Tools

Jesús Labarta

CEPBA-UPC

Technology Transfer | Research | Training | Mobility of Researchers
User Support | Education | HPC Facilities | Parallel Expertise

Performance tools

- **Objective:**
  - Identify performance problems and help optimize application
- **Phases**
  - Data acquisition
  - Processing
    - ✓ Compaction
    - ✓ Summarization: Statistics
  - Presentation
    - ✓ Textual
    - ✓ Graphical
    - ✓ Esoteric …

Performance tools

- **Data acquisition:**
  - Insert probes into the running program
- **Basic approaches**
  - Statistical profilers
    - ✓ Based on sampling program activity: PC,call stack.
    - ✓ Interrupt driven: correlated or not with program activity
    - ✓ Need to project samples to total program behavior
  - Tracing
    - ✓ Instrument every occurrence of relevant events
    - ✓ Instrumentation library / Dynamic instrumentation

Performance tools

- **Information processing**
  - A lot of samples are obtained
  - Some type of processing may be needed before presenting the information to the user
- **Basic approaches**
  - Inline processing
    - ✓ Data processing is performed during the program run.
    - ✓ A small output (statistics) is generated by the instrumentation tool
  - Off line (post mortem)
    - ✓ All data is emitted to a file
    - ✓ A tool post-processes the data
  - Different levels of intermediate approaches are possible

Performance tools

- **Information processing basic approaches**
  - Inline processing
    - ✓ Perturbation: Processing overhead
    - ✓ Simple, predefined set of statistics
    - ✓ No control of details: need to believe in way of accounting
    - ✓ No information on time distributions, ….
  - Off line (post mortem)
    - ✓ Perturbation: Buffering - I/O overhead
    - ✓ File size and management
    - ✓ All data is there: can the analysis tool extract it? Do we know what to look for?

Performance tools

- **Perturbation:**
  - Probe effect
  - f (granularity, overhead)
    - ✓ Granularity
      - – Program
      - – What to instrument
    - ✓ Overhead
      - – Control flow
      - – Time measurement
      - – Inline processing
      - – Storage to buffer
        - » Written to disk when full

Profiling @ SGI

- **Instrumentation program**
  - Ssrun: sampling every
    - ✓ Application time (user+system)
    - ✓ User time
    - ✓ Number of instructions, cache misses,...
- **Post processing / presentation**
  - Cvperf: graphical
  - Prof: textual

Profiling @ SGI

```
karnak 93% ssrun -usertime ./LUBb
karnak 94% prof LUBb.usertime.m6877966 >  seq.usertime.txt
karnak 95% prof -b LUBb.usertime.m6877966 >  seq.usertime.b.txt


karnak 98% ssrun  -pcsamp ./LUBb
karnak 103% prof -l LUBb.pcsamp.m6437465 >  seq.pcsamp.h.txt


karnak 113% ssrun -exp dc_hwc ./LUBb
```


Profiling @ SGI

```
SpeedShop profile listing generated Mon May 27 12:41:38 2002
   prof LUBb.usertime.m6877966
                    LUBb (n64): Target program
                      usertime: Experiment name
                         ut:cu: Marching orders
                R10000 / R10010: CPU / FPU
                            64: Number of CPUs
                           250: Clock frequency (MHz.)
  Experiment notes--
          From file LUBb.usertime.m6877966:
        Caliper point 0 at target begin, PID 6877966
                        ./ LUBb
        Caliper point 1 at exit(0)
-------------------------------------------------------------------------
Summary of statistical  callstack sampling data (usertime)--
                          154: Total Samples
                            0: Samples with incomplete  traceback
                        4.620: Accumulated Time ( secs.)
                         30.0: Sample interval ( msecs.)
-------------------------------------------------------------------------
```

Statistical significance?

Profiling @ SGI: prof

```
Function list, in descending order by exclusive time
-------------------------------------------------------------------------
[index] excl.secs excl.%  cum.% incl.secs incl.% samples procedure  ( dso: file, line)
    [4]    4.500  97.4%   97.4%    4.500  97.4%     150     bmod (LUBb: LUBb.f, 122)
    [5]    0.060   1.3%   98.7%    0.060   1.3%       2     genmat (LUBb: prepost.f, 1)
    [6]    0.060   1.3%  100.0%    0.060   1.3%       2     bdiv (LUBb: LUBb.f, 85)
    [1]    0.000   0.0%  100.0%    4.620 100.0%     154    __start ( LUBb: crt1text.s, 103)
    [2]    0.000   0.0%  100.0%    4.620 100.0%     154    main ( libftn.so: main.c, 76)
    [3]    0.000   0.0%  100.0%    4.620 100.0%     154     LUBb (LUBb: LUBb.f, 1)
```

Routine

Source and line

Just by this routine

Including functions called by this routine

Profiling @ SGI: prof -b

```
Butterfly function list, in descending order by inclusive time
-------------------------------------------------------------------
        attrib.% attrib.time                        incl.time  caller (callsite) [index]
 [index]   incl.%   incl.time    self%   self-time      procedure [index]
                             attrib.%  attrib.time    incl.time  callee (callsite) [index]
-------------------------------------------------------------------
     [1]   100.0%     4.620     0.0%       0.000        __start [1]
                               100.0%       4.620      4.620  main (@0x10001428;   LUBb: crt1text.s, 177) [2]
-------------------------------------------------------------------
           100.0%     4.620                             4.620  __start (@0x10001428;  LUBb: crt1text.s, 177) [1]
     [2]   100.0%     4.620     0.0%       0.000        main [2]
                               100.0%       4.620      4.620  LUBb (@0x0c45f980;  libftn.so: main.c, 97) [3]
-------------------------------------------------------------------
           100.0%     4.620                             4.620  main (@0x0c45f980;   libftn.so: main.c, 97) [2]
     [3]   100.0%     4.620     0.0%       0.000        LUBb [3]
                                97.4%       4.500      4.500   bmod (@0x10001608; LUBb: LUBb.f, 35) [4]
                                 1.3%       0.060      0.060  bdiv (@0x100015e0; LUBb: LUBb.f, 32) [6]
                                 1.3%       0.060      0.060  genmat (@0x100014d8; LUBb: LUBb.f, 16) [5]
-------------------------------------------------------------------
            97.4%     4.500                             4.620   LUBb (@0x10001608; LUBb: LUBb.f, 35) [3]
     [4]    97.4%     4.500    97.4%       4.500        bmod [4]
-------------------------------------------------------------------
             1.3%     0.060                             4.620   LUBb (@0x100014d8; LUBb: LUBb.f, 16) [3]
     [5]     1.3%     0.060     1.3%       0.060        genmat [5]
-------------------------------------------------------------------
             1.3%     0.060                             4.620   LUBb (@0x100015e0; LUBb: LUBb.f, 32) [3]
     [6]     1.3%     0.060     1.3%       0.060        bdiv [6]
```

