Tutorial

Application Performance Analysis Tools for Linux Clusters

Rick Kufrin

NCSA National Center for Supercomputing Applications

Phil Mucci

Royal Institute of Technology Sweden

Felix Wolf

University of Tennessee

Linux Clusters: The HPC Revolution 2004, Austin, TX, May 17, 2004

Introduction to PerfSuite

Rick Kufrin
National Center for Supercomputing Applications

Linux Clusters: The HPC Revolution 2004
Austin, TX
May 17, 2004

Tutorial Outline

- Downloading & installation (5 mins)
- Overview of PerfSuite (5 mins)
- Basic usage (15 mins)
- PerfSuite APIs (5 mins)
- Advanced use / examples (10 mins)
  - Parallel applications
  - Auto-collection
- Questions / support information

Download

- Two primary websites:
  - http://perfsuite.sourceforge.net/
  - http://perfsuite.ncsa.uiuc.edu/
- Software identical at each location
  - Currently no anonymous CVS repository
- Web site at NCSA has an additional “helper” script that retrieves PerfSuite and auxiliary software with one command
- What other software is necessary?
  - At a minimum, you’ll need the GNU C compiler and the expat library
  - Highly recommended: Tcl/Tk, tDOM Tcl extension, and PAPI (either version 2 or 3)

Configuration and Build

- PerfSuite follows the GNU Autotools model for configuration, build, install, packaging and directory structure
- Procedure is familiar “gunzip”, “configure”, “make”, “make check”, “make install”
- “configure –h” provides synopsis of configuration options. Some of the most important include:
  - --with-papi=*<PAPI toplevel directory>*
  - --with-tdom=*<tDOM lib directory>*
  - --enable-mpi
- $F77 environment variable is used to detect Fortran calling convention to use (underscore, case, etc)
- $MPICPPFLAGS is used to find <mpi.h>

Things To Note During Configure

```
checking whether make sets $(MAKE)... yes
checking for gawk... gawk
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes

checking whether we are using the GNU Fortran 77 compiler... yes

configure: configuring Tcl/Tk support
checking for tclsh8.4... /usr/local/bin/tclsh8.4
checking for wish8.4... /usr/local/bin/wish8.4
checking for use of Tcl library... success

configure: configuring PAPI support
checking for PAPI_library_init in -lpapi... yes
checking PAPI version... 2

configure: configuring MPI support
checking for mpi.h... yes

configure: configuring tDOM support
checking load of tDOM from the Tcl shell (/usr/local/bin/tclsh8.4)...
   success
```

Example Configuration

```
$ ./configure --prefix=/opt/perfsuite \
  --with-papi=/opt/papi \
  --with-tdom=/usr/local/tcl/lib F77=ifc \
  MPICPPFLAGS="-I/usr/local/mpich/include" \
  PTHREAD_LIBS="-lpthread"
```

- PerfSuite will be installed (entirely) under /opt/perfsuite
- PAPI's installation, specified as either DESTDIR or PREFIX when you built PAPI, is in /opt/papi
- The tDOM TCL extension was built with PREFIX=/usr/local/tcl
- Intel's ifc compiler will be used to determine C/Fortran calling convention
- The MPI installation is in /usr/local/mpich
- PTHREAD_LIBS not usually necessary (but doesn't hurt)
- Your configure command line will be stored in "config.log"

A Quick Tour Through the Directories

- Once PerfSuite is successfully built and installed, you'll have the following subdirectories under $PREFIX:
  - `bin`
    - `psconfig, psenv, psinv, psprocess, psrun`
  - `include`
    - `perfsuite.h, fperfsuite.h, pshwpc.h`
  - `lib`
    - `libperfsuite, libpshwpc`, and other runtime/extensions
  - `man`
    - Man pages for each command in `bin`
  - `share`
    - `perfsuite`
- The configure option `--libdir=` is helpful when installing with multiple Fortran compilers

The `perfsuite/share` Subdirectory

- This directory contains machine-independent files with the following structure:
  - `doc` – the license, a README, BUGS, and available documentation for commands and libraries (in progress).
  - `dtds` – Document Type Definitions (similar to a database schema) for PerfSuite XML documents. These give the specifics for what can/must be contained in a valid PerfSuite XML doc.
  - `examples` – several example programs and makefiles that can be helpful when getting started.
  - `tclbin & tcllib` – Tcl scripts and packages that are used internally.
  - `xml/pshwpc` – contains all the standard hardware event configuration files and associated XML docs and stylesheets. Can be used as-is or copied to your own directories to be modified as you like. We'll see more on these shortly.

A Little Bit of History

- PerfSuite grew out of experiences “in the field”, working with computational scientists on a daily basis and learning which types of tools “fit”, which don’t, and what gaps needed to be filled
  - First such tool was a Tk-based tool called Memory Placement Monitor that combined the information available from several different utilities to present a graphical layout of the placement of pages on large distributed shared memory machines (SGI Origin)
  - Next effort was a graphical display of very high-dimensional data: the output of the IRIX “perfex” tool when applied to highly-parallel programs (scaling runs on varying processor counts, each processor contributing 32 different hardware event counts)
  - Natural extension was to provide a perfex-like capability under Linux

Performance Analysis in Practice

- Observation: many application developers don't use performance tools at all (or rarely)
- Why?
  - Learning curve can be steep
  - Results can be difficult to understand
  - Investment (time) can be substantial
  - Maturity/availability of various tools
  - Not everyone is a computer scientist
- Although it's the norm for vendor-supplied tools to be available for proprietary HPC operating systems, Linux is just beginning to catch up with contributions from the open source community (independent or vendor-supported).

PerfSuite

- Design Goals
  - Remove the barriers to the initial steps of performance analysis (don't make it hard)
  - Separate data collection from presentation
  - Machine-independent representation
  - Holistic viewpoint: compiler, hardware counters, message-passing, etc.
    - (we'll only discuss counter support here)
  - Focus on the "Big Picture" (remember that 80/20 rule?)

- A primary goal is to provide an "entry point" that can help you to decide how to proceed

PerfSuite and XML

- In PerfSuite, nearly all data (input, output, configuration, etc) is represented as XML (eXtensible Markup Language) documents
- This provides the ability to manipulate & transform the data in many ways using standard software / skills
- Machine-independent (no binary files)
  - ...opens the data up to the user
- There are numerous high-quality XML-aware libraries available from either compiled or interpreted languages that can make it easy to transform the data for your needs
- Web browsers (e.g. Mozilla, IE) have built-in XML capabilities

PerfSuite Counter-Related Software

- Four performance counter-related utilities:
  - psconfig - configure / select performance events
  - psinv - query events and machine information
  - psrun - generate raw counter or statistical profiling data from an unmodified binary
  - psprocess - pre- and post-process data
- Four libraries (shared and static)
  - libperfsuite – the "core" library that can be used standalone and will be built regardless of the availability of other software
  - libpshwpc – HardWare Performance Counter library, also built regardless of other software. Without counter support, will only perform time-based profiling through `profil()`. A version suitable for threaded programs is available (`_r` suffix).
  - libpshwpc_mpi – a convenience library based on the MPI standard PMPI interface.

psinv

- Lists information about the characteristics of the computer
- This same information is also stored in psrun XML output and is useful for later generating derived metrics (or for remembering where you ran your program!)
- x86/x86-64 version also shows processor features and descriptions
- Lists available hardware performance events

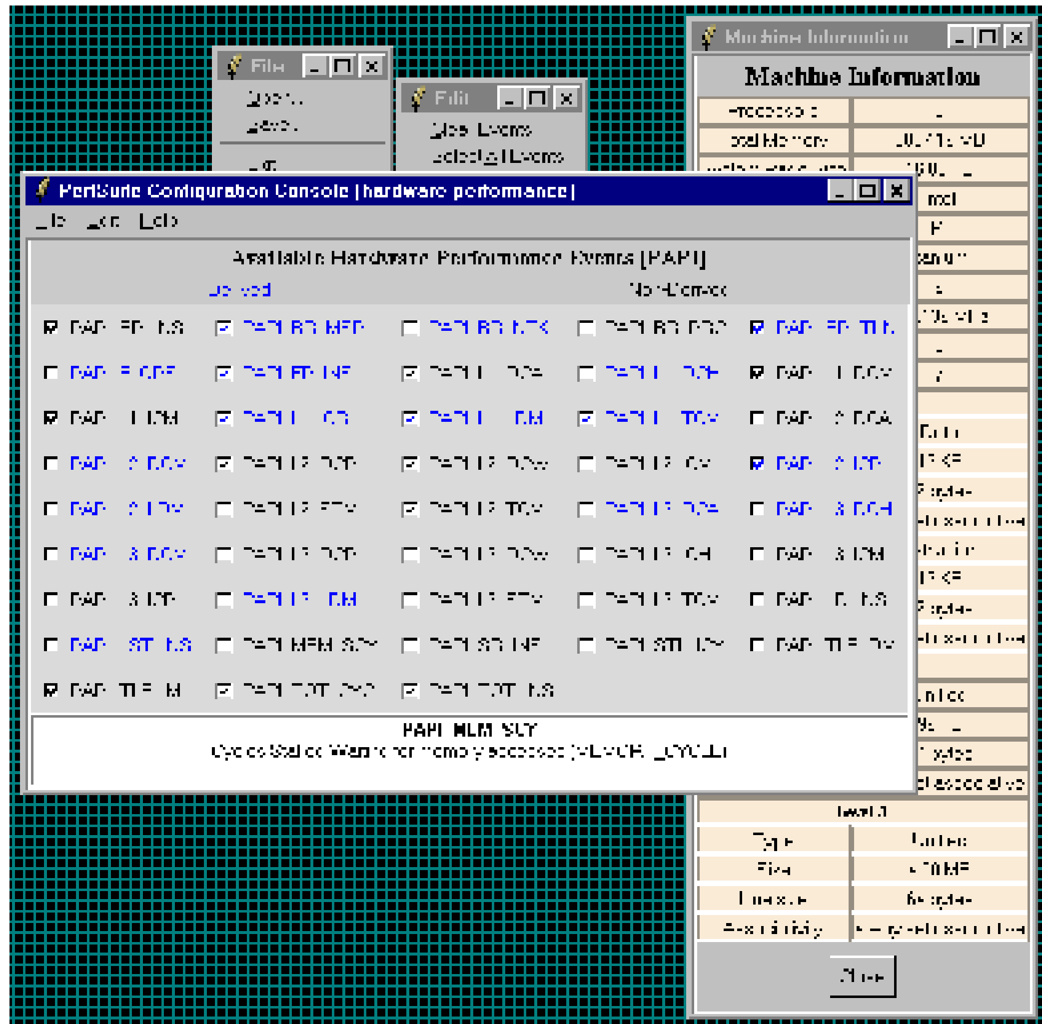
```
titan:~3% psinv -v
System Information -
Processors:                 2
Total Memory:               2007.16 MB
System Page Size:           16.00 KB

Processor Information -
Vendor:                     Intel
Processor family:           IPF
Model (Type):               Itanium
Revision:                   6
Clock Speed:                800.136 MHz

Cache and TLB Information -
Cache levels:               3
Caches/TLBs:                7

Cache Details -
Level 1:
        Type:               Data
        Size:               16 KB
        Line size:          32 bytes
        Associativity:      4-way set associative

        Type:               Instruction
        Size:               16 KB
        Line size:          32 bytes
        Associativity:      4-way set associative
```

psinv (cont'd)

```
PAPI Standard Event Information -
Standard events:    43
Non-derived events:26
Derived events:     17

PAPI Standard Event Details -
Non-derived:
        PAPI_BR_INS:    Branch instructions
        PAPI_BR_PRC:    Conditional branch instructions correctly
predicted
        PAPI_L1_DCA:    Level 1 data cache accesses
        PAPI_L1_DCM:    Level 1 data cache misses
        PAPI_L1_ICM:    Level 1 instruction cache misses
        PAPI_L2_DCA:    Level 2 data cache accesses
        PAPI_L2_DCR:    Level 2 data cache reads
        PAPI_L2_DCW:    Level 2 data cache writes
        PAPI_L2_ICM:    Level 2 instruction cache misses
        PAPI_L2_STM:    Level 2 store misses
        PAPI_L2_TCM:    Level 2 cache misses
Derived:
        PAPI_BR_MSP:    Conditional branch instructions mispredicted
        PAPI_BR_NTK:    Conditional branch instructions not taken
        PAPI_BR_TKN:    Conditional branch instructions taken
        PAPI_FLOPS:     Floating point instructions per second
        PAPI_FP_INS:    Floating point instructions
        PAPI_L1_DCH:    Level 1 data cache hits
```

psconfig

- Graphical user interface makes it easy to select events
- Can read in or write out valid XML documents to be used by psrun
- Provides text description of events with mouse click
- Searching capabilities



Example XML Event Document

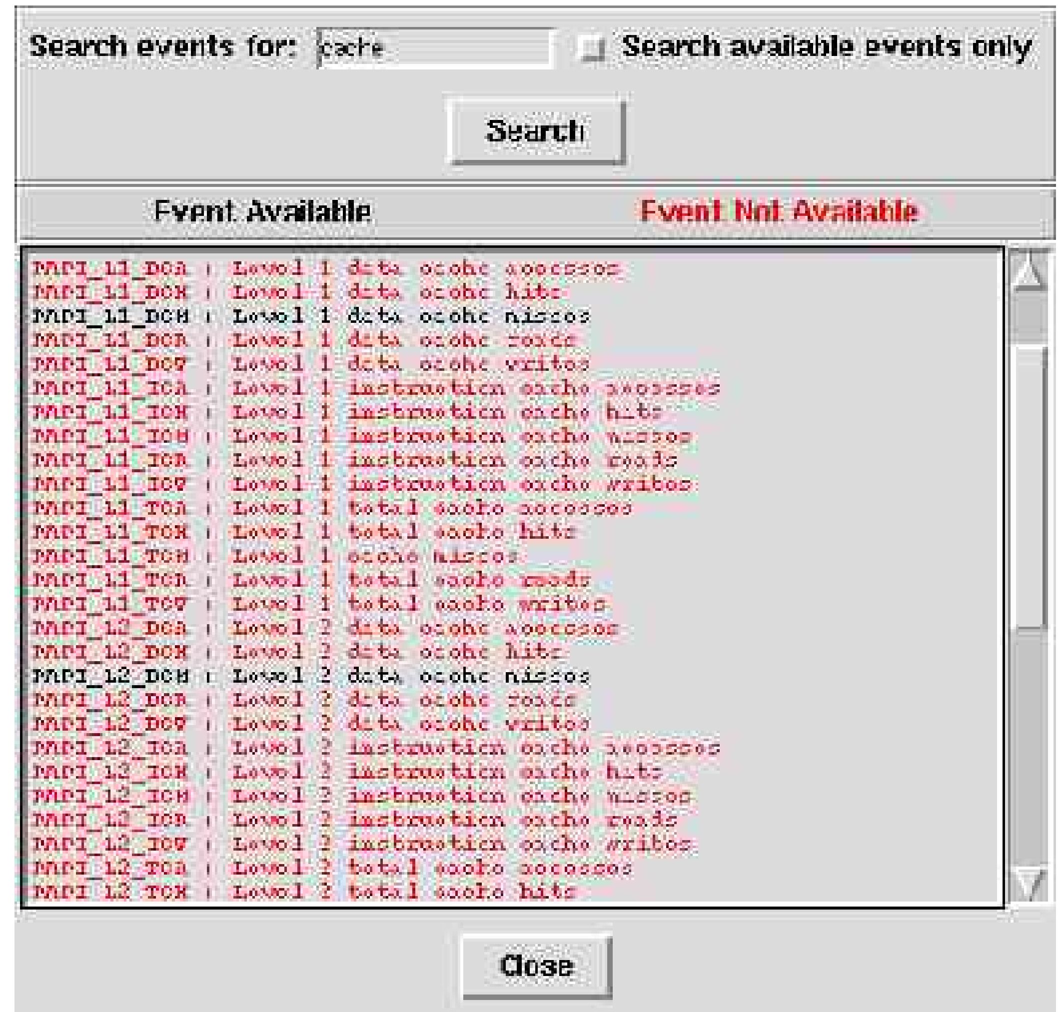
```
<?xml version="1.0" encoding="UTF-8" ?>
<ps_hwpc_eventlist class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_BR_MSP" />
  <ps_hwpc_event type="preset" name="PAPI_BR_PRC" />
  <ps_hwpc_event type="preset" name="PAPI_BR_TKN" />
  <ps_hwpc_event type="preset" name="PAPI_FP_INS" />
  <ps_hwpc_event type="preset" name="PAPI_TOT_CYC" />
  <ps_hwpc_event type="preset" name="PAPI_TOT_INS" />
  <ps_hwpc_event type="preset" name="PAPI_L1_DCA" />
  <ps_hwpc_event type="preset" name="PAPI_L1_DCH" />
  <ps_hwpc_event type="preset" name="PAPI_L1_DCM" />
  <ps_hwpc_event type="preset" name="PAPI_L1_ICR" />
  <ps_hwpc_event type="preset" name="PAPI_L1_TCM" />
  <ps_hwpc_event type="preset" name="PAPI_L2_DCA" />
  <ps_hwpc_event type="preset" name="PAPI_L2_DCM" />
  <ps_hwpc_event type="preset" name="PAPI_L2_DCR" />
  <ps_hwpc_event type="preset" name="PAPI_L2_DCW" />
</ps_hwpc_eventlist>

.
```

- You can edit this file like any text file, load it into psconfig, modify it, save it, etc.
- Select for use through env variable

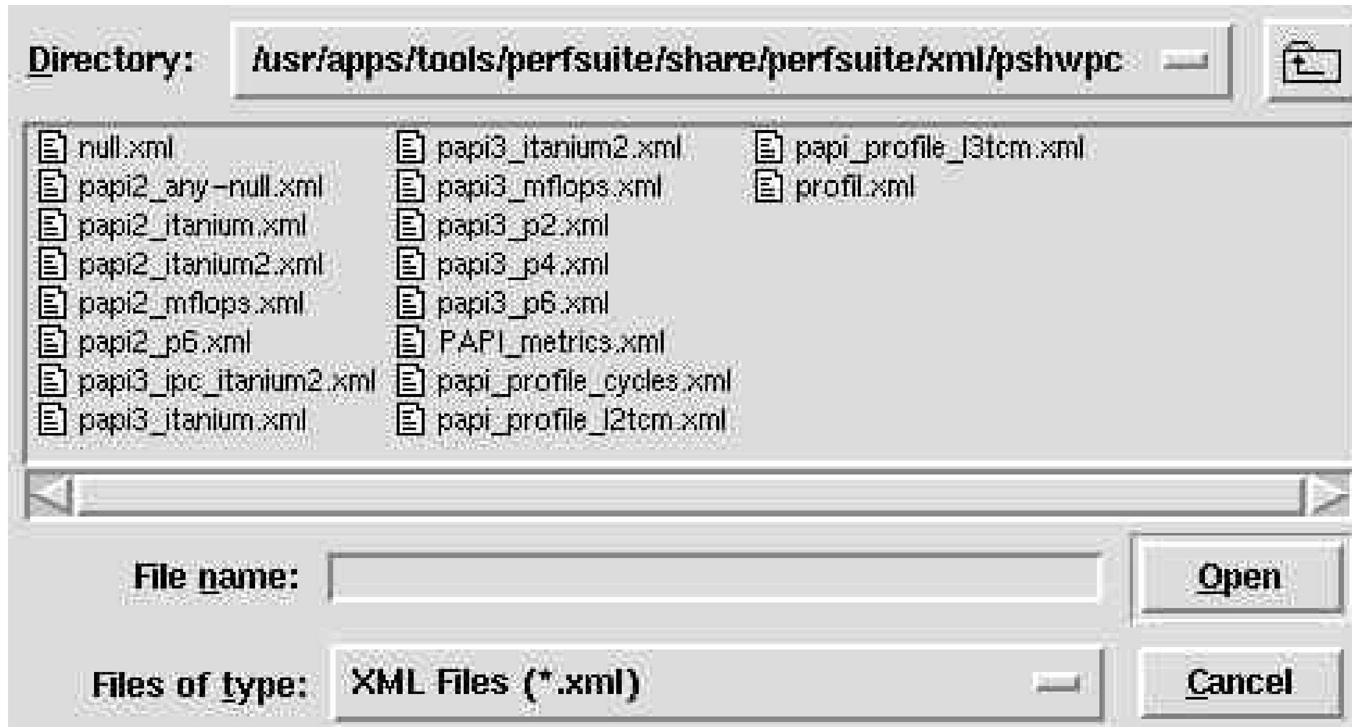
  PS_HWPC_CONFIG

Searching Events with psconfig

- Selecting “Edit”, “Search Events…” brings up a window like this that allows you to search events for keywords
- Can restrict the search to only events available on your computer
- The search is based on the event’s description, not it’s standard event name (PAPI_TOT_CYC)



