconds) ---------------------------------------------
Task    AppTime    MPITime     MPI%
   0       37.7       25.2    66.89
// ...
8191       37.6         26    69.21
   *   3.09e+05   2.04e+05    65.88

@--- Callsites: 26 -------------------------------------------------
 ID Lev File/Address      Line Parent_Funct              MPI_Call
  1   0 coarsen.c          542 hypre_StructCoarsen       Waitall
// 25 similiar lines

@--- Aggregate Time (top twenty, descending, milliseconds) --------
Call                 Site       Time    App%    MPI%     COV
Waitall                21   1.03e+08   33.27   50.49    0.11
Waitall                 1   2.88e+07    9.34   14.17    0.26
// 18 similiar lines
```

II-29

mpiP Text Output Example (cont.)

```
@--- Aggregate Sent Message Size (top twenty, descending, bytes) --
Call                 Site      Count      Total       Avrg  Sent%
Isend                  11  845594460   7.71e+11        912  59.92
Allreduce              10      49152   3.93e+05          8   0.00
// 6 similiar lines

@--- Callsite Time statistics (all, milliseconds): 212992 ---------
Name          Site Rank     Count     Max    Mean      Min   App%   MPI%
Waitall         21    0    111096     275     0.1 0.000707  29.61  44.27
//  ...
Waitall         21 8191     65799     882    0.24 0.000707  41.98  60.66
Waitall         21    * 577806664     882   0.178 0.000703  33.27  50.49
// 213,042 similiar lines

@--- Callsite Message Sent statistics (all, sent bytes) -----------
Name          Site Rank     Count       Max      Mean    Min       Sum
Isend           11    0     72917 2.621e+05     851.1      8 6.206e+07
//...
Isend           11 8191     46651 2.621e+05      1029      8 4.801e+07
Isend           11    * 845594460 2.621e+05     911.5      8 7.708e+11
// 65,550 similiar lines
```

II-30

"Swiss Army Knife" of Performance Analysis : TAU

- Very portable tool set for instrumentation, measurement and analysis of parallel multi-threaded applications
- Instrumentation API supports choice
  - between profiling and tracing
  - of metrics (i.e., time, HW Counter (PAPI))
- Uses Program Database Toolkit (PDT) for C, C++, Fortran source code instrumentation
- Supports
  - Languages: C, C++, Fortran 77/90, HPF, HPC++, Java, Python
  - Threads: pthreads, Tulip, SMARTS, Java, Win32, OpenMP
  - Systems: same as KOJAK + Windows + MacOS + …
- http://tau.uoregon.edu/
- http://www.cs.uoregon.edu/research/pdt/

Tuning and Analysis Utilities

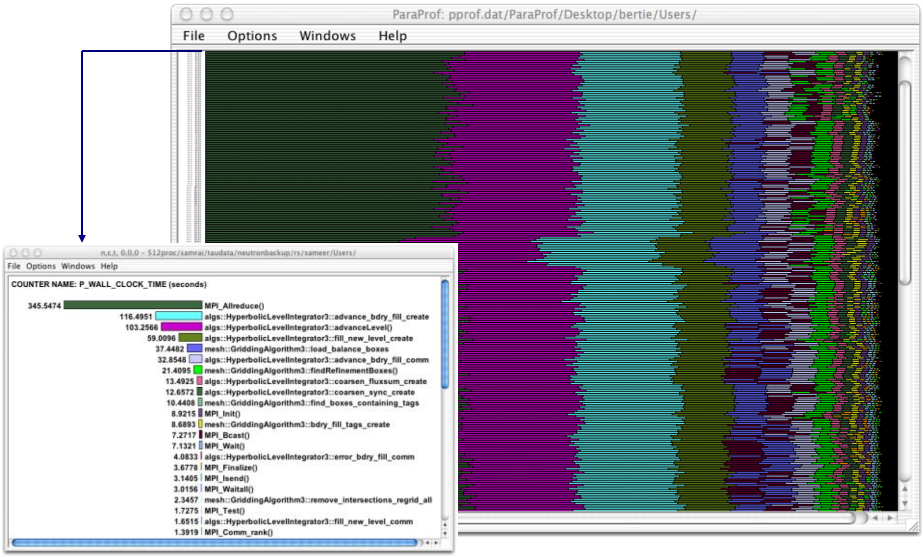
II-31

TAU Instrumentation

- Flexible instrumentation mechanisms at multiple levels
  - Source code
    - manual
    - automatic
      - C, C++, F77/90/95 (Program Database Toolkit (PDT))
      - OpenMP (directive rewriting with Opari)
  - Object code
    - pre-instrumented libraries (e.g., MPI using PMPI)
    - statically-linked and dynamically-loaded (e.g., Python)
  - Executable code
    - dynamic instrumentation (pre-execution) (DynInst)
    - virtual machine instrumentation (e.g., Java using JVMPI)
- Support for performance mapping
- Support for object-oriented and generic programming

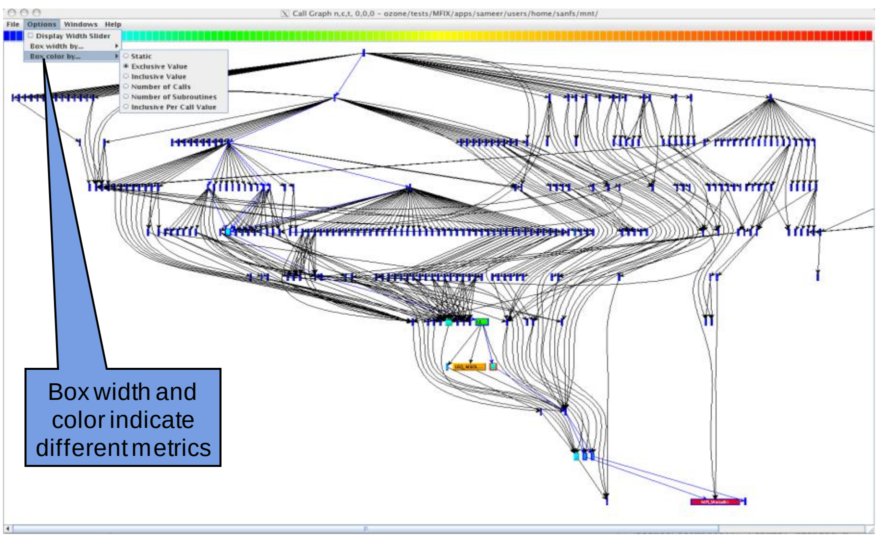
II-32

TAU Profiling, Large System



II-33

ParaProf –Callgraph View (MFIX)



II-34

TAU Usage

- Usage:
  - Specify programming model by setting `TAU_MAKEFILE` to one of `$<TAU_ROOT>/<arch>/lib/Makefile.tau-*`
    Examples from Linux cluster with Intel compiler and PAPI:
    - `MPI: Makefile.tau-icpc-papi-mpi-pdt`
    - `OMP: Makefile.tau-icpc-papi-pdt-openmp-opari`
    - `OMPI: Makefile.tau-icpc-papi-mpi-pdt-openmp-opari`
