Open Source Performance Analysis Tools for HPC/Linux Systems

Philip J. Mucci
mucci@cs.utk.edu
http://www.cs.utk.edu/~mucci
KTH/UTK/SiCortex

HPCiA '07
Tromsø, Norway

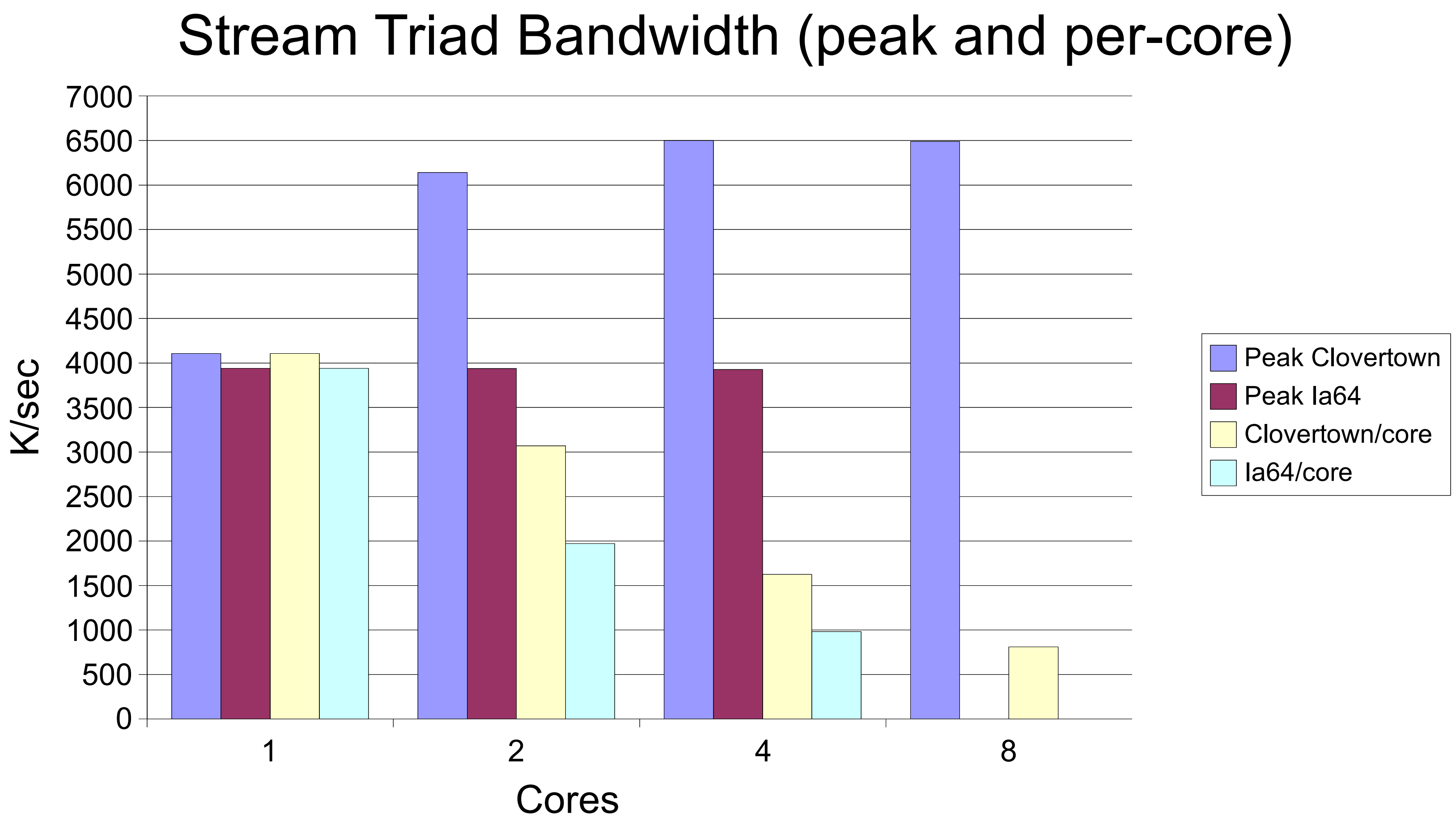
Outline

- Motivation
- Background
- Middleware
- Tools

Motivation

- Economic: Time is Money
  - Average lifetime of these large machines is a handful years before being decommissioned.
  - Many HPC centers charge departments by the CPU hour.
  - Consider the cost per day of a 4 million dollar machine, retired after 4 years, with annual support/maintenance/infrastructure/personnel costs of $300,000.
  - That's $1500.00 per hour of compute time.
    - Price per sq. ft/m, per KW/h, per employee, all going up.
    - Faster than delivered performance (vs. peak)?

Memory Bandwidth



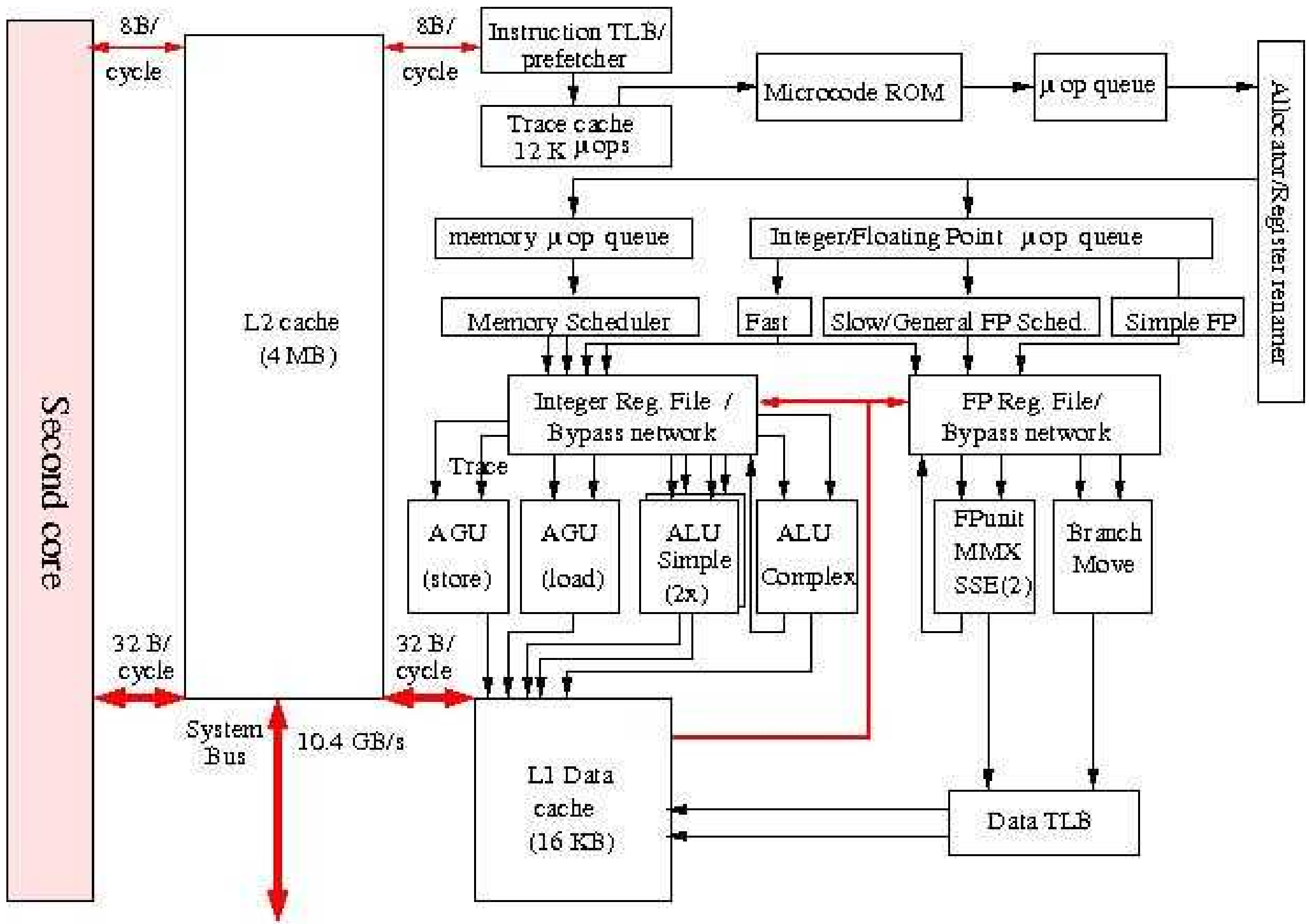
Motivation

- Qualitative: Improvements in Science
  - Poorly written code can easily run many times worse than an optimized version.
  - Consider how hard it is to write a decent:
    - DGEMM/FFT/2D/3D Convolution
- Performance IS the difference in resolving the phenomena of interest.
- Predictions about the limiting factors of performance codes are (usually) erroneous:
  - Working set size and memory bandwidth
  - P2P latency & collective performance
- How can you plan for additional resources?

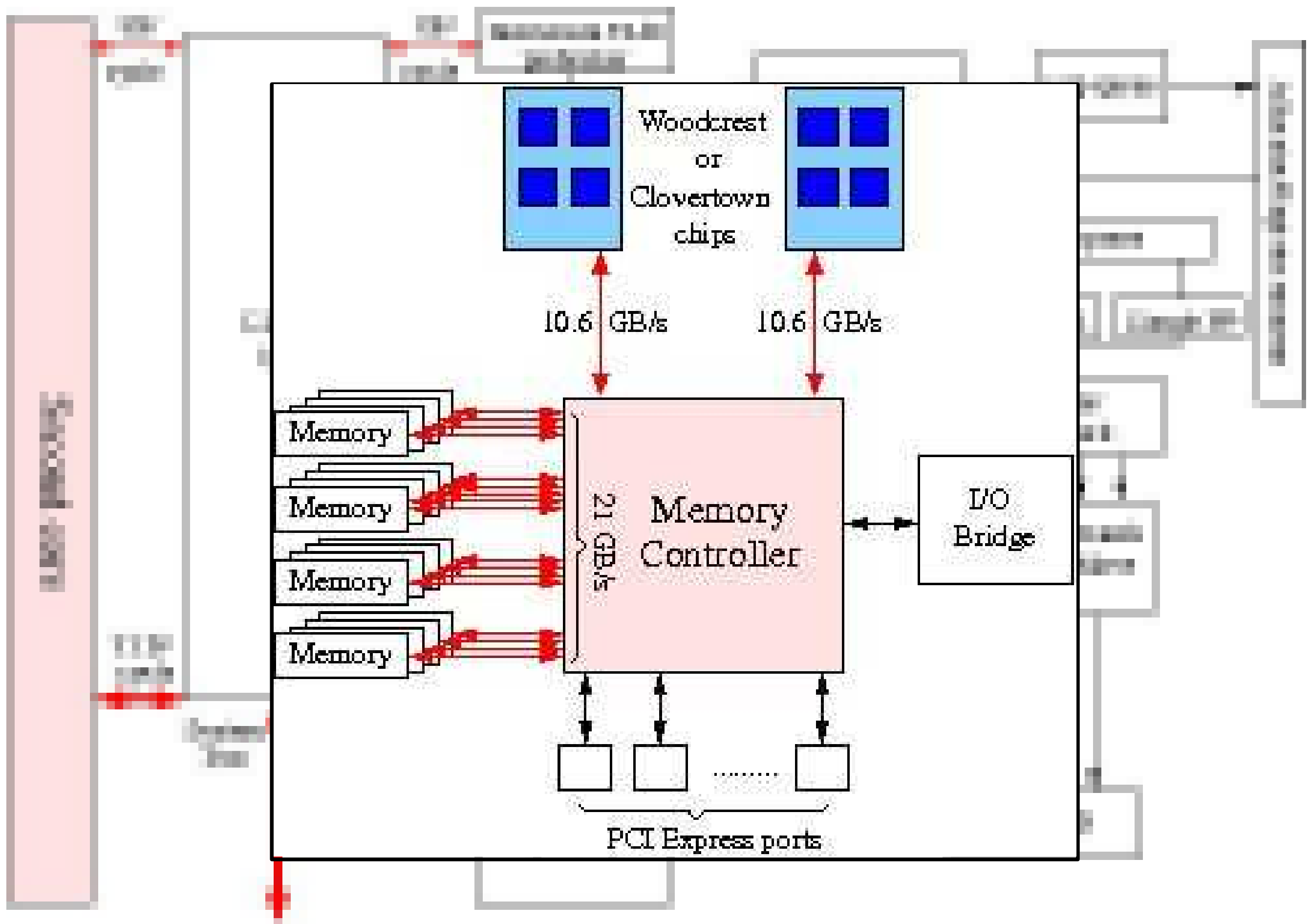
Rising Processor Complexity

- Architectural can innovations can 'hide' performance from the programmer.
  - Performance cannot be easily predicted.
    - Static/dynamic branch prediction
    - Hardware prefetching
    - Out-of-order scheduling/execution
    - Predication
    - Coherency
- A measure of wallclock time, even comparative, is not enough.
- How fast is fast, how high is up?

Intel Clovertown Block Diagram



Intel Clovertown 2x4 Node



Motivation

- Application Scientists
  - Plan, code and test for performance during entire development cycle. (K.R. 'Clever Contributors')
    - Retrofitting performance isn't possible without a significant investment.
    - Be wary of promises of 'free' performance, i.e. OpenMP, HPF, BSP, UPC. All contain trade-offs.
- Computer Scientists
  - Develop tools, infrastructure and expertise to develop **and track** performance in terms of architecture and applications in the long term.
- Education and cooperation is the key.

Performance Evaluation

- Traditionally, performance evaluation has been somewhat of an art form:
  - Limited set of tools (/bin/time, gprof)
  - Major differences between systems, SW and HW
  - Lots of intuition/experience/guesswork involved in looking 'behind the numbers'
    - Few individuals possessed all the knowledge
- Today, the situation is different.
  - Hardware support for performance analysis and a wide variety of quality Open Source tools to choose from.
  - Exporting the knowledge down into the user base.

The State of Linux Performance Tools

- Linux kernel has no code to support hardware performance measurements in production (non-root, shared) environments.*
  - Despite highly stable kernel patches being available for > 10 years on some platforms.
  - Patch deployment complicated for smaller users where support agreements preclude patching of the kernel.
- No major commercial Linux distribution contains anything beyond OProfile and Gprof.*
  - Gprof: requires recompilation, does not support threads
  - Oprofile: requires root privileges to use, does not allow sharing of the PMU resources

The State of Linux Performance Tools

- Vendors have developed some good tools, but kept much of the code private.
- Numerous quality open-source tools exist but many are lacking in good release engineering practices:
  - Documentation/Installation/Usage semantics
  - Parallel run-time integration and interoperability
  - Distribution (no RPM's, Ebuilds, Debs...)

Evaluation of Workloads

- Characterization
  - Overall evaluation of performance
  - Isolate specific components for focus.
- Analysis and Optimization
  - Establish baseline performance data
  - Focus experimentation and optimization passes.
- Performance Development
  - Integration of robust performance evaluation
  - Regular performance regression testing

The Trouble with Timers

- They depend on the load on the system.
  - Elasped wall clock time does not reflect the actual time the program is doing work due to:
    - OS work/interference.
    - Other processes.
  - Per-thread timers are coarse grained.
- The solution?
  - We need measurements that are accurate yet independent of external factors. (With some help from the OS)

Hardware Performance Counters

- Performance Counters are hardware registers dedicated to counting certain types of events within the processor or system.
  - Usually a small number of these registers (2,4,8)
  - Sometimes they can count a lot of events or just a few
  - Symmetric or asymmetric
  - May be on or off chip
- Each register has an associated control register that tells it what to count and how to do it.
  - Interrupt on overflow
  - Edge detection (cycles vs. events)
  - User vs. kernel mode

Sample Performance Data Available

- Cycle count
- Instruction count
  - All instructions
  - Floating point
  - Integer
  - Load/store
- Branches
  - Taken / not taken
  - Mispredictions
- Pipeline stalls due to
  - Memory subsystem
  - Resource conflicts
- Cache
  - I/D cache misses for different levels
  - Invalidations
- TLB
  - Misses
  - Invalidations
  -

Intel Core 2 PMU

- 5 counters
  - 2 fully programmable counters
  - 3 fixed function counters
    - INSTR_RETIRED.ANY
    - CPU_CLK_UNHALTED.CORE
    - CPU_CLK_UNHALTED.REF
- Each register can:
  - Count in any combination of the counting domains:
    - User, Kernel, etc..
  - Interrupt on overflow
- Plus an RTC

Hardware Performance Counter Virtualization

- Every process appears to have its own counters.
- OS accumulates counts into 64-bit quantities for each thread and process.
  - Saved and restored (lazily) on context switch.
- Both user, kernel and other domains can be measured.
- Explicit counting or histograms based on sampling upon counter overflow.
- Counts are largely independent of load.

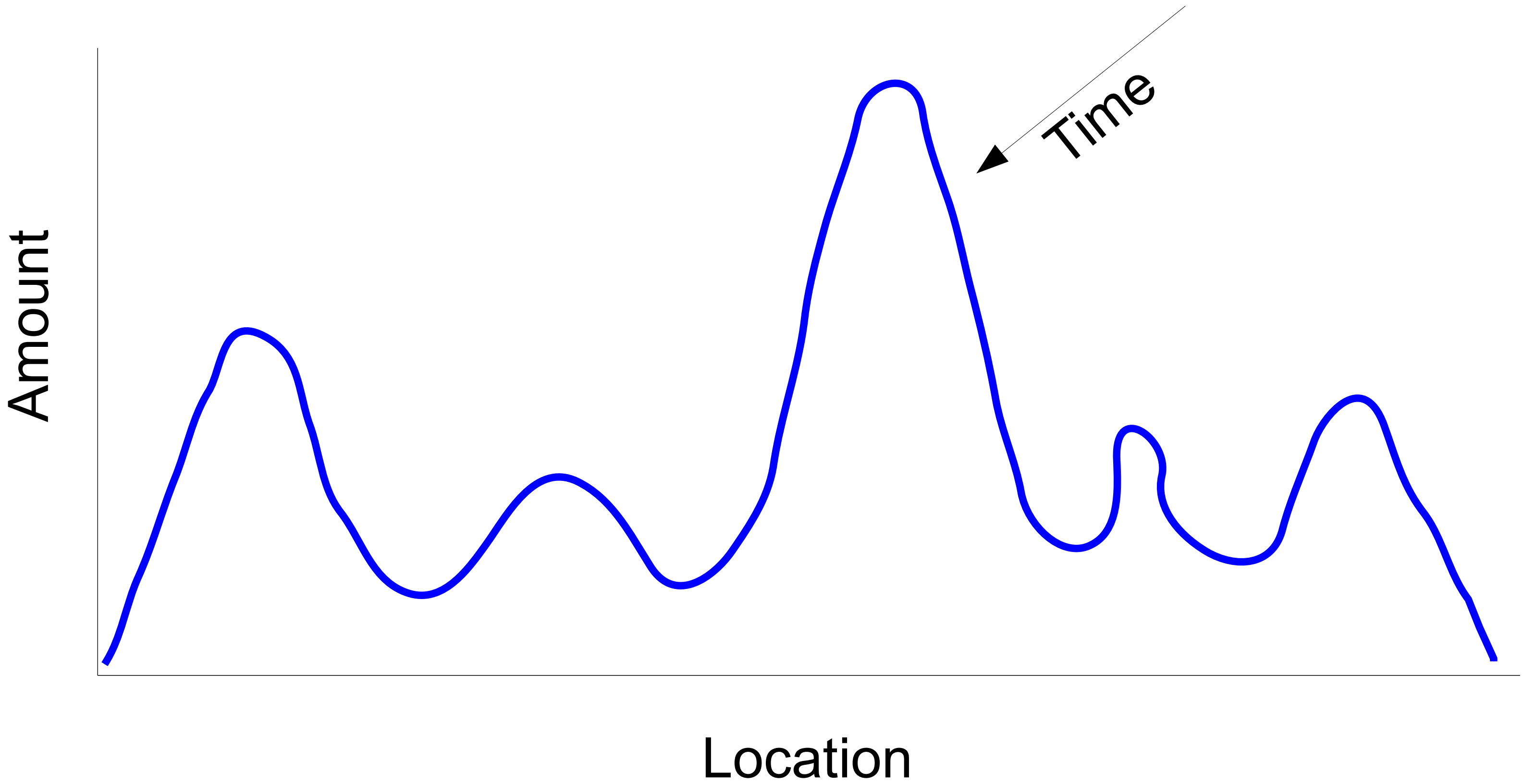
Direct Measurements

- Start/stop paradigm, usually requiring explicit instrumentation.
- Aggregate
  - Reduce data at run-time avg/min/max measurements.
  - Useful for application and architecture characterization and optimization.
- Tracing
  - Generate a record for each measured event.
  - Useful only when evidence of performance anomalies is present due to the large volume of data generated.