Time when called from

calls

Profiling @ SGI: prof -l (pcsamp)

```
Line list, in descending order by function-time and then line number
------------------------------------------------------------------------
          secs      %   cum.%   samples  function ( dso: file, line)

         0.050    1.0     1.0         5   bmod (LUBb: LUBb.f, 122)
         0.070    1.4     2.3         7   bmod (LUBb: LUBb.f, 143)
         0.050    1.0     3.3         5   bmod (LUBb: LUBb.f, 144)
         0.360    7.0    10.3        36   bmod (LUBb: LUBb.f, 145)
         4.330   83.8    94.0       433   bmod (LUBb: LUBb.f, 146)
         0.060    1.2    95.2         6   bmod (LUBb: LUBb.f, 151)

         0.010    0.2    95.4         1   genmat (LUBb: prepost.f, 10)
         0.090    1.7    97.1         9   genmat (LUBb: prepost.f, 12)

         0.010    0.2    97.3         1   bdiv (LUBb: LUBb.f, 104)
         0.050    1.0    98.3         5   bdiv (LUBb: LUBb.f, 106)
         0.040    0.8    99.0         4   bdiv (LUBb: LUBb.f, 108)

         0.040    0.8    99.8         4  fwd ( LUBb: LUBb.f, 183)

         0.010    0.2   100.0         1   LUBb (LUBb: LUBb.f, 35)
```

Profiling @ SGI: prof -l (dc_hwc)

```
Line list, in descending order by function-time and then line number
------------------------------------------------------------------------
        counts     %   cum.%   samples  function ( dso: file, line)

         16424   0.1     0.1         8   bmod (LUBb: LUBb.f, 122)
          2053   0.0     0.1         1   bmod (LUBb: LUBb.f, 143)
          6159   0.0     0.1         3   bmod (LUBb: LUBb.f, 144)
         26689   0.1     0.3        13   bmod (LUBb: LUBb.f, 145)
      17937061  97.3    97.6      8737   bmod (LUBb: LUBb.f, 146)
          8212   0.0    97.6         4   bmod (LUBb: LUBb.f, 151)

         22583   0.1    97.7        11   bdiv (LUBb: LUBb.f, 85)
          4106   0.0    97.8         2   bdiv (LUBb: LUBb.f, 104)
          6159   0.0    97.8         3   bdiv (LUBb: LUBb.f, 105)
         67749   0.4    98.2        33   bdiv (LUBb: LUBb.f, 106)
        117021   0.6    98.8        57   bdiv (LUBb: LUBb.f, 108)

          2053   0.0    98.8         1  fwd ( LUBb: LUBb.f, 160)
          8212   0.0    98.9         4  fwd ( LUBb: LUBb.f, 181)
        188876   1.0    99.9        92  fwd ( LUBb: LUBb.f, 183)

          4106   0.0    99.9         2   genmat (LUBb: prepost.f, 11)
          4106   0.0    99.9         2   genmat (LUBb: prepost.f, 12)

          4106   0.0    99.9         2   LUBb (LUBb: LUBb.f, 34)
          2053   0.0   100.0         1   LUBb (LUBb: LUBb.f, 35)

          2053   0.0   100.0         1  lu0 ( LUBb: LUBb.f, 70)
          4106   0.0   100.0         2  lu0 ( LUBb: LUBb.f, 72)

          2053   0.0   100.0         1   memset (libc.so.1: bzero.s, 160)
```

Higher percentage of L1 misses than time

CEPBA tools: Challenge

What can we say

about an unknown application/system

without looking at the source code

in short time?

Jesus Labarta, SC,2002

CEPBA tools: Thesis

"A **single instrumented run**

captures a **lot of information**

that is essentially **thrown away**

in current parallel programming practice"

Jesus Labarta, SC,2002

CEPBA-Tools

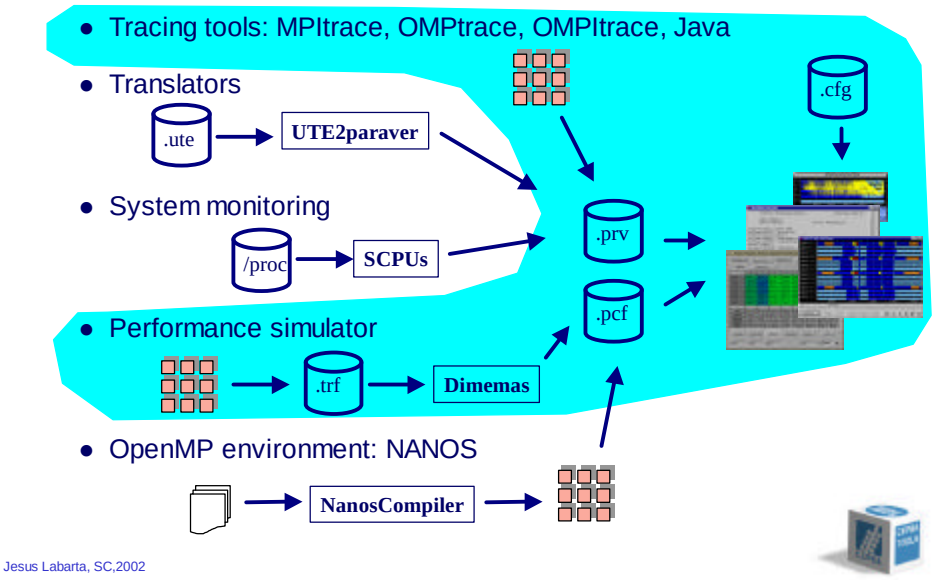
- **Philosophy: Flexibility, insight !!!**
  - Performance analysis ≡ search on a huge and fuzzy space
    - ✓ Be equipped with flexible tools ...
      - – No semantics in the tool
    - ✓ ...supporting quantitative analysis ...
    - ✓ and be ready for surprises
- **Core tools**
  - Paraver
  - Dimemas

. . . available through
http://www.cepba.upc.es/tools

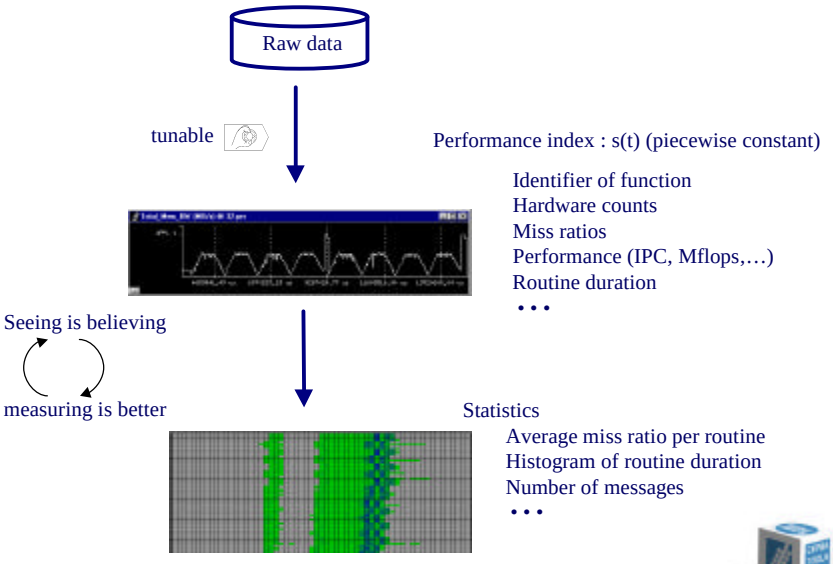
Index

- **CEPBA-Tools**
- **Paraver**
  - Description
  - Instrumentation
  - Understanding applications
- **Dimemas**
  - Description
  - Validation
  - Understanding applications