Browsing Default Event Configurations



- Selecting “File”, “Default Hardware Event Configurations…” brings up the directory with pre-selected configuration documents
- Opening one of them will show you which events will be used
- You can base custom configuration files using these as a start

Configuring for Profiling

- Setting up for profiling is similar to counting - all you have to do is modify the XML configuration document:
- The XML document “root element” is now <ps_hwpc_profile>, not <ps_hwpc_eventlist>
- You can supply an optional “threshold”, or sampling rate
- Only one event is allowed in the document
- psconfig does not yet support profiling, need to edit by hand

```
<?xml version="1.0" encoding="UTF-8" ?>
<ps_hwpc_profile class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_BR_MSP" threshold="100000" />
</ps_hwpc_profile>
```

psrun

- Hardware performance counting and profiling with unmodified dynamically-linked executables
- Available for x86, x86-64, and ia64
- POSIX threads support
- Automatic multiplexing
- Can be used with MPI
- Optionally collects resource usage
- Supports all PAPI standard events
- Input/Output = XML documents (can request plain text)

A Quick “Cookbook” for psrun

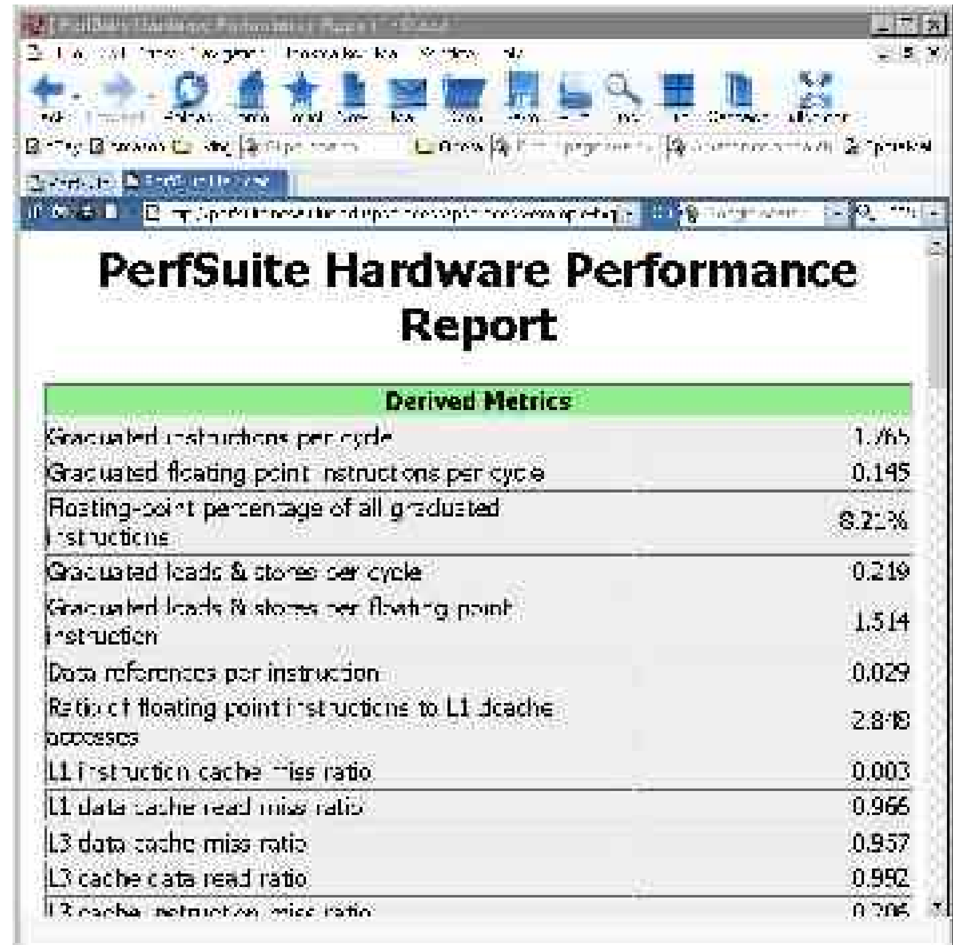
```
# First, be sure to set all paths properly (can do in .cshrc/.profile)

% set PSDIR=/opt/perfsuite
% source $PSDIR/bin/psenv.csh

# Use psrun on your program to generate the data,
# then use psprocess to produce an HTML file

% psrun myprog
% psprocess --html psrun.12345.xml > myprog.html

# Take a look at the results

% mozilla myprog.html

# Second run, but this time profiling instead of counting

% psrun -c $PSDIR/share/perfsuite/xml/pshwpc/profil.xml myprog
% psprocess -e myprog psrun.67890.xml
```

PerfSuite Environment Variables

- PS_HWPC: “off” or “on”, controls whether measurement takes place at all (for API)
- PS_HWPC_CONFIG: set to the name of the XML event file created with psconfig or “by hand”. A default is used if not set
- PS_HWPC_FILE: controls the prefix of the XML output document (default “psrun”)
- PS_HWPC_ANNOTATION - adds an arbitrary “note” to the XML output
- PS_HWPC_DOMAIN: controls whether counting at user or system level (or both)
- PS_HWPC_THRESHOLD: sets threshold for profiling
- PS_HWPC_FORMAT: “text” or “xml”, controls whether output is in an XML document or plain text (similar to a psprocess report)
- PSRUN_DOFORK: if set (to anything), monitors child processes also

psprocess (HTML mode)

- This style of output is customizable by you.
- By default, the information it contains and its visual appearance are based on PerfSuite-provided defaults, but these can be easily replaced to suit your needs.
- This output is generated by psprocess using XML Transformations. The stylesheet is in the share/perfsuite/xml/pshwpc subdirectory, with a “xsl” file extension



psprocess (text mode)

```
PerfSuite Hardware Performance Summary Report
Version        : 1.0
Created        : Mon Dec 30 11:31:53 AM Central Standard Time 2002
Generator      : psprocess 0.5
XML Source     : /u/ncsa/anyuser/performance/psrun-ia64.xml

Execution Information
============================
Date           : Sun Dec 15 21:01:20 2002
Host           : user01

Processor and System Information
============================
Node CPUs      : 2
Vendor         : Intel
Family         : IPF
Model          : Itanium
CPU Revision   : 6
Clock (MHz)    : 800.136
Memory (MB)    : 2007.16
Pagesize (KB)  : 16
```

psprocess (text mode, cont'd)

```
Cache Information
==========================
Cache levels : 3
--------------------------------
Level 1
Type          : data
Size (KB)     : 16
Linesize (B) : 32
Assoc         : 4
Type          : instruction
Size (KB)     : 16
Linesize (B) : 32
Assoc         : 4
--------------------------------
Level 2
Type          : unified
Size (KB)     : 96
Linesize (B) : 64
Assoc         : 6
```

The reports (text or HTML) generated by psprocess have several sections, covering:

- Report creation details
- Run details
- Machine information
- Raw counter listings
- Counter explanations and index
- Derived metrics
- Run annotation defined by you

psprocess (text mode, cont'd)

```
Index Description                                        Counter Value
======================================================================
1 Conditional branch instructions mispredicted.....         4831072449
4 Floating point instructions......................        86124489172
5 Total cycles.....................................       594547754568
6 Instructions completed...........................      1049339828741

Statistics
======================================================================
Graduated instructions per cycle...................              1.765
Graduated floating point instructions per cycle....              0.145
Level 3 cache miss ratio (data)....................              0.957
Bandwidth used to level 3 cache (MB/s).............            385.087
% cycles with no instruction issue.................             10.410
% cycles stalled on memory access..................             43.139
MFLOPS (cycles)....................................            115.905
MFLOPS (wallclock).................................            114.441
```

PerfSuite Library Access (API)

- All of the functionality is also available from within your program (C/C++/Fortran) through a small API
- Same XML documents are read, same XML documents are written, small additional functionality
- Why would you want to use this?
  - Primarily to gain finer control over where measurements are taken in your program. For example, you might defer measurement until program initialization has completed
- For complex uses, you are probably better off using an “industrial-strength” performance library
- The intent of the API is to “abstract out” the process of performance measurement to a very high level

libpshwpc Library Routines

C / C++

```
ps_hwpc_init (void)
ps_hwpc_start (void)
ps_hwpc_suspend (void)
ps_hwpc_stop (char *prefix)
ps_hwpc_shutdown (void)
```

Fortran

```
call psf_hwpc_init (ierr)
call psf_hwpc_start (ierr)
call psf_hwpc_suspend (ierr)
call psf_hwpc_stop (prefix,
   ierr)
call psf_hwpc_shutdown (ierr)
```

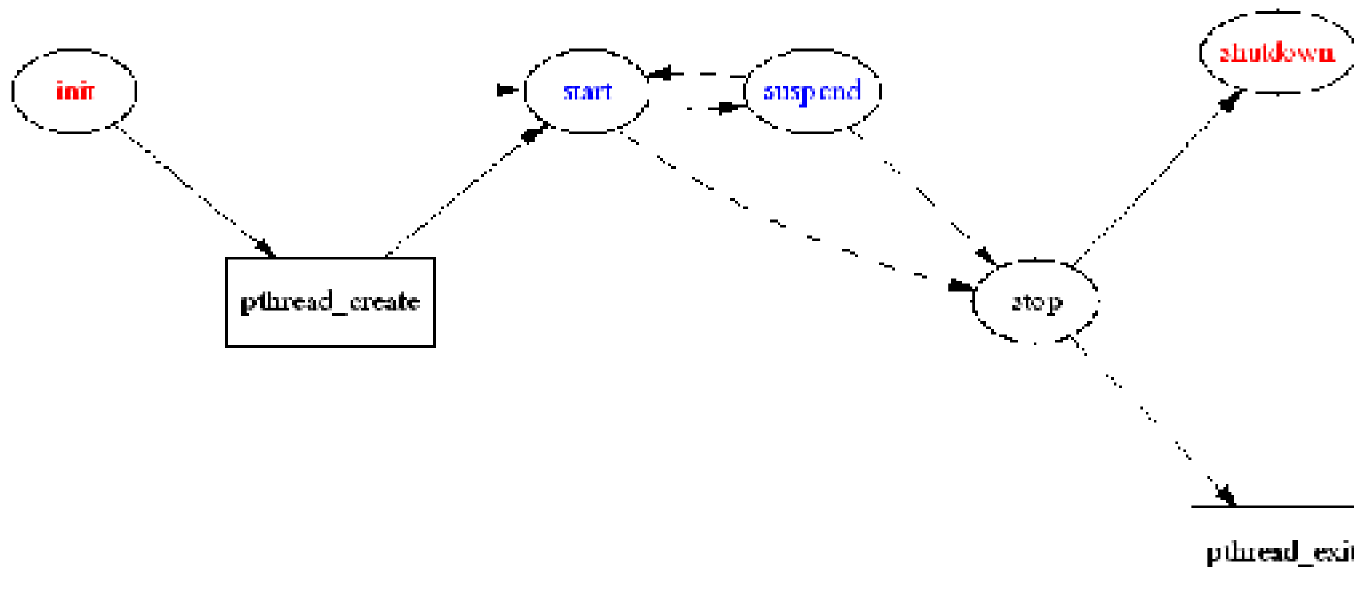
The `libpshwpc` API contains five routines that you can call from your C/C++ or Fortran program.

Call “init” once, call “start” and “suspend” as many times as you like. Call “stop” (supplying a file name prefix of your choice) to get the performance data XML document.

Optionally, call “shutdown”.

libpshwpc Usage

- This depicts the typical flow of calls to routines in the library
- Routines in red may only be called once, routines in blue can be called multiple times.
- Newly-created threads can start, suspend, stop, etc. Dashed lines show the typical flow of pthreads.
- fork/exec work if managed properly (call shutdown in the child process)



Example Fortran API Use

```
include 'fperfsuite.h'

call PSF_hwpc_init(ierr)

call PSF_hwpc_start(ierr)

do j = 1, n

   do i = 1, m

      do k = 1, l

         c(i,j) = c(i,j) + a(i,k)*b(k,j)

      end do

   end do

end do

call PSF_hwpc_stop('perf', ierr)

call PSF_hwpc_shutdown(ierr)
```

```
% ifc -c matmult.f -I/opt/perfsuite/include

% ifc matmult.o -L /usr/apps/tools/perfsuite/lib/intel
    -L/usr/apps/tools/papi/lib -lpshwpc -lperfsuite -lpapi
```

Using Processor “Native” Events

- It’s easy to work with native events in addition to PAPI standard events by modifying the configuration file slightly.
- Instead of using the XML attributes `type=“preset” name=“PAPI_EVENTNAME”`, use the attribute `type=“native”` and enclose the event name as the *content* of the element
- Must have PAPI 3 support
- Can be used with profiling configurations

```
<ps_hwpc_event type=“native”>NOPS_RETIRED<ps_hwpc_event>
<ps_hwpc_event type=“native”>BACK_END_BUBBLE_ALL</ps_hwpc_event>
```

Advanced Use (psrun)

- psrun supports a few options that can be useful in working with shared or distributed memory programs:

- `-p / --pthreads`
  - uses a POSIX thread-aware variant of the library that captures thread creation and measures performance of each, depositing the results in an XML document with the thread ID embedded:
- `-f / --fork`
  - monitors child processes that are created. Not enabled by default.
- `-a / --annotate`
  - inserts an XML “element” with a user-supplied annotation (text)

Advanced Use (psprocess)

- psprocess is meant to be a "generic" processor for different XML document types generated by PerfSuite. For hardware counting, the most common type is `<hwpcreport>`
- Individual documents can be combined into a "multi-document" with the option `-c / --combine`. With hardware counter data, psprocess summarizes the information contained in them with descriptive statistics (mean, max, min, sum, stddev)
- `-s LIST` is a very useful option to be used with profiling runs. `LIST` is a comma-separated list of `modules, files, functions, lines` used to limit the amount of output
- `-t THRESHOLD` is also helpful in limiting the output of profiling runs. `THRESHOLD` is a number that specifies the minimum % of samples required for a given entry to be displayed. Example: "`-t 2`" means "don't show me anything that didn't account for at least 2% of the samples collected"

Application Example: CX3D

- Fortran 90 / MPI code (Forschungszentrum Jülich) that simulates Czochralski crystal growth.
- Spatial decomposition across processors can be specified at runtime.
- We'll look at the steps involved in using PerfSuite on 8 processors to obtain profiling and counting information.
- The application measures elapsed time internally with system_clock (). For the 8-proc run, the measured wall clock time for a 4x2 decomposition is 40.88 secs.
- We can also measure parallel runs using gprof by using the environment variable GMON_OUT_PREFIX to override the default "gmon.out" filename.

Profile Procedure

- We have two executables: one compiled for gprof-style profiling and the other compiled as normal with symbols retained (-g).
- Run with mpirun as usual
  - gprof runs produce 8 ${GMON_OUT_PREFIX}.PID files that can be looked at individually or first combined with “-s” into a “gmon.sum” file that can be post-processed as usual
  - psrun runs produce 8 XML documents that can be post-processed with psprocess
- Note: gprof also retains the call graph information (psrun does not)

Profiling Results (gprof summary)

```
 %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 76.79    246.25   246.25     8000    30.78    30.93  velo_
  9.01    275.15    28.90     8000     3.61     3.64  temp_
  3.74    287.14    11.99     8000     1.50     1.50  curr_
  2.04    293.68     6.54
   gmpi_net_lookup
  1.81    299.49     5.81                             gm_ntoh_u8
  1.31    303.69     4.21
   MPID_RecvComplete
  0.75    306.12     2.42                             _gm_ntoh_u8
  0.71    308.38     2.27     8008     0.28     0.32  bound_
```

% time attributed to the highest routine (velo) ranges from 79.21 to 74.42.

```
$ gprof -s cx.gprof ${GMON_OUT_PREFIX}.*
$ gprof -s cx.gprof gmon.sum
```

Profiling Results (psprocess individual)

```
Profile Information
==========================================================================
Class                          : PAPI
Event                          : PAPI_TOT_CYC (Total cycles)
Period                         : 30600000
Samples                        : 4012
Domain                         : user
Run Time                       : 40.65 (seconds)
Min Self %                     : (all)

Module Summary
--------------------------------------------------------------------------
Samples   Self %  Total %  Module

   3942   98.26%   98.26%  /u/ncsa/rkufrin/apps/cx3d/cx
     69    1.72%   99.98%  /opt/gm/lib/libgm.so.0.0.0
      1    0.02%  100.00%  /lib/tls/libpthread-0.34.so
```

Profiling Results (psprocess, cont'd)

```
File Summary
--------------------------------------------------------------------------------
Samples   Self %  Total %  File

   3182   79.31%   79.31%  /u/ncsa/rkufrin/apps/cx3d/velo.f
    384    9.57%   88.88%  /u/ncsa/rkufrin/apps/cx3d/temp.f
    164    4.09%   92.97%  /u/ncsa/rkufrin/apps/cx3d/testin.f
    143    3.56%   96.54%  /u/ncsa/rkufrin/apps/cx3d/curr.f
     53    1.32%   97.86%  ./include/gm_send_queue.h
     23    0.57%   98.43%  ??
     22    0.55%   98.98%  /u/ncsa/rkufrin/apps/cx3d/bound.f
     15    0.37%   99.35%  /u/ncsa/rkufrin/apps/cx3d/csendxs.f
     14    0.35%   99.70%  ./libgm/gm_send.c
     10    0.25%   99.95%  /u/ncsa/rkufrin/apps/cx3d/crecvxs.f
      1    0.02%   99.98%  ./libgm/gm_ptr_hash.c
      1    0.02%  100.00%  ./libgm/gm_hash.c
Function Summary
--------------------------------------------------------------------------------
Samples   Self %  Total %  Function

   3182   79.31%   79.31%  velo
    384    9.57%   88.88%  temp
    164    4.09%   92.97%  testin
    143    3.56%   96.54%  curr
     54    1.35%   97.88%  gm_send_with_callback
```

Profiling Results (psprocess, cont'd)

```
Function:File:Line Summary
--------------------------------------------------------------------------------
Samples   Self %  Total %  Function:File:Line

    687   17.12%   17.12%  velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:232
    535   13.33%   30.46%  velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:260
    509   12.69%   43.15%  velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:210
    378    9.42%   52.57%  velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:356
    189    4.71%   57.28%  velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:493
```

```
$ mpirun -np 8 psrun -c profile_cycles.xml ./cx
$ psprocess -e cx psrun.PID.xml
```

profile_cycles.xml:

```
<ps_hwpc_profile class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_TOT_CYC"
  threshold="30600000"/>
</ps_hwpc_profile>
```

Summary Information (psprocess)

```
Aggregate Statistics                          Min       Max    Median      Mean    StdDev       Sum
===================================================================================================
% CPU utilization....................       97.88     98.41     98.09     98.12      0.17    784.93
% cycles stalled on any resource......       0.00      0.00      0.00      0.00      0.00      0.00
CPU time (seconds)...................       39.95     40.15     39.99     40.01      0.07    320.11
Floating point operations per cycle...       0.05      0.05      0.05      0.05      0.00      0.39
Floating point operations per graduated instruction
                                             0.04      0.04      0.04      0.04      0.00      0.31
Graduated instructions per cycle......       1.27      1.30      1.29      1.29      0.01     10.28
Graduated instructions per issued instruction
                                             0.99      1.00      1.00      1.00      0.00      7.97
Issued instructions per cycle.........       1.28      1.31      1.29      1.29      0.01     10.33
Level 2 cache hit rate (data).........       0.96      0.97      0.97      0.97      0.00      7.74
Level 2 cache line reuse (data).......      27.49     30.82     29.57     29.28      1.22    234.26
MFLOPS (cycles).......................     145.53    154.10    151.18    150.40      3.63   1203.21
MFLOPS (wall clock)...................     142.45    151.50    148.37    147.57      3.64   1180.56
MIPS (cycles).........................    3881.34   3952.56   3924.68   3922.56     28.18  31380.47
MIPS (wall clock).....................    3799.24   3877.19   3854.91   3848.68     30.42  30789.40
MVOPS (cycles)........................       0.00      0.00      0.00      0.00      0.00      0.00
MVOPS (wall clock)....................       0.00      0.00      0.00      0.00      0.00      0.00
Mispredicted branches per correctly predicted branch
                                             0.00      0.01      0.01      0.01      0.00      0.05
Vector instructions per cycle.........       0.00      0.00      0.00      0.00      0.00      0.00
Vector instructions per graduated instruction
 0.00      0.00      0.00      0.00      0.00      0.00
Wall clock time (seconds).............      40.60     40.88     40.79     40.78      0.10    326.25
```

```
$ psprocess -c psrun.*.xml > combined.xml
$ psprocess combined.xml
```