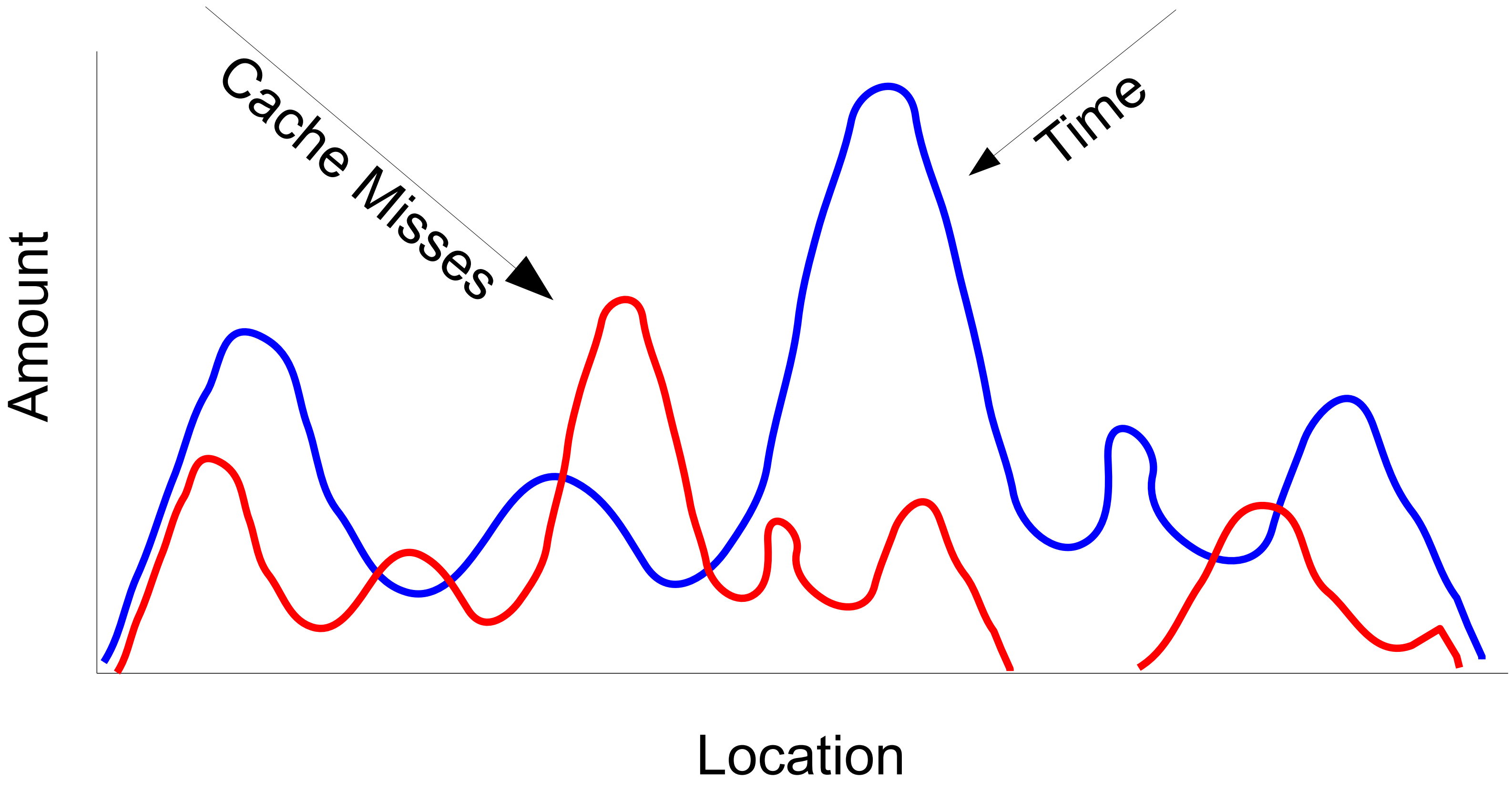
Indirect Measurements

- Form probabilistic estimates of the distribution of performance related events.
- Profiling
  - Histogram pointer values based on interrupts every N performance events. (i.e. Histogram the IP every 1000 L1 D-cache misses.)
- Sampling
  - Record values of performance related events based on the trigger of some other event. (i.e. Record effective data address of cache miss, IP, timestamp and miss latency type every 1000 L1 D-cache misses.)
- Boundary between indirect and direct is somewhat fuzzy.

Statistical Profiling



Hardware Statistical Profiling



Parallel Performance

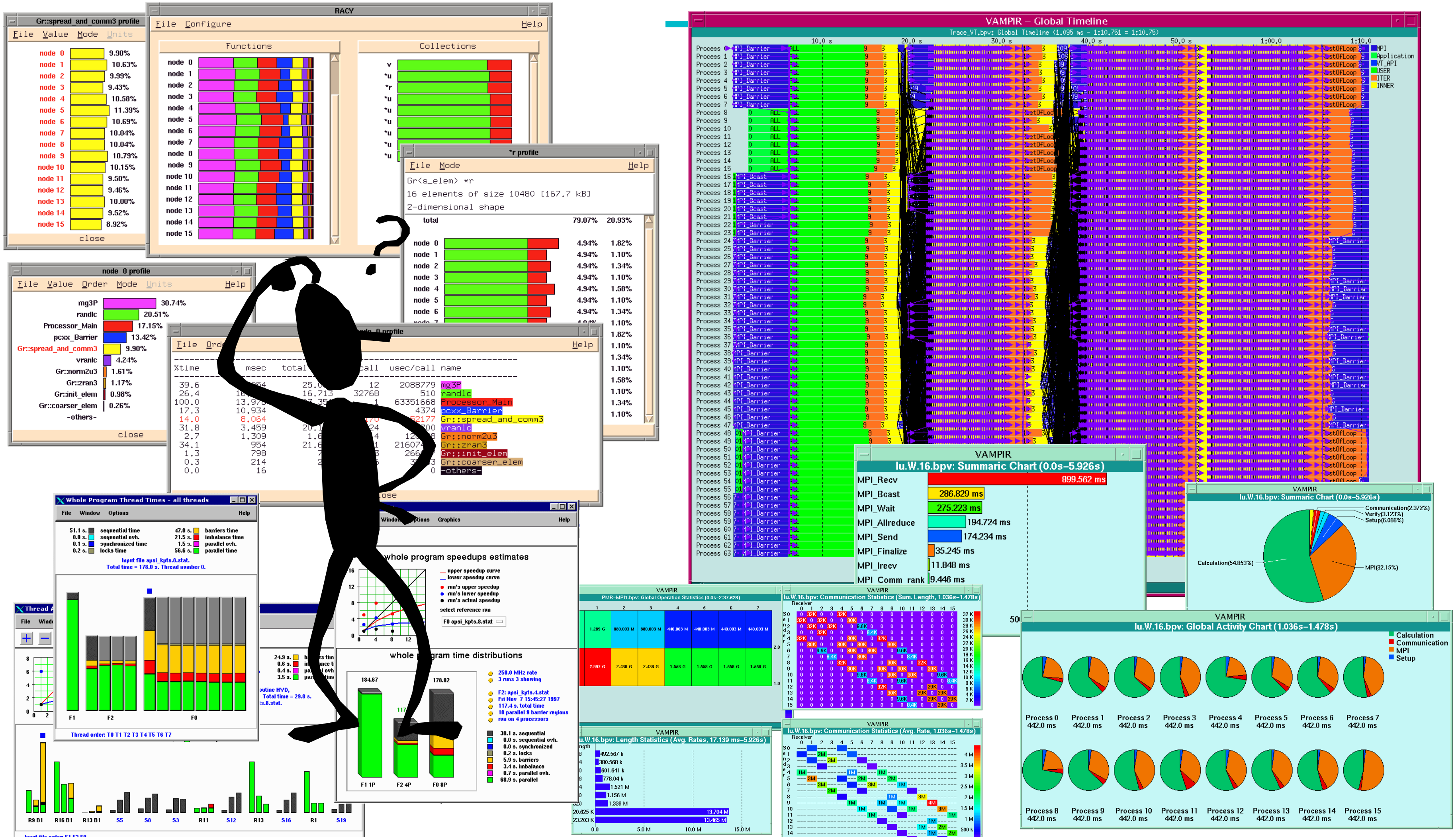
"The single most important impediment to good parallel performance is *still* poor single-node performance."

- William Gropp
  Argonne National Lab

Beware the Fallacy of Reported Linear Scalability

- But what about per-core performance?
- With a slow code, overall performance of the code is not vulnerable to other system parameters like communication bandwidth, latency.
- Very common on tightly integrated systems where you can simple add PE's for performance.

Which Tool?



Gr::spread_and_comm3 profile
RACY
Functions
Collections
*r profile
node 0 profile
Whole Program Thread Times - all threads
whole program speedups estimates
whole program time distributions
VAMPIR – Global Timeline
VAMPIR
lu.W.16.bpv: Summaric Chart (0.0s–5.926s)
MPI_Recv
MPI_Bcast
MPI_Wait
MPI_Allreduce
MPI_Send
MPI_Finalize
MPI_Irecv
MPI_Comm_rank
lu.W.16.bpv: Global Activity Chart (1.036s–1.478s)

Why a Kernel Patch is Needed?

- PMU registers are of finite and limited precision
  - PMU registers are virtualized into 64 bit quantities.
- Measure multiple users/processes/threads simultaneously
  - Context switch boundaries are preserved just like the FPU registers
  - Measurements should be independent of system load/activity.

PerfCtr Kernel Subsystem

- Lightweight and thin kernel patch that provides minimal set of functionality for hardware performance analysis.
  - Efficient code structure with lazy updates.
  - System wide and per-thread counting.
  - First-person and third-person (attach) operation.
  - Platform support:
    - x86[_64] and ppc
    - Many kernel revisions and distributions
  - Actively supported
- The recommended patch as of today for production installation.

PerfCtr: More Information

- Websites
  - http://user.it.uu.se/~mikpe/linux/perfctr/
  - http://sourceforge.net/projects/perfctr/
- Mailing Lists
  - perfctr-devel@lists.sourceforge.net

The Perfmon2 Kernel Subsystem

- Additional features over PerfCtr:
  - Buffered interrupts with sampling.
  - Kernel mode multiplexing.
  - Flexible event sampling interface for advanced hardware.
    - Event address registers
    - Branch trace buffers
  - Additional platforms: ia64,ppc,sparc64,mips,cell
- In the pipe for adoption (see LKML).
- Support of many vendors.

PerfMon2: More Information

- Websites
  - http://perfmon2.sourceforge.net/
- Mailing Lists
  - perfmon2-devel@lists.sourceforge.net

PAPI

- Performance Application Programming Interface
- The purpose of PAPI is to implement a standardized portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- The goal of PAPI is to facilitate the optimization of parallel and serial code performance by encouraging the development of cross-platform optimization tools.

PAPI

- Ad-hoc standard library for the implementation of application performance analysis tools.
- 2 level API, high-level (apps) and low-level (tools)
- Provides first and third person semantics for ‘thread-centric’ counting and sampling based on PMU events.
- Handles the ‘gory details’ and allows one to focus on tool development.
- Portable: write once, run anywhere.

3.

PAPI Events

- Preset events: proposed set of events deemed most relevant for application performance tuning.
  - Mappings from symbolic names to machine specific definitions for a particular hardware resource.
    - Total Cycles is PAPI_TOT_CYC
- Native events: performance metrics as specified by the hardware reference documentation.
  - Usually require a more detailed understanding of the architecture.
  - Names are different for every architecture.
- PAPI also supports presets that may be derived from the underlying hardware metrics.

PAPI Presets on IA64

Preset	Description
PAPI_L1_DCM	Level 1 data cache misses
PAPI_L1_ICM	Level 1 instruction cache misses
PAPI_L2_DCM	Level 2 data cache misses
PAPI_L2_ICM	Level 2 instruction cache misses
PAPI_L3_DCM	Level 3 data cache misses
PAPI_L3_ICM	Level 3 instruction cache misses
PAPI_L1_TCM	Level 1 cache misses
PAPI_L2_TCM	Level 2 cache misses
PAPI_L3_TCM	Level 3 cache misses
PAPI_CA_SNP	Requests for a snoop
PAPI_CA_INV	Requests for cache line invalidation
PAPI_L3_LDM	Level 3 load misses
PAPI_L3_STM	Level 3 store misses
PAPI_TLB_DM	Data translation lookaside buffer misses
PAPI_TLB_IM	Instruction translation lookaside buffer misses
PAPI_TLB_TL	Total translation lookaside buffer misses
PAPI_L1_LDM	Level 1 load misses
PAPI_L2_LDM	Level 2 load misses
PAPI_L2_STM	Level 2 store misses
PAPI_L3_DCH	Level 3 data cache hits
PAPI_STL_ICY	Cycles with no instruction issue
PAPI_STL_CCY	Cycles with no instructions completed
PAPI_BR_MSP	Conditional branch instructions mispredicted
PAPI_BR_PRC	Conditional branch instructions correctly predicted
PAPI_TOT_IIS	Instructions issued
PAPI_TOT_INS	Instructions completed
PAPI_LD_INS	Load instructions
PAPI_SR_INS	Store instructions
PAPI_BR_INS	Branch instructions
PAPI_RES_STL	Cycles stalled on any resource
PAPI_FP_STAL	Cycles the FP unit(s) are stalled
PAPI_TOT_CYC	Total cycles

Other Noteworthy Middleware

- Monitor
  - Dynamically insert callbacks for relevant events, thread creation, destruction, library loading, fork/exec, etc...
- DyninstAPI
  - Full dynamic instrumentation infrastructure
    - In-memory and on-disk binary instrumentation
- SymtabAPI
  - Efficient symbol table lookup
- Mrnet
  - Multicast-reduction network for efficient exchange of data by parallel tools
- Libpfm/libperfctr: low level counter access libs

Internal Timers

- C/C++
  - **getrusage()**, process (user & kernel)
  - **times()**, process (user & kernel)
  - **gettimeofday()**, wallclock
  - **clock_gettime()**,wallclock
- Fortran
  - **call cpu_time (value)**, process (user+kernel)
  - **call etime (array,cpu_time)**, process (user & kernel, sum)
  - **call second (value)**, process (user+kernel)
  - **call system_clock (count,rate,max)**, wallclock

Internal Timers(2)

- **`MPI_Wtime()`**, microseconds
- Timers are usually not synchronized between nodes beyond what NTP can achieve.
- Latency is different than resolution.
- Timers may be expensive,
  - Many calls to this function will affect your wall clock time.
- Arch specific timers:
  - TSC register (Intel, PPC, etc)
    - May not even be synchronized among cores
    - May change rate
  - Central switch clock (BG, SP, Altix)

PAPI Performance **Ex**periment Tools

- Set of commands that provide the interface to the underlying performance monitoring tools.
  - All are based on Monitor and PAPI
- papiex, mpipex, ioex, hpcex, gptlex
  - Easy to use as /bin/time, generating concise text output where appropriate.
  - Take the same arguments, except for tool-specific options.
  - Provide standard and HTML man pages and documentation.
  - Requires no recompilation.
  - Monitors all subprocesses/threads.
  - Output goes to stderr or a file.