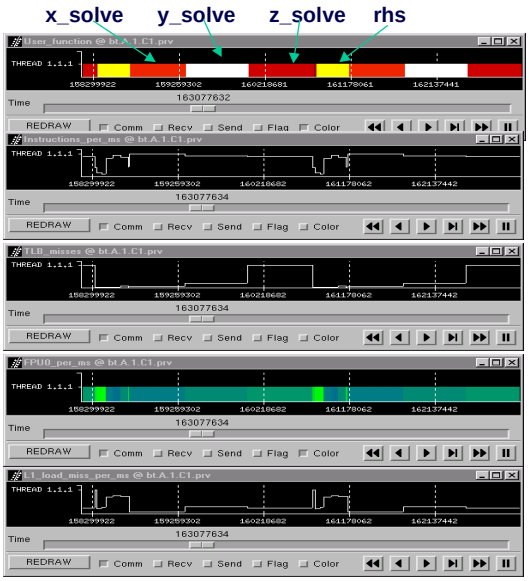
CEPBA-Tools



Paraver: Performance Data browser



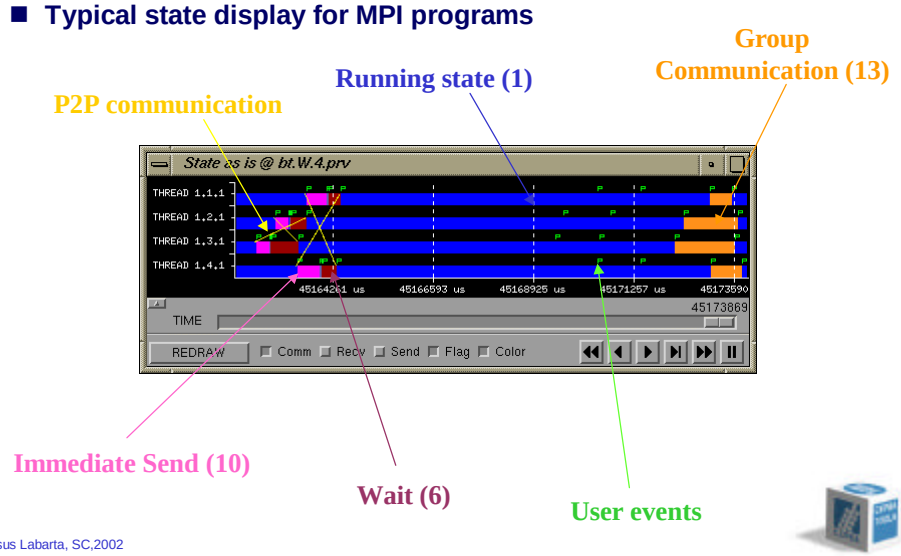
Visualization



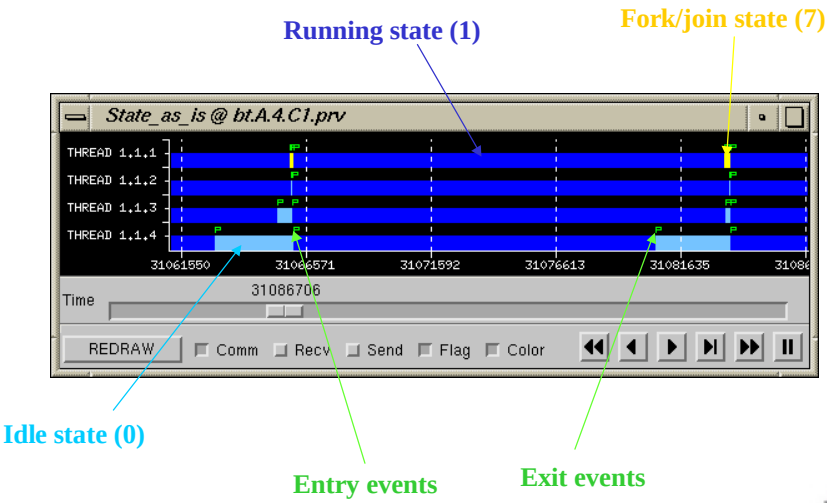
- **Encoding**
  - Discrete colors
  - Function
  - Gradient color
  - Not null gradient color
- **Navigation**
- **Zoom/Undo**
- **Y - scale**
- **Multiple windows**
  - synchronize
- **Time measurement**
- **Multiple traces**

Visualization: MPI

- **Typical state display for MPI programs**

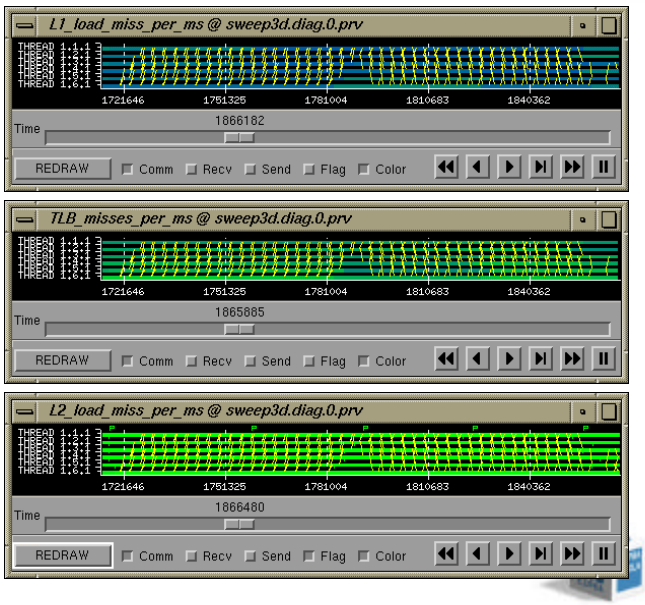


Visualization: OpenMP



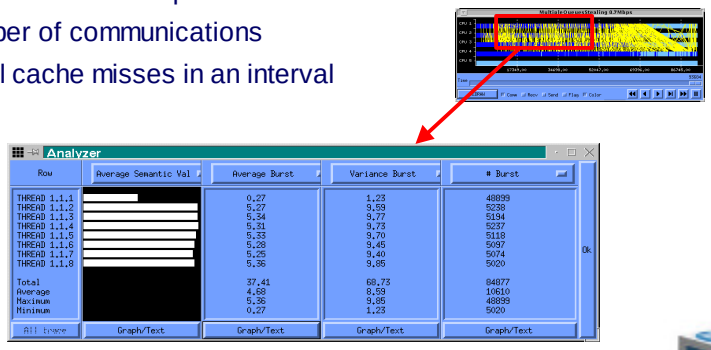
Visualization

- L1 misses/ms
- TLB misses/ms
- L2 misses/ms

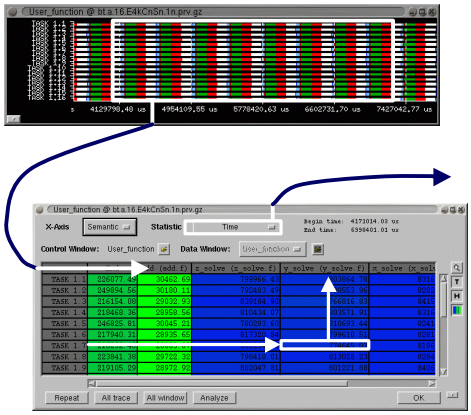


Analysis modules: 1D

- **Example measures**
  - Average processor utilization
  - Average duration/variance of specific function (if within range)
  - Number of calls to specific function
  - Number of communications
  - Total cache misses in an interval
  - ...