  - Compile and link with
    - `tau_cc.sh file.c ...`
    - `tau_cxx.sh file.cxx...`
    - `tau_f90.sh file.f90 ...`
  - Execute with real input data
    Environment variables control measurement mode
    - TAU_PROFILE, TAU_TRACE, TAU_CALLPATH, …
  - Examine results with `paraprof`

Vampirtrace MPI Tracing Tool

- Library for Tracing of MPI and Application Events
  - Records MPI-1 point-to-point and collective communication
  - Records MPI–2 I/O operations and RMA operations
  - Records user subroutines
- Uses the standard MPI profiling interface
- Usage:
  - Compile and link with
    - `vtcc -vt:cc mpicc file.c ...`
    - `vtcxx -vt:cxx mpiCC file.cxx...`
    - `vtf90 -vt:f90 mpif90 file.f90 ...`
  - Execute with real input data (⇨ generates `<exe>.otf`)

II-36

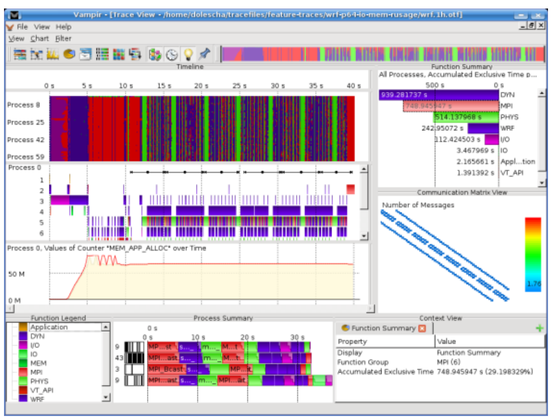
Vampirtrace MPI Tracing Tool

- Versions up to 4.0 until 2003 were commercially distributed by PALLAS as VampirTrace
- Current status
  - Commercially distributed by Intel
    - Version 5 and up distributed as Intel Trace Collector
    - For Intel based platforms only
    - http://software.intel.com/en-us/articles/intel-cluster-toolkit/
  - New open-source VampirTrace version 5 distributed by Technical University Dresden
    - Based on KOJAK's measurement system with OTF backend
    - http://www.tu-dresden.de/zih/vampirtrace/
    - Is also distributed as part of Open MPI

II-37

Vampir Event Trace Visualizer

- Visualization and Analysis of MPI Programs



- Commercial product
- Originally developed by Forschungszentrum Jülich
- Now all development by Technical University Dresden

II-38

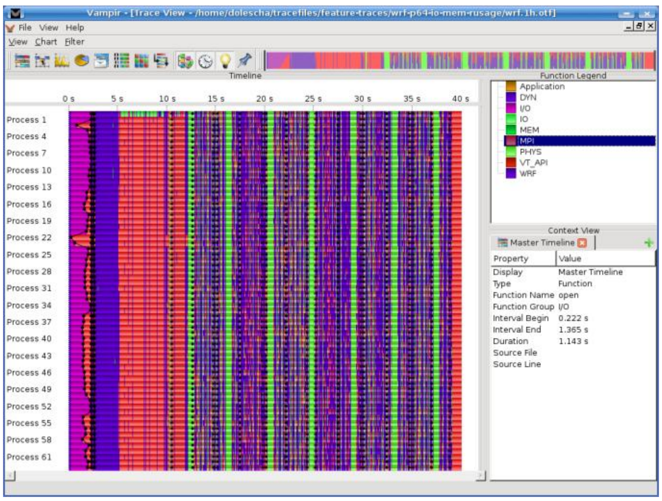
Vampir Event Trace Visualizer

- Versions up to 4.0 until 2003 were commercially distributed by PALLAS as Vampir
- Current status
  - Commercially distributed by Intel
    - Version 4 distributed as Intel Trace Analyzer
    - For Intel based platforms only
    - Intel meanwhile released own new version (ITA V6 and up)
    - http://software.intel.com/en-us/articles/intel-cluster-toolkit/
  - Original Vampir (and new VNG) commercially distributed by Technical University Dresden
    - http://www.vampir.eu

II-39

Vampir: Time Line Diagram

- Functions organized into groups
- coloring by group
- Message lines can be colored by tag or size

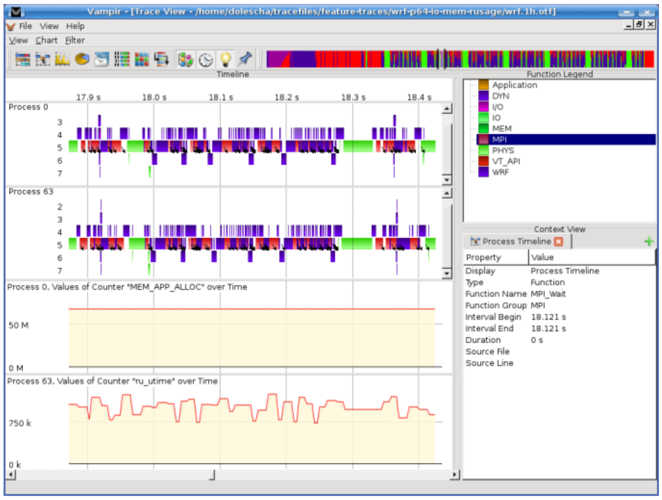


- Information about states, messages, collective and I/O operations available through clicking on the representation

II-40

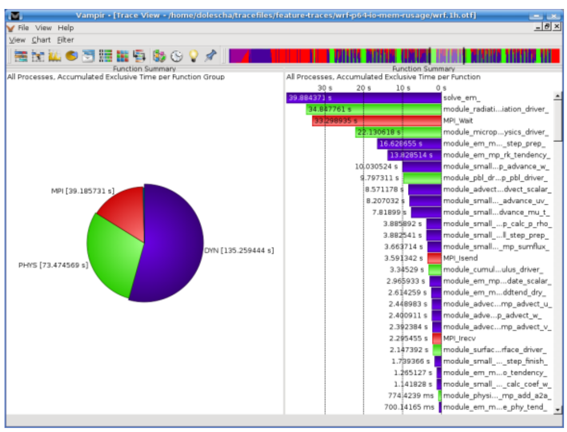
Vampir: Process and Counter Timelines

- Process timeline show call stack nesting
- Counter timelines for hardware or software counters



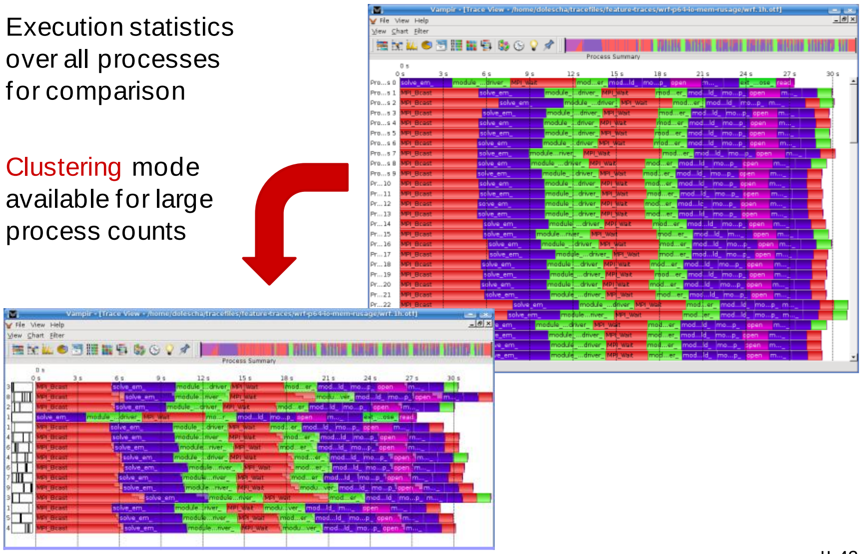
Vampir: Execution Statistics

- Aggregated profiling information: execution time, number of calls, inclusive/exclusive
- Available for all / any group (activity) or all routines (symbols)
- Available for any part of the trace
  ⇨ selectable through time line diagram



Vampir: Process Summary

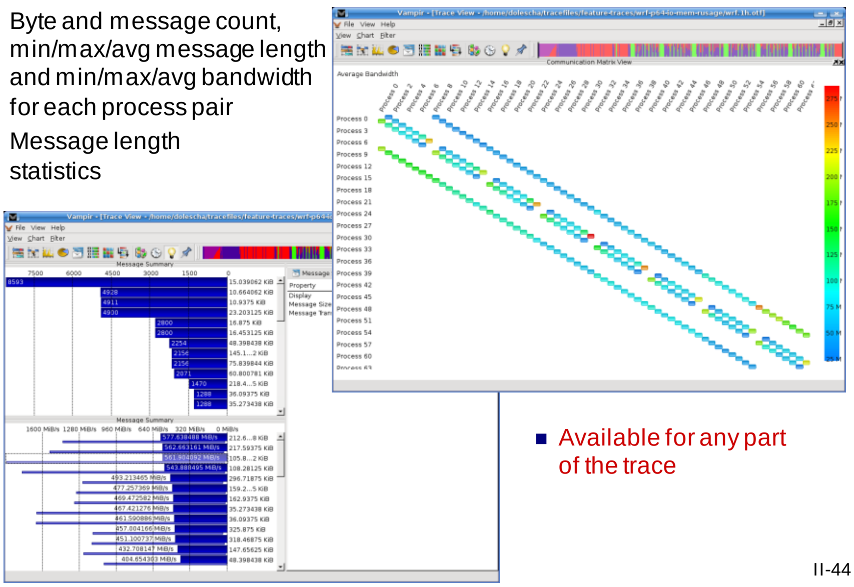
- Execution statistics over all processes for comparison
- Clustering mode available for large process counts



II-43

Vampir: Communication Statistics

- Byte and message count, min/max/avg message length and min/max/avg bandwidth for each process pair
- Message length statistics



- Available for any part of the trace

II-44

Other Profiling Tools

- gprof
  - Available on many systems
  - Compiler instrumentation (invoked via –g –pg)
- FPMPI-2 (ANL)
  - http://www.mcs.anl.gov/fpmpi/
  - MPI profiler (invoked via re-linking)