Case Study: Automatic Performance Collection on NCSA Linux Clusters

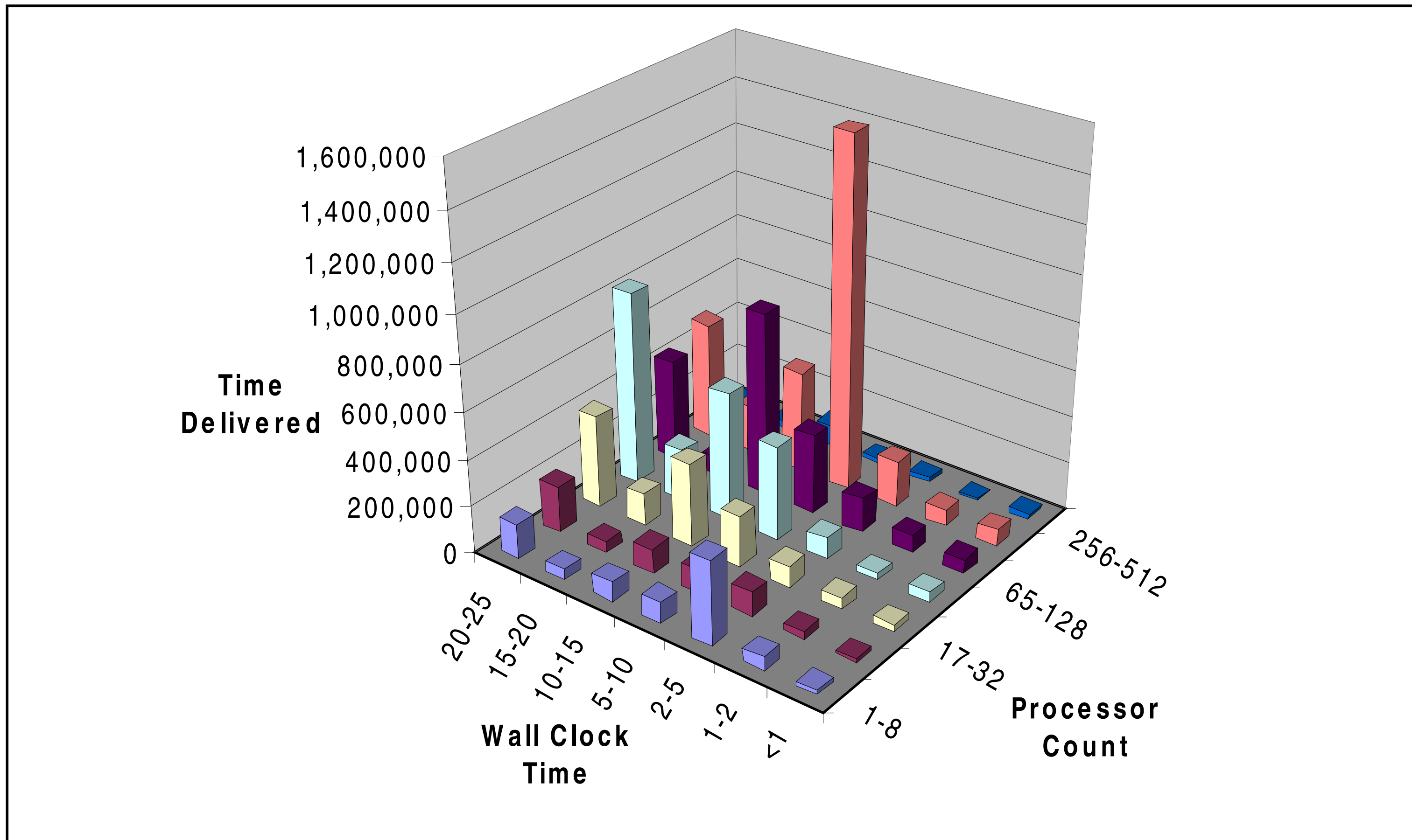
- NCSA transition (c. 2000) from shared-memory “traditional” supercomputers to cluster technology is a major shift:
  - Does it translate *in practice* to high-performance cycles **delivered**?
  - What is the percentage of users making efficient use of the resource?
  - How can knowledge improve services (i.e., feedback loop)?

Project Requirements

- Initial project definition (Jan 2003):
  - Measure the aggregate performance of all user applications on Linux clusters, (new) IBM p690, and (retiring) Origin 2000 systems
  - Unmodified binaries – no impact on or effort required of users
  - Operational within existing job management system – no “special queues” or contacts. Avoid self-selecting users.
  - In-place and operational by March ’03 in order to gather sufficient data for NSF reporting by late summer.
- Implementation
  - Focus narrowed to Linux clusters and PerfSuite used to gather the performance data 24x7
  - Implemented as a wrapper to the standard MPI launch commands
    - Could be extended to serial applications relatively easily
  - Integrated with system support efforts (file management) and a relational database back-end
  - By Supercomputing ’03, nearly 5 million records of performance data gathered

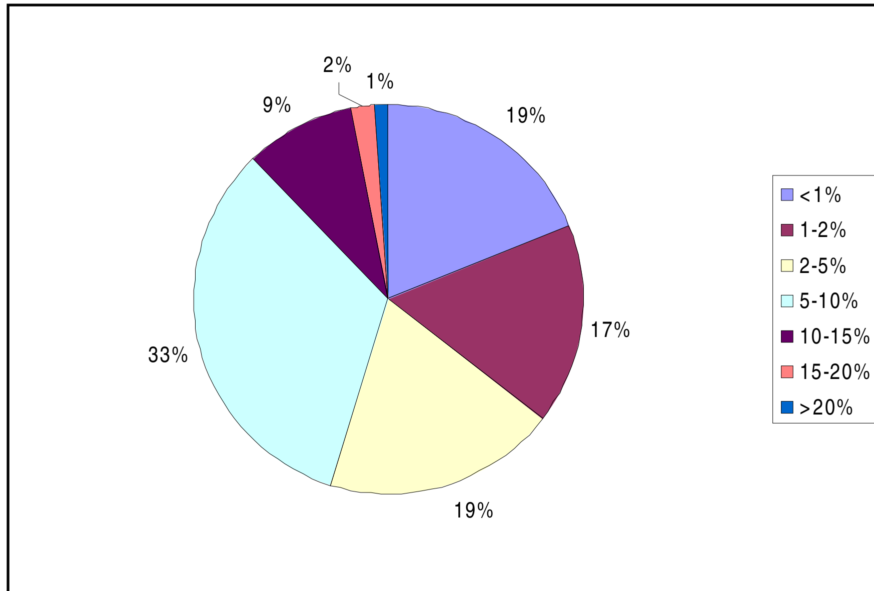
Job Scale (time, processors)

Pentium III FY03

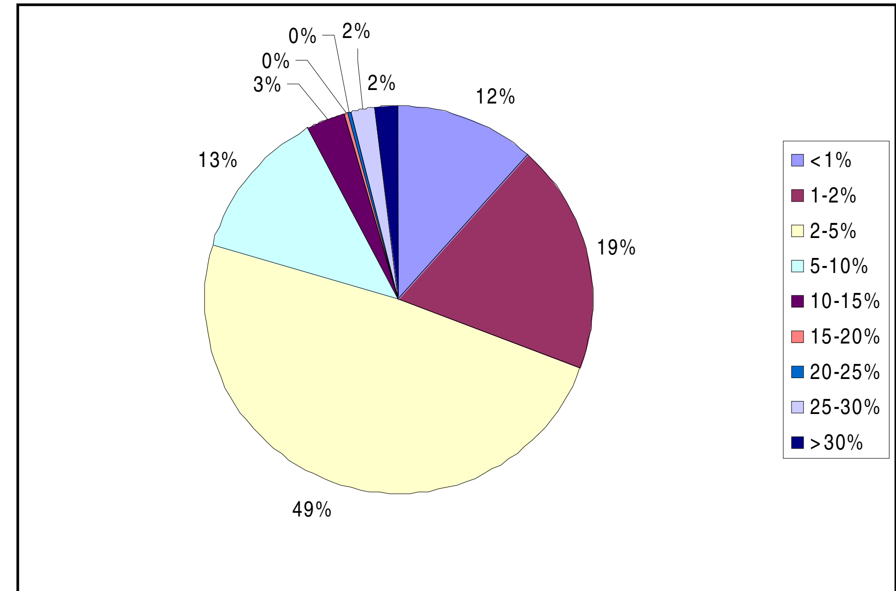


% Peak FP Performance

Pentium III



Itanium



- 10% of peak or greater: 12% on Pentium III, 7% on Itanium
- Note: vector/SIMD instructions not counted as FP_INS / FP_OPS by PAPI

Are Performance Counters Enough?

- Performance counters provide valuable information required for an analysis like this, but:
  - They only provide a CPU-centric view
  - They are not directly comparable across architectures
  - There is no single metric suitable for determining whether an arbitrary application is making “good use” of a machine
- Extensions are being planned to address “off-chip” performance factors

Wouldn’t it be nice *if the computer told you* how it was doing?

PerfSuite Current Status

- Available for download since December 2003
- Currently in a beta cycle with minor enhancements and/or bug fixes
- Expect next beta release (version 0.6.1 beta 5) in June 2004, with the following additions:
  - Support for AMD x86 and x86-64 processors
  - Improved process-handling support
  - New text-only basic mode for psrun (eliminates the need to use psprocess for quick performance runs)
- Likely last changes to be made before focusing on substantial new version for the end of 2004

For More Information & Support

- Three mailing lists available through SourceForge:
  - perfsuite-announce, perfsuite-bugs, and perfsuite-users.
- All are low-volume and no-spam (so far!)
- Must be subscribed to post a note, but they are enabled for archiving. Pointers to subscribing available at http://perfsuite.sourceforge.net/
- The Perfctr, Perfmon, and PAPI lists are all good resources for the underlying software that PerfSuite uses for counter access – you can find links to them at http://perfsuite.ncsa.uiuc.edu/perftools/

DynaProf

A Tutorial on Dynamic Performance Analysis

Philip J. Mucci

mucci@cs.utk.edu

Innovative Computing Laboratory

University of Tenessee, Knoxville

The 5th LCI International Conference on Linux Clusters

The HPC Revolution 2004

May 17th, 2004

Outline

- **DynaProf Introduction**
  - Goal and Overview
  - Architecture
- **Obtaining and Installing DynaProf**
- **Using DynaProf**
  - Instrument running executable
  - Collect and browse performance data
- **DynaProf Current Status**

What is DynaProf?

- A portable tool to dynamically and selectively instrument serial and parallel programs for the purpose of performance analysis and application understand.
- **DynaProf is a portable tool to gather hardware performance data at run time for an unmodified application.**
- **Instrumentation is done through the dynamic insertion of function calls to specially developed performance probes.**
- **DynaProf provides a simple and intuitive command line interface.**

Why the "Dyna" in DynaProf?

- Dynamic Instrumentation means the application' s object code is modified at run-time.
- The instrumentation is contained in simple shared libraries DynaProf calls probes.
- Object code to those functions is generated and then inserted into the program's address space.
- DPCL and DynInst do all the dirty work.

DPCL vs. DynInst

- ***DPCL***
  - ***Based on an early version of DynInst.***
  - ***Supports Asynch./Sync. Operation.***
  - ***Provides functions for getting data back to tool.***
  - ***Integrated with IBM's Parallel Operating Environment.***
  - ***It's Stale! And it requires a working rsh/ssh. (AIX only)***
- ***DynInst***
  - ***Shared libraries, Loops, Basic blocks, Arbitrary locations***
  - ***Provides breakpoints, CFG***
  - ***Single process model.***
  - ***Actively supported on many platforms.***

DynaProf Goals

- Make collection of run-time performance data easy!
- Avoiding the instrumentation/recompilation cycle.
- Avoiding interference with compiler optimization.
- Using the same tool with different probes.
- Providing useful and meaningful probe data.
- Providing different kinds of probes.
- Allow easy development of custom probes.
- Providing complete language independence.

DynaProf Probes

- ***papiprobe***
  - ***Measure any combination of PAPI presets and native events.***
- ***papiclockprobe***
  - ***Measure accurate wallclock and virtual time using PAPI timers.***
- ***wallclockprobe***
  - ***Highly accurate elapsed wallclock time in microseconds.***
- ***perfometerprobe***
  - ***Visualize hardware counter traces in pseudo real-time.***
- ***tauprobe***
  - ***Support all TAU measurement methodologies including timing, memory tracking, hardware counters and call stack tracing.***
- ***vmonprobe***
  - ***Statistical profiling of hardware counter events ala gprof.***

DynaProf Probe Design

- Probes export a few functions with loosely standardized interfaces.
- Easy to roll your own.
  - If you can code a timer, you can write a probe.
- DynaProf detects thread model and will load a special version of the probe.
- The probes dictate how the data is recorded and visualized.

Papiprobe, Papiclockprobe & Wallclockprobe

- These are well tested probes.
- papiprobe
  - Counts hardware counters using PAPI, either PAPI presets or Native events.
  - Supports counter multiplexing:
    - Not good for fine grained instrumentation.
- papiclockprobe
  - Counts cycles using PAPI wallclock and virtual timers.
- wallclockprobe
  - Counts microseconds using real time cycle counter available on each platform.

DynaProf Status

- **Currently supported platforms with DynInst 4.1:**
  - Linux 2.4, 2.6
    - IA/64
    - X86
    - X86_64 on 32-bit executables.
  - Solaris 2.8+
  - IRIX 6.x

DynaProf Installation

- **Download appropriate DynaProf binary distribution from web site and follow the instructions.**
  - **http://www.cs.utk.edu/~mucci/dynaprof**
- **Requirements:**
  - PAPI 2.x/3.x
  - GNU Readline
  - On Linux 2.x:
    - DynInst 3.0, 4.0 or later
    - May require a new binutils, libdwarf and libelf. Available from http://www.paradyn.org.
  - On AIX 4.3/5:
    - DPCL (See /usr/lpp/ppe.dpcl)
    - PMAPI (See /usr/pmapi/*, /usr/lib/libpmapi.a)

Performance Probes

- ***Three probes provide the ability to instrument specific regions of code.***
  - ***Papiprobe***
  - ***Papiclock***
  - ***Wallclock***
- ***These probes generate the following data for each instrumented function:***
  - ***Inclusive:*** $T_{function} = T_{self} + T_{children}$
  - ***Exclusive:*** $T_{function} = T_{self}$
  - ***1-Level Call Tree:*** T_{child} = ***Inclusive*** $T_{function}$

Performance Probe Data

- **The papiprobe, papiclock and wallclock probes produce data in an identical format.**
- **These three probes always measure the entire executable "TOTAL" in addition to any additional instrumentation points the user has specified.**
- **All use a Perl script to display the data in a human readable format. Usage:**
  - papiproberpt <file>
  - papiclockrpt <file>
  - wallclockrpt <file>

Papiprobe

- By default, it measures PAPI_FP_INS or PAPI_TOT_INS if the former is not available.
- Takes a comma separated list of options or PAPI events, either preset or native.
- Passing 'help' as option prints out list of available PAPI presets.
- Passing 'mpx' or “multiplex” as an option enables the use of counter multiplexing if needed.

Making Sense of Papiprobe Data

- Sometimes the data might not make sense so we must to understand the EXACT semantics of the events.
- There is a command that will list all the available PAPI events and their native mappings.
  - Type “papi_avail -a<cr>” for PAPI preset mappings.
  - Type “papi_native_avail -a<cr>” for PAPI native mappings.
- Note the information at the end of each line between the parenthesis. This can be cross-referenced with that in Architecture Manual and the kernel header files. Doesn't that sound fun?

DynaProf Exercises 1 & 2

- **We will use DynaProf to evaluate different versions of SWIM, the shallow water benchmark code.**
  - Discover delivered MFLOP/S and IPC of an entire serial run.
  - Evaluate memory subsystem efficiency of the core compute solvers of a serial run. (Cache miss counts.)

Exercise Preparation

- **Setup the DynaProf tutorial module.**
  - <PREFIX> will be announced at tutorial time.
  - For csh: Type “source <PREFIX>/dynaprof-setup.csh<cr>”
  - For sh: Type “. <PREFIX>/dynaprof-setup.sh<cr>”
  - This will set current environment variables and edit your .cshrc or .profile login script.
- **Build the swim executable.**
  - Type “cd ~/dynaprof/swim; make<cr>”

Exercise 1: Global MFLOP/S & IPC

- **Type "dynaprof<cr>"**
- **Type "load swim<cr>"**
- **Type "use papiprobe PAPI_TOT_CYC, PAPI_FP_INS, PAPI_TOT_INS<cr>"**
- **Type "run<cr>"**
- **Type "quit<cr>"**
  - Note name of the output file at beginning of run.
- **Type "papiproberpt <output_file> | more<cr>"**

Exercise 1 cont.

- **Compute MFLOP/S & IPC:**
  - CPU Seconds = PAPI_TOT_CYC/(Mhz*1.0e6)
  - TMFLOP = PAPI_FP_INS/(1.0e6)
  - MFLOP/S = TMFLOP/Seconds
  - IPC = PAPI_TOT_INS/PAPI_TOT_CYC

Yee_bench

- Benchmark developed at PDC and PSCI.
- Kernel of FDTD method for Maxwell equations in C.E.M.
- 5 versions of the same serial code:
  - Fortran 90: Yee_bench_i386-linux_f90
  - Fortran 77: Yee_bench_i386-linux_f77
  - C: Yee_bench_i386-linux_c
  - C++: Yee_bench_i386-linux_cxx
  - C with non-contiguous arrays: Yee_bench_i386-linux_c_noncont
- Cd Yee_bench
  - Type “make”.

Other Things to Try

- **Listing the available PAPI events.**
  - Type “use papiprobe help” to DynaProf to list all PAPI events.
- **Use multiplexing with lots of PAPI events.**
  - Type “use papiprobe mpx, <event>, ...”
  - **Use only with large granularity measurements!**
- **Attaching to a process instead of loading:**
  - Type “attach <exe> <pid>” after externally starting an application.

List command

- The list command allows you to browse the executable's object code.
- list - List modules in process.
- list module[s] [module] - List modules.
- list function[s] <module> [function] - List functions.
- list child[ren] <module> <function> - List call points in function.

Instr command

- The instr command controls the instrumentation.
- instr - List instrumented locations.
- instr module[s] <module> - Instrument modules.
- instr function[s] <module> <function> - Instrument functions in module.

The Fortran DEFAULT_MODULE

- Normally the DEFAULT_MODULE contains the Fortran runtime libraries.
- When -g is not specified on the command line, PGI and GNU compilers put all user functions in this module.

Exercise 2: Cache Misses

- **Measure solver routines to cache miss counts.**
  - Type “dynaprof<cr>”
  - Type “load swim<cr>”
  - Type “list<cr>”
  - Type “list module swim.F<cr>”
  - Type “list functions swim.F calc*<cr>”
  - Type “list children swim.F inital<cr>”
  - Type “list children swim.F shalow<cr>”

DynaProf Command Line Editing

- **Provides robust command line editing**
  - Arrow Keys and Emacs Bindings:
    - Delete char under cursor
    - C-a Beginning of Line
    - C-e End of line
    - C-<spc> Set mark
    - C-w Cut to mark
    - C-y Yank cut text
  - <TAB> triggers filename completion

Exercise 2 cont.