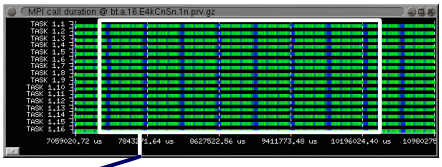


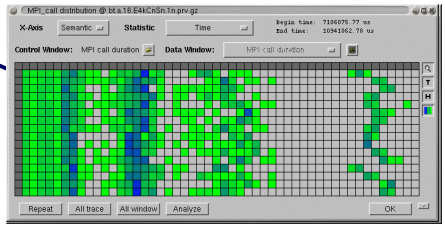
Analysis modules: 2D



- **1 column per control window value**
  - User Function
- **Per thread statistics**
  - # times function is called
  - Per function execution profile
  - average function call duration

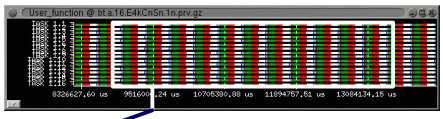
Analysis modules: 2D

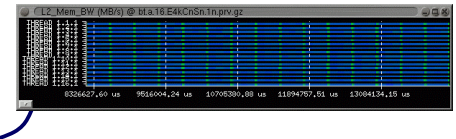


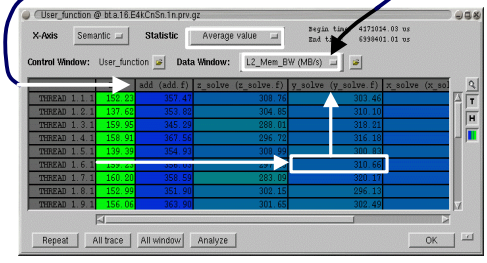


- **1 column per control window value**
  - Range of Durations
- **Per thread statistics**
  - Histogram of durations
  - Cumulative time at each duration

Analysis modules: 2D

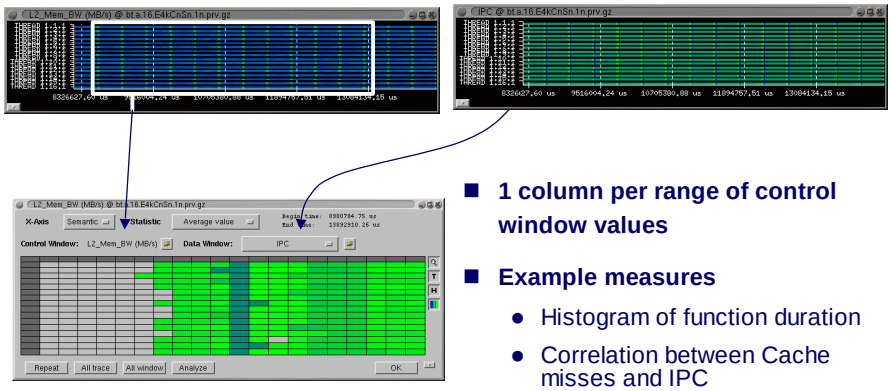






- **Statistics on data window accumulate on column specified by control window**
- **Per thread statistics**
  - Average of data window
  - Integral of data window
  - #data window value changes
  - …

Analysis modules: 2D



- **1 column per range of control window values**
- **Example measures**
  - Histogram of function duration
  - Correlation between Cache misses and IPC
  - Correlation between function duration and hardware counts

Complex?

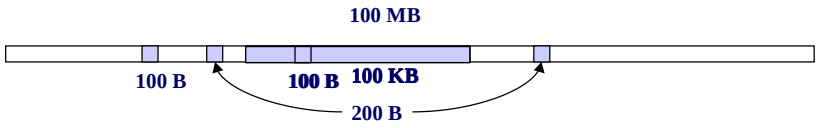
- **Window configuration files: Capture views**
  - Expert knowledge on how to compute performance index
  - Knowledge on “reasonable” values (scales)
  - Specific time and scales to expose behavior
- **What is useful for**
  - Performance analysis by non expert
  - Training
  - Checkpointing of studies
  - Cooperative work
  - Bug reporting

Configuration files: Directory tree

- **OMPItrace**
  - General
    - ✓ State as is, user functions, user function distribution
  - OpenMP
    - ✓ Parallel functions, parallel function distribution,
  - MPI
    - ✓ MPI call profile, MPI call distribution, message size, send BW, …
  - Counters
    - ✓ Program
      - – Memory ops mix, Memory access direction, …
    - ✓ Architecture
      - – L2 miss ratio, …
    - ✓ Performance
      - – IPC, cycles per ms, MIPS, Memory BW, processor BW, …

Methodology

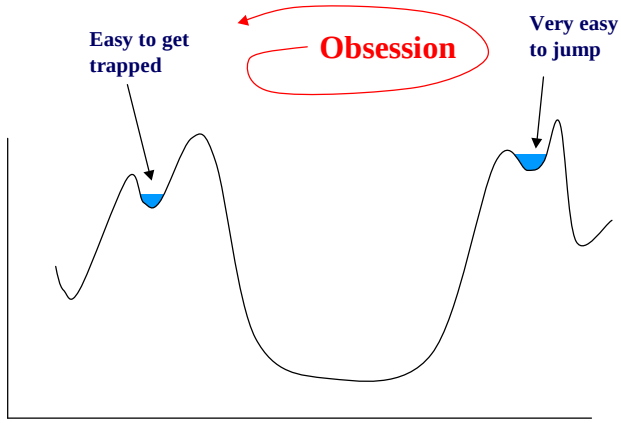
- **Where to look?**
  - Potentially very complex
  - Intricate relations between huge number of factors
  - Microscopic phenomena may have macroscopic effects
  - Delayed effects



- **What to look for?**
- **How?**

Methodology

- The Optimization process



Methodology: a wish

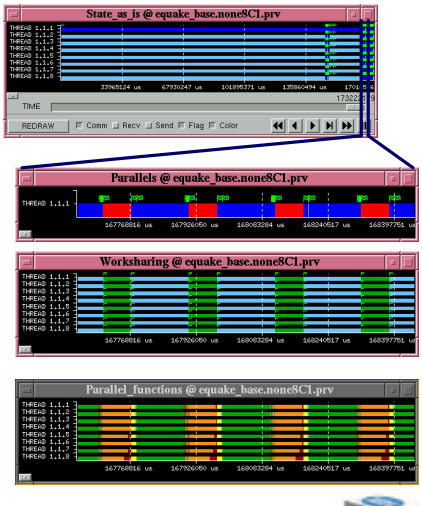
- A set of performance indices to look at
- Expected observations. Good and bad references
- A mechanism to drive the search process based on precious observations
  - Sequence
  - tree
- Conclusion: exit path when performance problem identified
- Reality: still a wish