PapiEx

- Used to obtain summary information about an application using PAPI and other metrics.
- Represents the first pass of application performance evaluation.
- It provides:
  - Memory footprint
  - Percent of time in I/O
  - Percent of time in MPI
  - PAPI, native and derived metrics
  - Provides per-thread, per-task and per-job summaries
  - Simple instrumentation API for further focus

PapiEx Output

```
PapiEx Version:          0.99rc2
Executable:              /afs/pdc.kth.se/home/m/mucci/summer/a.out
Processor:               Itanium 2
Clockrate:               900.000000
Parent Process ID:       8632
Process ID:              8633
Hostname:                h05n05.pdc.kth.se
Options:                 MEMORY
Start:                   Wed Aug 24 14:34:18 2005
Finish:                  Wed Aug 24 14:34:19 2005
Domain:                  User

Real usecs:              1077497
Real cycles:             969742309
Proc usecs:              970144
Proc cycles:             873129600

PAPI_TOT_CYC:            850136123
PAPI_FP_OPS:             40001767

Mem Size:                4064
Mem Resident:            2000
Mem Shared:              1504
Mem Text:                16
Mem Library:             2992
Mem Heap:                576
Mem Locked:              0
Mem Stack:               32
```

Papiex: Workload Characterization

```
MFLIPS ..........................................     66.51
IPC .............................................      0.40
CPU Utilization .................................      0.96
% Memory Instructions ...........................     39.02
% FP Instructions ...............................     33.38
% Branch Instructions ...........................     18.87
% Integer Instructions ..........................     66.62
Loads/Stores Ratio ..............................     18.14
L1 D-cache Hit % ................................     97.22
L1 I-cache Hit % ................................    100.00
D-TLB Hit % .....................................     87.43
I-TLB Hit % .....................................     99.97
FP ins. per D-cache Miss ........................     30.72
Computational Intensity .........................      0.86
Branch Misprediction % ..........................     14.47
Dual Issue % ....................................     11.41
Est. Stall % ....................................     17.06
Est. L1 D-cache Miss Stall % ....................      7.79
Est. L1 I-cache Miss Stall % ....................      0.02
Est. D-TLB Miss Stall % .........................      3.91
Est. I-TLB Miss Stall % .........................      0.03
Est. TLB Trap Stall % ...........................      0.00
Est. Mispred. Branch Stall % ....................      1.09
Dependency Stall % ..............................      4.22
T: Actual/Ideal Cycles ..........................      3.77
T: Ideal (max dual) MFLIPS ......................    250.55
P: Actual/Ideal Cycles ..........................      2.83
P: Ideal (curr dual) MFLIPS .....................    188.41
% MPI Cycles ....................................     18.49
% I/O Cycles ....................................      0.02
```

PapiEx Caliper Fortran Example

```
#include "papiex.h"

      program zero

      real a, b, c;
      a = 0.1
      b = 1.1
      c = 2.1

      PAPIEX_START_ARG(1,"write")
      print *, "Doing 10000000 iters. of a += b * c on doubles."
      PAPIEX_STOP_ARG(1)

      PAPIEX_START_ARG(2,"do loop")
      do i=1,100000000
         a = a + b * c
      end do
      PAPIEX_STOP_ARG(2)

      end
```

Ioex

- Used to characterize the I/O performance of an application.
  - Based on concepts from IOtrack written at PDC/KTH.
- Per-file statistics:
  - Flags
  - Access type
  - Bandwidth
  - Chunk size
  - Time spent

Ioex: Per-file profile

```
File: /dev/zero
  open64
    calls                                   :            1
  read
    calls                                   :           10
    usecs                                   :          587
    usecs/call                              :           58
    bytes                                   :     10485760
    bytes/call                              :      1048576
    MB/s                                    :        17863
File: /home/out
  open64
    calls                                   :            1
    flags                                   : O_WRONLY|O_CREAT|O_TRUNC
  write
    calls                                   :           10
    usecs                                   :       157444
    usecs/call                              :        15744
    bytes                                   :     10485760
    bytes/call                              :      1048576
    MB/s                                    :           66
```

PapiEx: More Information

- Multiple tools in one:
  - MpiP: mpipex
  - HPCToolkit: hpcex
  - GPTL: gptlex
  - PAPI: papiex
  - IO: ioex
- Project websites
  - http://www.cs.utk.edu/~mucci/papiex
- Mailing list
  - ptools-perfapi@cs.utk.edu

MPI Performance Analysis

- There are 2 modes, both accomplished through intercepting the calls to MPI.
  - Aggregate
  - Tracing
- Often aggregate is sufficient.
  - MPIP, FMPI
- Tracing
  - Jumpshot and the MPE libraries.
  - Vampir
  - Intel Trace Analyzer

mpiP: Lightweight MPI Profiling

- Used to characterize the MPI performance of an application and quickly find MPI bottlenecks.
- It provides:
  - MPI load balance
  - MPI function profile
  - Message size distribution
  - Call site information: file, function and line
- Used by simply relinking with the mpiP library.
  - Preloading can be used

MPI Profile by Callsite

```
---------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) -
---------------------------------------------------------------
Call                 Site       Time    App%    MPI%     COV
Barrier                29   9.65e+05    4.96   30.20    0.00
Barrier                18    6.1e+05    3.14   19.10    0.21
Allgather              12   3.68e+05    1.89   11.51    0.47
Barrier                43   3.25e+05    1.67   10.18    0.43
Sendrecv               78    2.2e+05    1.13    6.88    2.19
Sendrecv               21   1.57e+05    0.81    4.92    0.51
```

Load Balance

```
-------------------------------------------
@--- MPI Time (seconds) ---------------
-------------------------------------------
Task    AppTime    MPITime     MPI%
   0   1.06e+03       79.8     7.53
   1   1.06e+03       89.9     8.47
   2   1.06e+03       85.2     8.03
   3   1.06e+03       85.8     8.09
   4   1.06e+03       85.1     8.03
   5   1.06e+03        111    10.42
   6   1.06e+03        144    13.54
   7   1.06e+03        142    13.37
   8   1.06e+03        139    13.12
   9   1.06e+03        147    13.85
  10   1.06e+03        140    13.16
  11   1.06e+03        141    13.33
  12   1.06e+03        143    13.47
  13   1.06e+03        138    13.03
  14   1.06e+03        144    13.55
  15   1.06e+03        182    17.19
   *    1.7e+04      2e+03    11.76
```

MPIP Output

```
@ Command : /afs/pdc.kth.se/home/m/mucci/mpiP-2.7/testing/./sweep-ops-stack.exe
/tmp/SPnodes-mucci-0
@ Version                    : 2.7
@ MPIP Build date            : Aug 17 2004, 17:04:36
@ Start time                 : 2004 08 17 17:08:48
@ Stop time                  : 2004 08 17 17:08:48
@ MPIP env var               : [null]
@ Collector Rank             : 0
@ Collector PID              : 17412
@ Final Output Dir           : .
@ MPI Task Assignment        : 0 h05n05-e.pdc.kth.se
@ MPI Task Assignment        : 1 h05n35-e.pdc.kth.se
@ MPI Task Assignment        : 2 h05n05-e.pdc.kth.se
@ MPI Task Assignment        : 3 h05n35-e.pdc.kth.se

@--- MPI Time (seconds) ---------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0      0.084     0.0523    62.21
   1     0.0481      0.015    31.19
   2      0.087     0.0567    65.20
   3     0.0495     0.0149    29.98
   *      0.269      0.139    51.69

@--- Aggregate Time (top twenty, descending, milliseconds) ---------------
---------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%
Barrier                 1        112   41.57   80.42
Recv                    1       26.2    9.76   18.89
Allreduce               1      0.634    0.24    0.46
Bcast                   1        0.3    0.11    0.22
Send                    1      0.033    0.01    0.02
```

MPIP Output

```
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
---------------------------------------------------------------------------
Call                 Site      Count      Total       Avrg  Sent%
Allreduce               1          8    4.8e+03        600  46.15
Bcast                   1          8    4.8e+03        600  46.15
Send                    1          2        800        400   7.69
---------------------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 16 --------------------
---------------------------------------------------------------------------
Name              Site Rank  Count      Max     Mean      Min   App%   MPI%
Allreduce            1    0      2    0.105    0.087    0.069   0.21   0.33
Allreduce            1    1      2    0.118     0.08    0.042   0.33   1.07
Allreduce            1    2      2     0.11    0.078    0.046   0.18   0.27
Allreduce            1    3      2    0.102    0.072    0.042   0.29   0.97
Barrier              1    0      3     51.9     17.3    0.015  61.86  99.44
Barrier              1    1      3    0.073   0.0457    0.016   0.29   0.91
Barrier              1    2      3     54.9     18.8    0.031  64.90  99.53
Barrier              1    3      3     1.56     1.02    0.035   6.20  20.68
Bcast                1    0      2    0.073   0.0535    0.034   0.13   0.20
Bcast                1    1      2    0.037    0.023    0.009   0.10   0.31
Bcast                1    2      2    0.084    0.046    0.008   0.11   0.16
Bcast                1    3      2     0.03   0.0275    0.025   0.11   0.37
Recv                 1    1      1     14.6     14.6     14.6  30.48  97.71
Recv                 1    3      1     11.6     11.6     11.6  23.37  77.98
Send                 1    0      1    0.013    0.013    0.013   0.02   0.02
Send                 1    2      1     0.02     0.02     0.02   0.02   0.04
Send                 1    *     32     54.9     4.34    0.008  51.69 100.00
---------------------------------------------------------------------------
@--- Callsite Message Sent statistics (all, sent bytes) ------------------
---------------------------------------------------------------------------
Name              Site Rank   Count       Max      Mean       Min       Sum
Allreduce            1    0       2       800       600       400      1200
Allreduce            1    1       2       800       600       400      1200
Allreduce            1    2       2       800       600       400      1200
Allreduce            1    3       2       800       600       400      1200
Bcast                1    0       2       800       600       400      1200
Bcast                1    1       2       800       600       400      1200
Bcast                1    2       2       800       600       400      1200
Bcast                1    3       2       800       600       400      1200
Send                 1    0       1       400       400       400       400
Send                 1    2       1       400       400       400       400
Send                 1    *      18       800     577.8       400  1.04e+04
---------------------------------------------------------------------------
@--- End of Report --------------------------------------------------------
```

mpiP: More Information

- Project websites
  - http://mpip.sourceforge.net/
- Mailing list
  - mpip-help@lists.sourceforge.net
  - mpip-users@lists.sourceforge.net
- Similar tool:
  - FMPI: Fast MPI Profiling

HPCToolkit

- A statistical profiling package based on interrupts from the performance monitoring hardware.
- No instrumentation required, but compiling with -g helps.
- 3 phase:
  - Collection
  - Analysis (optional)
  - Presentation/visualization.

Hpcrun

- Used to produce statistical profiles without instrumentation.
  - Based on HPCToolkit from Rice University.
- Take interrupts when a counter overflows a certain threshold.
  - i.e. every 10000 cache misses, interrupt/sample the PC.
  - Supports multiple simultaneous profiles
- Data is viewed with hpcprof (text) and hpcviewer (Java GUI)
  - Advanced source code correlation and visualization through bloop (a binary analyzer) and hpcviewer.
- Profile by load module, file, function, line and even instruction.

Hpcprof: Hotspot analyses

```
Columns correspond to the following events [event:period (events/sample)]
  PAPI_TOT_CYC:999999 - Total cycles (2553 samples)

Load Module Summary:
  65.5% testconv2d
  34.5% /lib64/libc-2.5.so

File Summary:
  36.9% <<testconv2d>>/home/phil/ISC/new/convolution/simplest_conv.c
  34.5% <</lib64/libc-2.5.so>><unknown>
  10.0% <<testconv2d>>/home/phil/ISC/new/convolution/support.c
   9.8% <<testconv2d>>/home/phil/ISC/new/convolution/testconv2d.c
   8.8% <<testconv2d>>/home/phil/ISC/new/convolution/convCore.c

Function Summary:
  36.9% <<testconv2d>>conv2d_simple
  17.0% <</lib64/libc-2.5.so>>random
  12.9% <</lib64/libc-2.5.so>>random_r
  10.0% <<testconv2d>>makeRandomDouble
   9.8% <<testconv2d>>main
   8.8% <<testconv2d>>conv2dBy3TileZero
   4.6% <</lib64/libc-2.5.so>>rand

Line Summary:
  34.5% <</lib64/libc-2.5.so>><unknown>:0
  26.1% <<testconv2d>>/home/phil/ISC/new/convolution/simplest_conv.c:27
   6.5% <<testconv2d>>/home/phil/ISC/new/convolution/simplest_conv.c:24
```

ICL UT

KTH

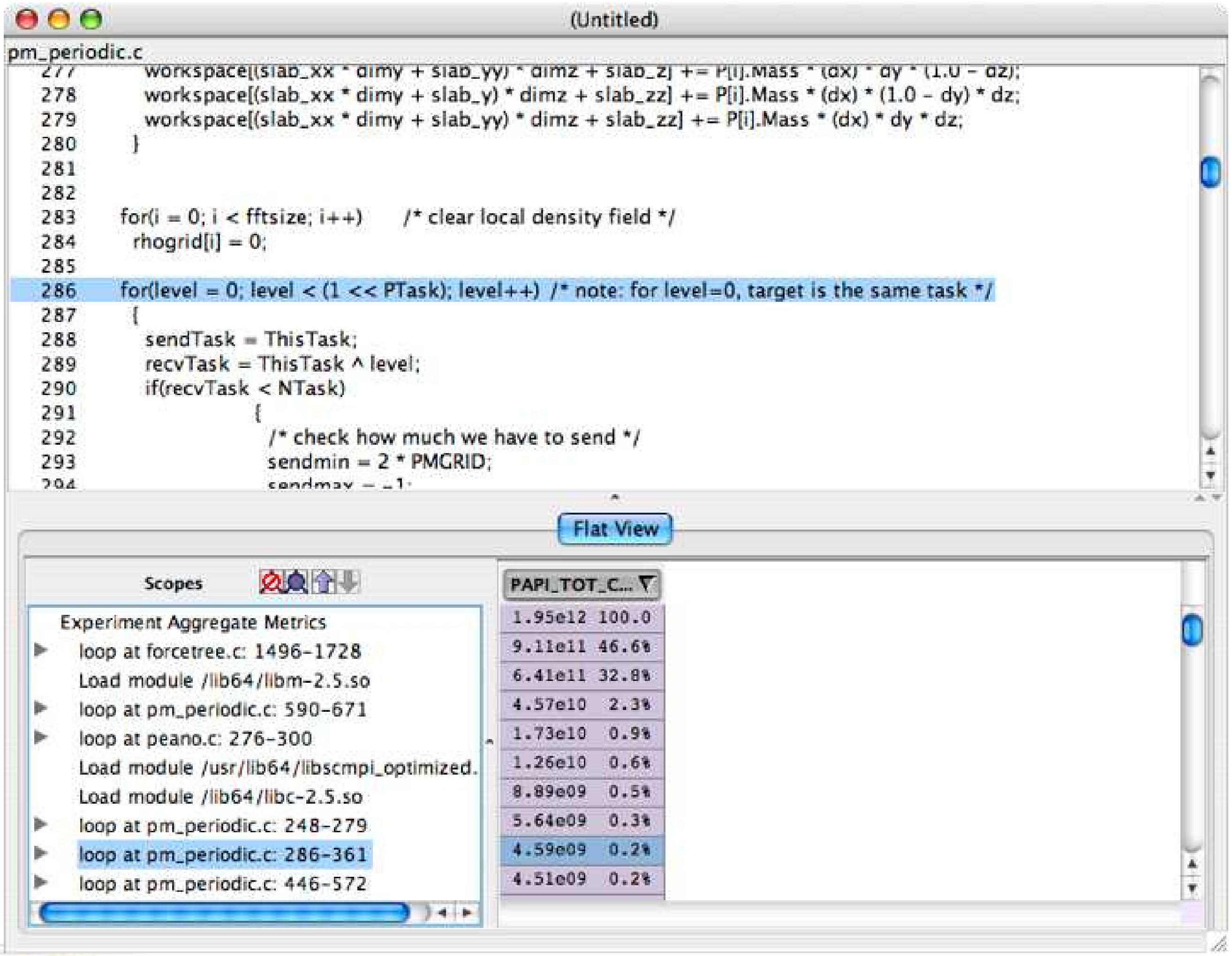
Hpcprof: Source code annotation

```
 0.8%     for (j = coff; j < nca-coff; j++)
            {
 0.1%       out = 0.0;
 2.5%       for (ki = 0; ki < nrk; ki++)
              {
 6.5%         for (kj = 0; kj < nck; kj++)
                {
                // out += a[i+ki][j+kj] * k[ki][kj];
26.1%           out += *(a+(i+ki-roff)*nca + j+kj-coff) * *(k+(ki*nck)+kj);
                }
              }
            // c[i+roff][j+coff] = out;
 1.0%       *(c+(i)*nca + j) = out;
            }
          }
```

Hpcprof: Assembly annotation

```
0x1200068c0:    0.01% move      v0,v1
0x1200068c4:    0.06% daddu     a0,a2,v0
0x1200068c8:    0.60% dsll      a1,a0,0x3
0x1200068cc:    5.48% ld        v0,48(s8)
0x1200068d0:    0.01% daddu     v1,a1,v0
0x1200068d4:    4.18% ldc1      $f0,0(v1)
0x1200068d8:          mul.d     $f2,$f3,$f0
0x1200068dc:    0.03% ldc1      $f1,8(s8)
0x1200068e0:          add.d     $f0,$f1,$f2
0x1200068e4:    0.04% sdc1      $f0,8(s8)
0x1200068e8:    5.04% lw        v0,16(s8)
0x1200068ec:    0.01% addiu     v1,v0,1
0x1200068f0:    6.60% sw        v1,16(s8)
0x1200068f4:    7.80% lw        v0,16(s8)
0x1200068f8:    0.02% lw        v1,60(s8)
0x1200068fc:    0.03% slt       a0,v0,v1
0x120006900:    0.02% bnez      a0,0x12000683c
```

Hpcviewer: Loop-level profiling



HPCToolkit: More Information

- Project websites
  - http://www.hipersoft.rice.edu/hpctoolkit/
  - http://lacsi.rice.edu/software/hpctoolkit/

Pfmon

- Used to perform highly focused instrumentation and/or advanced sampling.
  - Uses libpfm and the Perfmon2 kernel subsystem..
- Per-thread, per-CPU, system-wide sampling and counting.
- Allows one to attach to a running code.
- Limited but highly accurate dynamic instrumentation support.
- Very rich feature set, but complicated interface.
- Not MPI aware.

Pfmon: More Information

- Project websites
  - http://www.hpl.hp.com/research/linux/perfmon/pfmon.php4
  - http://perfmon2.sourceforge.net/
- Mailing list
  - perfmon2-devel@lists.sourceforge.net

GPTL

- Used to easily instrument applications for the generation of performance data.
  - Developed at NCAR for inclusion into their applications.
- Optimized for usability.
- Provides access to timers as well as PAPI events.
- Thread-safe and per-thread statistics.
- Provides estimates of overhead.
- Call-tree generation.
- Preserves parent/child relationships.

GPTL: More Information

- Project websites
  - http://www.burningserver.net/rosinski/gptl/index.html

TAU Parallel Performance System

- Parallel Performance Evaluation Tool for Fortran, C, C++, Python and Java
- Used for in-depth performance studies of an application throughout its lifecycle.
- Supports Parallel Profiling
  - Flat, callpath, and phase based profiling
  - PerfDMF performance database and PerfExplorer cross experiment analysis tool
  - PAPI counters, wallclock time, CPU time
- Supports Event Tracing
  - Generates traces in OTF, Vampir, Kojak formats..
  - Supports Memory and PAPI counters in trace files with synchronized time stamps.