  - special: Optionally identifies synchronization time
  - single output file: count, sum, avg, min, max over ranks
- ThreadSpotter (Rogue Wave) [commercial product]
  - http://www.roguewave.com/products/threadspotter.aspx
  - Sampling-based memory and thread performance analyzer
  - Works on un-modified, optimized executables

Other Profiling Tools II

- ompP (UC Berkeley)
  - http://www.ompp-tool.com
  - OpenMP profiler (invoked via OPARI source instrumentation)
- HPCToolkit (Rice University)
  - http://www.hpctoolkit.org
  - Multi-platform sampling-based callpath profiler
  - Works on un-modified, optimized executables
- Open|SpeedShop (Krell Insitute with support of LANL, SNL, LLNL)
  - http://www.openspeedshop.org
  - Comprehensive performance analysis environment
  - Uses binary instrumentation (via DynInst)

Other Tracing Tools

- MPE / Jumpshot (ANL)
  - Part of MPICH2
  - Invoked via re-linking
  - Only supports MPI P2P and collectives; SLOG2 trace format
- Extrae / Paraver (BSC/UPC)
  - http://www.bsc.es/paraver
  - Measurement system (Extrae) and visualizer (Paraver)
  - Powerful filter and summarization features
  - Very configurable visualization

CONTENT

- Fall-back
  - Simple timers
  - Hardware counter measurements
- Overview of some performance tools
  - mpiP, TAU, Vampir + Vampirtrace
- Practical Performance Analysis and Tuning
- Challenges and open problems in performance optimization
  - Heterogeneous systems
  - Automatic performance analysis: KOJAK/Scalasca
  - Beyond execution time and flops
  - Extremely large HPC systems

II-48

Practical Performance Analysis and Tuning

- Successful tuning is combination of
  - Right algorithm and libraries
  - Compiler flags and pragmas / directives (Learn and use them)
  - THINKING
- Measurement is better than reasoning / intuition (= guessing)
  - To determine performance problems
  - To validate tuning decisions / optimizations (after each step!)

II-49

Practical Performance Analysis and Tuning

- It is easier to optimize a slow correct program than to debug a fast incorrect one
  - ⇨ Debugging before Tuning
  - ⇨ Nobody really cares how fast you can compute the wrong answer
- The 80/20 rule
  - Program spents 80% time in 20% of code
  - Programmer spends 20% effort to get 80% of the total speedup possible in the code
  - ⇨ Know when to stop!
- Don't optimize what doesn't matter
  - ⇨ Make the common case fast

II-50

Typical Performance Analysis Procedure

1. Do I have a performance problem at all?
   - ⇨ Time / hardware counter measurements
   - ⇨ Speedup and scalability measurements
2. What is the main bottleneck (computation/communication...) ?
   - ⇨ Flat profiling (sampling / prof)
3. Where is the main bottleneck?
   - ⇨ Call graph profiling (gprof)
   - ⇨ Detailed (basic block) profiling
4. Why is it there?
   - ⇨ Hardware counters analysis
   - ⇨ Trace selected parts to keep trace files manageable
5. Does my code have scalability problems?
   - ⇨ Load Imbalance analysis
   - ⇨ Profile code for typical small and large processor count
   - ⇨ Compare profiles function-by-function

II-51

CONTENT

- Fall-back
  - Simple timers
  - Hardware counter measurements
- Overview of some performance tools
  - mpiP, TAU, Vampir + Vampirtrace
- Practical Performance Analysis and Tuning
- Challenges and open problems in performance optimization
  - Heterogeneous systems
  - Automatic performance analysis: KOJAK/Scalasca
  - Beyond execution time and flops
  - Extremely large HPC systems

II-52

Heterogenous Systems