- Type “use papiprobe PAPI_FP_INS, PAPI_TOT_INS, PAPI_L2_DCM<cr>”
- Type “instr function swim.F calc*<cr>”
- Type “run<cr>”
- Type “quit<cr>
- Note name of the output file at beginning of run.
- Type “papiproberpt <output_file> | more<cr>”

• Try repeating with PAPI_L1_DCM

DynaProf and Threads

- For threaded code, just specify the the same probe!
- DynaProf detects a threaded executable and loads a special version of the probe library.
- The probe detects thread creation and termination.
- All threads share the instrumentation.
- Output goes to <exe>.<probe>.<pid>.<tid>

DynaProf and MPI

- **With DynInst, DynaProf must be run in batch mode as part of the line to mpirun.**
- **DynaProf provides a special load that waits until MPI_Init finishes before continuing.**
  - mpiload <exe> <args>
- **On AIX with DPCL, DynaProf talks directly to the parallel run-time system. (POE)**
  - poeattach <exe> <pid_of_poe>
  - poeload <exe> <poe args>

DynaProf Batch Mode

- **DynaProf can run from a script via command line arguments:**
  - -c <FILE> Specifies the name of a script
  - -b Exits after processing the script
  - -q Suppress printing any output
- **You can see all DynaProf's arguments by using the -h flag.**
- **All arguments have long versions.**

•Exercise 3: Instrument an MPI Application

- **Edit the DynaProf script.**
  - Type “vi swim_mpi.ex3.dp<cr>”
- **Edit the Load Leveler script.**
  - Type “vi swim_mpi.ex3.ll<cr>”
  - Type “llsubmit swim_mpi.ex3.ll<cr>”
- **Look in “swim_mpi.ex3.out” for name of probe output files**
- **Type “papiproberpt <output_file> | more<cr>”**

DynaProf and MPI cont.

- To make it easy, DynaProf comes with a utility called dynaprof_mpi that generates scripts for use with mpirun.

```
[mucci@torc0 llcbench]$ dynaprof_mpi -h
dynaprof_mpi [-lmh] executable-file [-- executable-args]

        -l      Generate script for LAM MPI implementation.
        -m      Generate script for MPICH MPI implementation. (default)
        -h      Print this message.

This is the script generation tool for using DynaProf with MPI
programs.

This program produces two files:

<executable-file>.sh to be used with mpirun.
<executable-file>.dp to be used as a skeleton DynaProf script.
```

dynaprof_mpi Example

```
[mucci@torc0 llcbench]$ dynaprof_mpi ./mpi_bench -- -b
Using MPICH MPI configuration.

Created /home/mucci/LCI04/llcbench/mpi_bench.sh
Created /home/mucci/LCI04/llcbench/mpi_bench.dp

Now edit the /home/mucci/LCI04/llcbench/mpi_bench.dp file
and insert your dynaprof commands.

Then run:

mpirun <mpirun args> /home/mucci/LCI04/llcbench/mpi_bench.sh
-- or --
mpirun <mpirun args> dynaprof -q -b -c /home/mucci/LCI04/llcbench/mpi_bench.dp
```

PGI Fortran 90 Name Mangling

- **PGI Fortran 90 name mangling consist of 3 parts:**
  - Name of source file + _
  - “mod” + _
  - Lowercase function name
- **Example: update.f90, subroutine updateH_homo**
  - update_mod_updateh_homo

References

- **DynaProf and PAPI**
  - http://www.cs.utk.edu/~mucci/dynaprof
  - http://icl.cs.utk.edu/projects/papi
- **DynInst**
  - http://www.dyninst.org
  - http://www.paradyn.org
- **DPCL**
  - http://oss.software.ibm.com/dpcl

References

- **Yee_bench**
  - http://www.pdc.kth.se
  - http://www.psci.kth.se
  - Technical Report: *Yee_bench – A PDC benchmark code*, Report No: TRITA-PDC-2002:1, ISRN KTH/PDC/R--02/1--SE, November 2002, Ulf Andersson

References 2

- **GNU Binutils**
  - http://ftp.gnu.org/gnu/binutils
  - http://sources.redhat.com/binutils
- **GNU Readline**
  - http://cnswww.cns.cwru.edu/~chet/readline/rltop.html
  - http://ftp.gnu.org/gnu/readline

References 3

- **Libdwarf - DWARF Debugging Library**
  - http://reality.sgi.com/davea
- **Libelf – ELF Object File Access Library**
  - http://www.stud.uni-hannover.de/~michael/software/english.html

Acknowledgments

- This work was supported by:
  - DOE SciDAC via PERC

Thank You.

KOJAK / CUBE

Tutorial

Felix Wolf

University of Tennessee, ICL

Linux Clusters: The HPC Revolution 2004

Austin, TX

May 17, 2004

KOJAK / CUBE

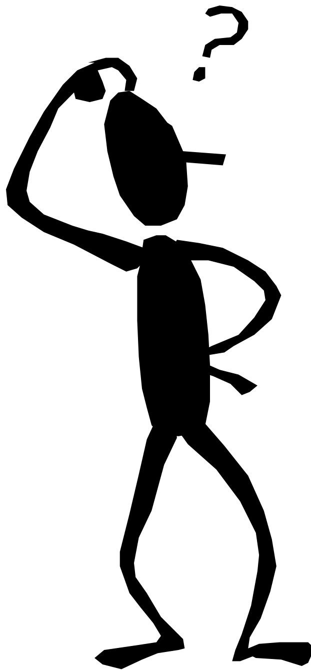
- Collaborative research project between
  - Forschungszentrum Jülich
  - University of Tennessee
- Automatic performance analysis
  - MPI and/or OpenMP applications
  - Parallel communication analysis
  - CPU and memory analysis
- WWW
  - http://www.fz-juelich.de/zam/kojak/
  - htttp://icl.cs.utk.edu/kojak/
- Contact
  - kojak@cs.utk.edu

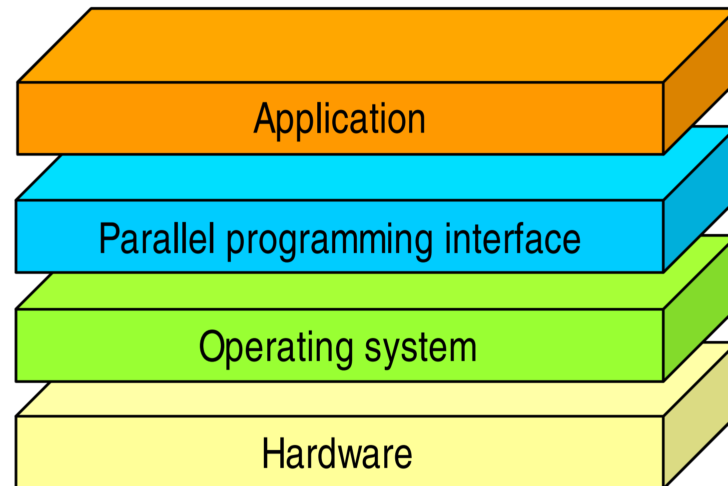
Outline

- Introduction
- Overall architecture
- Functionality
  - Instrumentation
  - Analysis
  - Presentation
- Installation
- Usage

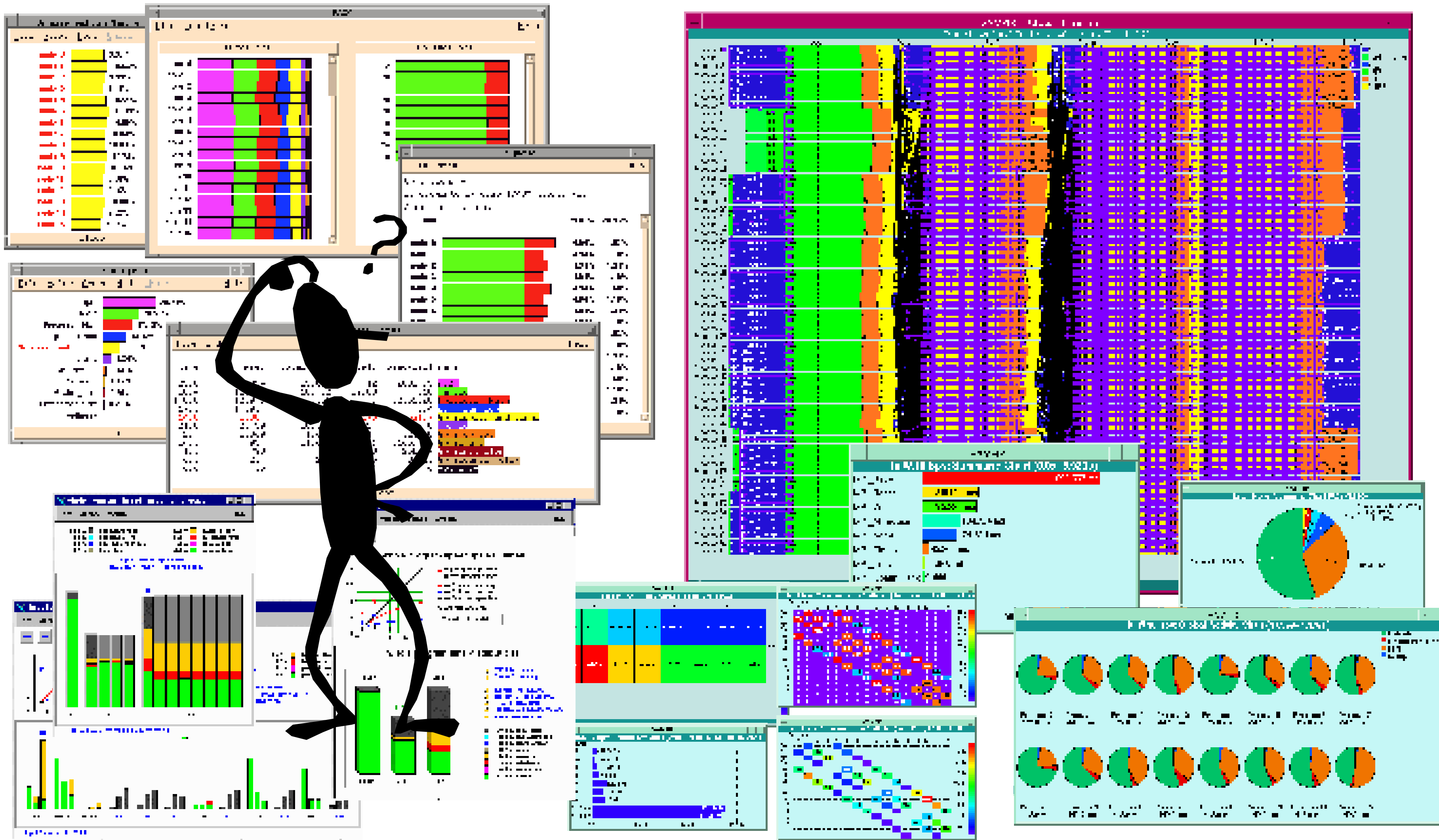
Complexity in Parallel Systems

- Parallel applications rarely achieve available performance
- Performance behavior hard to understand
- Complex interactions between different system layers



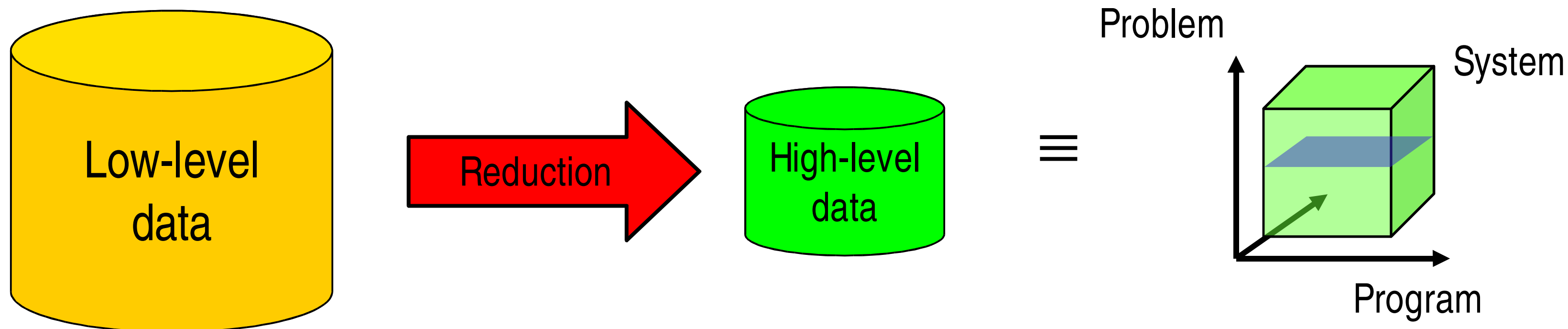


Low-level View of Performance Behavior

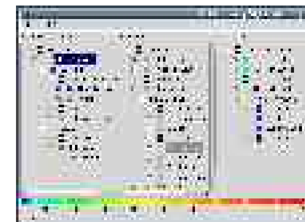


Automatic Performance Analysis

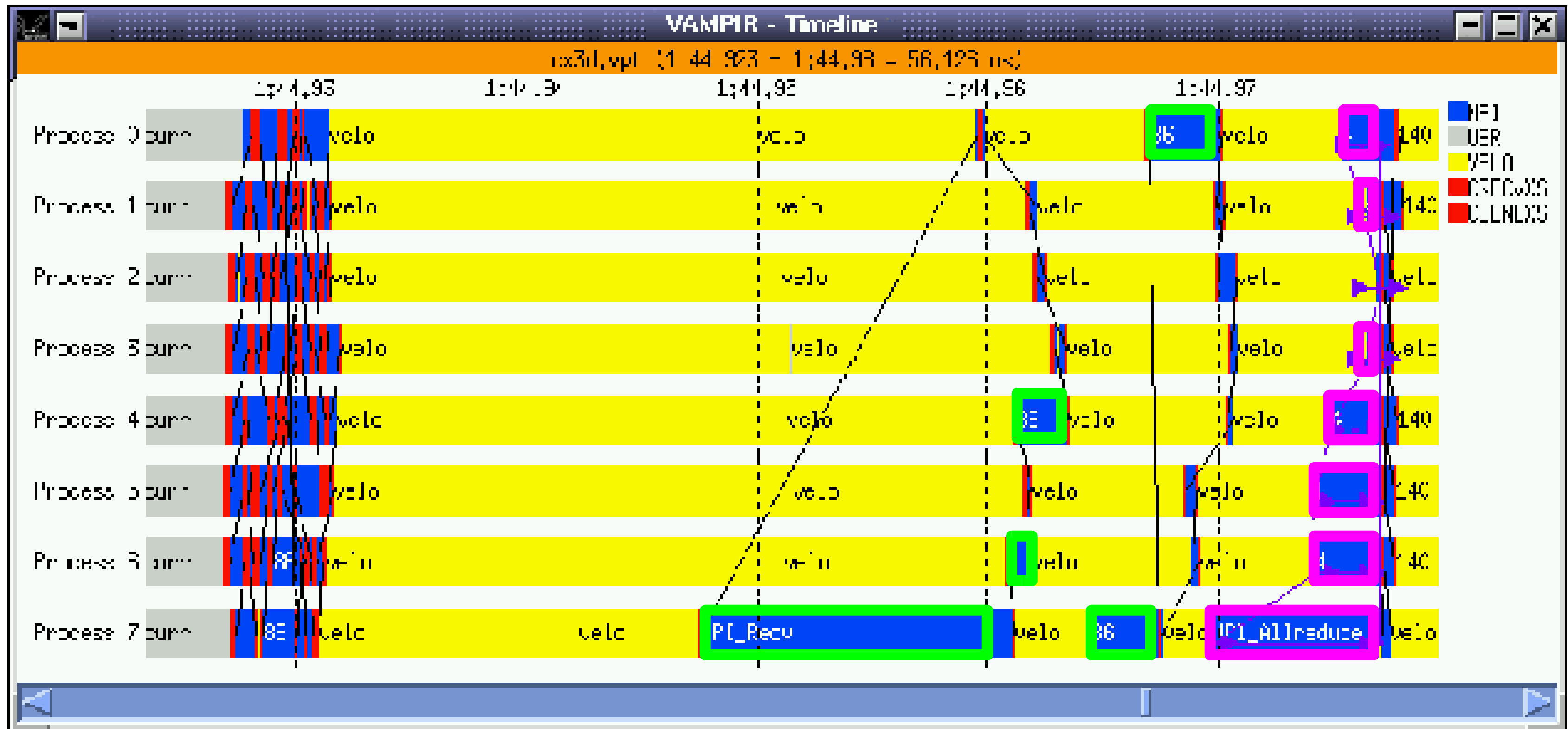
- Transformation of low-level performance data



- Take event traces of MPI/OpenMP applications
- Search for execution patterns
- Calculate mapping
  - Problem, call path, system resource $\Rightarrow$ time
- Display in performance browser

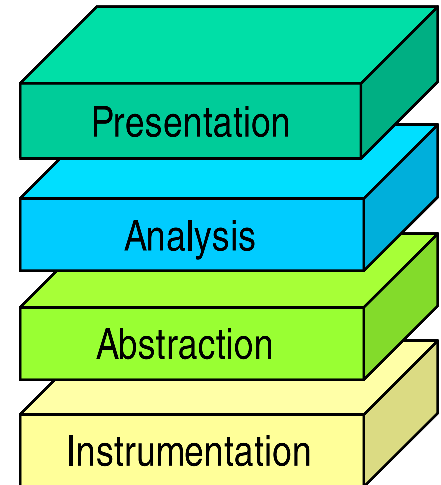


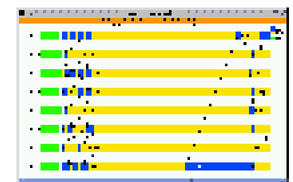
Example



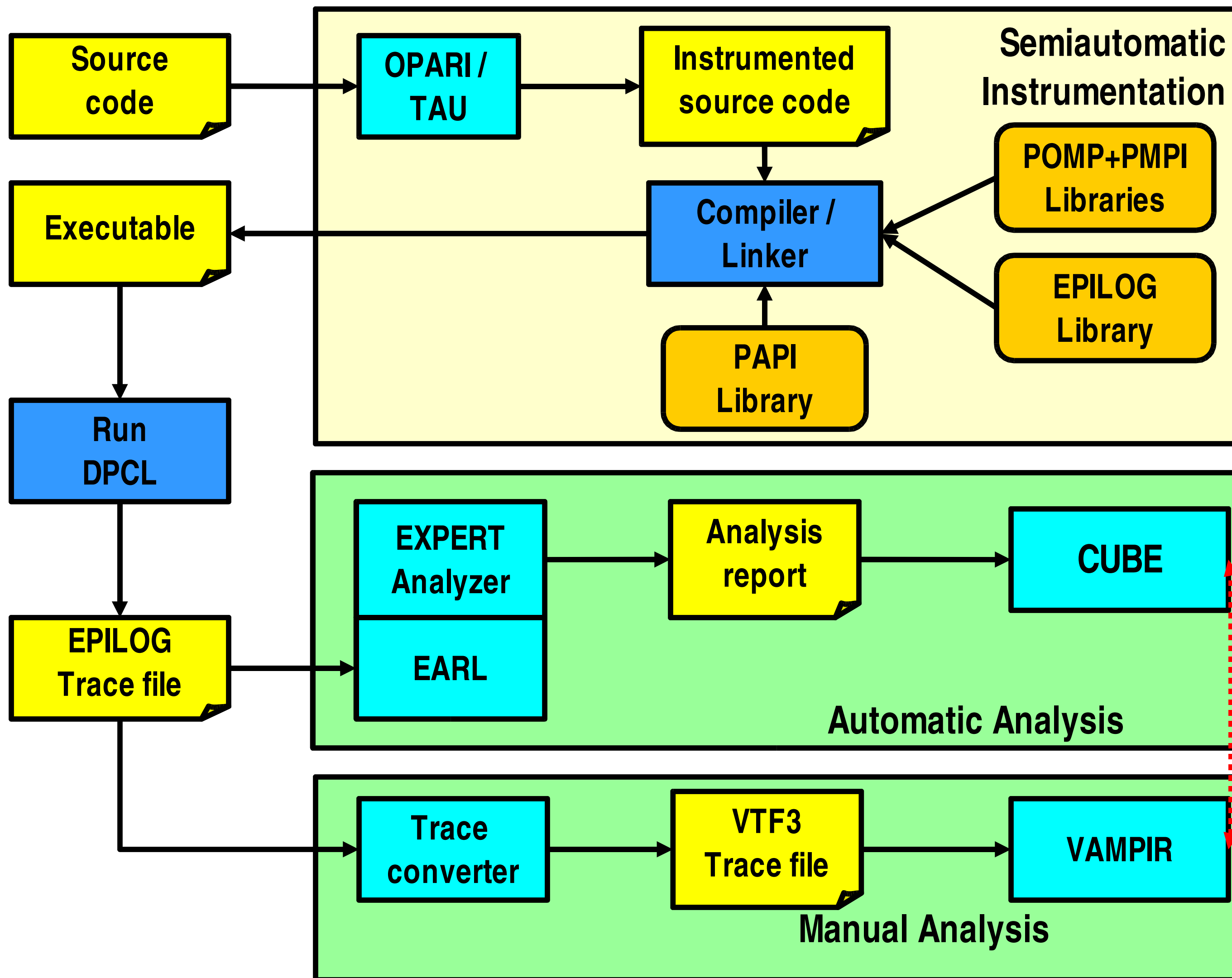
Overall Architecture

- Automatic instrumentation
  - Profiling interface PGI compiler / TAU
- Abstract representation of event trace
  - Simplified specification of performance problem
  - Simplified extension of predefined problems
- Automatic analysis
  - Classification and quantification of performance behavior
  - Automatic comparison of multiple experiments
    - Not yet released
- Presentation
  - Navigating / browsing through performance space
  - Can be combined with time-line display





Overall Architecture



Release Version 2.0b

- Platforms
  - **Instrumentation, measurement, analysis, and display**
    - Linux IA-32 clusters with GNU, PGI, or Intel compilers
    - IBM Power3 / Power4 based clusters
    - SGI Mips (O2K, O3K) and IA-64 based (Altix) clusters
    - SUN Sparc based clusters
  - **Instrumentation and measurement only**
    - Hitachi SR-8000
    - Cray T3E
    - NEC SX
- Requirements
  - **CUBE** 1.0.2 (separate download)
    - **Requires wxWidgets, libxml2**
  - **PAPI** Performance Application Programming Interface

Tracing

- Recoding of individual time-stamped program events as opposed to aggregated information
  - Entering and leaving a function
  - Sending and receiving a message
- Typical event records include
  - Timestamp
  - Process or thread identifier
  - Event type
  - Type-specific information
- Event trace
  - Sequence of events in chronological order

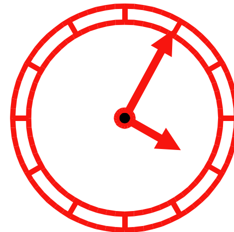
Tracing (2)

Process A:

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

Process B:

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

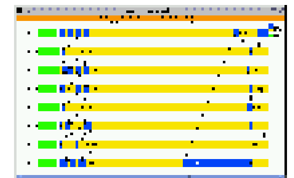


1	master
2	slave
3	...