TAU Parallel Performance System

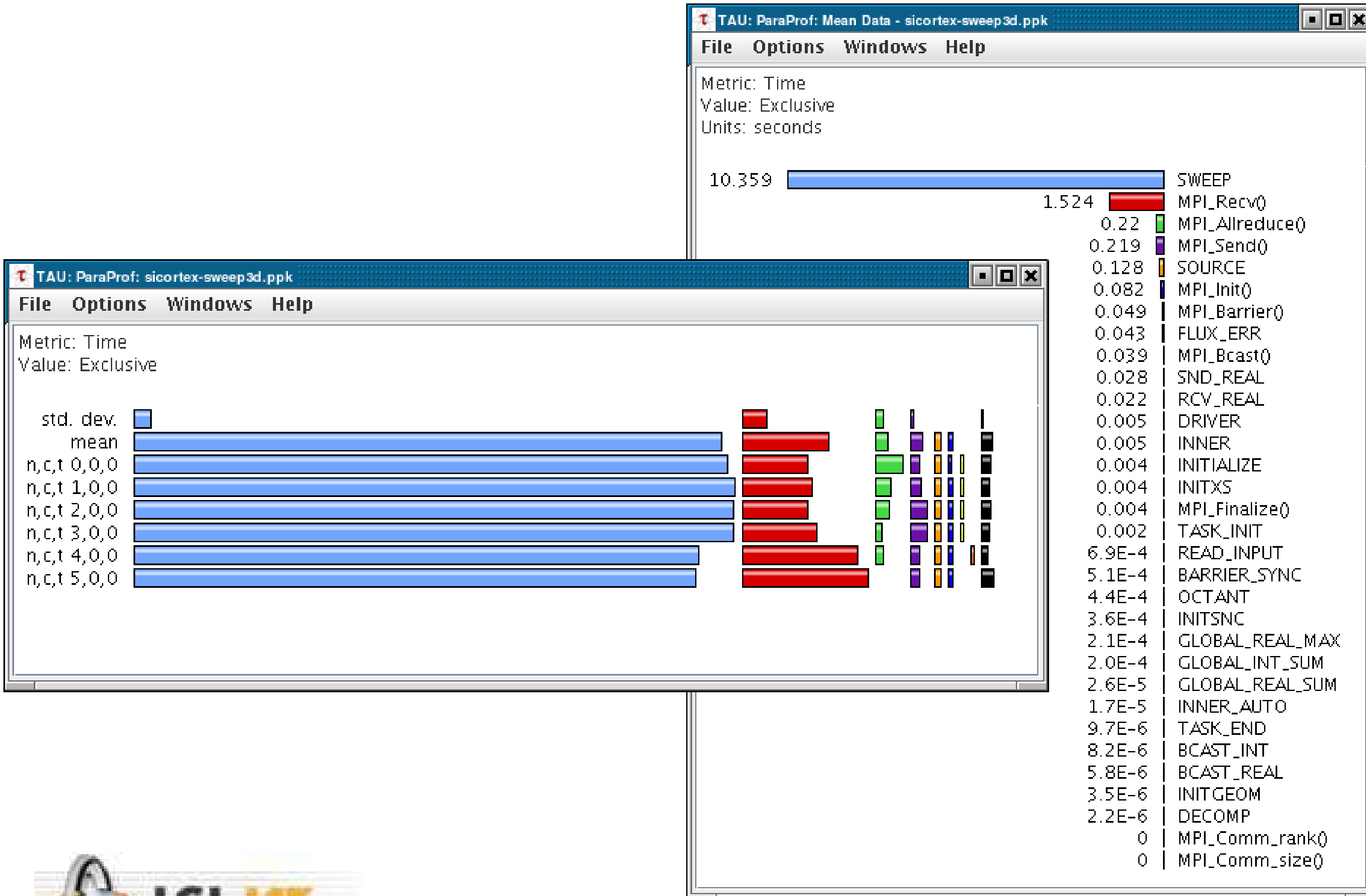
Tuning and Analysis Utilities

- Multi-level instrumentation
  - Source code (manual), pre-processor (Program Database Toolkit, PDT), MPI library
  - Memory, I/O instrumentation in Fortran and C/C++
  - Supports runtime throttling, selective instrumentation at routine and loop level.
- Widely-ported parallel performance profiling system.
  - All HPC systems, compilers, MPI-1 and 2 implementations, OpenMP and pthreads .

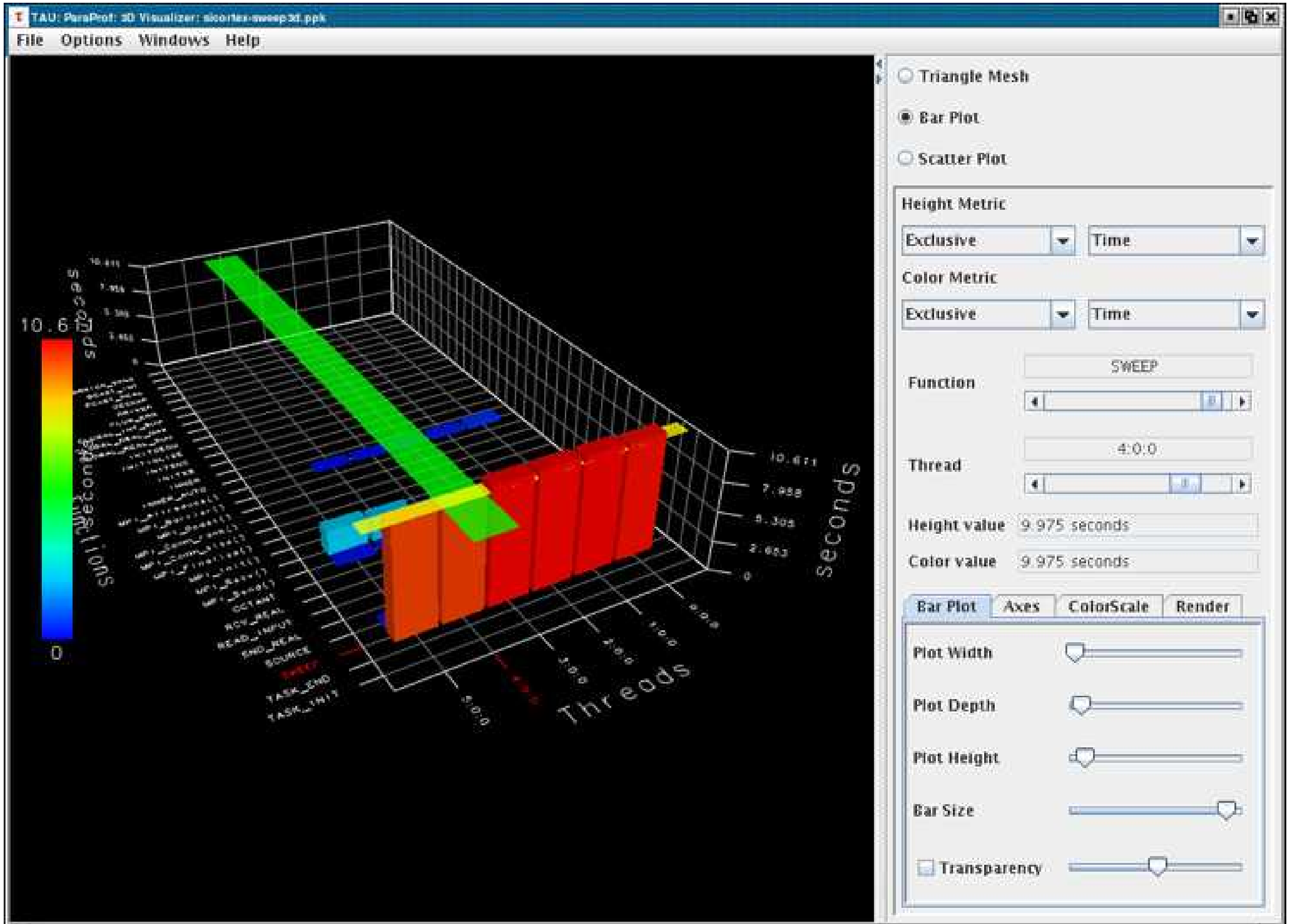
Tauex

- Used to control the behavior of the TAU performance system on instrumented and uninstrumented executables.
- Previously, TAU required extensive setup and relinking when options changed.
  - Now, all TAU options can be changed at run-time.

Paraprof Function Profile



ParaProf 3D Profile



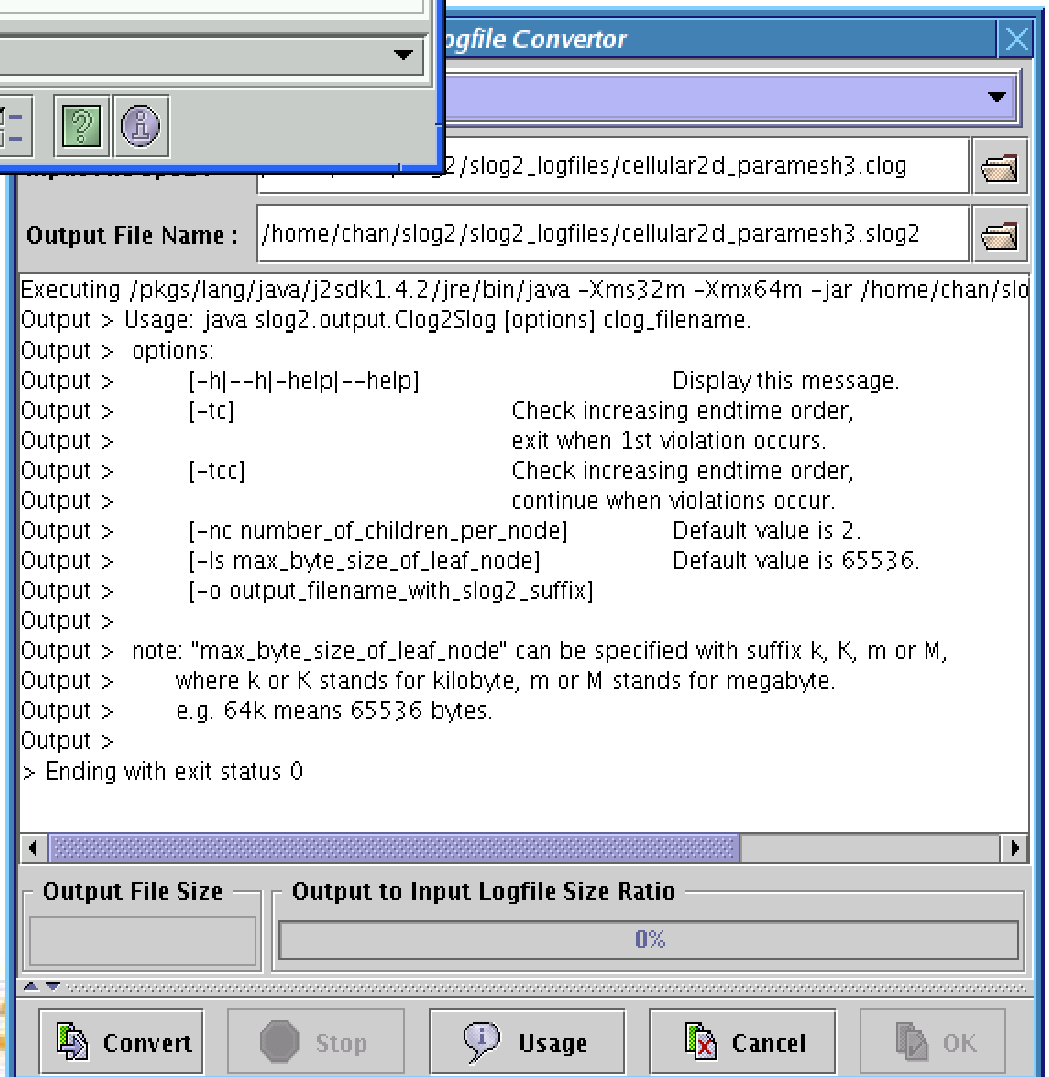
TAU: More Information

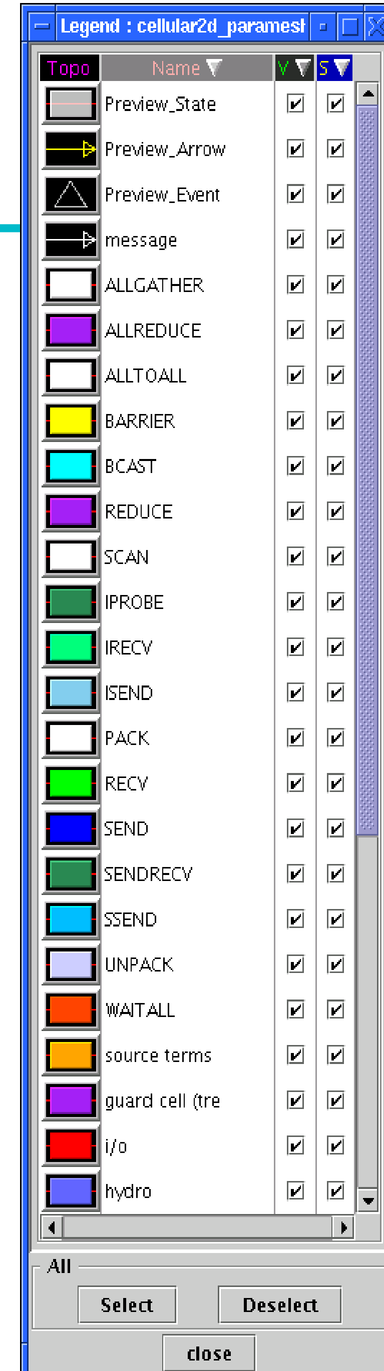
- Project websites
  - http://www.cs.uoregon.edu/research/tau/

MPI Tracing with Jumpshot

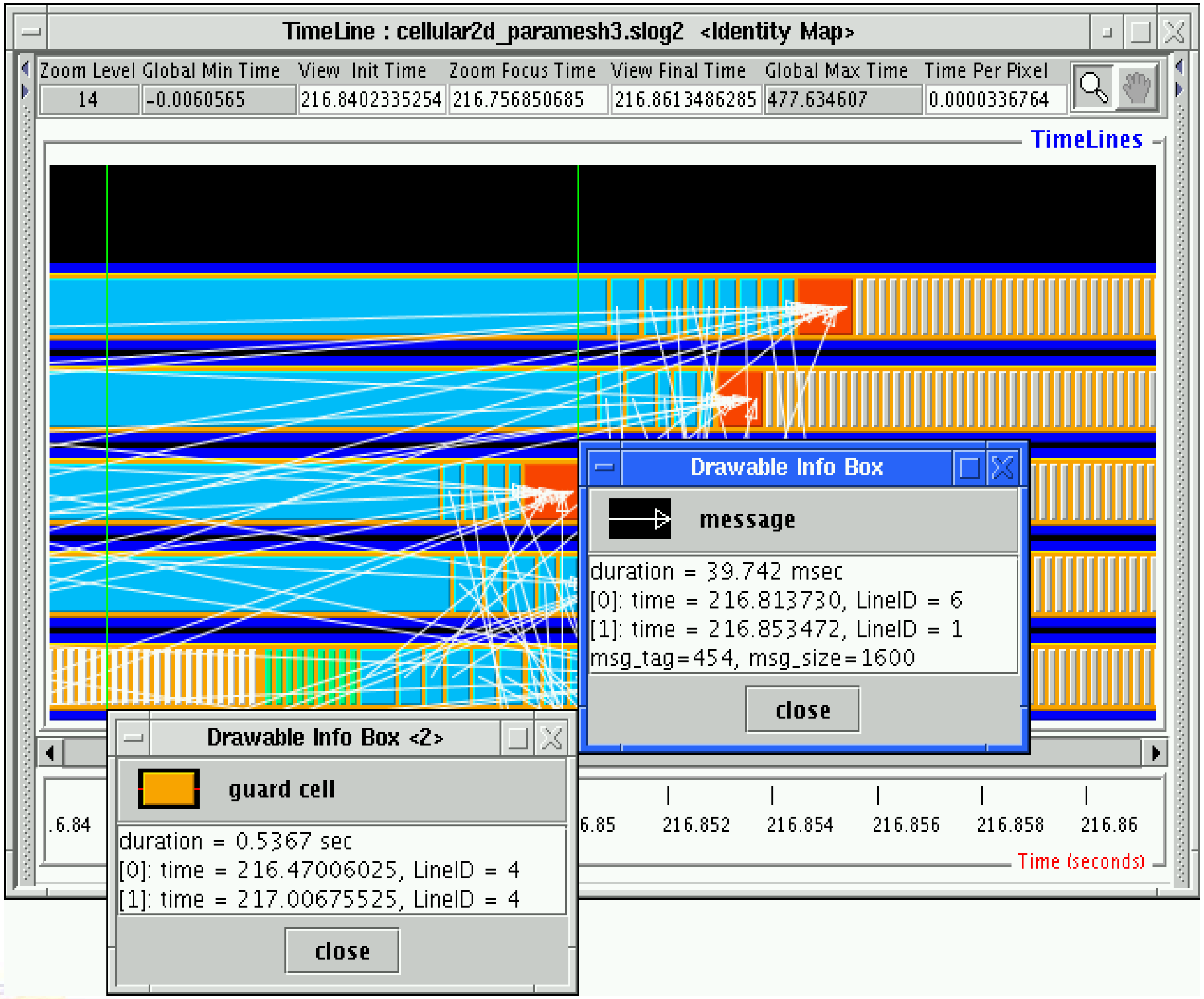
- Visualization and tracing infrastructure for MPI
  - Can also be used to instrument user code
- Based around the MPE tracing library
  - Relink your codes with MPE and run
- GUI and trace format has limited scalability yet it's a good Open Source solution for small(ish) runs.