- Current trend to use hardware acceleration to speedup calculations
  - IBM Cell (e.g. LANL Roadrunner)
  - Clearspead
  - FPGA-based acceleration
  - GPU-based acceleration
- In the long run: new programming models needed
- Very little to not existing tool support

⇨ Tool research opportunities!

II-53

GPU Performance Tools

- CUDA profiler
- TAU
  - CUDA + OpenCL profiling (host side)
- VampirTrace
  - CUDA tracing (host side)

II-54

Current Related Projects

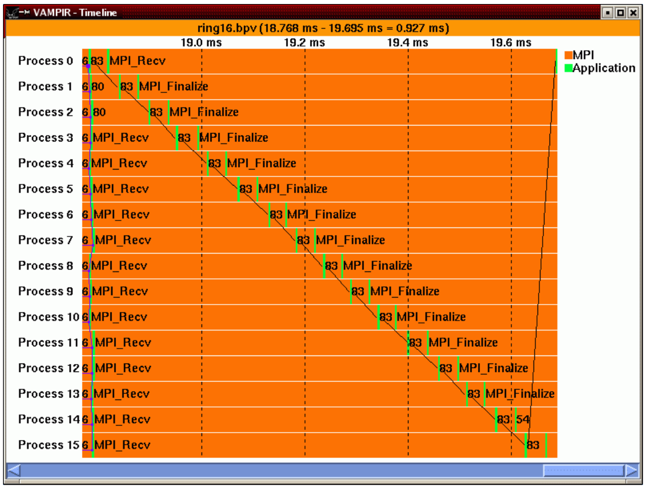
- Hybrid Programming for Heterogeneous Architectures
  - EU ITEA2 funded project 2010 – 2013
  - Successor of successful ParMA project
  - 25 partners for France, Germany, Spain, Sweden including Bull, TUD, BSC, USQV, JSC, Acumem
  - Develop programming models and tools
  - Evaluation with large set of industrial codes

CONTENT

- Fall-back
  - Simple timers
  - Hardware counter measurements
- Overview of some performance tools
  - mpiP, TAU, Vampir + Vampirtrace
- Practical Performance Analysis and Tuning
- Challenges and open problems in performance optimization
  - Heterogeneous systems
  - Automatic performance analysis: KOJAK/Scalasca
  - Beyond execution time and flops
  - Extremely large HPC systems

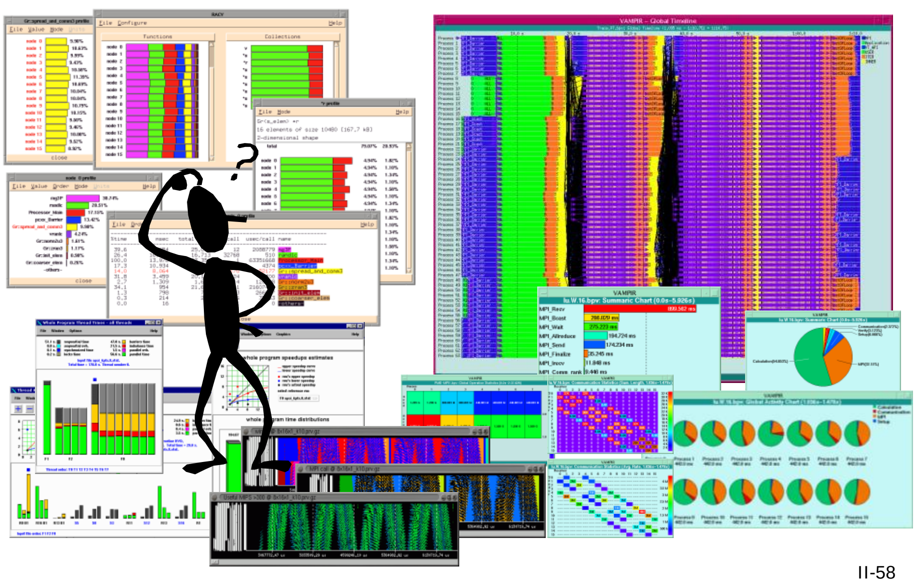
II-56

"A picture is worth 1000 words…"

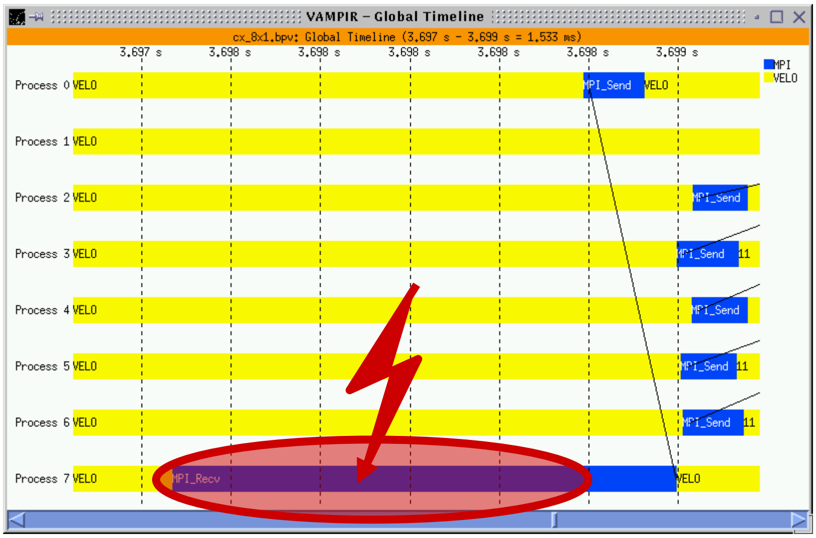


- MPI class example: Send messages in a ring program

"What about 1000's of pictures?" (with 100's of menu options)



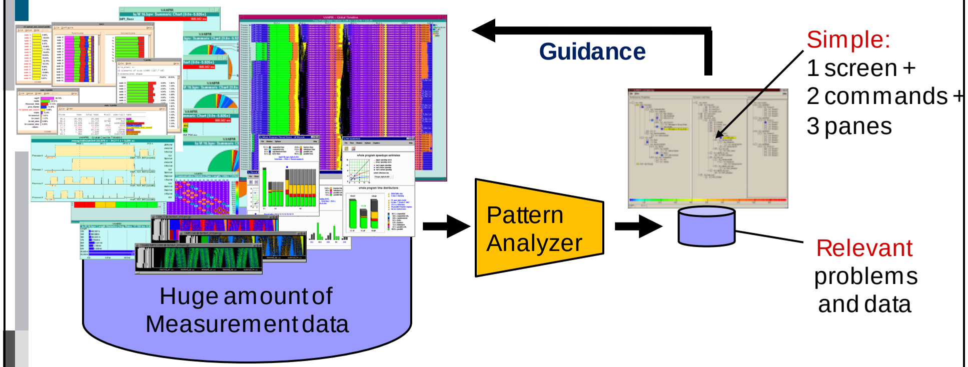
Example Automatic Analysis: Late Sender



Example Automatic Analysis (2): Wait at NxN

Basic Idea Automatic Performance Analysis

- "Traditional" Tool
- Automatic Tool



- For non-standard / tricky cases (10%)
- For expert users

- For standard cases (90% ?!)
- For "normal" users
- Starting point for experts

⇨ More productivity for performance analysis process!

II-61

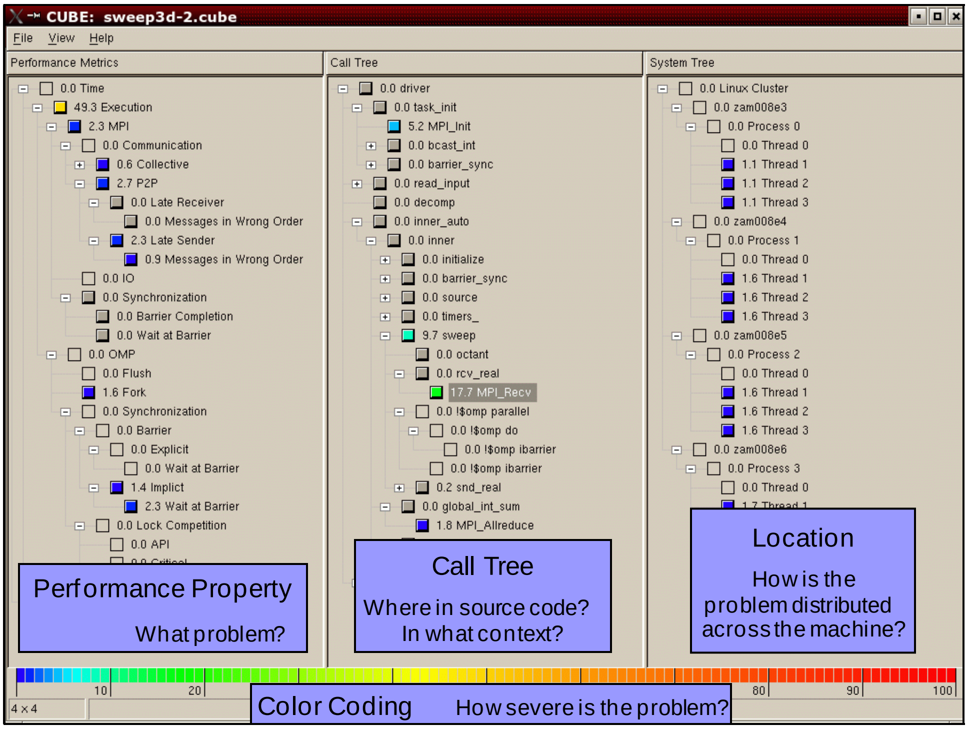
The KOJAK Project

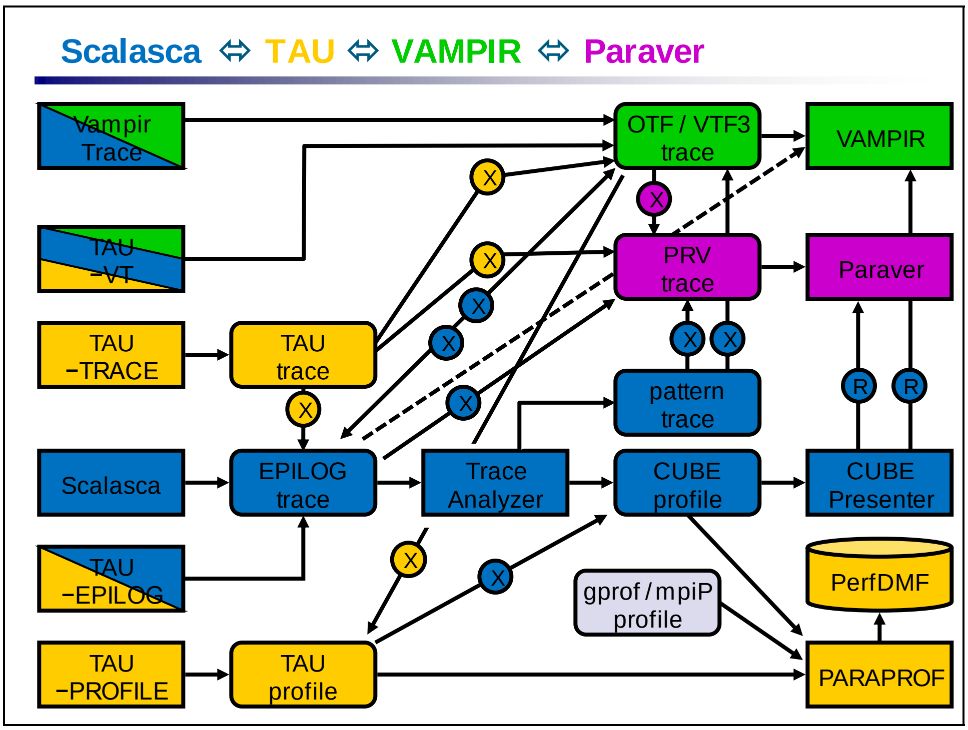
- Kit for Objective Judgement and Automatic Knowledge-based detection of bottlenecks
- Forschungszentrum Jülich
- Innovative Computing Laboratory, TN
- Started 1998

- Approach
  - Instrument C, C++, and Fortran parallel applications
    - Based on MPI, OpenMP, SHMEM, or hybrid
  - Collect event traces
  - Search trace for event patterns representing inefficiencies
  - Categorize and rank inefficiencies found

- http://www.fz-juelich.de/jsc/kojak/

II-62


CUBE: sweep3d-2.cube
File
View
Help
Performance Metrics
Call Tree
System Tree
0.0 Time
49.3 Execution
2.3 MPI
0.0 Communication
0.6 Collective
2.7 P2P
0.0 Late Receiver
0.0 Messages in Wrong Order
2.3 Late Sender
0.9 Messages in Wrong Order
0.0 IO
0.0 Synchronization
0.0 Barrier Completion
0.0 Wait at Barrier
0.0 OMP
0.0 Flush
1.6 Fork
0.0 Synchronization
0.0 Barrier
0.0 Explicit
0.0 Wait at Barrier
1.4 Implicit
2.3 Wait at Barrier
0.0 Lock Competition
0.0 API
0.0 driver
0.0 task_init
5.2 MPI_Init
0.0 bcast_int
0.0 barrier_sync
0.0 read_input
0.0 decomp
0.0 inner_auto
0.0 inner
0.0 initialize
0.0 barrier_sync
0.0 source
0.0 timers_
9.7 sweep
0.0 octant
0.0 rcv_real
17.7 MPI_Recv
0.0 !$omp parallel
0.0 !$omp do
0.0 !$omp ibarrier
0.0 !$omp ibarrier
0.2 snd_real
0.0 global_int_sum
1.8 MPI_Allreduce
0.0 Linux Cluster
0.0 zam008e3
0.0 Process 0
0.0 Thread 0
1.1 Thread 1
1.1 Thread 2
1.1 Thread 3
0.0 zam008e4
0.0 Process 1
0.0 Thread 0
1.6 Thread 1
1.6 Thread 2
1.6 Thread 3
0.0 zam008e5
0.0 Process 2
0.0 Thread 0
1.6 Thread 1
1.6 Thread 2
1.6 Thread 3
0.0 zam008e6
0.0 Process 3
0.0 Thread 0
Performance Property
What problem?