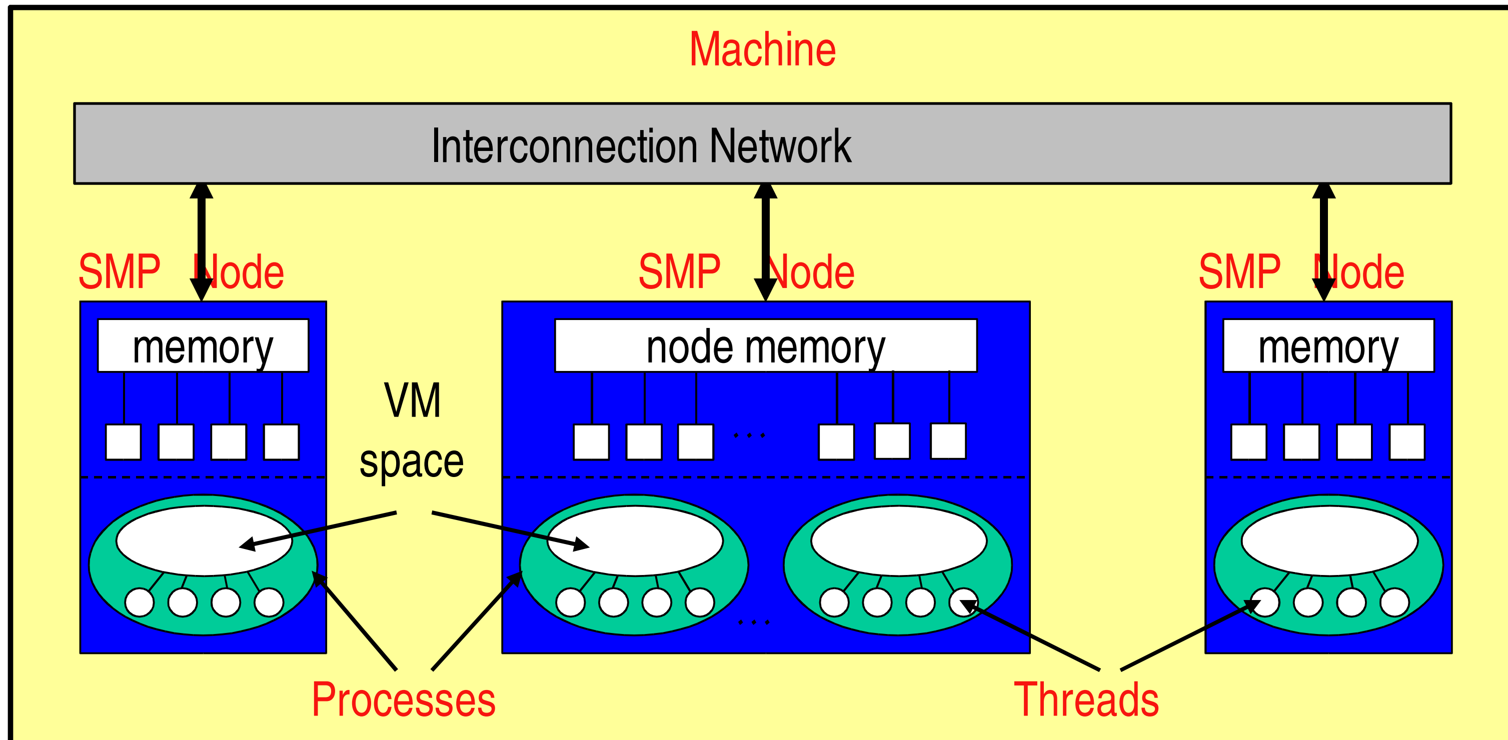
...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			

EPILOG Trace File Format

- Event Processing, Investigation, and LOGging
- MPI and OpenMP support (i.e., thread-safe)
  - Region enter and exit
  - Collective region enter and exit (MPI & OpenMP)
  - Message send and receive
  - Parallel region fork and join
  - Lock acquire and release
- Stores source code + HW counter information
- Input of the EXPERT analyzer
- Visualization using VAMPIR
  - EPILOG VTF3 converter



EPILOG Trace File Format (2)



- Hierarchical location ID
  - (machine, node, process, thread)
- Specification
  - http://www.fz-juelich.de/zam/docs/autoren2004/wolf

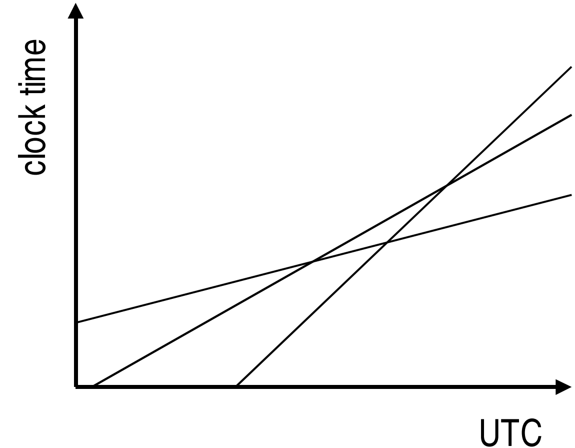
Clock Synchronization

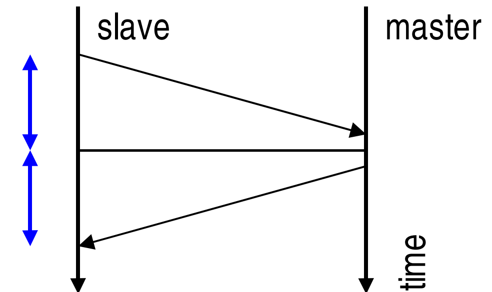
- Time ordering of parallel events require global time
- Accuracy requirements
  - Correct order of message events (latency!)
- Linux clusters usually provide only distributed local clocks
- Local clocks may differ in drift and offset
  - Drift: clocks may run differently fast
  - Offset: clocks may start at different times
- Clock synchronization
  - Hardware:cannot be changed by tool builder
  - Software: online / offline
- Online: (X)NTP accuracy usually too low

Offline Clock Synchronization

- Model
  - Different offset
  - Different but constant drift (approximation!)
  - One master clock
- Algorithm
  - Measure offset slave ↔ master (2x)
  - Request time from master (Nx)
    - Take shortest propagation time
    - Assume symmetric propagation
  - Get two pairs of (slave time s_i, offset o_i)
  - Master time

$$m(s) := s + \frac{(o_2 - o_1)}{(s_2 - s_1)} * (s - s_1) + o_1$$





Instrumentation

- Generating event traces requires extra code to be inserted into the application
- Supported programming languages
  - C, C++, Fortran
- Automatic instrumentation of MPI
  - PMPI wrapper library
- Automatic instrumentation of OpenMP
  - POMP wrapper library
- Automatic instrumentation of user code / functions
  - Using PGI compiler and kinst tool
  - Using TAU
- Manual instrumentation of user code / functions
  - Using POMP directives and kinst-pomp tool

PGI Compiler and kinst Tool

- Put kinst in front of every compile and link line in your makefile

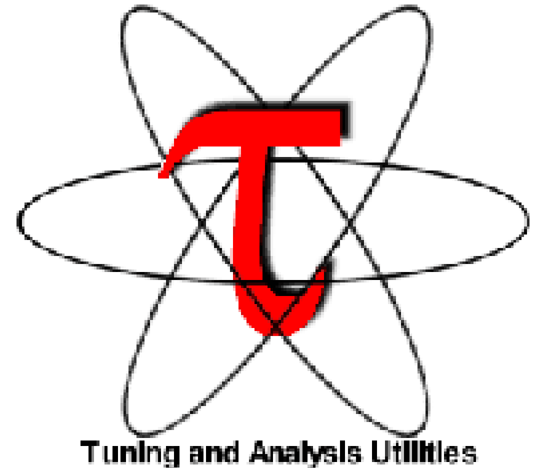
```
# compiler
CC       = kinst pgcc
F90      = kinst pgf90

# compiler MPI
MPICC    = kinst mpicc
MPIF90   = kinst mpif90
```

- Build as usual, everything else is taken care off
  - Instrumentation of OpenMP constructs
  - Instrumentation of user functions
- If trace file becomes too big, exclude certain files from instrumentation
  - Compile those files without kinst

Other Platforms

- Compiler-supported automatic instrumentation of user functions also available on the following platforms
  - HITACHI SR-8000
  - SUN Solaris (Fortran90 only)
  - NEC SX 6
- Autmatic instrumentation with DPCL
  - IBM AIX

TAU Source Code Instrumentor



- Based on PDTOOLKIT
- Part of the TAU performance framework
- Supports
  - f77, f90, C, and C++
  - OpenMP, MPI
  - HW performance counters
  - Selective instrumentation
- http://www.cs.uoregon.edu/research/paracomp/tau/
- Configure with -epilog=<dir> to specify location of EPILOG library

POMP Directives

- Instrumentation of user-specified arbitrary (non-function) code regions
- C/C++

```
#pragma pomp inst begin(name)
      ...
  [ #pragma pomp inst altend(name) ]
      ...
#pragma pomp inst end(name)
```

- Fortran

```
!$POMP INST BEGIN(name)
      ...
  [ !$POMP INST ALTEND(name) ]
      ...
!$POMP INST END(name)
```

POMP Directives (2)

- Insert once as the first executable line of the main program

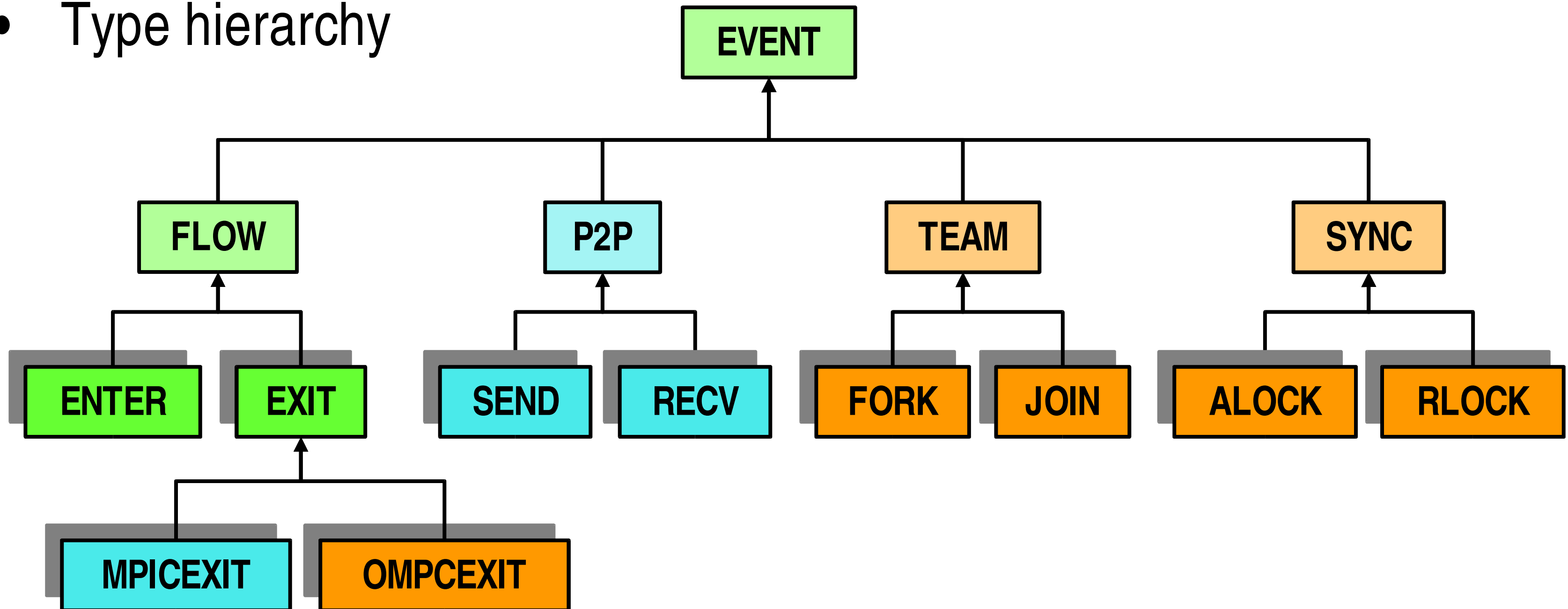
```
#pragma pomp inst begin(name)
```

```
!$POMP INST BEGIN(name)
```

```
# compiler
CC       = kinst-pomp pgcc
F90      = kinst-pomp pgf90

# compiler MPI
MPICC    = kinst-pomp mpicc
MPIF90   = kinst-pomp mpif90
```

KOJAK Event Model

- Type hierarchy



- Event type
  - Set of attributes (time, location, position, …)
- Event trace
  - Sequence of events in chronological order

Automatic Analysis with EXPERT

- Offline trace-analyzer
  - EPILOG input format
- Searches for execution patterns that indicate inefficient behavior
  - Performance properties of an application
- Transforms traces into compact representation of performance behavior
  - Mapping of call paths, process or threads into metric space
- Implemented in C++
  - Earlier version in Pyhton
- Uses EARL library to access event trace

Abstraction with EARL

- EARL library provides random access to individual events
- Computes links between corresponding events
  - E.g., From RECV to SEND event
- Identifies groups of events that represent an aspect of the program's execution state
  - E.g., all SEND events of messages in transit at a given moment
- Implemented in C++
- Language bindings
  - C++
  - Python

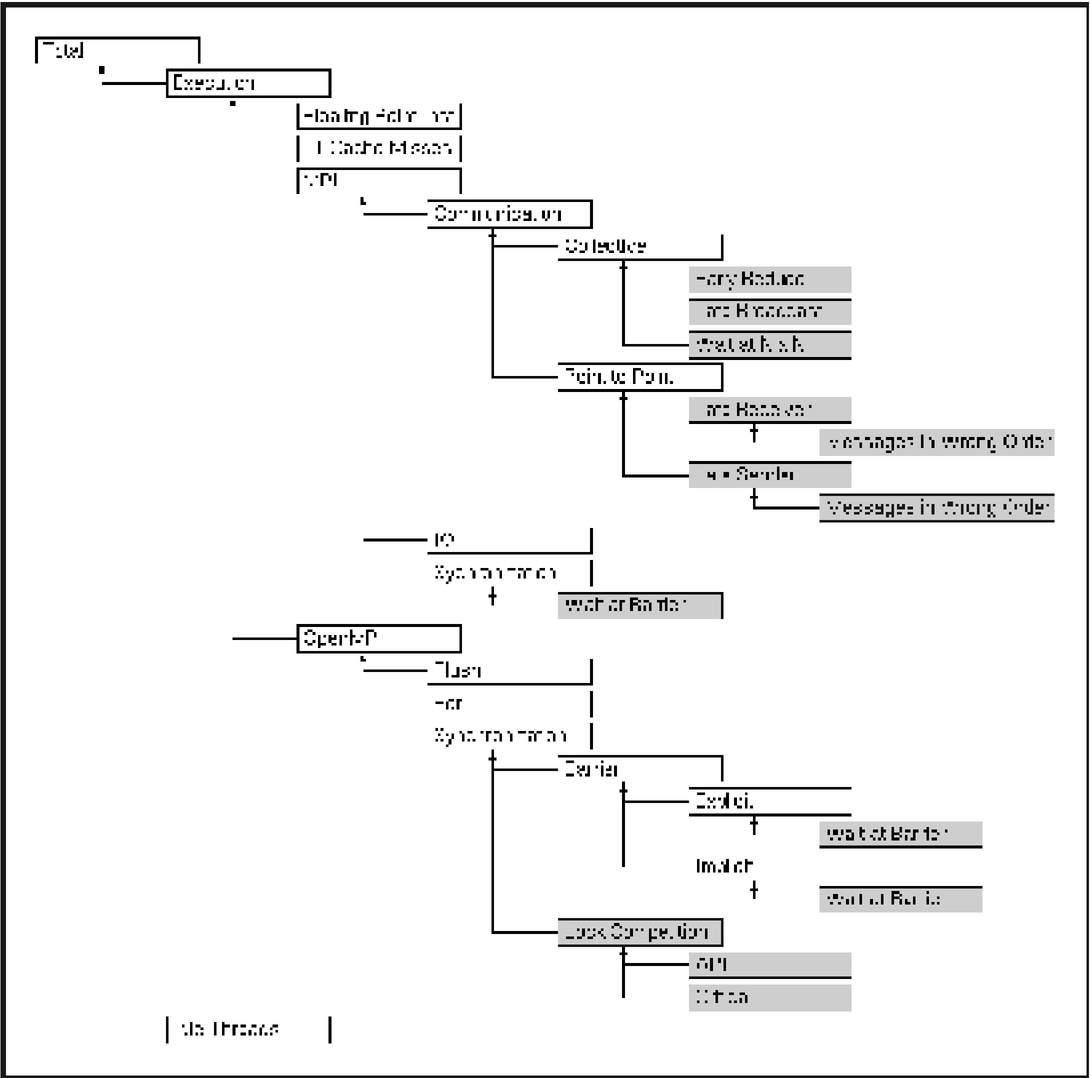
Pattern Specification

- Pattern
  - Compound event
  - Set of primitive events (= constitutents)
  - Relationships between constituents
  - Constraints
- Patterns specified as a C++ class
  - Provides callback method to be called upon occurrence of a specific event type in event stream (root event)
  - Uses links or state information to find remaining constituents
  - Calculates (call path, location) matrix containing the time spent on a specific behavior in a particular (call path, location) pair
  - Location can be a process or a thread

Pattern Specification (2)