Jumpshot Basics

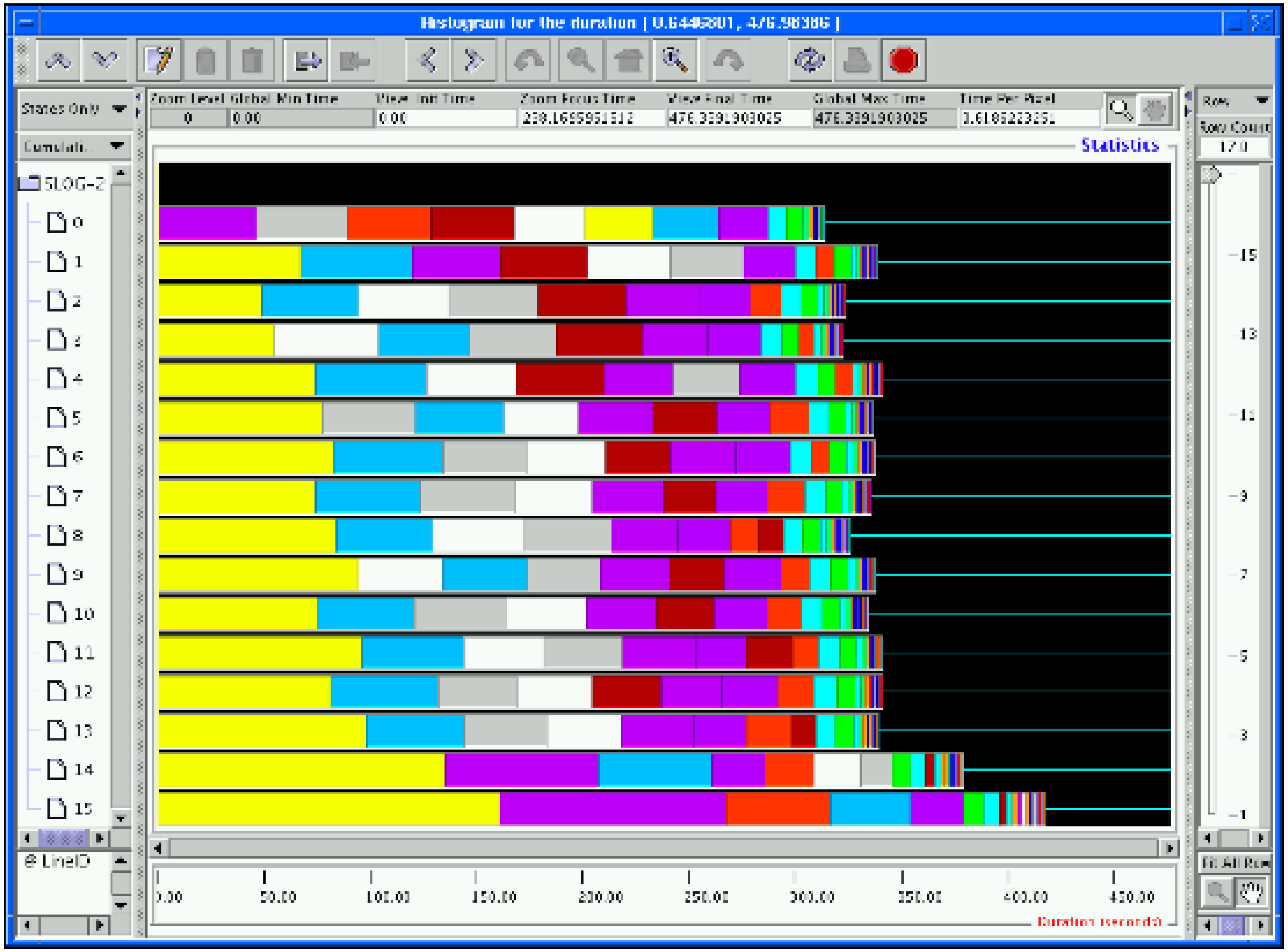




Jumpshot Zoomed Timeline



Jumpshot Histogram Window



Jumpshot: More Information

- Project websites
  - http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm
- Included with MPICH(1/2)

PerfSuite

- Small set of libraries and tools built upon them to provide basic, commonly-requested performance measurement capabilities for the average (not expert) HPC user
- Flexible data file formats (XML applications)
- Motivated by NCSA's move from traditional "supercomputers" (e.g., Cray, SGI, Convex, Thinking Machines) to Linux clusters
- Greatly influenced by SGI's perfex and SpeedShop
- Key enabling technologies: PAPI (UTK), Perfctr (Uppsala), and Perfmon (H-P)

PerfSuite Tools

- psrun
  - the PerfSuite variant of SGI's "perfex". Monitors unmodified dynamically-linked executables using libperfsuite/libpshwpc
  - Optionally provides resource measurement by creating a second monitoring thread
- psprocess
  - pre- and post-processing utility that interprets and presents raw data contained in PS XML docs to human-consumable form
  - Entirely written in Tcl scripting language + Tcl/C-coded extensions (for things like accessing PAPI/Perfmon from a Tcl script)
- psinv
  - system information utility, a la hinv, sysinfo, cpuinfo, etc.
  - What is the CPU type? (family, model, revision, etc)
  - What counters are available?
  - What are cache/TLB sizes?
  - Memory size, OS info
- psconfig
  - Tk point-and-click tool to make it easier to configure measurements
  - Not heavily used or maintained (lately)

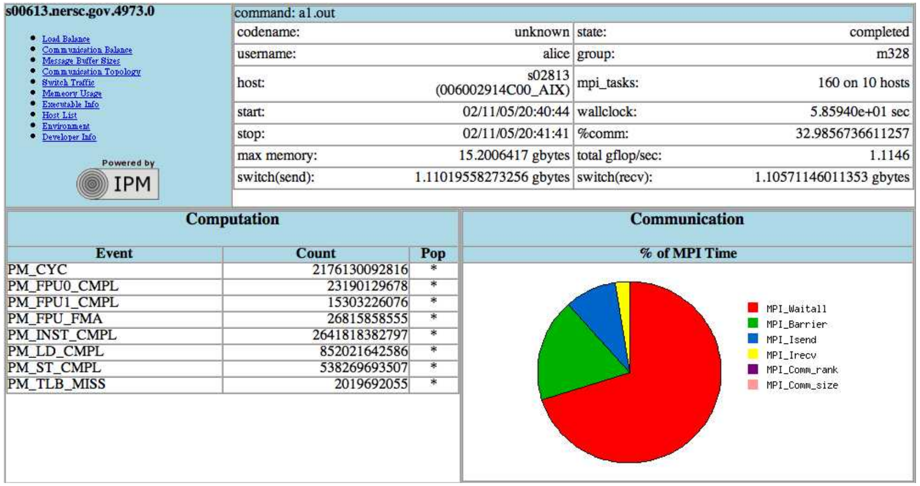
PerfSuite: More Information

- Project websites
  - http://www.sourceforge.net/projects/perfsuite/
  - http://perfsuite.ncsa.uiuc.edu/
- Email contacts and mailing lists
  - perfsuite@ncsa.uiuc.edu
  - perfsuite-users@lists.sourceforge.net
  - perfsuite-announce@lists.sourceforge.net
  - perfsuite-bugs@lists.sourceforge.net

IPM – Integrated Performance Monitor

- It is an easy-to-use, lightweight profiling infrastructure.
  - Based on MPI call interception and PAPI.
- It provides a concise summary of the performance of a parallel calculation.
- It has a low memory and CPU overhead.
- It is scalable to high concurrencies.
- It allows for the direct comparison of performance between different architectures.

IPM: HTML Output


s00613.nersc.gov.4973.0

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memory Usage
- Executable Info
- Host List
- Environment
- Developer Info

Powered by IPM

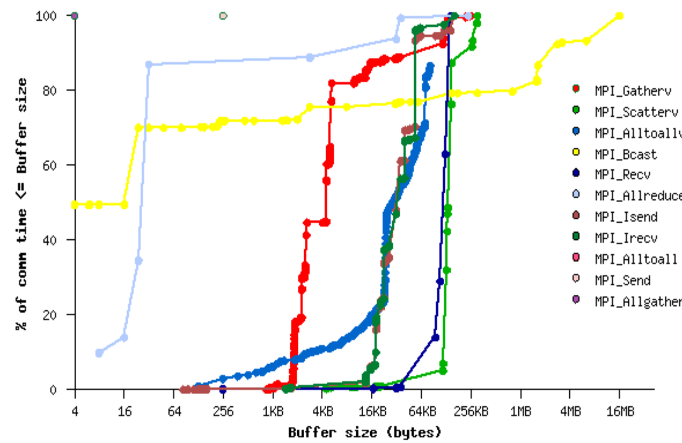
command: a1.out			
codename:	unknown	state:	completed
username:	alice	group:	m328
host:	s02813 (006002914C00_AIX)	mpi_tasks:	160 on 10 hosts
start:	02/11/05/20:40:44	wallclock:	5.85940e+01 sec
stop:	02/11/05/20:41:41	%comm:	32.9856736611257
max memory:	15.2006417 gbytes	total gflop/sec:	1.1146
switch(send):	1.11019558273256 gbytes	switch(recv):	1.10571146011353 gbytes

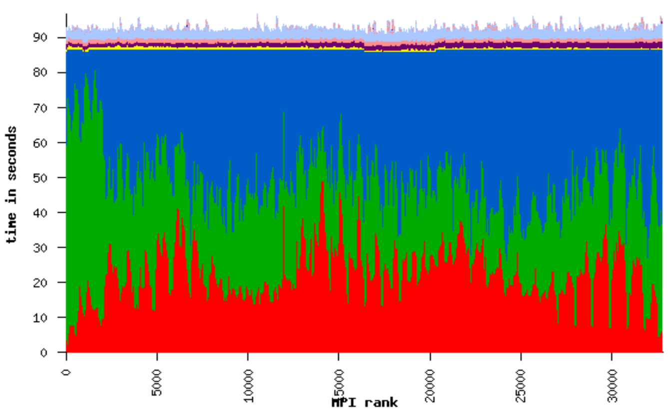
Computation

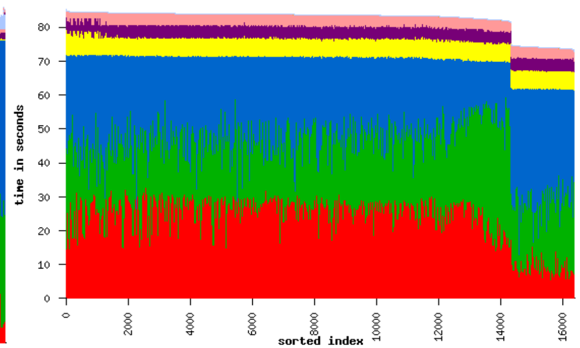
Event	Count	Pop
PM_CYC	2176130092816	*
PM_FPU0_CMPL	23190129678	*
PM_FPU1_CMPL	15303226076	*
PM_FPU_FMA	26815858555	*
PM_INST_CMPL	2641818382797	*
PM_LD_CMPL	852021642586	*
PM_ST_CMPL	538269693507	*
PM_TLB_MISS	2019692055	*

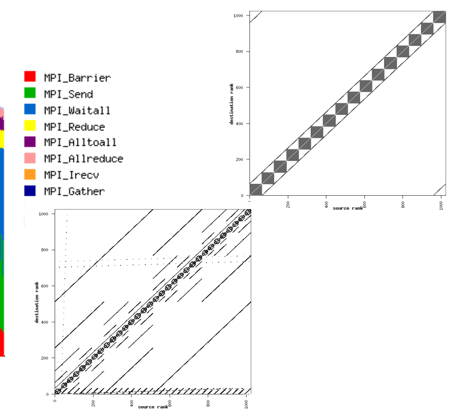
Communication

% of MPI Time









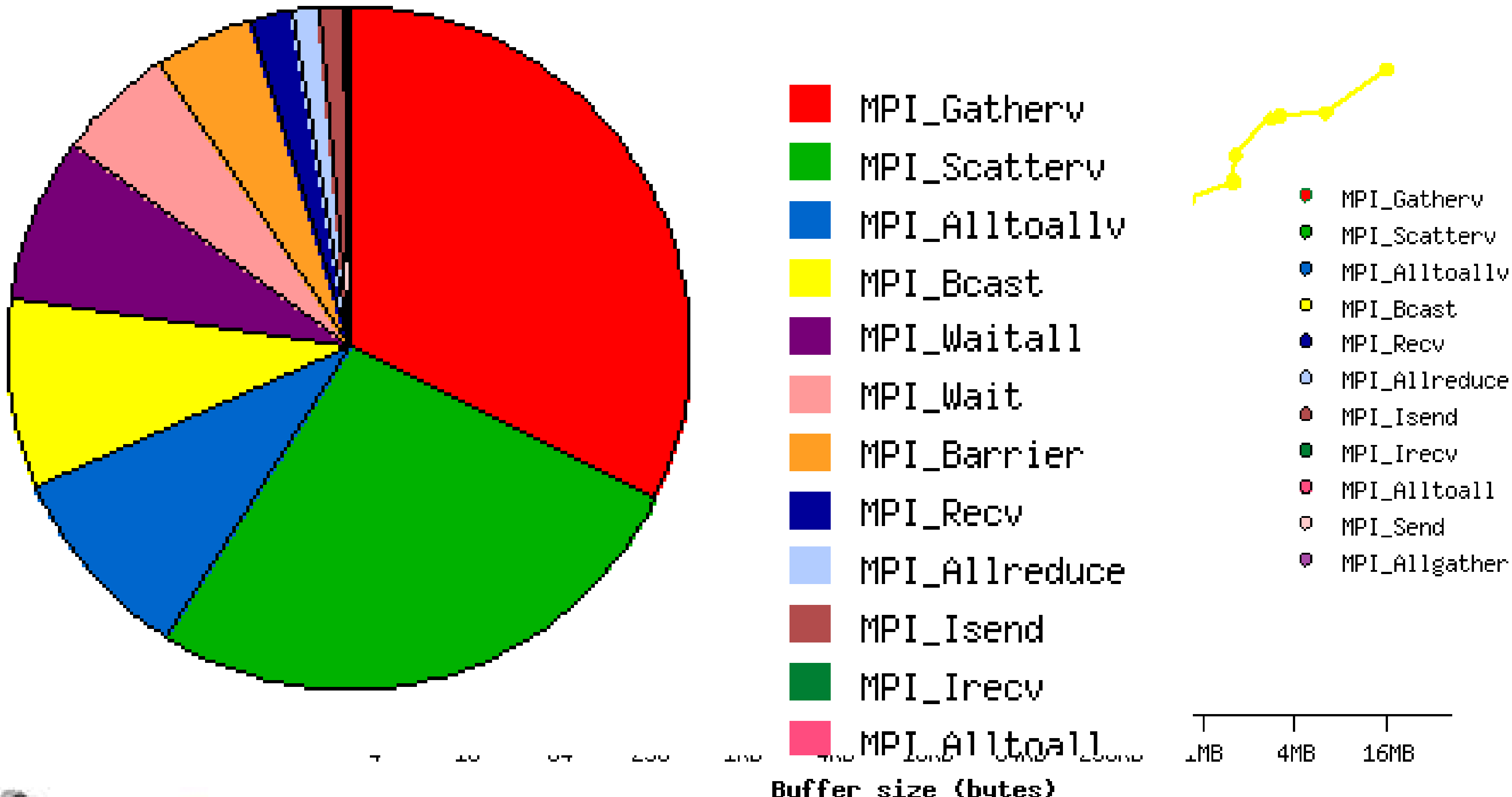
IPM: XML log files

- There's a lot more information in the logfile than you get to stdout. A logfile is written that has the hash table, switch traffic, memory usage, executable information, ...
- Parallelism in writing of the log (when possible)
- The IPM logs are durable performance profiles serving:
  - https://www.nersc.gov/nusers/status/llsum/
  - http://www.sdsc.edu/user_services/top/ipm/
  - http://www.nersc.gov/projects/ipm/ex3/
  - your own XML consuming entity, feed, or process

Message Sizes : CAM 336 way

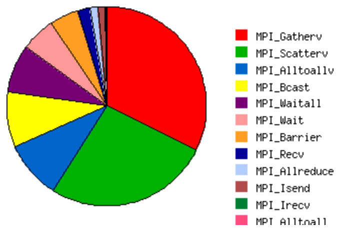
per MPI call

per MPI call & buffer size

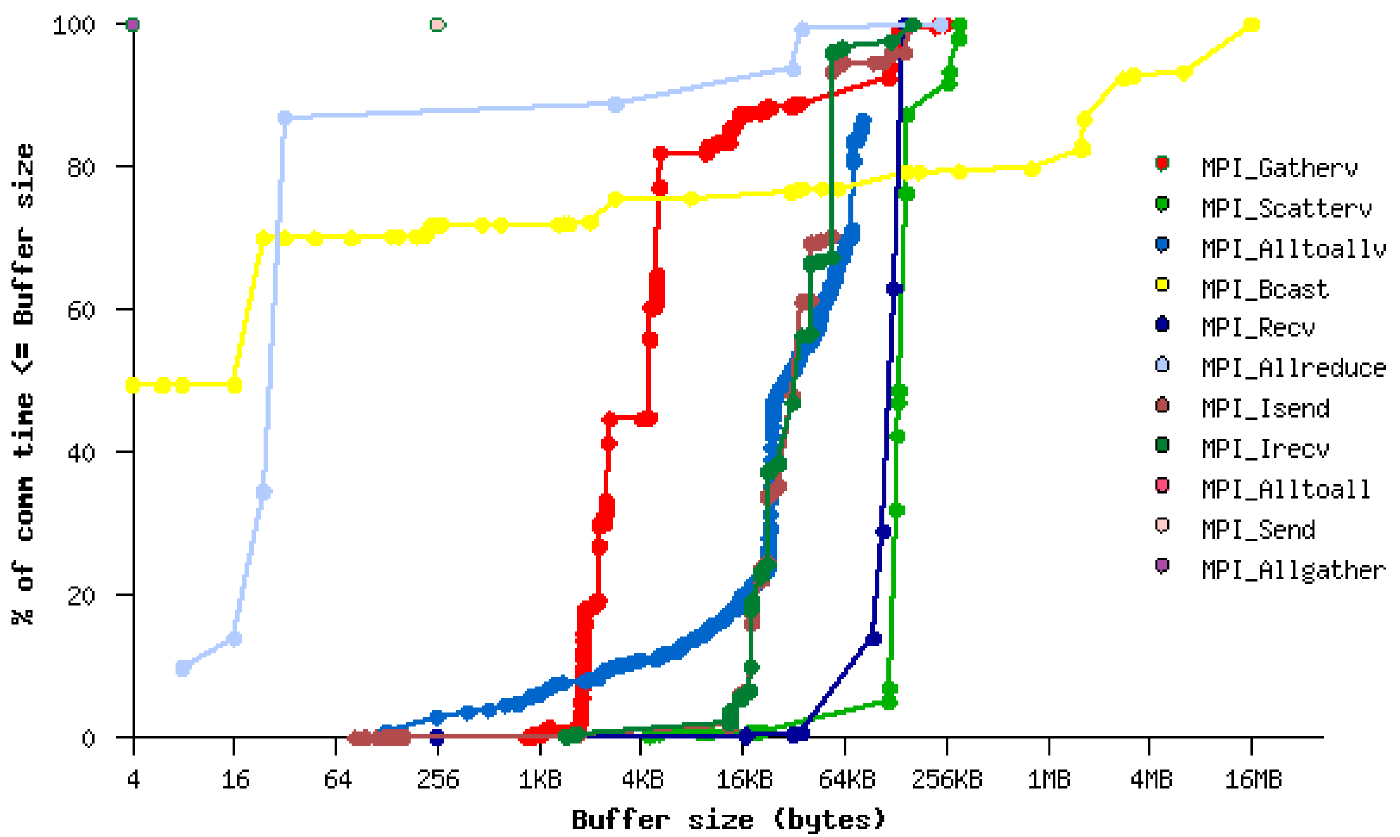


Message Sizes : CAM 336 way

per MPI call

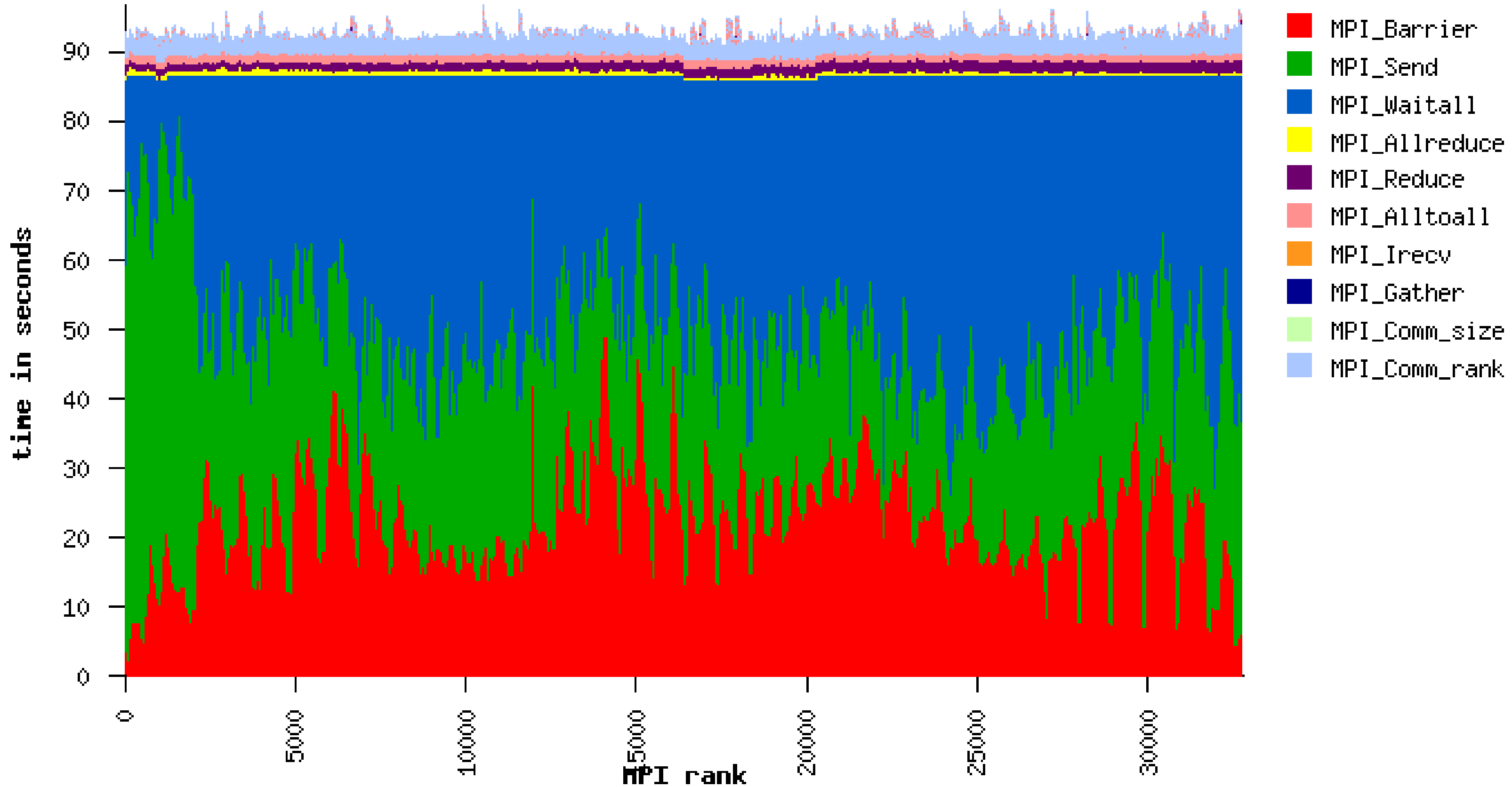


per MPI call & buffer size



IPM: Scalability

32K tasks AMR code



IPM: More than a pretty picture

Discontinuities in performance are often key to 1st order improvements

IPM: More Information

- Project websites
  - http://ipm-hpc.sourceforge.net/
  - http://sourceforge.net/projects/ipm-hpc/
- Email contacts and mailing lists
  - perfsuite@ncsa.uiuc.edu
  - perfsuite-users@lists.sourceforge.net
  - perfsuite-announce@lists.sourceforge.net
  - perfsuite-bugs@lists.sourceforge.net

What is Open|SpeedShop?