Methodology and Paraver

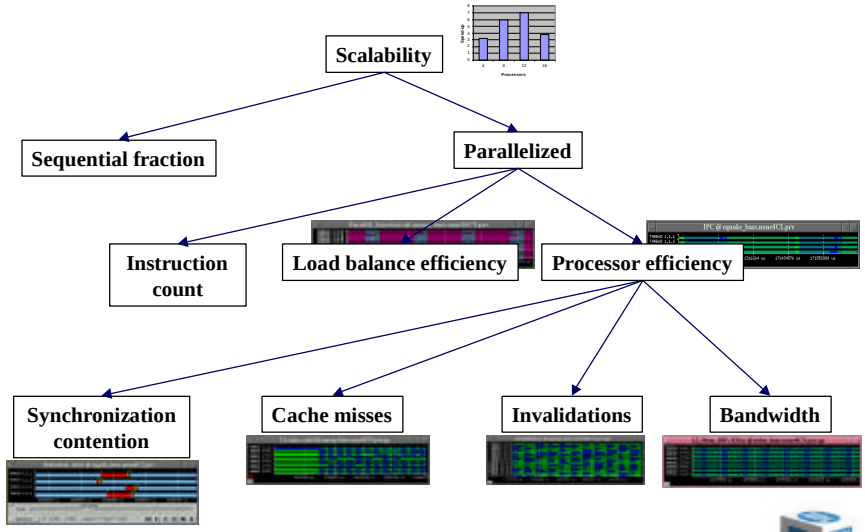
- **Paraver is not a methodology**
- **Paraver and the configuration files support the development of methodologies appropriate for families of applications or environments**

Methodology: Reference

- **What to trace? For how long?**
- **What is the application structure?**
- **Objective**
  - Representative duration
    - ✓ Relevant part: not initialization
    - ✓ Sufficient periods
  - Representative events
    - ✓ Routines
    - ✓ Variables
  - Small trace



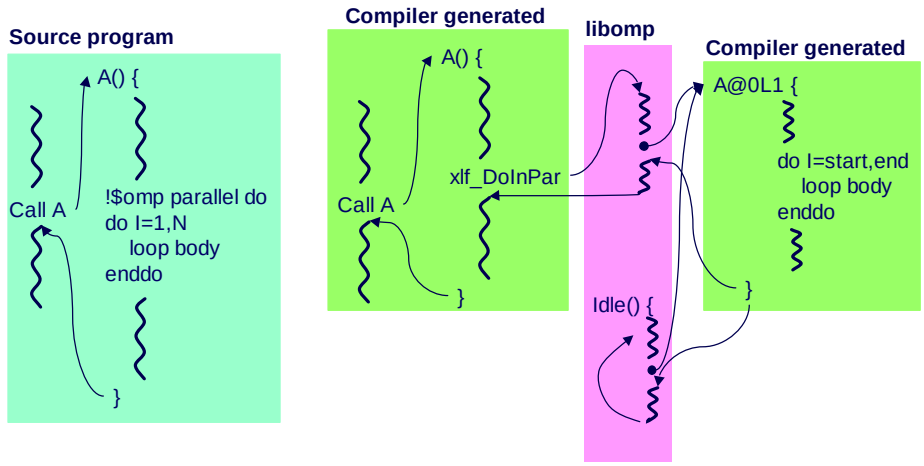
Analysis methodology



Instrumentation

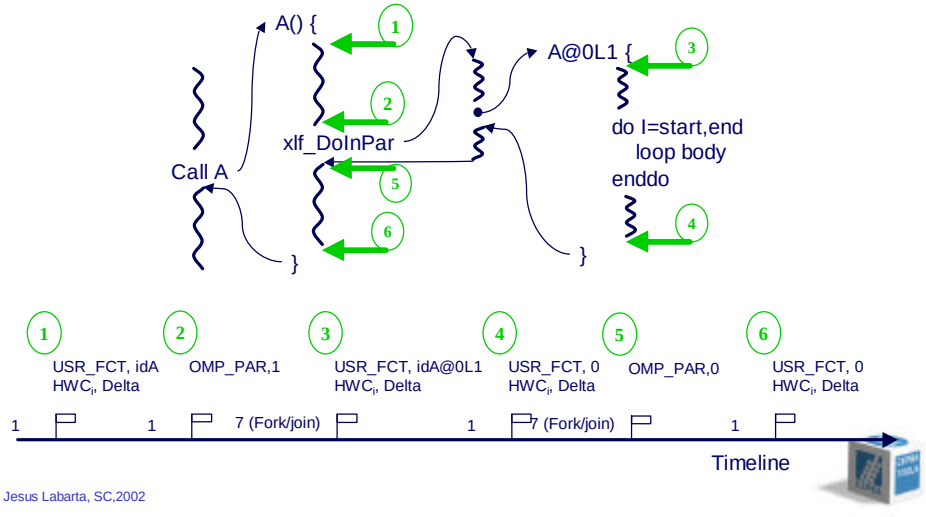
- **Probes**
  - Timing
  - Hardware counters
    - ✓ Portable: PAPI (http://icl.utk.edu/projects/papi)
    - ✓ Vendor specific: PMAPI, …
- **Insertion of probes**
  - Dynamic
    - ✓ DItools (http://personals.ac.upc.es/alberts/fpc.html)
    - ✓ DPCL (http://oss.software.ibm.com/developerworks/opensource/dpcl)
      - – IBM, Linux?
      - – Dyninst based (http://www.dyninst.org/)
  - Static:
    - ✓ PMPI

OpenMP compilation and Run Time



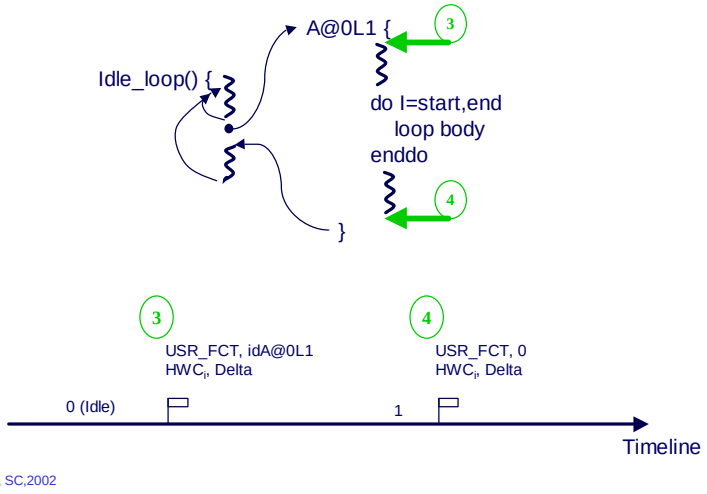
OpenMP instrumentation points

- **Main thread**



OpenMP instrumentation points

- **Slave threads**



Linux Instrumentation : MPI

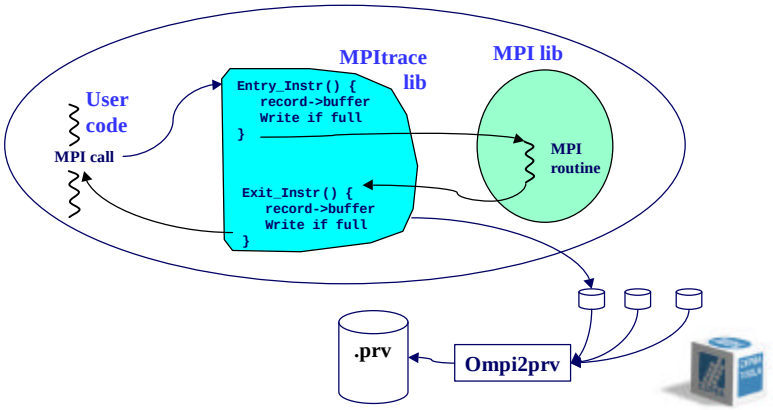
- **Using PMPI interface: Events at entry/exit of MPI calls**
- **PAPI: include hardware counter events**
- **MPItrace API: include user callable routines to**
  - emit events
    - ✓ mpitrace_event(int type, int value)
    - ✓ mpitrace_eventandcounters(int type, int value)
    - ✓ mpitrace_counters()
  - stop/resume tracing
    - ✓ mpitrace_shutdown(), mpitrace_resume()