Call Tree
Where in source code?
In what context?
Location
How is the problem distributed across the machine?
10
20
80
90
100
4 x 4
Color Coding
How severe is the problem?


Scalasca ⇔ TAU ⇔ VAMPIR ⇔ Paraver
Vampir Trace
TAU -VT
TAU -TRACE
TAU trace
Scalasca
EPILOG trace
TAU -EPILOG
TAU -PROFILE
TAU profile
Trace Analyzer
OTF / VTF3 trace
PRV trace
pattern trace
CUBE profile
gprof / mpiP profile
VAMPIR
Paraver
CUBE Presenter
PerfDMF
PARAPROF
X
R

CONTENT

- Fall-back
  - Simple timers
  - Hardware counter measurements
- Overview of some performance tools
  - mpiP, TAU, Vampir + Vampirtrace
- Practical Performance Analysis and Tuning
- Challenges and open problems in performance optimization
  - Heterogeneous systems
  - Automatic performance analysis: KOJAK/Scalasca
  - Beyond execution time and flops
  - Extremely large HPC systems

II-65

Beyond Execution Time and Flops

- High performance on today’s architectures requires extremely effective use of the cache and memory hierarchy of the system
  - Very complex designs
  - Will get “worse” with advanced multi- and many core chips

⇨ tools for memory performance analysis needed
- Current approach:
  - Measure cache, memory, TLB events (per execution unit)
  - Rarely tell what data object(s) are cause of the problem
- Main problem:
  - if cause known, what is the fix?
  - Or worse, what is the portable fix?

⇨ Should be really better left to the compiler!

II-66

Beyond Execution Time and Flops II

- Do we need tools for I/O performance analysis?
  - Currently only very few tools available (CrayPat, Pablo I/O, …)
  - Perhaps scientific programmers only need training in parallel I/0 facilities already available today?
- Tools for new programming paradigms?
  - Obvious next candidate: one-sided communication
    - SHMEM, MPI-2 RMA, ARMCI, …
      - "Easy" to monitor; intercept calls e.g., using wrapper funcs
    - CAF, UPC
      - "Harder"; probably need compiler support
  - Other candidates?

II-67

Beyond Execution Time and Flops III

- Tool (sets and environments) must be able to handle "hybrid" cases!
  - Message passing (MPI)
  - Multi-threading (OpenMP, pthreads, …)
  - One-sided communication
  - (Parallel) I/O
  - …
- BIGGEST challenge:
  - Many tools for instrumentation, measurement, analysis, and visualization