- Comprehensive Open Source Performance Analysis Environment
  - Targeting both End Users and Tool Developers
  - Performance analysis with single look & feel
  - Infrastructure to develop/prototype new tools
- Funding
  - DOE/NNSA as part of ASC PathForward
  - Initial phase co-funded by SGI
- Partners
  - DOE/NNSA Tri-Labs (LLNL, LANL, SNLs)
  - Krell Institute
  - Universities of Wisconsin and Maryland

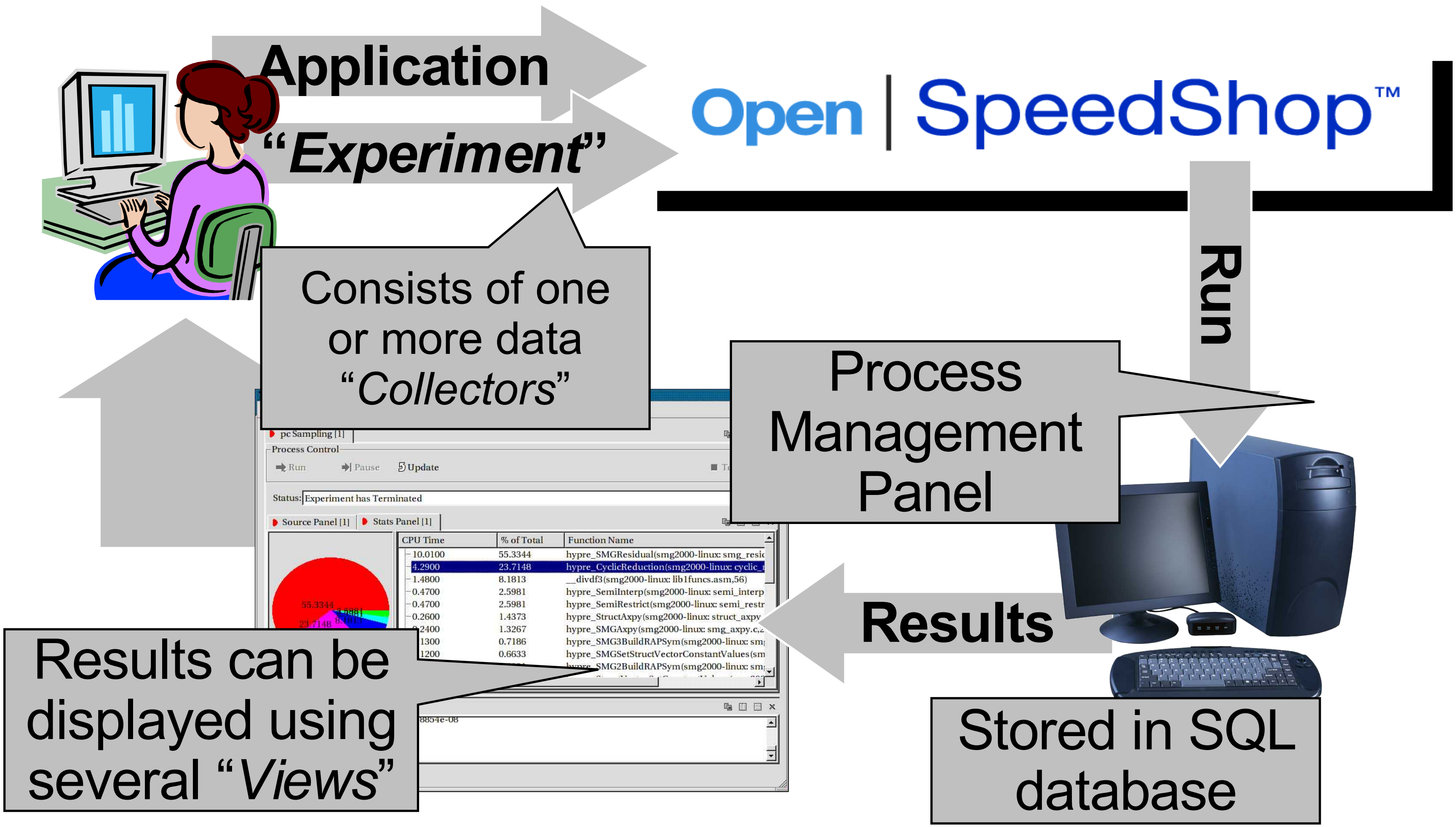
Highlights

- Open Source Performance Analysis Tool Framework
  - Most common performance analysis steps all in one tool
  - Extensible by using plugins for data collection and representation
- Instrumentation at Runtime
  - Use of unmodified application binaries
  - Attach to running applications
- Flexible and Easy to use
  - User access through GUI, Command Line, and Python Scripting

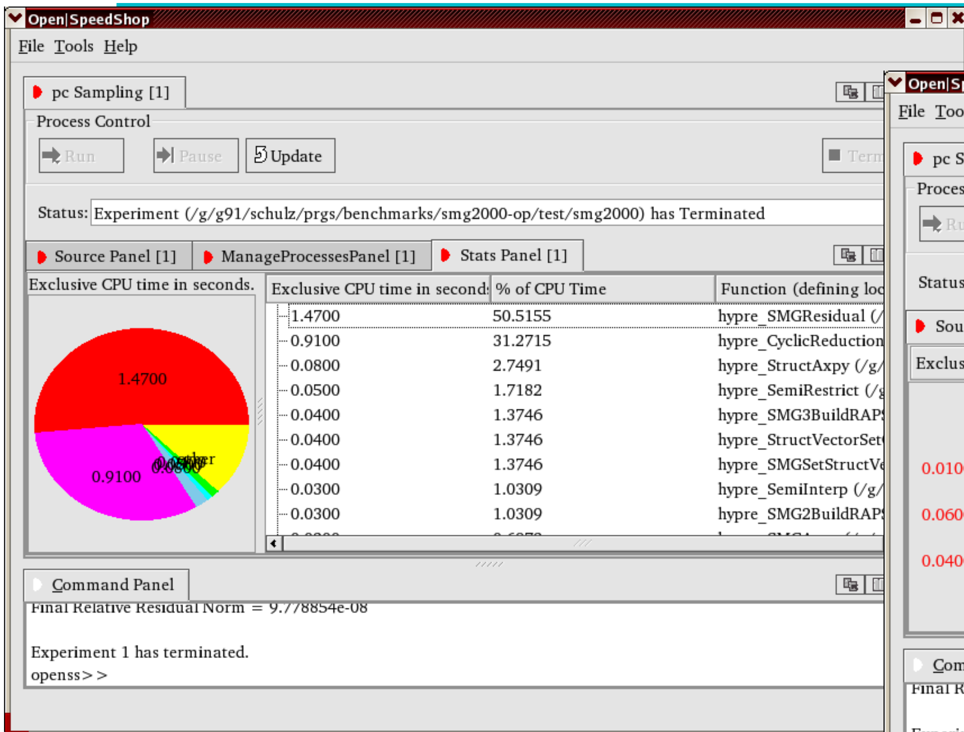
Highlights

- Large Range of Platforms
  - Linux Clusters with x86, IA-64, Opteron, and EM64T CPUs
  - Designed with portability in mind
- Version 1.5 released in Nov. 2007
  - Used at all three ASC labs with lab-size applications
  - Source and RPM versions at http://www.openspeedshop.org/

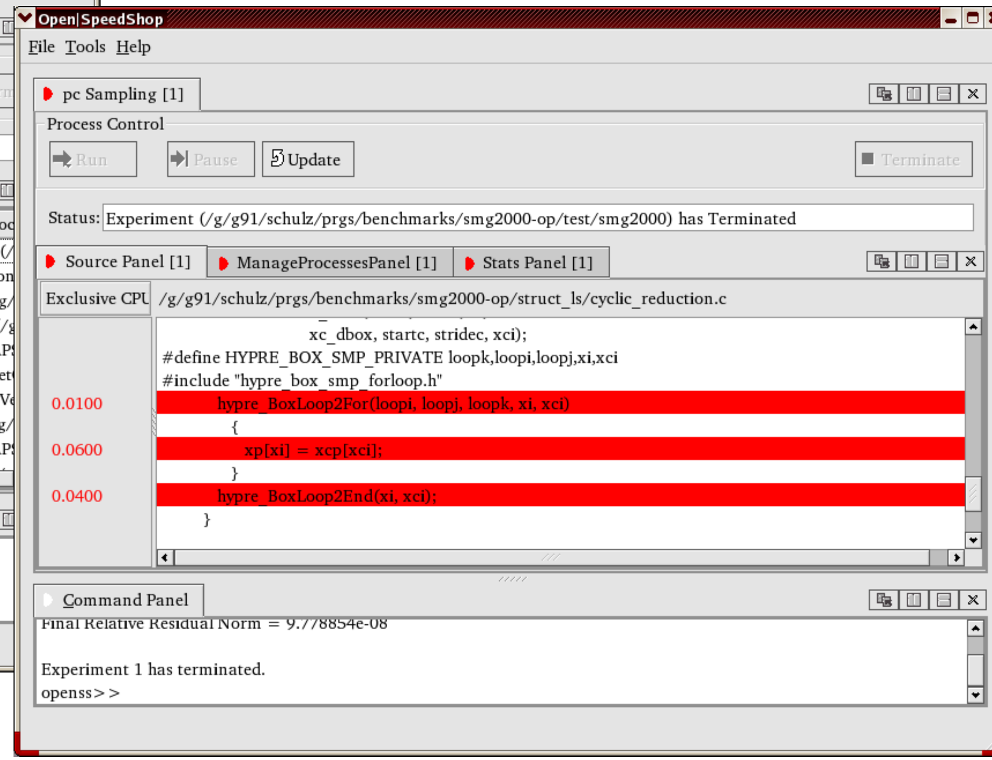
Workflow & High-level Design



Performance Experiments



Mapping to source code

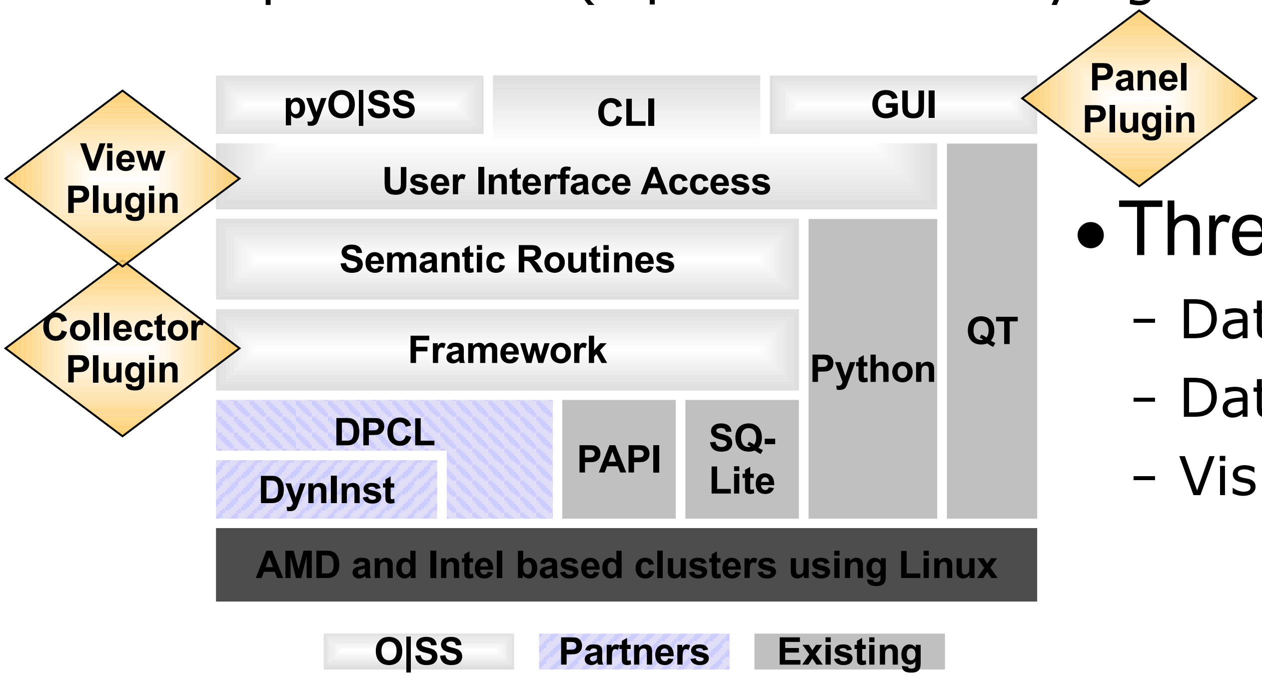


Per line/function display

- Existing Experiments
  - Profiling: PC sampling, User time, Hardware counter
  - Tracing: MPI calls, I/O calls, Floating Point Exceptions

Extensible Infrastructure

- Support for performance tool developers
  - Reusable tool infrastructure and user interfaces
  - Plugin architecture
  - Open source (O|SS and underlying libraries)



- Three plugin types
  - Data Collection
  - Data View Preparation
  - Visualization in Panels

Open SpeedShop: More Information

- Project websites
  - http://oss.sgi.com/openspeedshop/
  - http://sourceforge.net/projects/openss/
  - http://www.openspeedshop.org/
- Email contacts and mailing lists
  - oss-questions@openspeedshop.org

Scalasca project

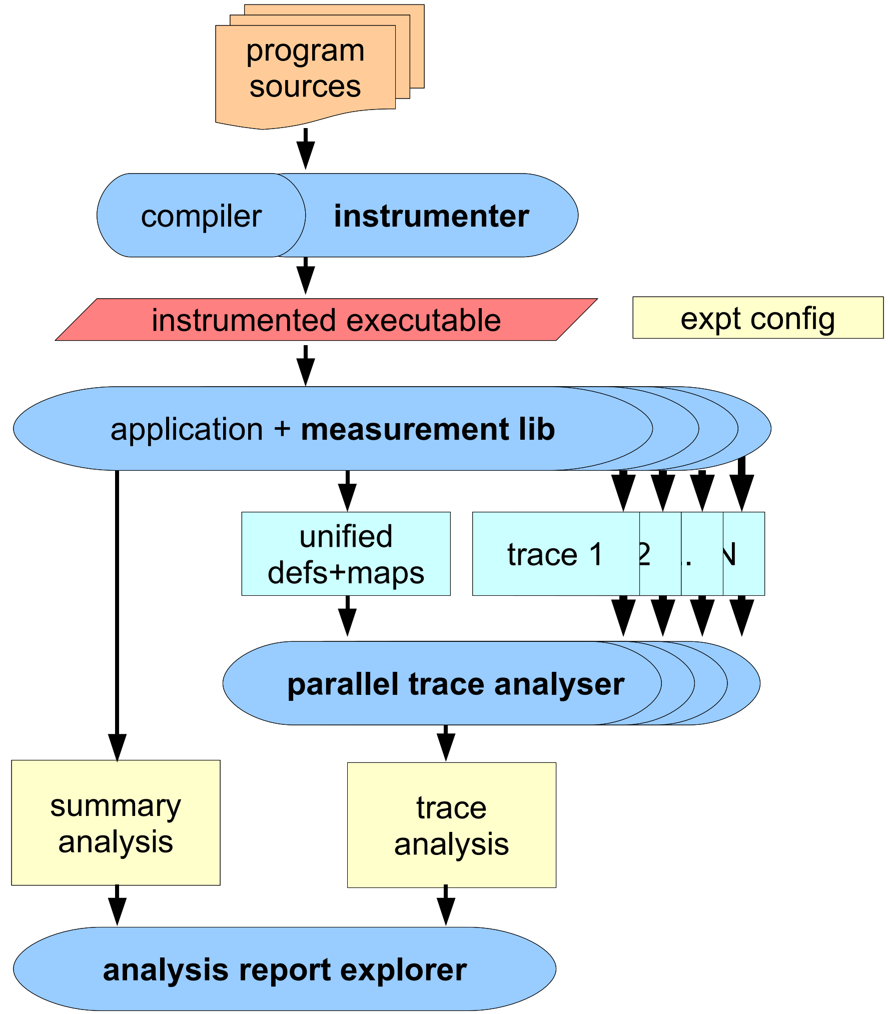
- Research group led by Prof. Felix Wolf
  - funded by Helmholtz Initiative & Networking Fund
  - in collaboration with University of Tennessee
- Follow-up to pioneering KOJAK project
  - automatic pattern-based trace analysis
- Developing toolset for <u>scalable</u> performance analysis of <u>large-scale</u> parallel applications
  - started January 2006 with initial focus on MPI-1
  - open-source release v0.9 in August 2007