- Two types of patterns
- Profiling patterns
  - Simple profiling information
    - How much time was spent in MPI calls?
  - Described by pairs of events
    - ENTER and EXIT of certain routine (e.g., MPI)
- Patterns describing complex inefficiency situations
  - Usually described by more than two events
  - E.g., late sender or synchronization before all-to-all operations
- All patterns are arranged in an inclusion hierarchy
  - Inclusion of execution-time interval sets exhibiting the performance behavior
  - E.g., execution time includes communication time

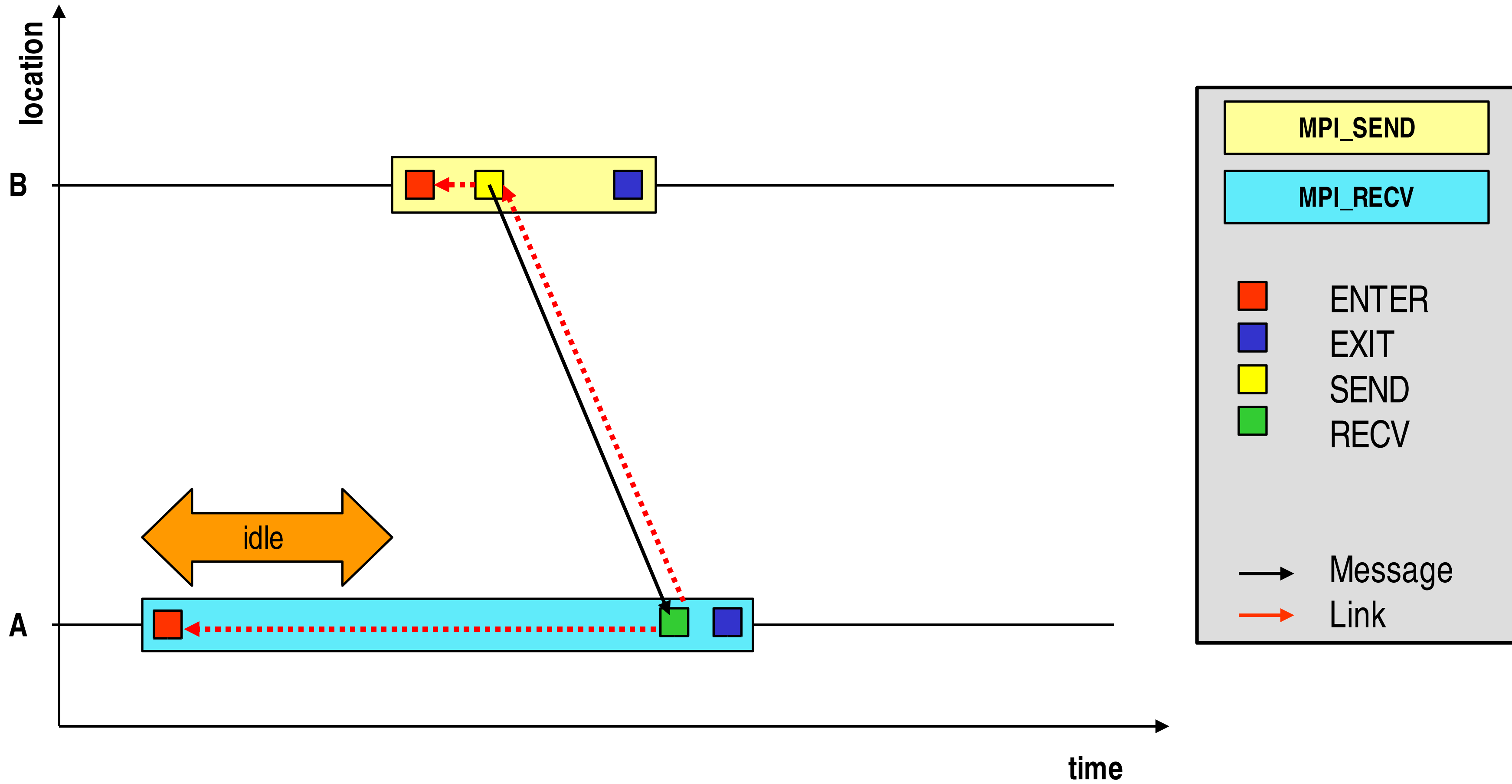
Pattern Hierarchy



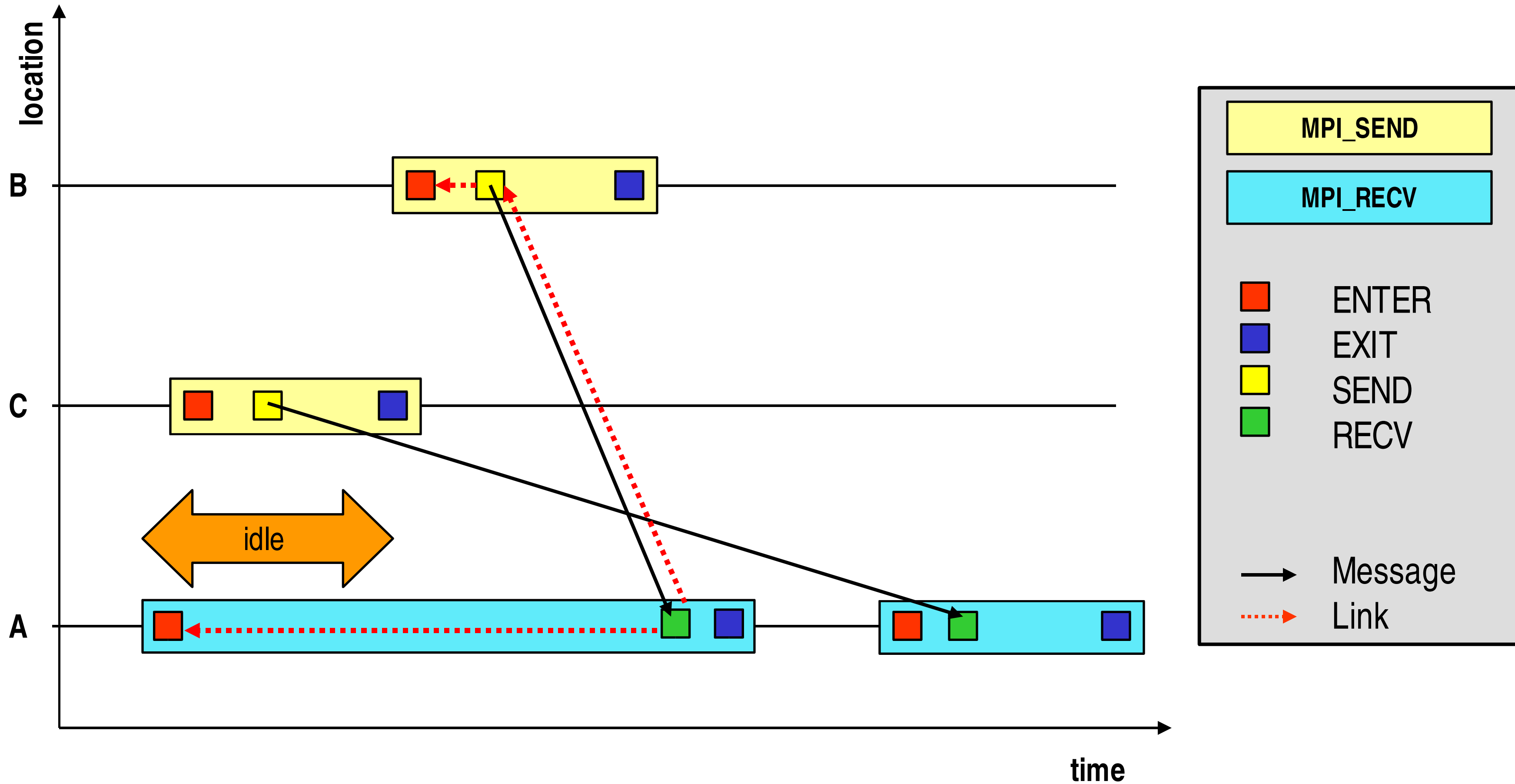
Pattern Search

- Register each pattern for specific event type
  - Type of root event
- Read the trace file one from the beginning to the end
  - Depending on the type of the current event
    - Invoke callback method of pattern classes registered for it
  - Callback method
    - Accesses additional events to identify remaining constituents
    - To do this it may follow links or obtain state information
- Pattern from an implementation viewpoint
  - Set of events hold together by links and state-set boundaries

Late Sender



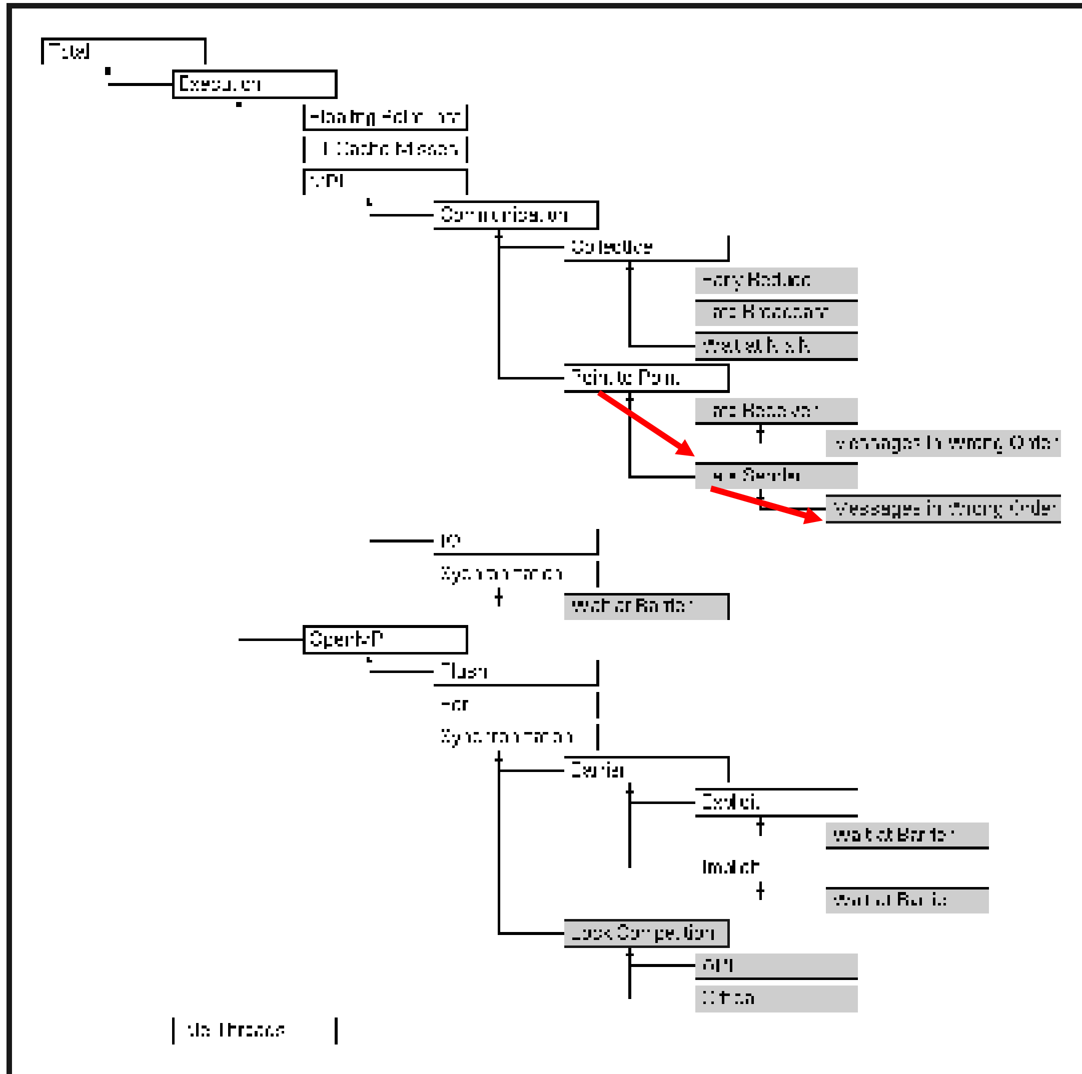
Late Sender / Wrong-Order



Successive Refinement

- Exploit specialization relationships among different patterns
- Pass on compound-event instances from more general pattern (class) to more specific pattern (class)
  - Along a path in the pattern hierarchy
- Previous strategy
  - Patterns could register only for primitive events (e.g., RECV)
- New strategy in KOJAK 2.0
  - Patterns can publish compound events
  - Patterns can register for primitive events and compound events

Potential Pathway of a Pattern Instance

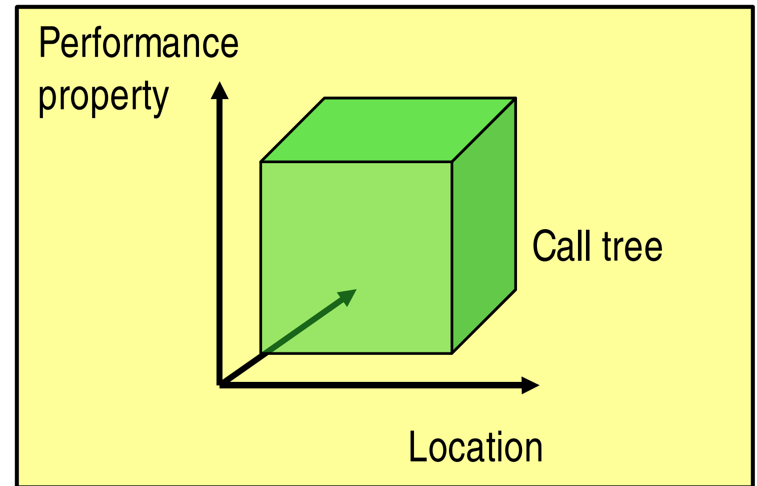


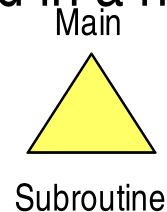
Hardware Counters

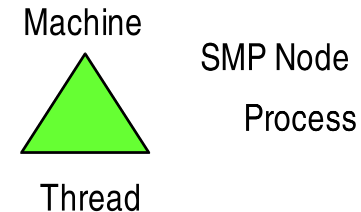
- Small set of CPU registers that count events
  - Events: signal related to a processor's function
- Original purpose
  - Verification and evaluation of CPU design
- Can help answer question
  - How efficiently is my application mapped onto the underlying architecture?
- KOJAK and hardware counters
  - Can be recorded as part of ENTER/EXIT event records
  - KOJAK identifies tuples (call path, thread) whose event rate is below/above average
  - GUI computes aggregated time of those tuples

Representation of Performance Behavior

- Three-dimensional matrix
  - Performance property (problem)
  - Call-tree node
  - Process or thread
- Uniform mapping onto time
  - Each cell contains fraction of (severity)
  - E.g. waiting time, overhead
- Each dimension is organized in a hierarchy



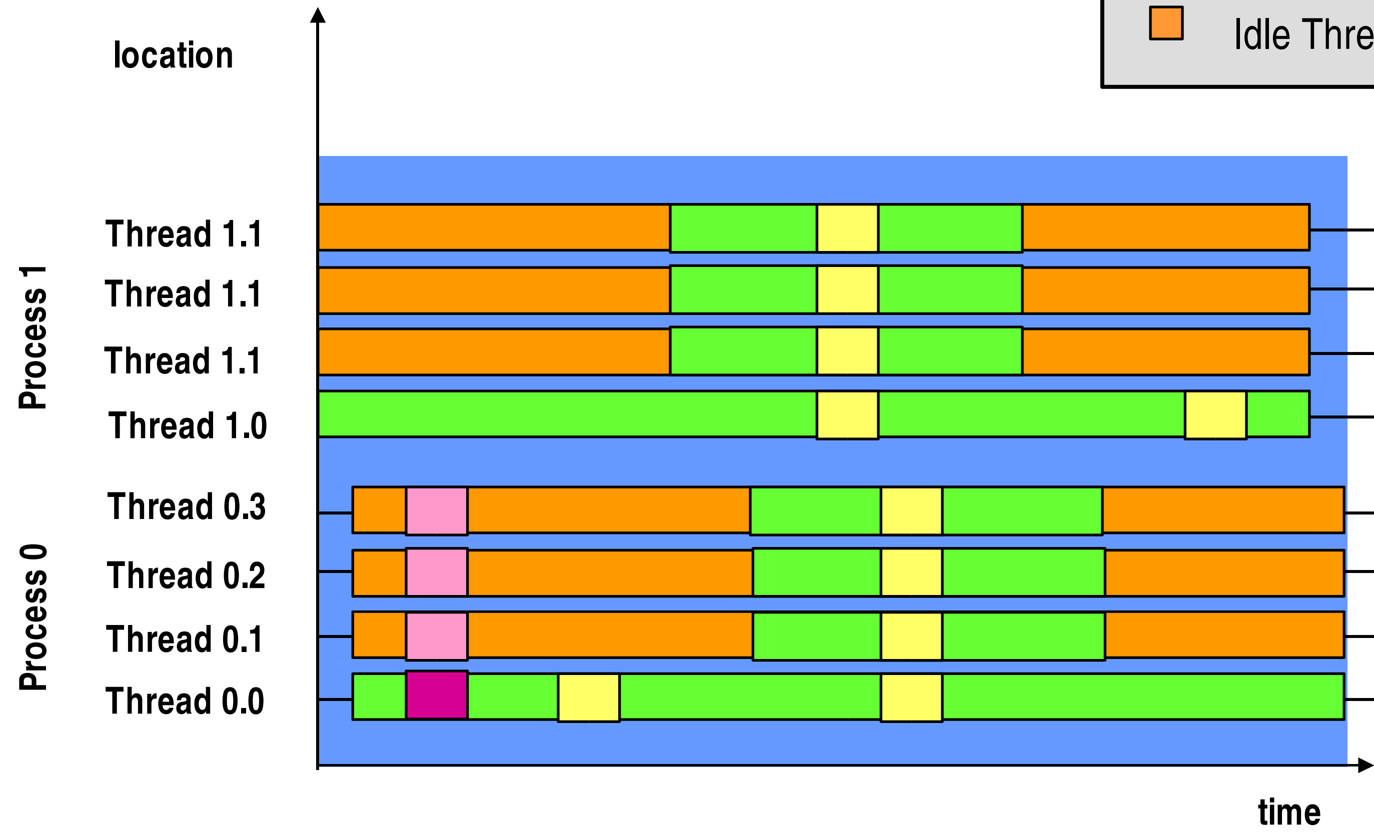




KOJAK Time Model

Performance Properties

- CPU Reservation
- Execution
- Idle Threads



Profiling Patterns (Samples)

- Execution time

Total Execution	# Execution time including idle threads # Execution time

- CPU and memory performance

L1 Data Cache Floating Point	# L1 data miss rate above average # FP rate below average

- MPI and OpenMP

MPI OpenMP Idle Threads	# MPI API calls # OpenMP API calls # Time lost on unused CPUs during OpenMP sequential execution

Complex Patterns (Samples)

- MPI

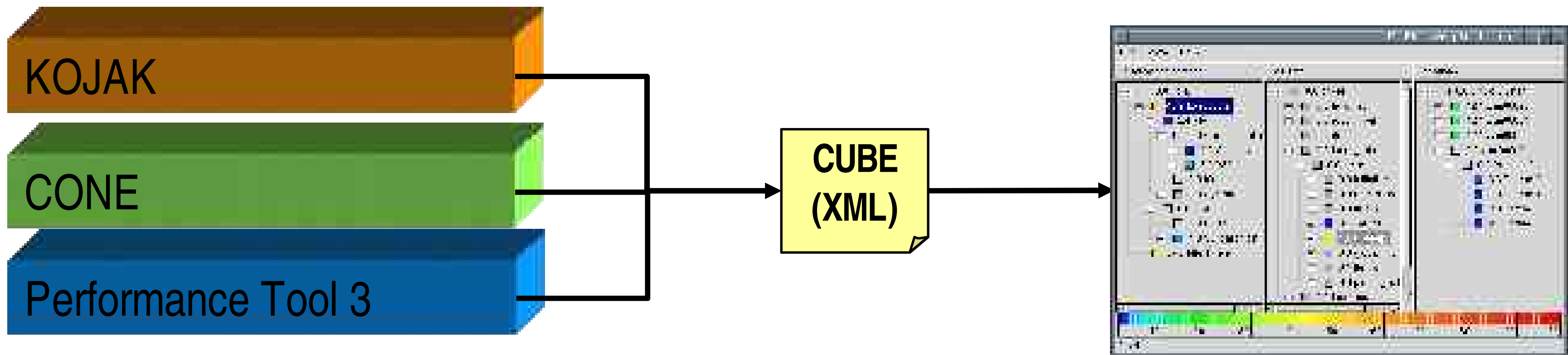
Pattern	Description
Late Sender	# Blocked receiver
Late Receiver	# Blocked sender
Messages in Wrong Order	# Waiting for new messages although older messages ready to be received
Wait at N x N	# Waiting for last participant in N-to-N operation
Late Broadcast	# Waiting for sender in broadcast operation