Linux tracing

- **Use**

  setenv MPTRACE_COUNTERS <list_of_hwc_events>

  mpitrace mpirun -np <nb procs> -machinefile <hosts> myprogram

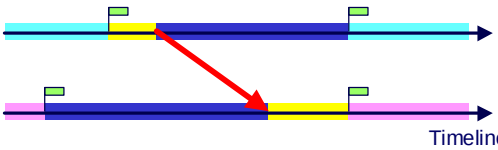


Trace format

Type of record: CPU: thread: Record specific information

Application:task:thread

State	**time_start, time_end, state**
Event	**time, type, value**
Relation	**src, dst, time_src, time_dst, tag, size**



Trace records (.prv)

```
#Paraver (22/02/02 at 11:17):45221692:4:1:4(1:1,1:2,1:3,1:4)
2:1:1:1:1:0:40000001:1
2:1:1:1:1:0:50000001:1
1:1:1:1:1:0:3596:15
1:2:1:2:1:0:141:2
1:3:1:3:1:0:1413:2
1:4:1:4:1:0:4503:2
2:2:1:2:1:141:40000001:1
1:2:1:2:1:141:0:1
...
1:2:1:2:1:83562:84123:10
3:2:1:2:1:83562:84123:1:1:1:1:91028:111409:11520:2000
2:2:1:2:1:84123:50000022:0
1:2:1:2:1:84123:84132:1
2:2:1:2:1:84132:50000022:1
1:2:1:2:1:84132:84309:10
3:2:1:2:1:84132:84309:1:1:1:1:91019:111401:11520:3000
2:2:1:2:1:84309:50000022:0
1:2:1:2:1:84309:84314:1
2:2:1:2:1:84314:50000022:1
1:2:1:2:1:84314:84716:10
3:2:1:2:1:84314:84716:4:1:4:1:109427:113206:11520:4000
2:2:1:2:1:84716:50000022:0
```

Application:process:thread

Type and value

State

Size and tag

Symbolic information (.pcf)

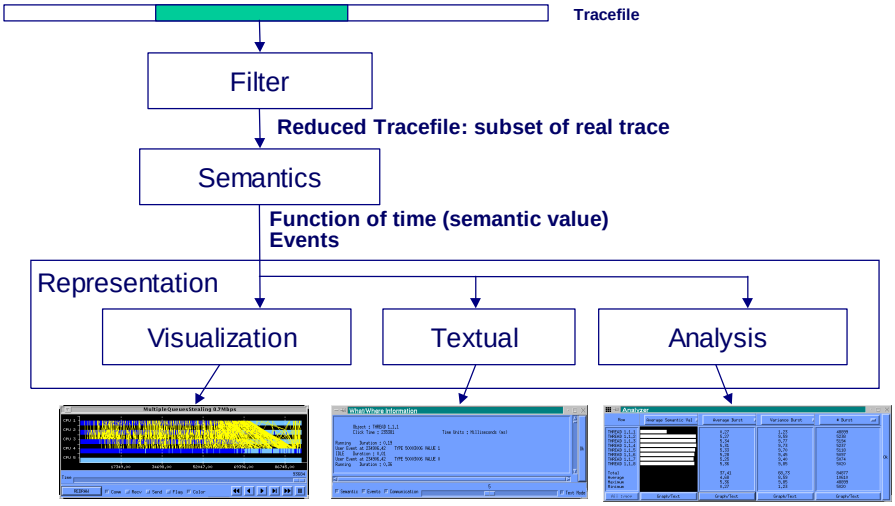
```
STATES
0    Idle
1    Running
2    Not created
3    Waiting a message
4    Blocked
5    Thd. Synchr.

DEFAULT_OPTIONS
LEVEL               THREAD
UNITS               MICROSEC
LOOK_BACK           100
SPEED               1
FLAG_ICONS          ENABLED

DEFAULT_SEMANTIC
THREAD_FUNC          State As Is

EVENT_TYPE
1    50000002    Buffered Send (MPI)
1    50000003    Synchronous Send (MPI)
1    50000004    Barrier (MPI)
1    50000005    Broadcast (MPI)
1    50000018    Send (MPI)
1    50000019    Receive (MPI)
VALUES
1      Begin
0      End
```

```
EVENT_TYPE
1    50000035    AllReduce (MPI)
VALUES
0      End
100      MPI_MAX
101      MPI_MIN
102      MPI_SUM
103      MPI_PROD
104      MPI_LAND
105      MPI_BAND
106      MPI_LOR
107      MPI_BOR
108      MPI_LXOR
109      MPI_BXOR
111      MPI_MAXLOC
110      MPI_MINLOC
```

Overhead

- **Application elapsed time w/o instrumentation**
  - Similar → user happy
  - Very different → the application has a problem
  - Very different → still very useful
    - ✓ I.e.: Hardware counts
- **Learn how to live with it**
  - Don't relax try to extract as much information as possible
- **Overhead of tracing in Linux**
  - No hardware counters: 1-2 µs
  - Hardware counters:
    - ✓ 1 Process: 6-7 µs
    - ✓ 4 Processes SMP: 16-19 µs (PAPI)

Structure



Demand Driven evaluation

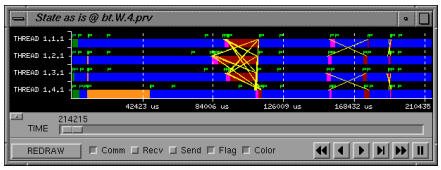
Semantic value

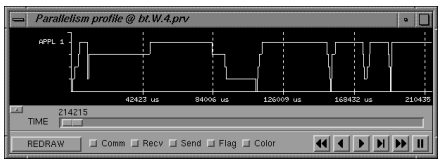
- **S = f(t)**
  - Piecewise constant function of time represented by one window
  - Depends on the filtered subtrace: subset of records of the trace left through by the filter. Each window may see a different subtrace.
  - The semantic value at time t may depend on records with time stamps potentially very far apart from t.

Filter module

- **What : restrict records that pass to the semantic module**
  - Events
    - ✓ by type
    - ✓ by value
  - Communications
    - ✓ by tag
    - ✓ by size
    - ✓ by source / destination
    - ✓ logical / physical
- **What for**
  - Reduce amount of information to display
  - Feed properly the semantic module

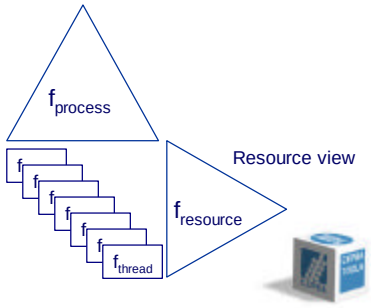
Semantic module





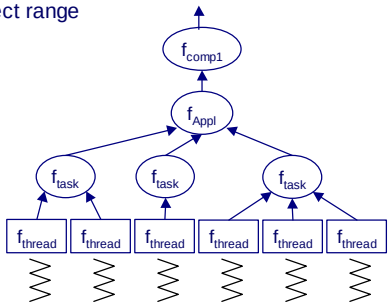
- **Angle:**
  - Process model
    - ✓ Thread, task, application, workload
  - Resource model
    - ✓ CPU, node, system

Process view



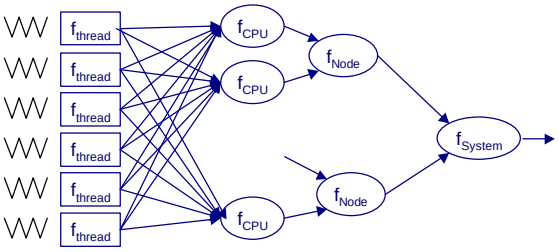
Semantic module: process model view

- Semantic value: f(t)
- $f = f_{comp2} \circ f_{comp1} \circ f_{Application} \circ f_{task} \circ f_{thread}$
- Semantic functions
  - ✓ f_{comp2}, f_{comp1}: sign, mod, div, in range, select range
  - ✓ $f_{Application}$: add, average, max, select
  - ✓ f_{task} : add, average, max, select
  - ✓ f_{thread}: in state, useful, given state,
    last event value,
    next event value,
    average next event value
    interval between events, ...