  - What about (automatic) optimization tools?

II-68

CONTENT

- Fall-back
  - Simple timers
  - Hardware counter measurements
- Overview of some performance tools
  - mpiP, TAU, Vampir + Vampirtrace
- Practical Performance Analysis and Tuning
- Challenges and open problems in performance optimization
  - Heterogeneous systems
  - Automatic performance analysis: KOJAK/Scalasca
  - Beyond execution time and flops
  - Extremely large HPC systems

II-69

Increasing Importance of Scaling

- Number of Processors share for TOP 500 Jun 2011

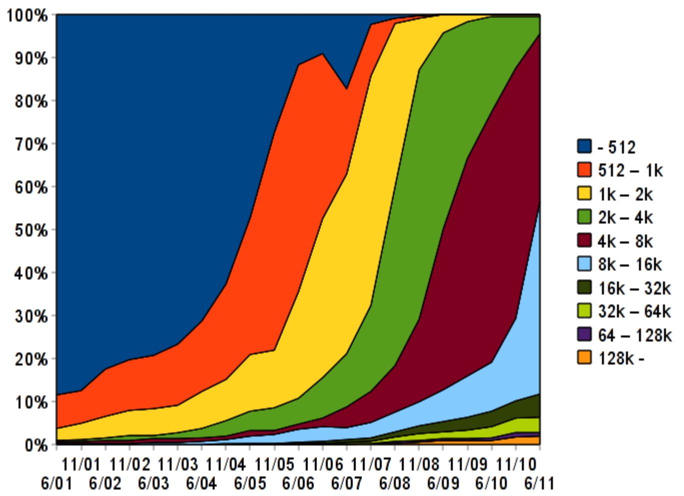
NProc	Count	Share	∑Rmax	Share	∑NProc
1025-2048	2	0.4%	168 TF	0.3%	2,632
2049-4096	20	4.0%	1,177 TF	2.0%	71,734
4097-8192	195	39.0%	9,759 TF	16.6%	1,262,738
8193-16384	224	44.8%	15,216 TF	25.8%	2,337,998
> 16384	59	11.8%	32,556 TF	55.3%	4,100,234
Total	500	100%	58,876 TF	100%	7,775.336

- Average system size: 15,551 cores
- Median system size: 8,556 cores

II-70

Increasing Importance of Scaling II

- Number of Processors share for TOP 500 Jun 2001 – Jun 2011



II-71

Projection for a Exascale System*

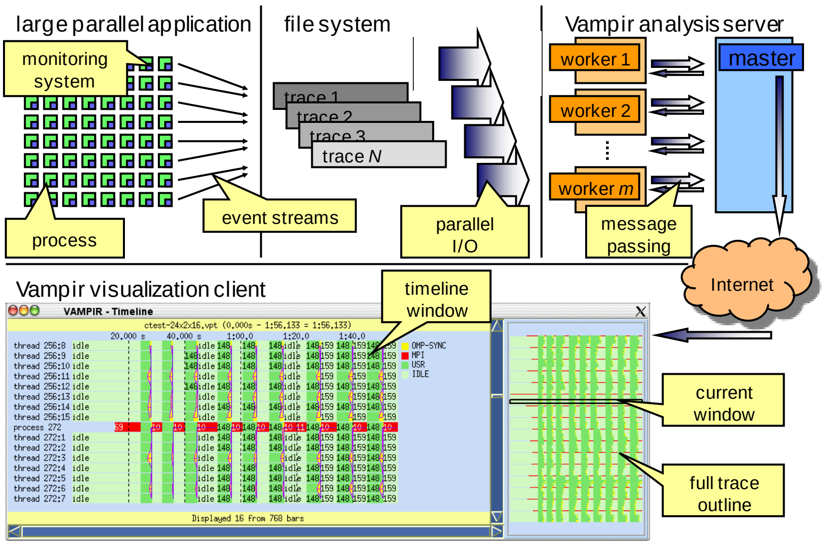
System attributes	2010	"2015"		"2018"		Difference 2010 & 2018
System peak	2 Pflop/s	200 Pflop/s		1 Eflop/sec		O(1000)
Power	6 MW	15 MW		~20 MW		
System memory	0.3 PB	5 PB		32-64 PB		O(100)
Node performance	125 GF	0.5 TF	7 TF	1 TF	10 TF	O(10) – O(100)
Node memory BW	25 GB/s	0.1 TB/sec	1 TB/sec	0.4 TB/sec	4 TB/sec	O(100)
Node concurrency	12	O(100)	O(1,000)	O(1,000)	O(10,000)	O(100) – O(1000)
Total Concurrency	225,000	O(10^8)		O(10^9)		O(10,000)
Total Node Interconnect BW	1.5 GB/s	20 GB/sec		200 GB/sec		O(100)
MTTI	days	O(1day)		O(1 day)		- O(10)

* From http://www.exascale.org

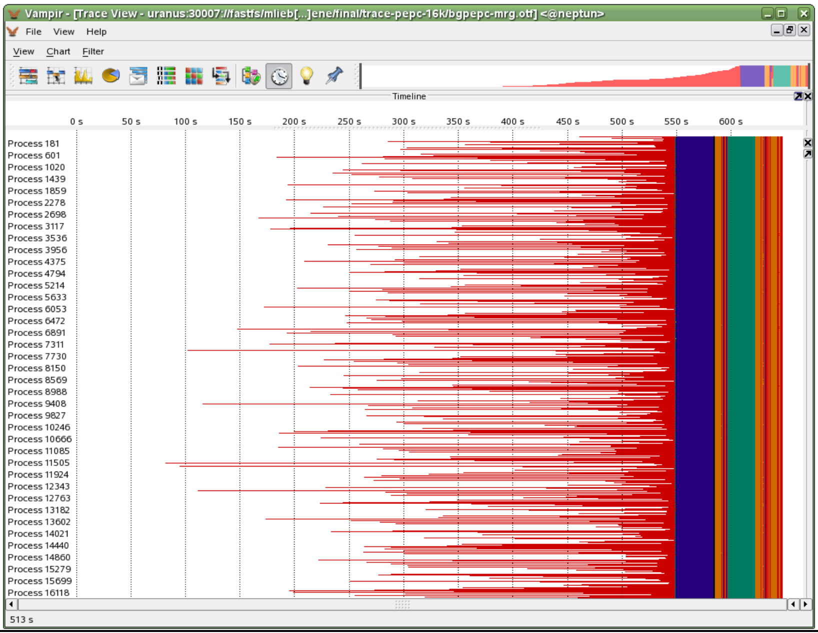
Extremely large HPC systems

- Many HPC computing centers have systems with some 1000s of PEs
  - Today's tools (e.g., gprof, Vampir, TAU) can handle 500–1000 PEs
  - Measurements give enough insight to optimize for 2000-4000 PEs
- Now: IBM BlueGene, Cray XT5 ⇨ some 100,000s PEs
  - Small customer base
  - Very unique performance problems:
    - Why does it work on 16.000 PEs but not for 32.000 PEs?
  - Load balance problems amplify performance degradation
  - Interesting challenging problem,