Key functionality of Scalasca v0.9

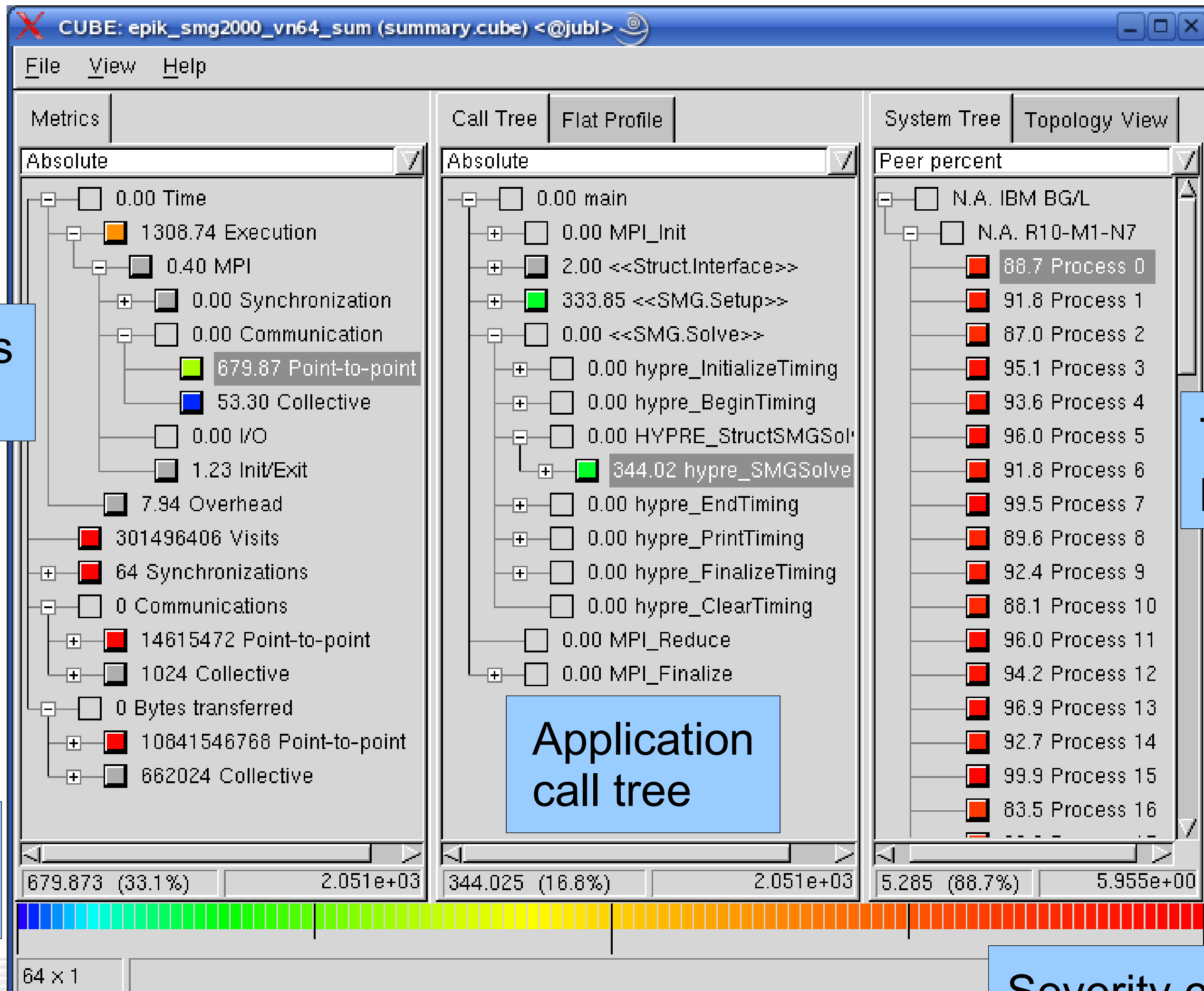
- Search for patterns representing inefficient behavior
- Common event & measurement manager
  - Identifier registration and unification at finale
  - Callpath tracking and measurement forwarding
- Scalable parallel runtime summarization
  - Aggregation of measurements per call path
  - Collation into integrated report at finalization
- Scalable parallel trace collection & analysis
  - Distributed trace recording per process
  - Replay-based distributed trace analysis

Scalasca components

- Automatic/manual code instrumenter
- Measurement library for runtime summary & traces
- Replay-based trace analyser
- Common analysis report explorer



Scalasca summary analysis presentation



Hierarchies of metrics

Application call tree

Tree of MPI processes

Selected values

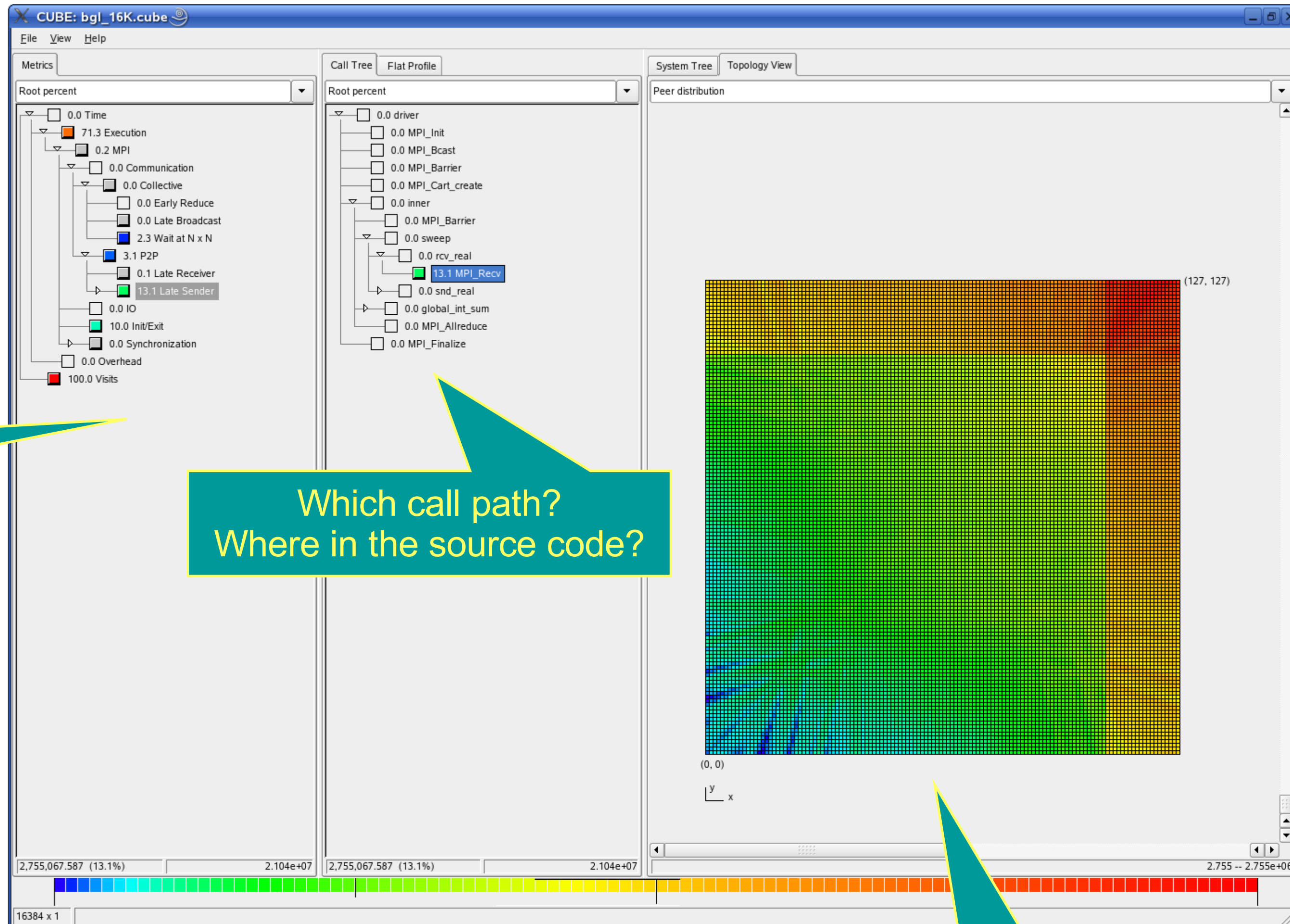
Severity colour scale

Scalasca trace metrics

- Trace analysis based on parallel replay
  - receivers determine severities from sent msg
- Aggregated for each call-path (& thread)
  - Visits & message statistics as for summary
  - Waiting & Imbalance times
    - Pt2Pt: Late Sender, Late Sender/Wrong Order, ...
    - Collective synch: Wait at Barrier/Barrier Completion
    - Collective comm: Wait at N x N/N x N Completion, Early Reduce/Scan, Late Broadcast
- Post-processing remaps into hierarchies

SWEEP3D

Late Sender
16K CPUs



Which type of problem?

Which call path?
Where in the source code?

Virtual topology

Scalasca: More Information

- Project websites
  - http://www.fz-juelich.de/jsc/kojak/
  - http://www.fz-juelich.de/jsc/scalasca/
  - http://icl.cs.utk.edu/kojak/
- Email contacts and mailing lists
  - kojak@cs.utk.edu
  - kojak@fz-juelich.de
  - scalasca@fz-juelich.de

Vampir

- Commercial offering from Dresden
- Used to visualize traces of performance data.
- 3 Components
  - VampirTrace, can be invoked from TAU or directly
  - VampirServer
  - VampirServer Browser

VampirTrace

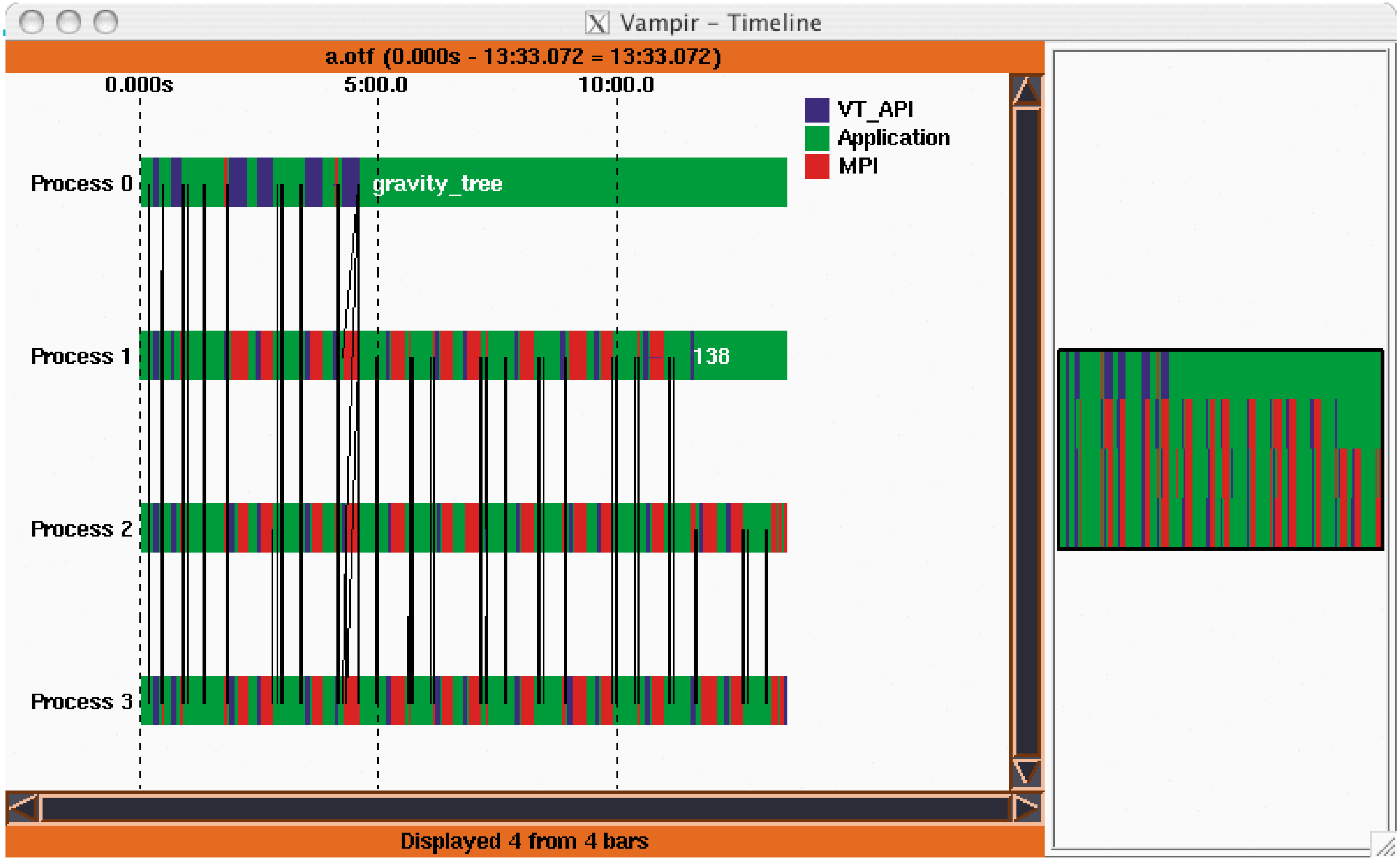
- Recorded events
  - Function entry/exit if compiler instrumentation is used.
  - MPI and OpenMP events
  - Hardware/software performance counters (e.g. PAPI)
  - OS events: Process creation, resource management
- Collected event properties
  - Time stamp
  - Location (process / thread / MPI)
  - MPI specifics like message size etc.
- Generates data in Open Trace Format (OTF)
  - Human readable
  - Fast searching and indexing
  - On-the-fly compression

1(

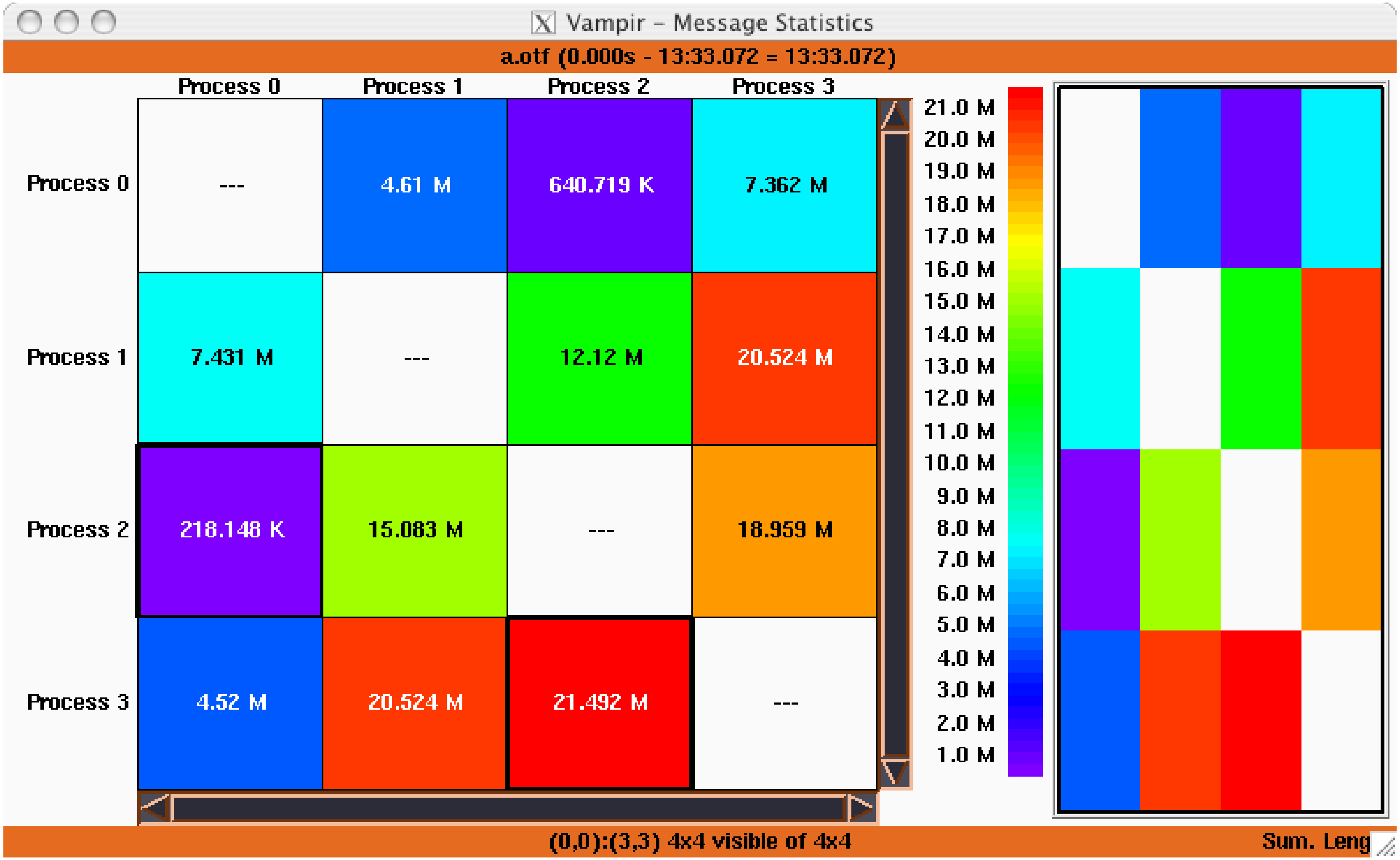
VampirServer

- VampirServer: Distributed high-end performance visualization
  - Client/server architecture
  - Parallel event processing
  - Runs on a (part of a) production environment
  - No need to transfer huge traces, uses parallel I/O
- VampirServer Browser: Lightweight client on local workstation
  - Outer appearance identical to Vampir
  - Highly scalable display engine
  - Statistics, profiles and summary charts
  - Message traffic and timelines
  - Receives visual content only
  - Already adapted to display resolution (but no images)
  - Moderate network bandwidth and latency requirements
  - Scales to trace data volumes > 40GB