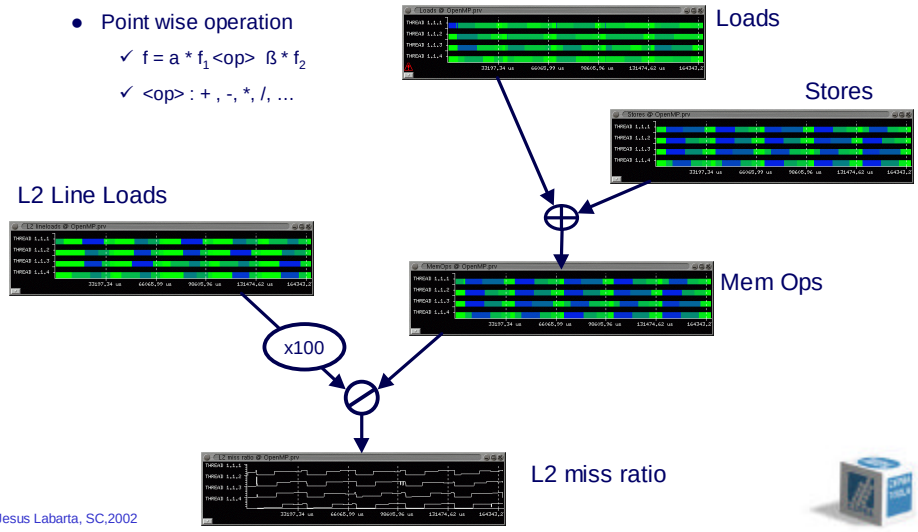
Semantic module: resource view

- $f_{resource} = f_{System} \circ f_{Node} \circ f_{CPU} \circ f_{thread}$
- Semantic functions
  - ✓ f_{System} : add, average, max, select
  - ✓ f_{Node} : add, average, max, select
  - ✓ f_{CPU} : select
  - ✓ f_{thread}: in state, useful, given state, next event value, thread_id



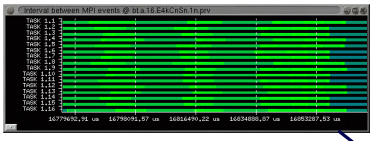
Semantic module

- **Derived windows**
  - Point wise operation
    - ✓ f = a * f_1 <op> ß * f_2
    - ✓ <op> : + , -, *, /, ...

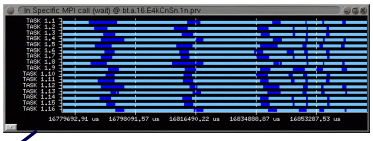


Semantic module

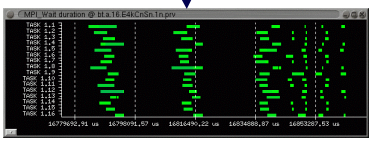
- **Derived windows**
  - Point wise operation
    - ✓ f = a * f_1 <op> ß * f_2
    - ✓ <op> : + , -, *, /, …



Interval between MPI events

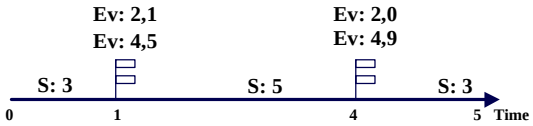


In MPI call

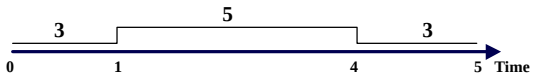


MPI call duration

Semantic module: Examples

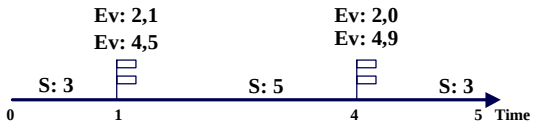


- **Thread function: State as is**



- **Useful for**
  - Global thread activity: computing, idle, fork/join, waiting,…..

Semantic module: Examples

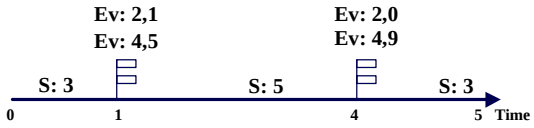


- **Filter: type == 2**

  Thread function: Last event value

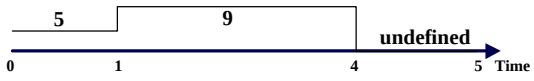
- **Useful for**
  - In parallel region
  - Mutual exclusion
  - Variable values: iteration,….

Semantic module: Examples



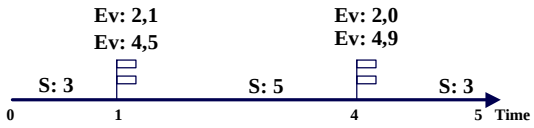
- **Filter: type == 4**

  Thread function: Next event value

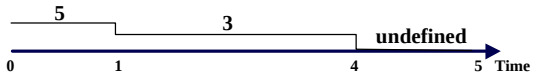


- **Useful for**
  - Hwc events (TLB, L1 misses,...) within interval

Semantic module: Examples



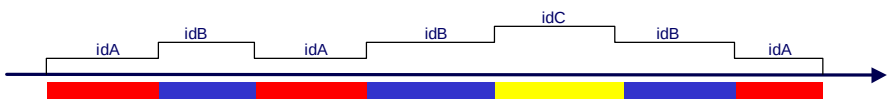
- **Filter: type == 4**
  Thread function: Average next event value



- **Useful for**
  - Hwc events (TLB, L1 misses,...) per time unit within interval

Semantic module: Examples

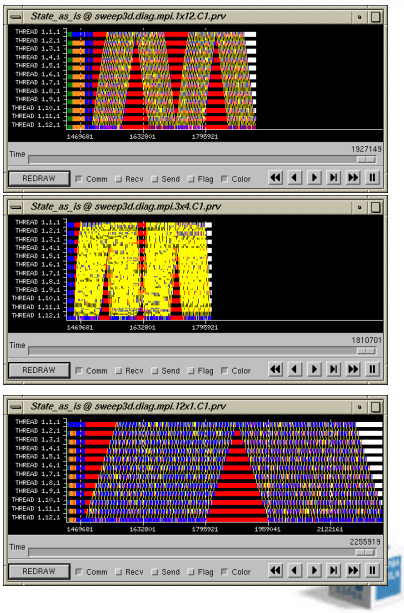
- **Filter: type == USR_FCT**
  Thread function: Last event value
  Compose: Stacked value