    but given scarce resources in tools research? …
    - Lots of data ⇨ need better data mgmt, parallel analysis!?
    - Bigger problem: scalable result displays / visualizations

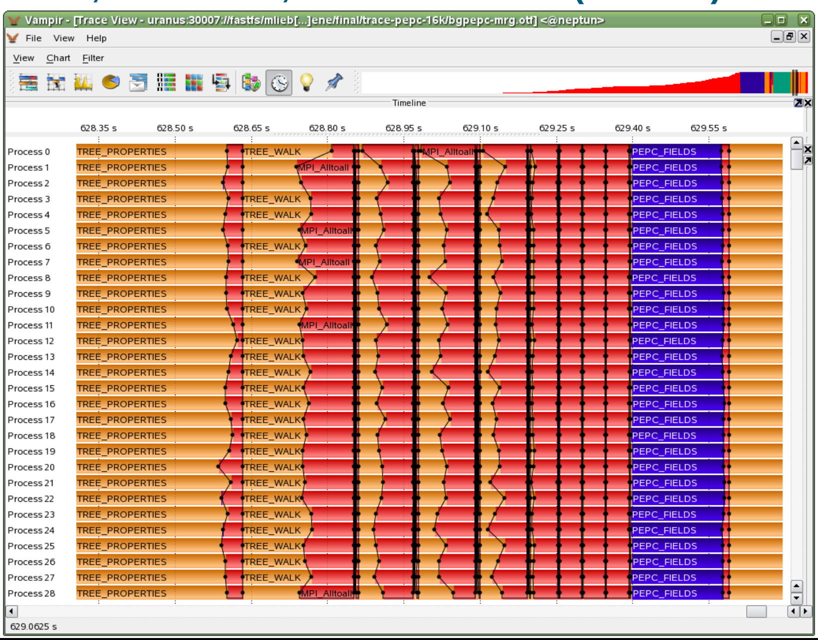
VampirServer Architecture



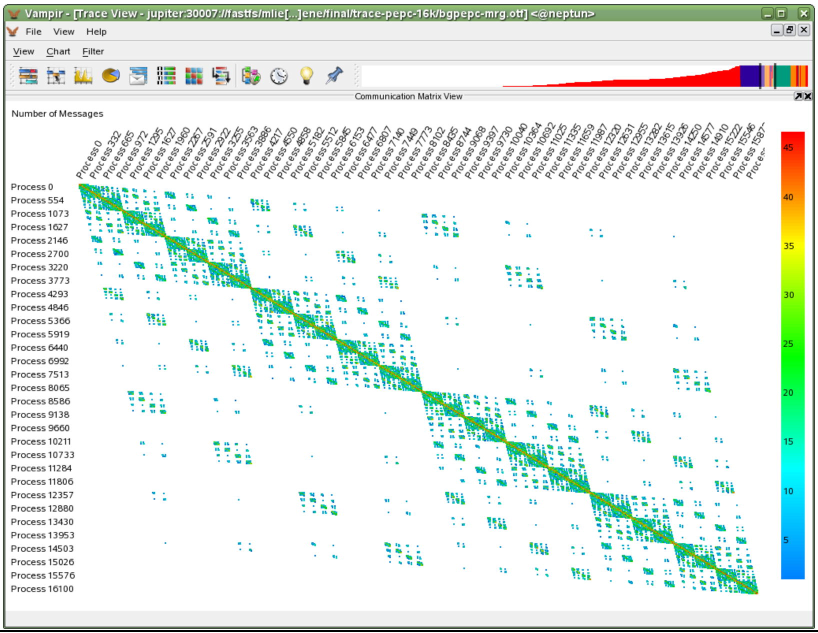
VampirServer:
PEPC, 16384 PEs, Global Timeline



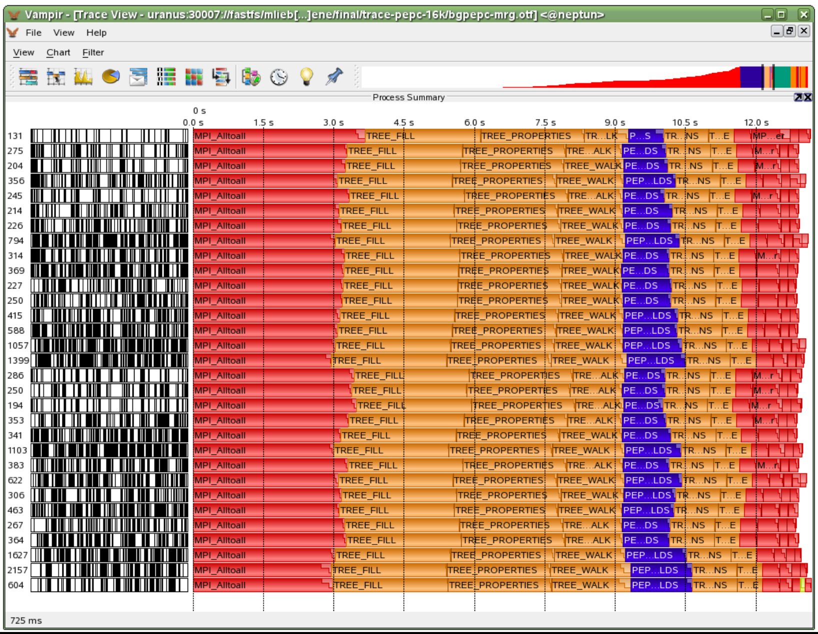
VampirServer:
PEPC, 16384 PEs, Global Timeline (zoomed)



VampirServer: PEPC, 16384 PEs, Message Statistics



VampirServer: PEPC, 16384 PEs, Cluster Analysis



Vampirserver

- Parallel client/server version of Vampir
  - Can handle much larger trace files
  - Remote visualization possible
- Usage
  - Start VampirServer daemon
    - `vngd -n <t>` # on frontend, uses <t> threads
    - `mpiexec -n <p> vngd` # cluster, uses <p> processes
  - Start the Vampir visualizer (`vampir`)
    - on local system if available
    - on frontend in 2nd shell

    then connect client to server, load and analyze trace file <exe>.otf
    - finally: `vngd-shutdown`

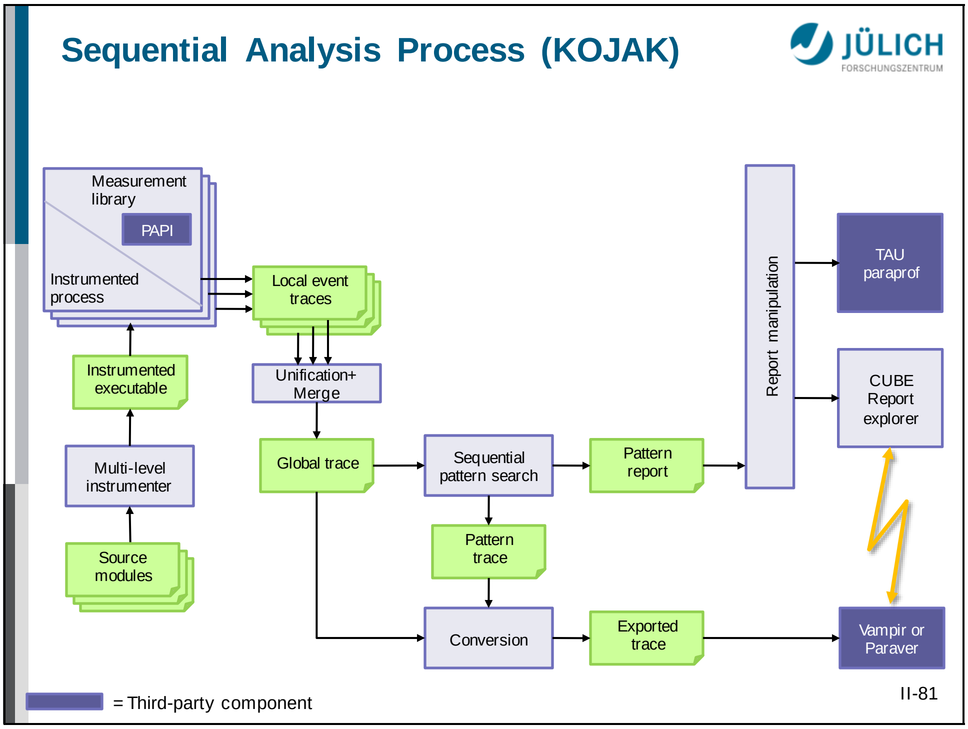
The Scalasca Project

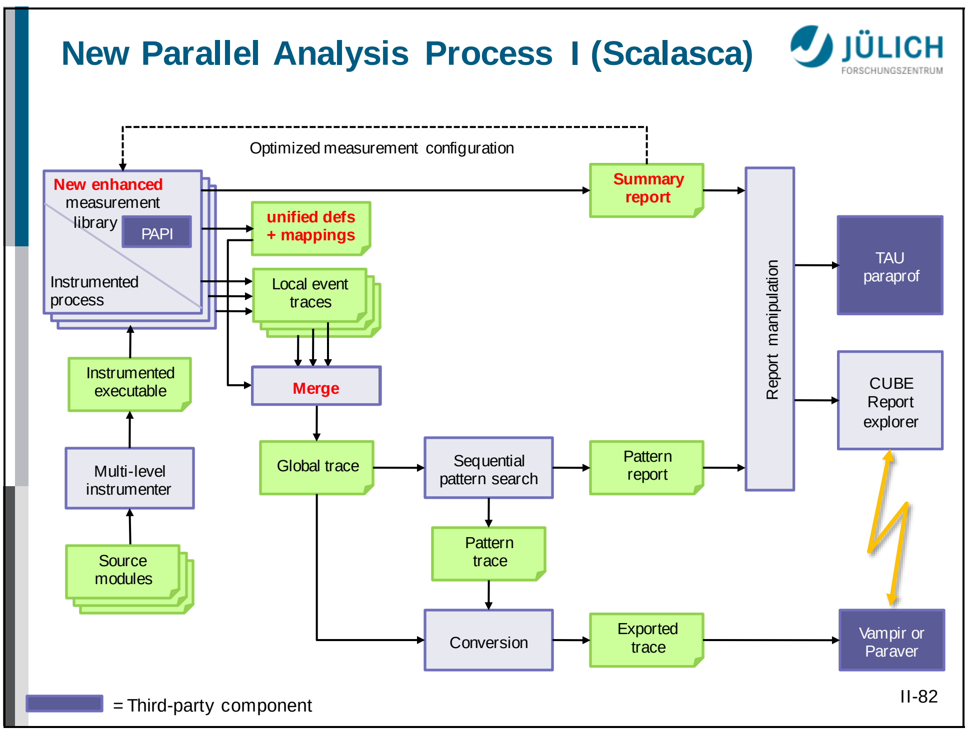
- Scalable Analysis of Large Scale Applications
- Follow-up project to KOJAK

http://www.scalasca.org/

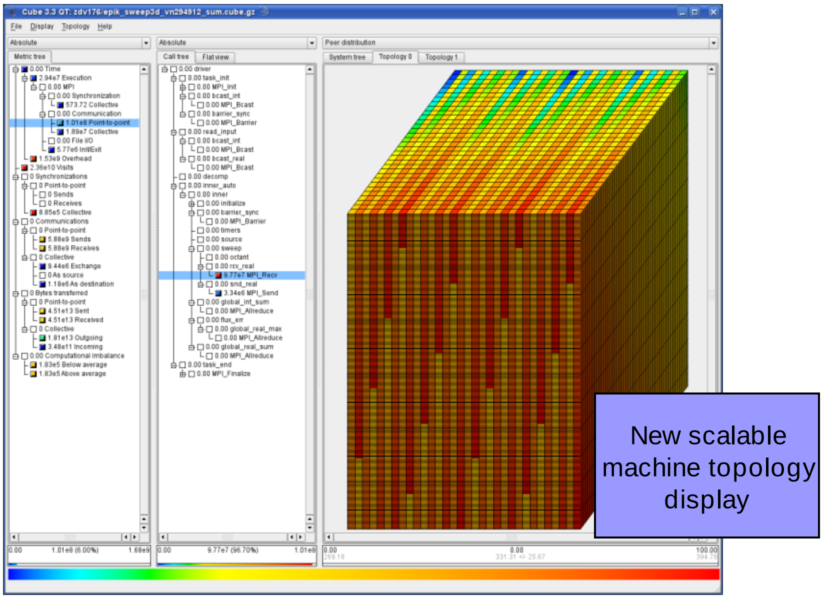
- Approach
  - **Instrument** C, C++, and Fortran parallel applications
    - Based on MPI, OpenMP, SHMEM, or hybrid
  - **Option 1: scalable call-path profiling**
  - **Option 2: scalable event trace analysis**
    - **Collect** event traces
    - **Search** trace for event patterns representing inefficiencies
    - **Categorize and rank** inefficiencies found
- Supports MPI 2.2 and basic OpenMP

German Research School for Simulation Sciences

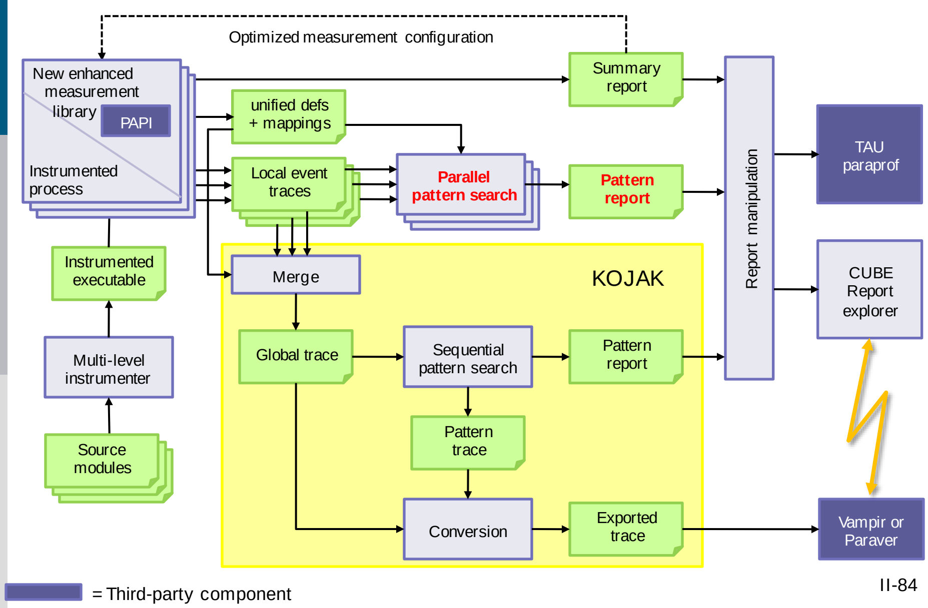

Sequential Analysis Process (KOJAK)
JÜLICH FORSCHUNGSZENTRUM
Measurement library
PAPI
Instrumented process
Local event traces
Unification+ Merge
Instrumented executable
Multi-level instrumenter
Source modules
Global trace
Sequential pattern search
Pattern report
Pattern trace
Conversion
Exported trace
Report manipulation
TAU paraprof
CUBE Report explorer
Vampir or Paraver
= Third-party component
II-81


New Parallel Analysis Process I (Scalasca)
JÜLICH FORSCHUNGSZENTRUM