- OpenMP

Pattern	Description
Wait at Barrier	# Waiting time in explicit or implicit barriers
Lock Synchronization	# Waiting for lock owned by another thread

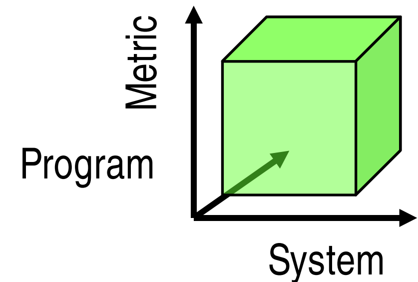
CUBE Uniform Behavioral Encoding

- Abstract data model of performance behavior
- Portable data format (XML)
- Documented C++ API to write CUBE files
- Generic presentation component
- Performance-data algebra



CUBE Data Model

- Most performance data are mappings of aggregated metric values onto program and system resources
  - Performance metrics
    - Execution time, floating-point operations, cache misses
  - Program resources (static and dynamic)
    - Functions, call paths
  - System resources
    - Cluster nodes, processes, threads



- Hierarchical organization of each dimension
  - Inclusion of metrics, e.g., cache misses $\subseteq$ memory accesses
  - Source code hierarchy, call tree
  - Nodes hosting processes, processes spawning threads

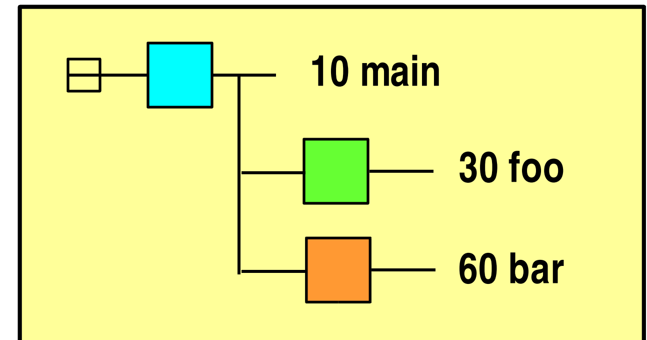
CUBE API

Writing CUBE data files

- Defining metric hierarchies
  - Times, events, sizes
- Defining the call tree
  - Flat profiles considered special case of tree profiles
- Defining system hierarchies consisting of
  - Machines, nodes, processes, and threads
- Entering metric values
  - Provide value for each tuple (metric, call path, thread)

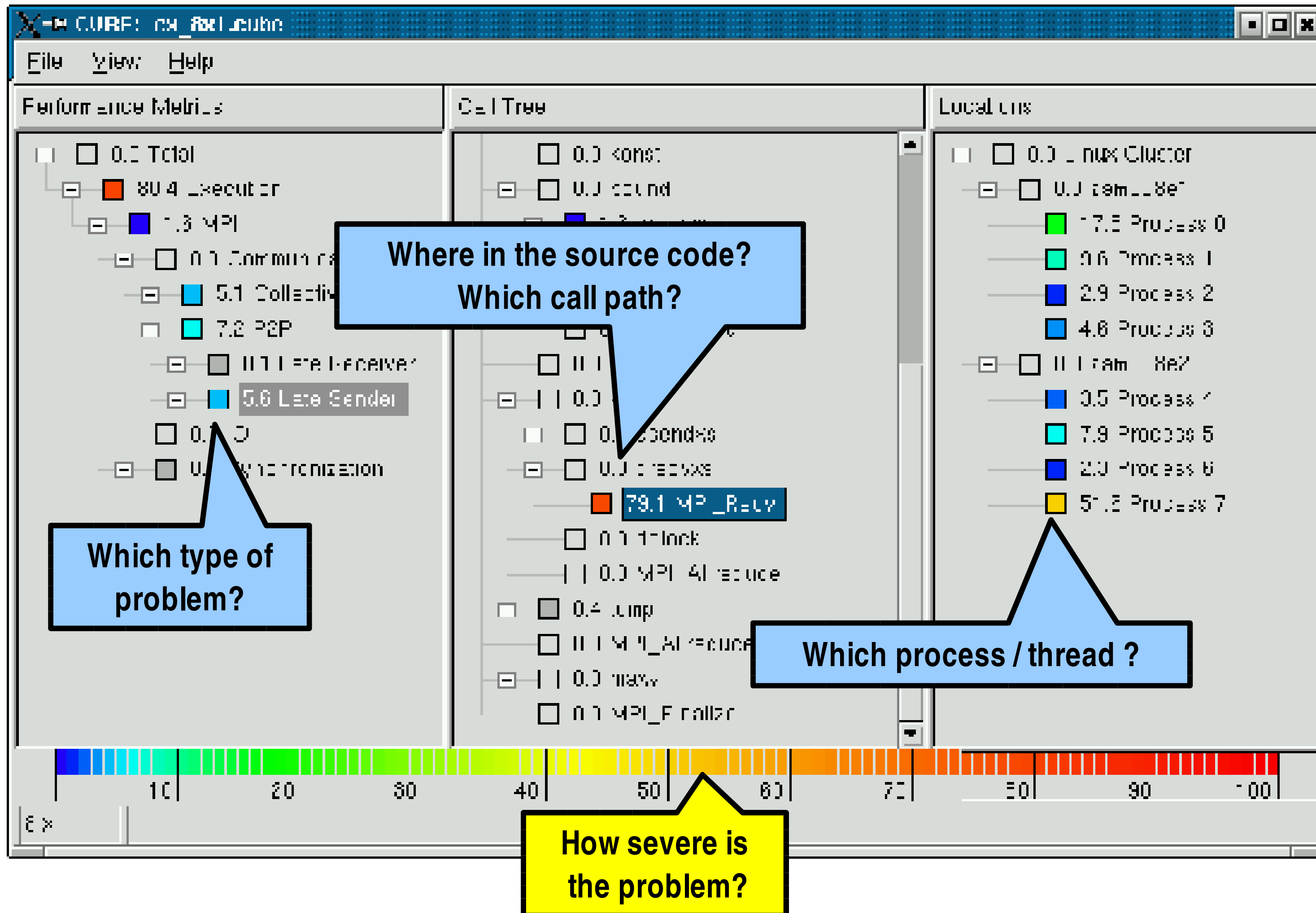
CUBE GUI

- Design emphasizes simplicity by combining a small number of orthogonal features
- Three coupled tree browsers
- Each node labeled with metric value
- Limited set of actions
- Selecting a metric / call path
  - Break down of aggregated values
- Expanding / collapsing nodes
  - Collapsed node represents entire subtree
  - Expanded node represents only itself without children
- Scalable because level of detail can be adjusted

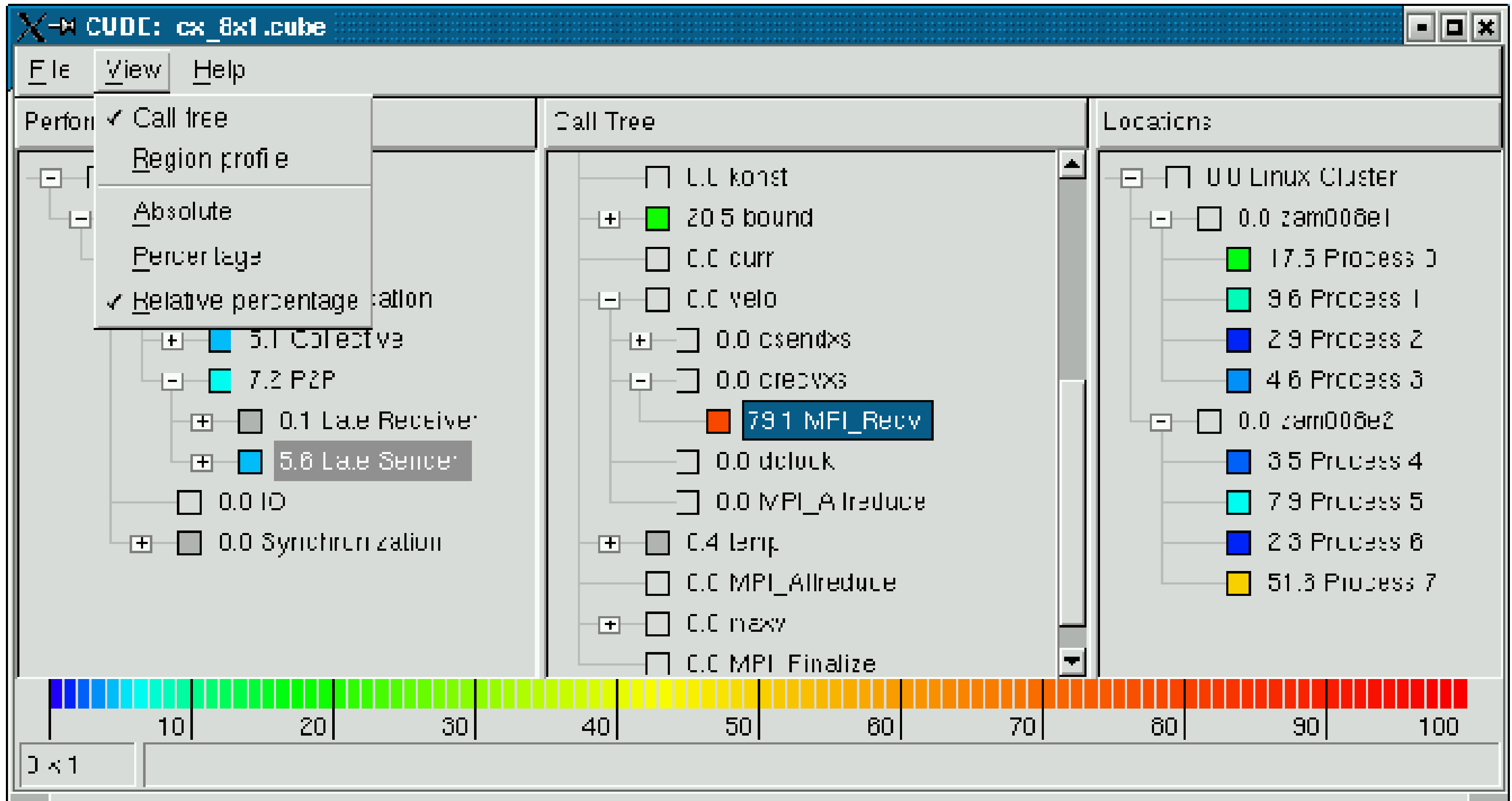


CUBE GUI (2)

- CX3D application Forschungszentrum Jülich, IFF



View Options

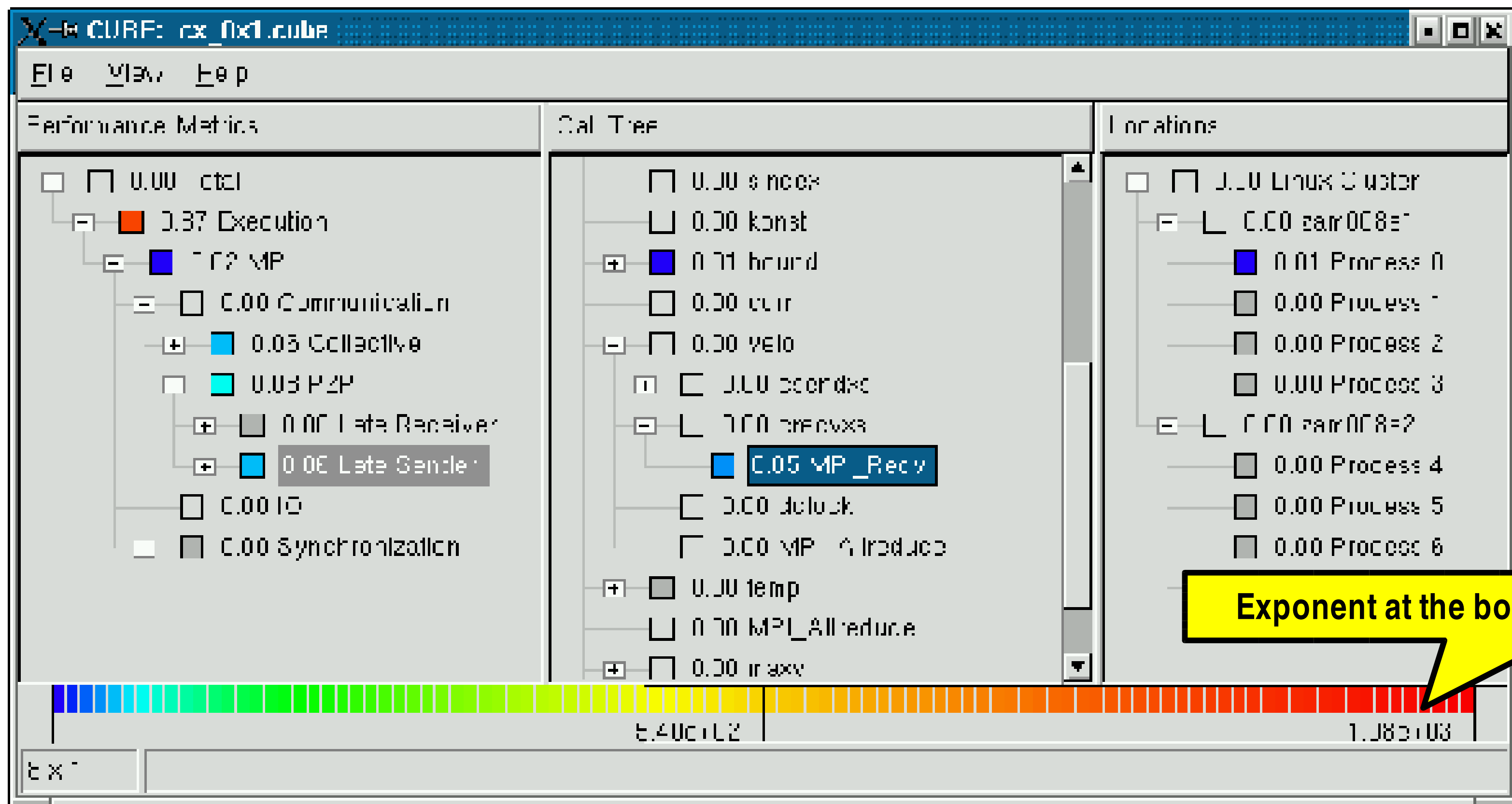


View Options (2)

- Number representation
  - Absolute
    - All values accumulated time values in seconds
    - Scientific notation, color legend shows exponent
  - Percentage
    - All values percentages of the total execution time
  - Relative percentage
    - All values percentages of the selection in the left neighbor tree
- Program resources
  - Call tree
  - Flat region profile
    - Module, region, subregions
- Note that the more general CUBE model also allows for other metrics (e.g., cache misses)

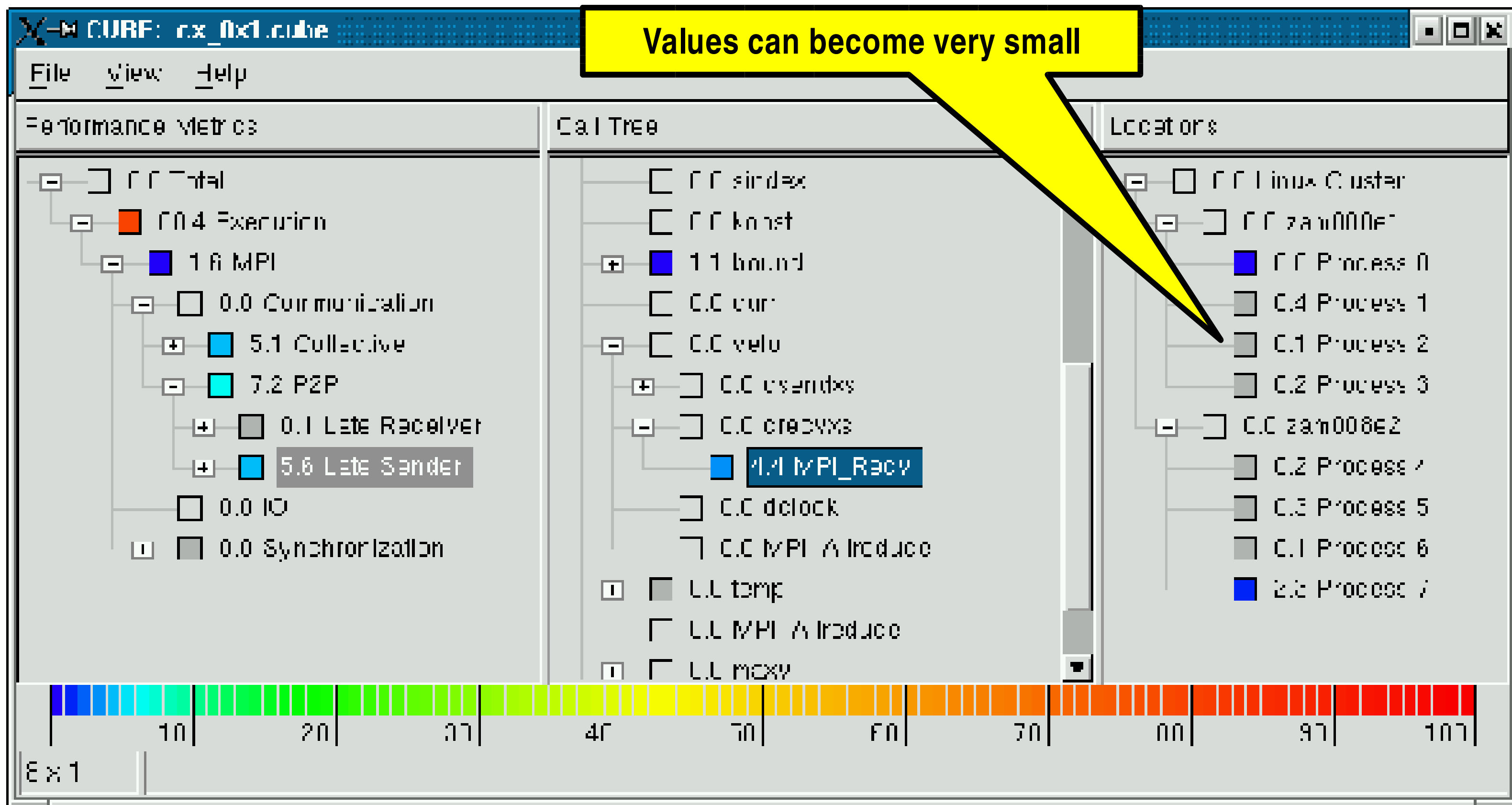
Absolute Mode

- All values accumulated time values in seconds
- Scientific notation, color legend shows exponent