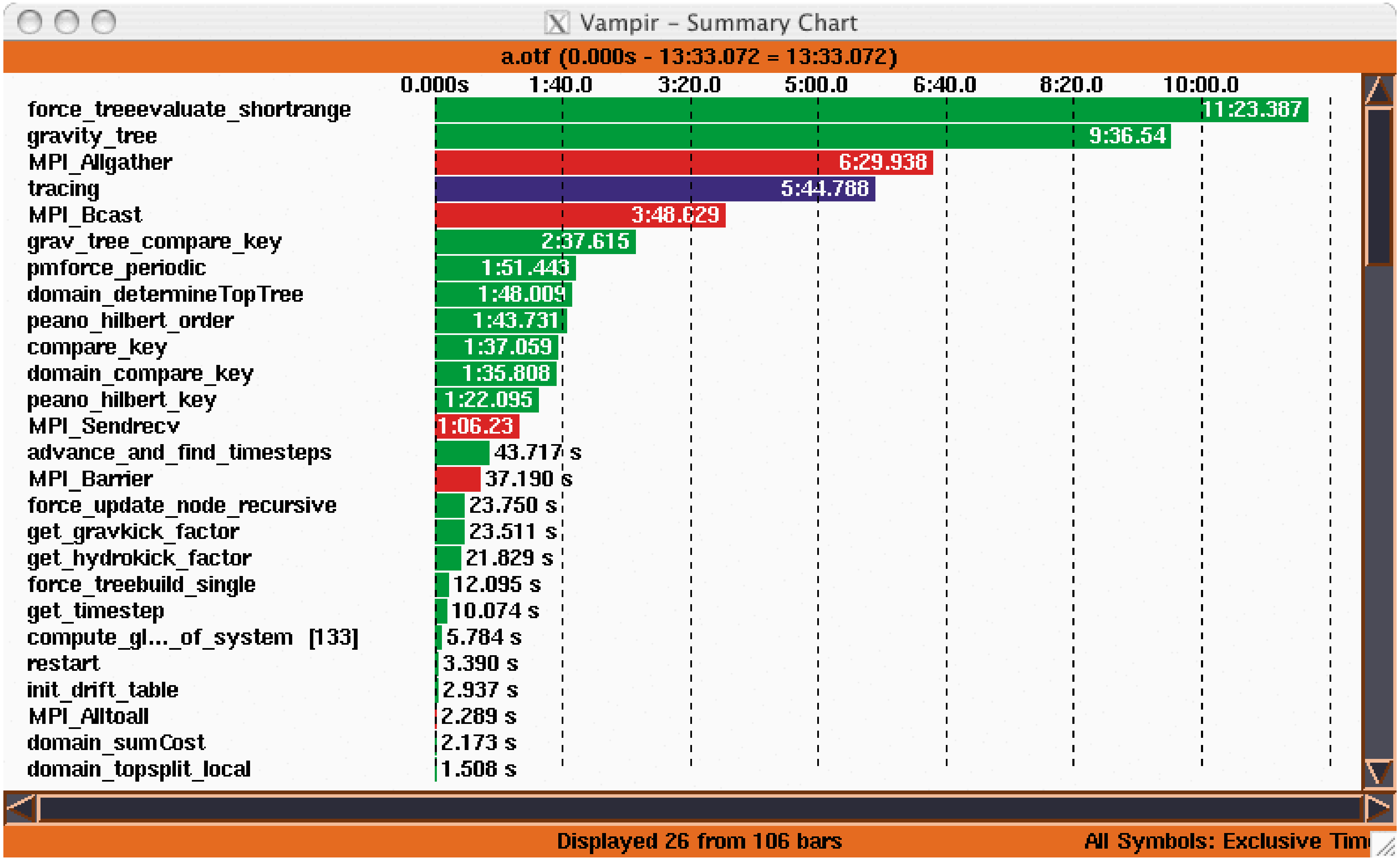
Vampir Timeline



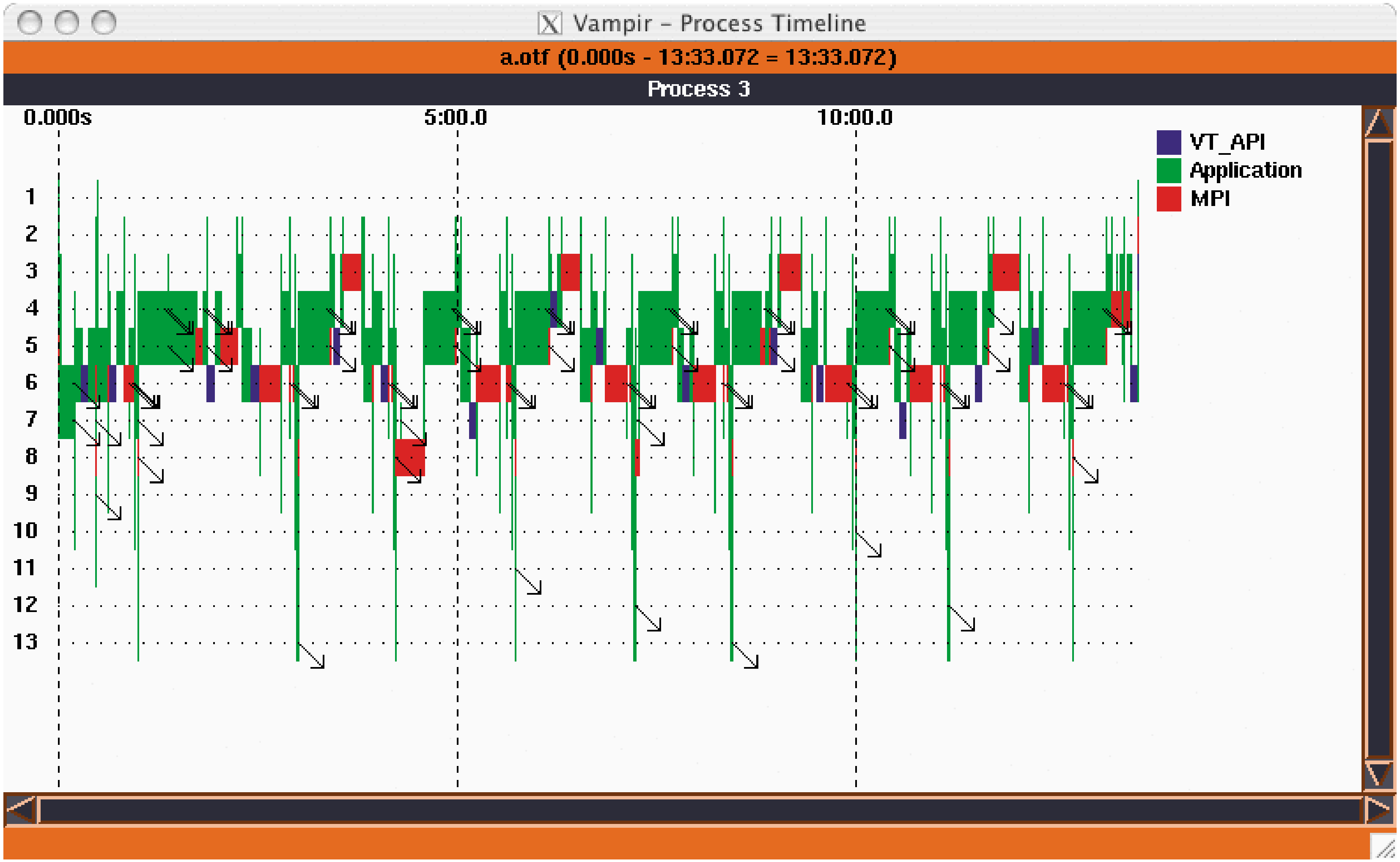
Vampir Message Statistics



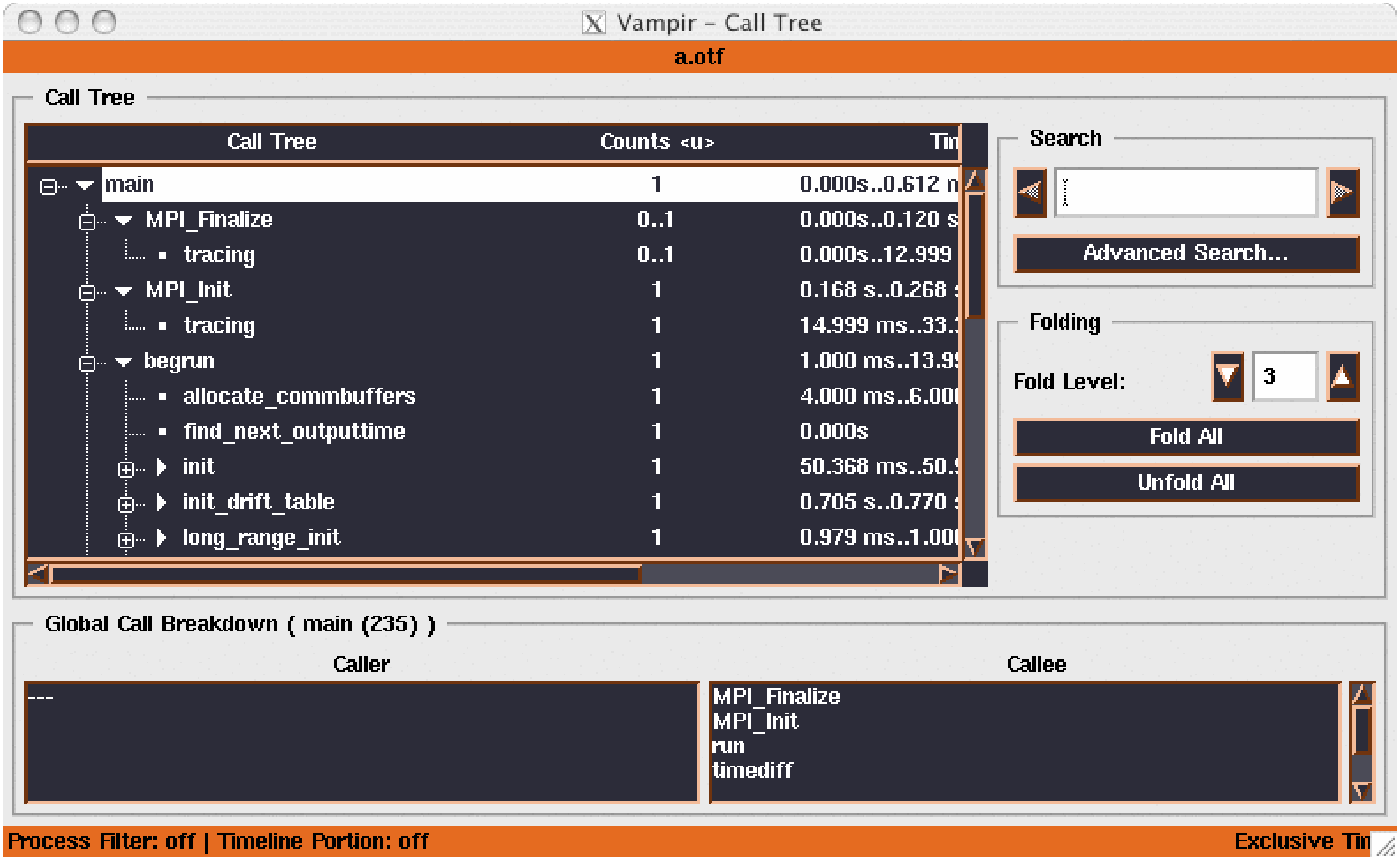
Vampir Summary Chart



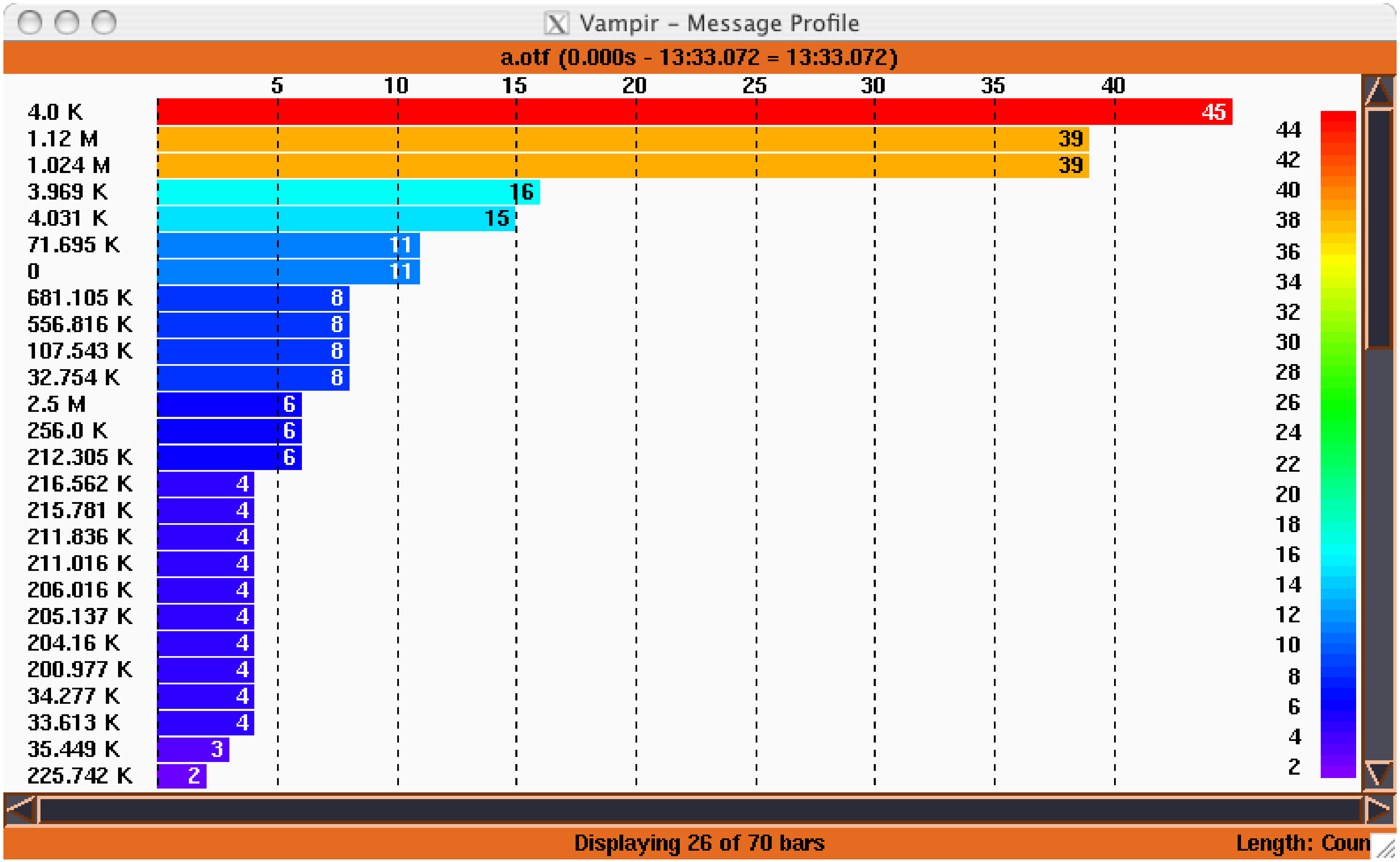
Vampir Process Timeline



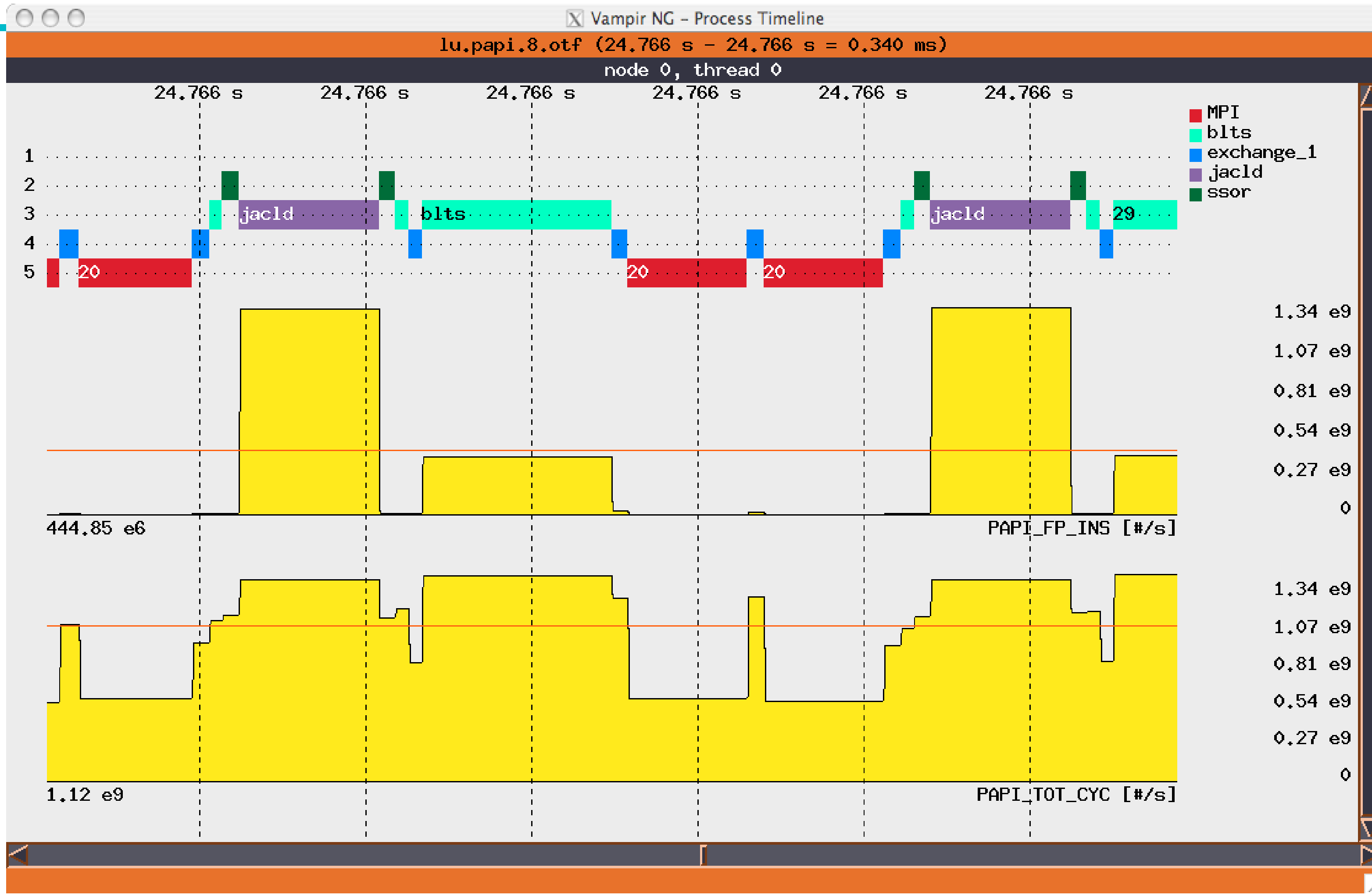
Vampir Call Tree



Vampir Message Profile



Visualizing TAU Traces with VampirNG



Vampir: More Information

- Project websites
  - http://www.vampir.eu

Acknowledgments

- Argonne National Laboratory.
- HiPerSoft, Rice University.
- Innovative Computing Laboratory, University of Tennessee, Knoxville.
- Lawrence Livermore National Laboratory.
- Lawrence Berkeley National Laboratory.
- National Center for Atmospheric Research.
- ParaTools, Inc.
- Research Centre Juelich, Germany.
- SiCortex Inc.
- Technische Universität Dresden, Germany.
- U. Oregon

Acknowledgments

- Stefane Eranian of HP Laboratories/Google
- Rick Kufrin, NCSA
- John Mellor-Crummey, Rice University
- Bernd Mohr, Felix Wolf, Brian Wylie, FZ Juelich
- Alan Morris, Allen Malony, Sameer Shende, Paratools, U. Oregon
- Martin Schulz, Lawrence Livermore National Laboratory
- David Skinner, NERSC and Lawrence Berkeley National Laboratory
- Nick Wright, San Diego Super Computing Center