Optimized measurement configuration
New enhanced measurement library
PAPI
Instrumented process
Summary report
unified defs + mappings
Local event traces
Merge
Instrumented executable
Multi-level instrumenter
Source modules
Global trace
Sequential pattern search
Pattern report
Pattern trace
Conversion
Exported trace
Report manipulation
TAU paraprof
CUBE Report explorer
Vampir or Paraver
= Third-party component
II-82

Scalasca Summary Analysis sweep3D@294,912



New Parallel Analysis Process II (Scalasca)

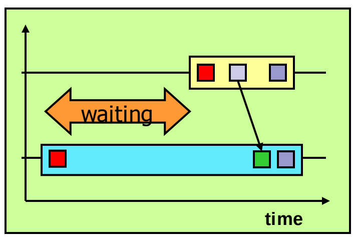


II-84

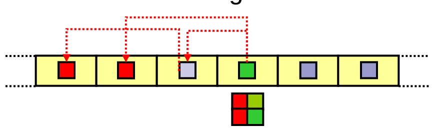
Scalable Automatic Trace Analysis

- Parallel pattern search to address wide traces
  - As many processes / threads as used to run the application
    ⇨ Can run in same batch job!!
  - Each process / thread responsible for its “own” local trace data
- Idea: “parallel replay” of application
  - Analysis uses communication mechanism that is being analyzed
  - Use MPI P2P operation to analyze MPI P2P communication, use MPI collective operation to analyze MPI collectives, …
  - Communication requirements not significantly higher and (often lower) than requirements of target application
- In-memory trace analysis
  - Available memory scales with number of processors used
  - Local memory usually large enough to hold local trace

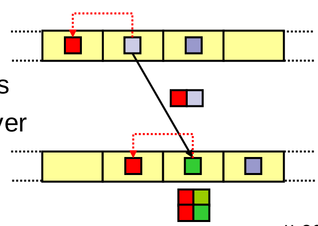
Example: Late Sender



- Sequential approach (EXPERT)
  - Scan the trace in sequential order
  - Watch out for receive event
  - Use links to access other constituents
  - Calculate waiting time

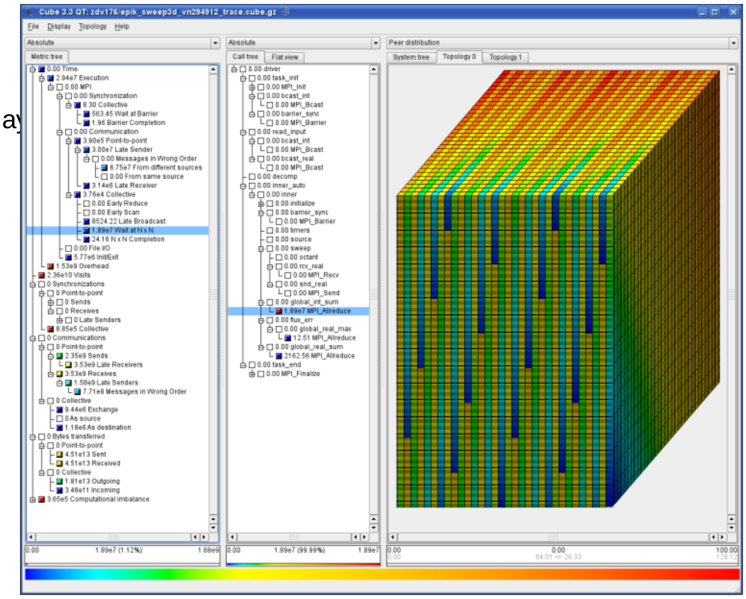


- New parallel approach (SCOUT)
  - Each process identifies local constituents
  - Sender sends local constituents to receiver
  - Receiver calculates waiting time

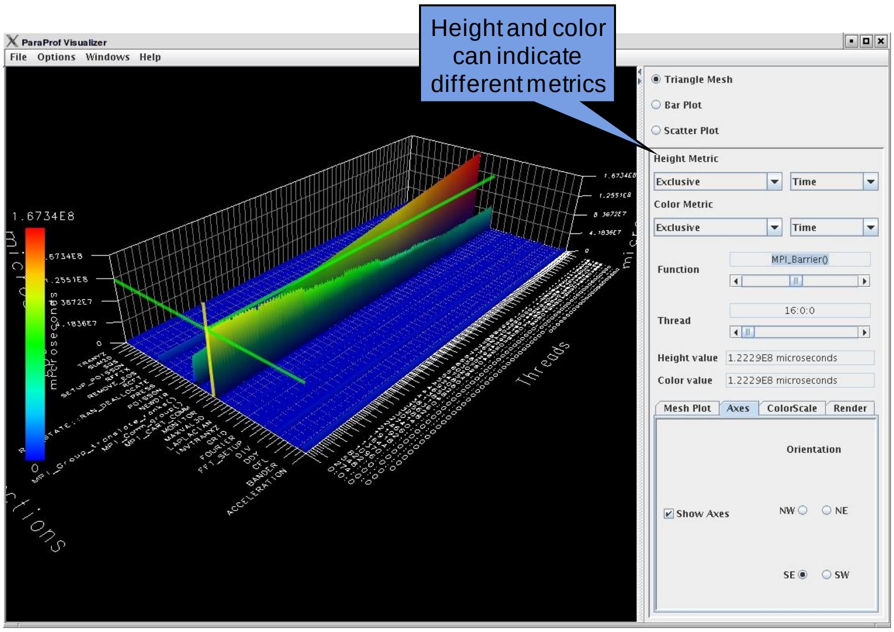


Scalasca Trace Analysis sweep3D@294,912

- 10 min sweep3D runtime
- 11 sec replay
- 4 min trace data write/read (576 files)
- 7.6 TB buffered trace data
- 510 billion events



TAU ParaProf: 3D Profile, Miranda, 16K PEs



Performance Tuning: Still a Problem?

[Slide left intentionally blank]

II-89

Further Documentation

- http://www.vi-hps.org/training/material/
  - Performance Tools LiveDVD image
  - Links to tool websites and documentation
  - Tutorial slides