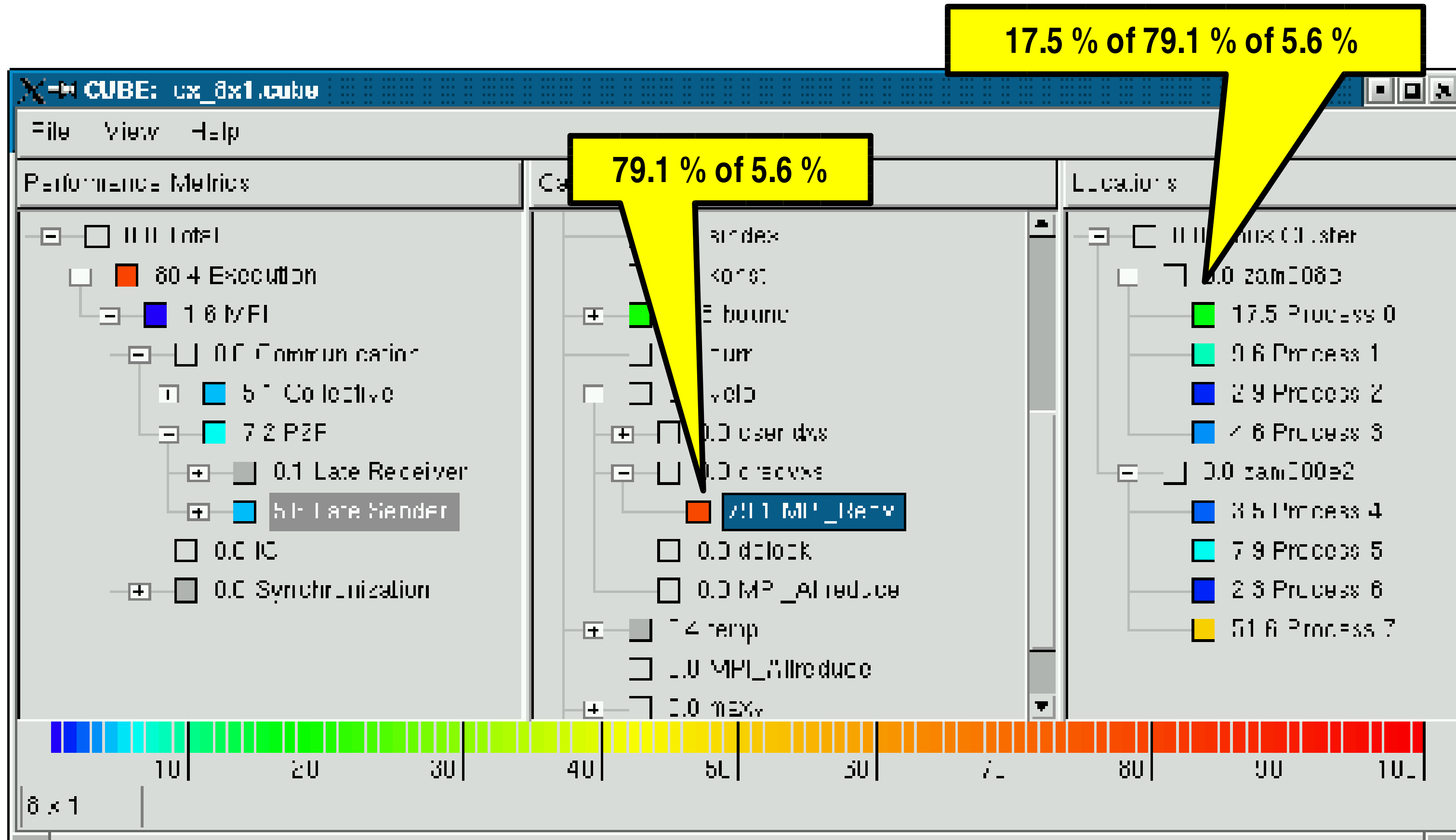
Percentage Mode

- All values percentages of the total execution time

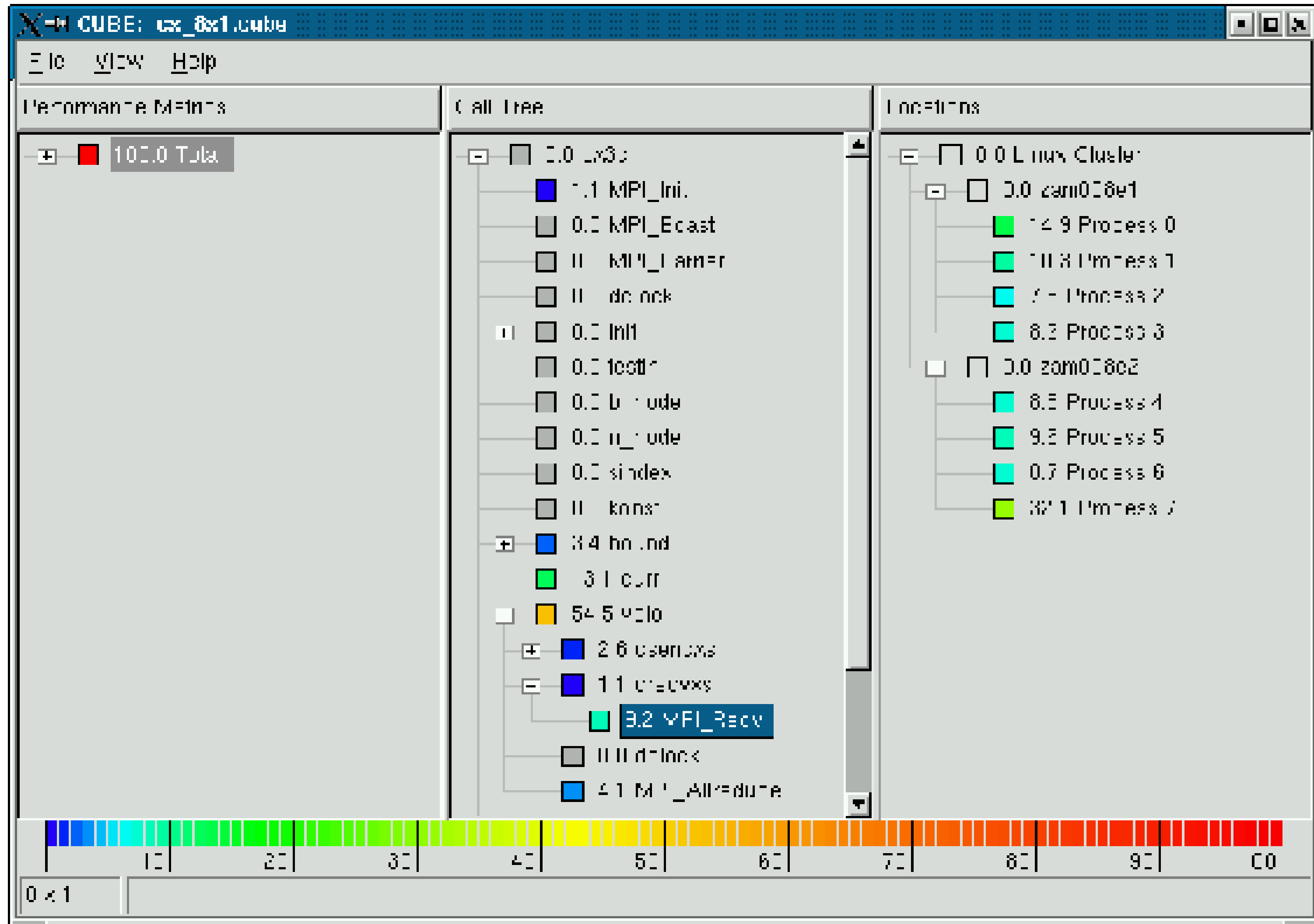


Relative Percentage Mode

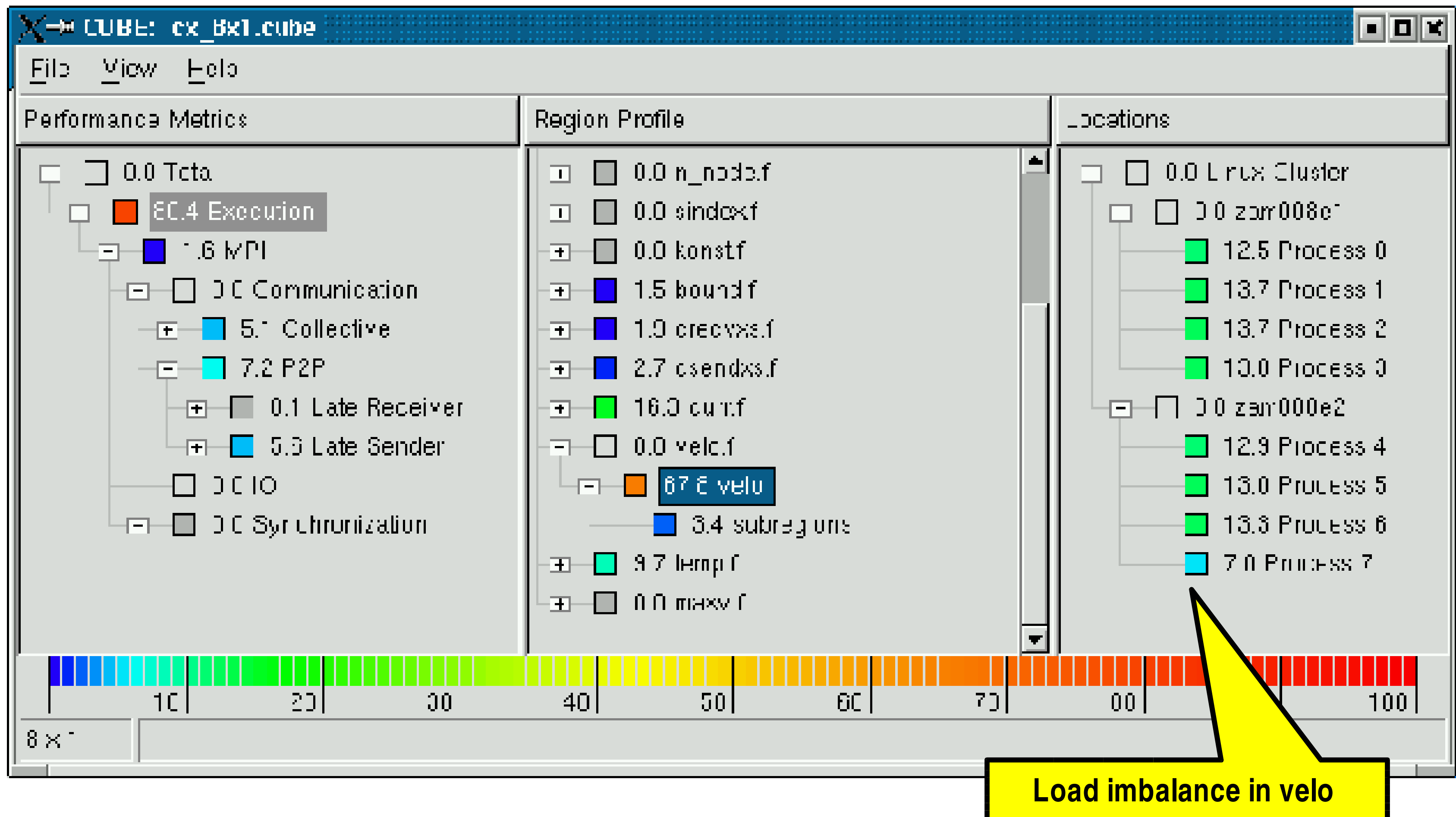
- All values percentages of the selection in the left neighbor tree



Call Tree View

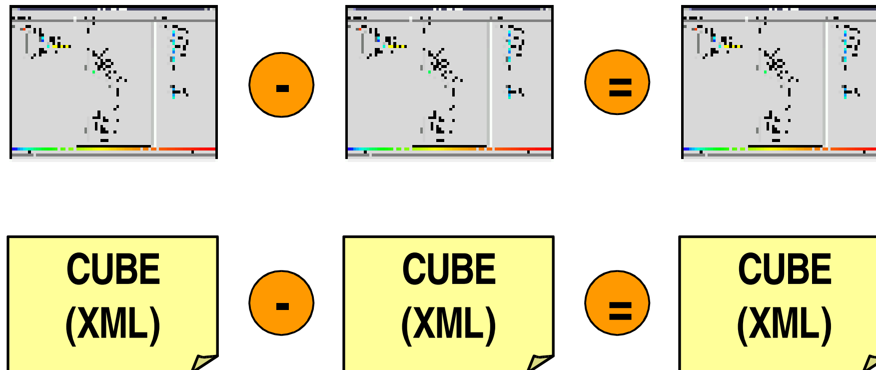


Region Profile View



Performance Algebra (will be released soon)

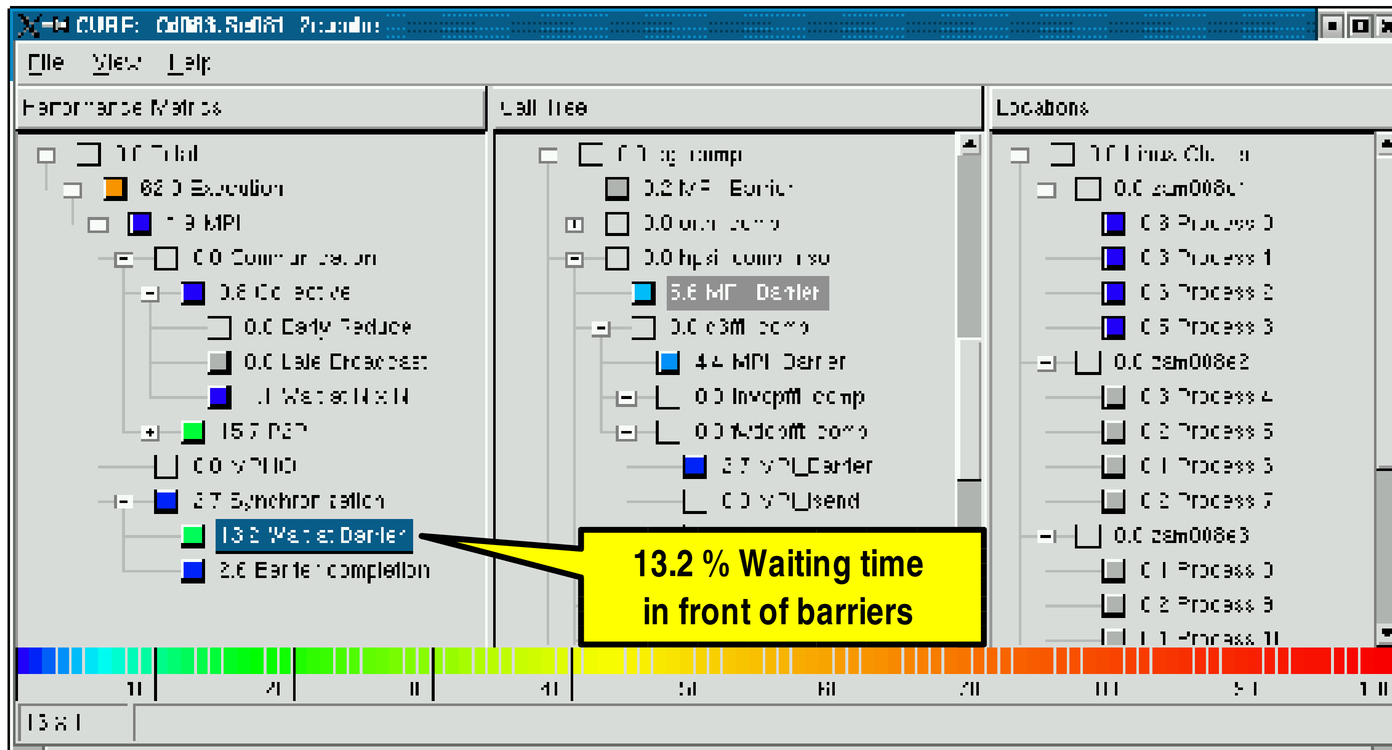
- Cross-experiment analysis
  - Different execution configuration
  - Different measurement tools
  - Different random errors



- Arithmetic operations on CUBE instances
  - Difference, mean, merge
  - Obtain CUBE instance as result
  - Display it like ordinary CUBE instance

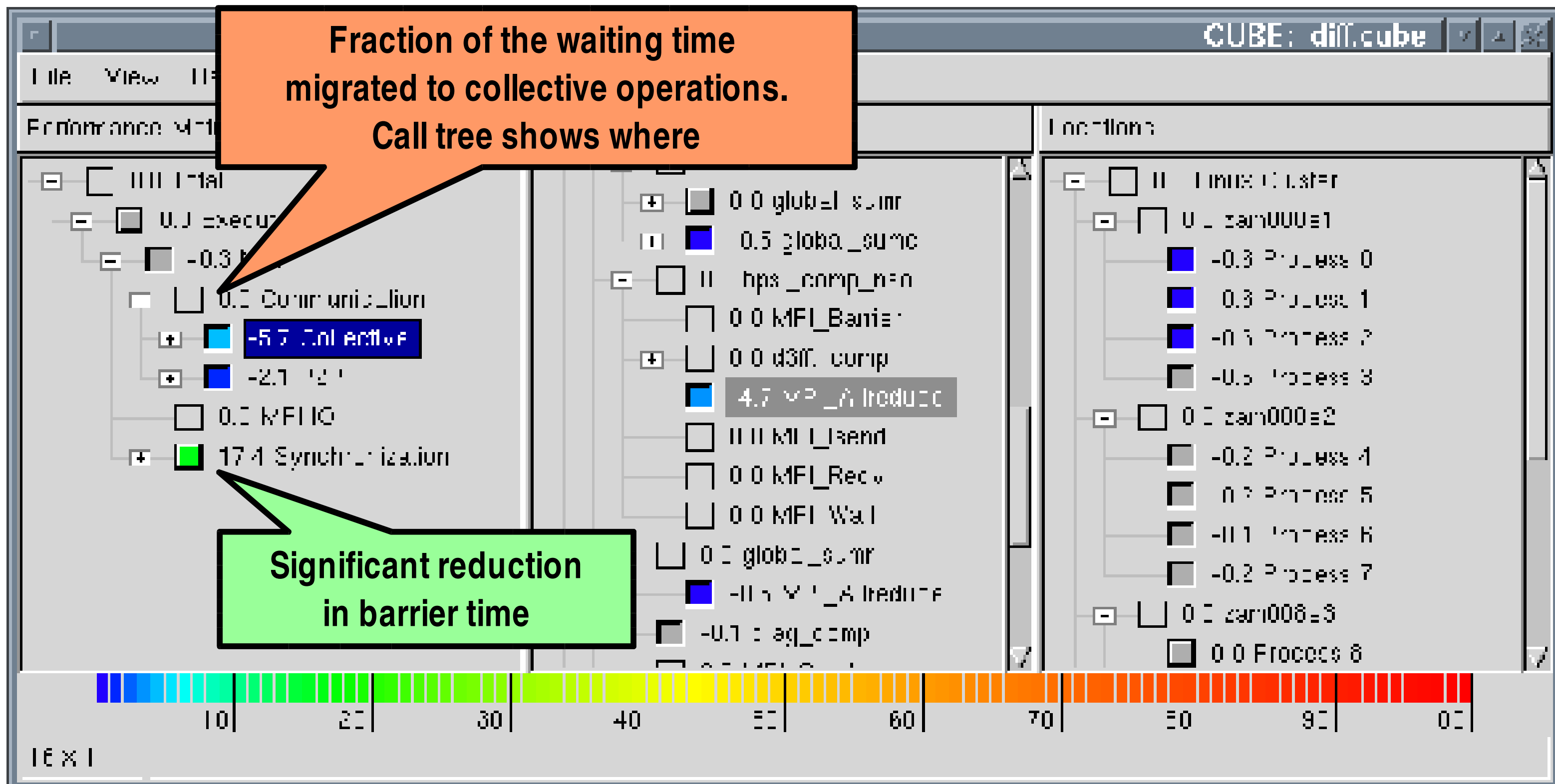
Nano-particle simulation PESCAN

- Application Lawrence Berkeley National Lab
- Numerous barriers to avoid buffer overflow when using large processor counts – not needed for smaller counts



(Re)moving Waiting Times

- Difference between before / after barrier removal
- Raised relief shows improvement
- Sunken relief shows degradation



Installation

- Install wxWidgets http://www.wxwidgets.org
- Install libxml2 http://www.xmlsoft.org
- The following commands should be in your search path
  - xml2-config
  - wx-config
- Install CUBE http://icl.cs.utk.edu/kojak/cube/
  - Two options
    - CUBE library only (analysis without presentation)
    - CUBE library + GUI
  - Follow the installation instructions in the manual
- Install KOJAK http://www.fz-juelich.de/zam/kojak/
  - Follow the installation instructions in ./INSTALL

KOJAK Runtime Parameters

- Global directory to store final trace file
- Local directory to store local files
- Trace file prefix
- Size of the internal event buffer
- Printing of EPILOG related control messages
- Hardware counters to be recorded as part of the trace file
  - EPILOG defines names for most common counters

```
export ELG_METRIC=L1_D_MISS:FLOATING_POINT
```

- Buffer parameters of EXPERT analyzer

Usage

- Run your instrumented application
- Application will generate a trace file a.elg
- Run analyzer and generate CUBE intput file a.cube

```
> expert a.elg
> cube cube.elg&
```

Acknowledgment

- People
  - Bernd Mohr
  - Fengguang Song
- Sponsors
  - DOE
  - Forschungszentrum Jülich

Thank you!

Questions ?