- **Useful for**
  - Routine

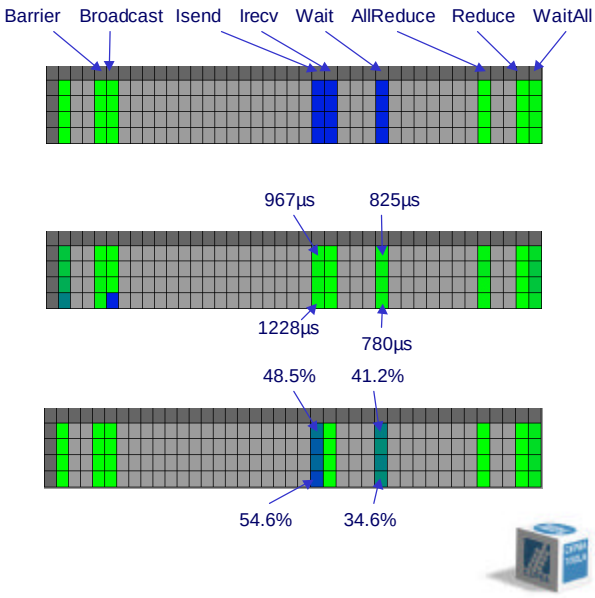
Understanding applications/system

- **Effect of domain partition (sweep3d)**
  - 1 x 12
  - 3 x 4
  - 12 x 1



Understanding applications/system

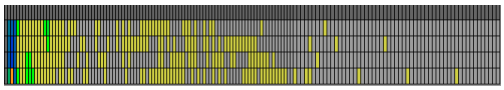
- **MPI calls analysis**
  - # MPI calls
  - Average MPI call duration
  - Percentage of time in MPI call (vs. Total time in MPI calls)



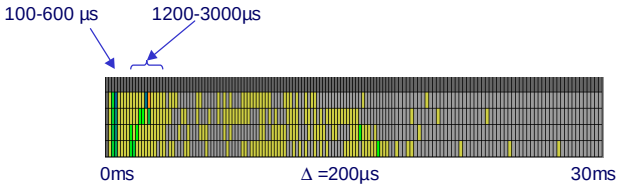
Understanding applications/system

- **Isend duration histogram**
  - % Number of calls

Blue 30%
Green 3%

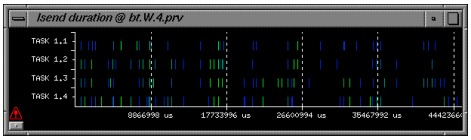


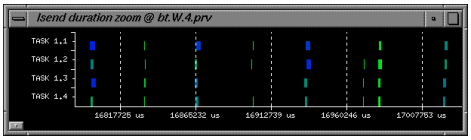
  - Total time

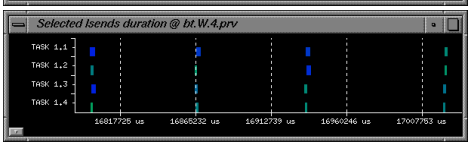


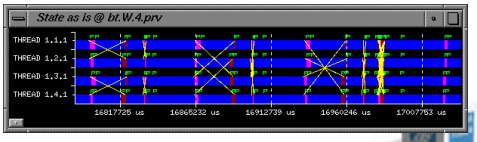
Understanding applications/system

- **Isend analysis**
  - Duration of Isend calls
  - Zoom
  - Select those with 1200µs<duration<3000 µs
  - Correlate to general view
    - ✓ Longer messages

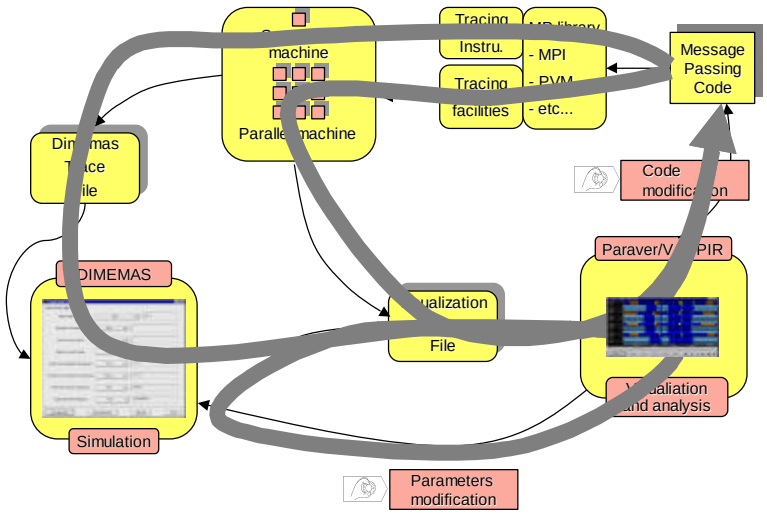






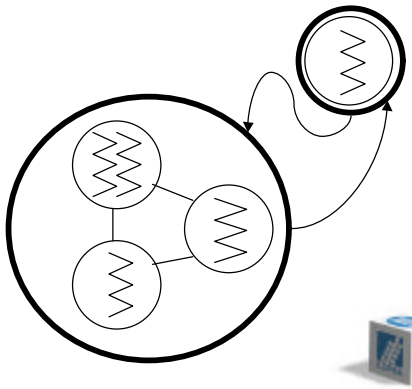


Dimemas



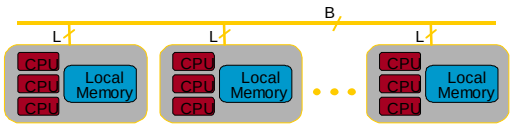
Tracefile

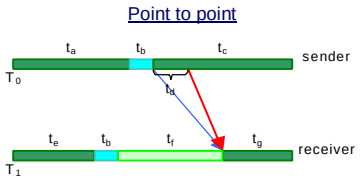
- **Characterises application**
  - Sequence of resource demands for each task
  - Sequence of events: communication
- **Application model**



Simulated Architecture

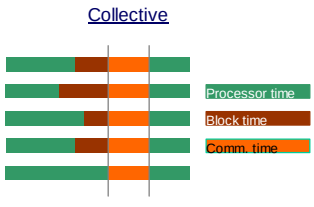
- **"Abstract" architecture**
  - Simple/general
    - ✓ network of SMPs
  - Fast simulation
  - **Key factors influencing performance**
  - Abstract interconnect
    - ✓ Local/remote Latency/BW
    - ✓ Injection mechanism
      - – #links
      - – half/full duplex)
    - ✓ Bisection BW, contention



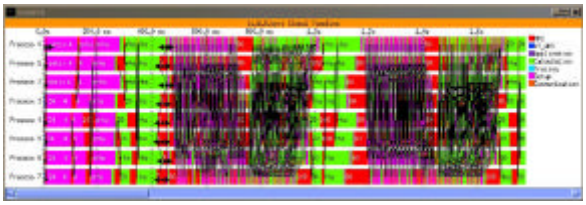


Simulated Architecture

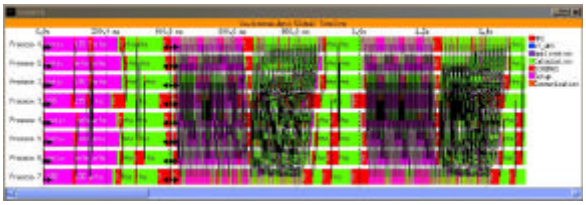
- **Collective communication model**
  - Fan-in / Fan-out
  - Size of message
  - Lin / log / const
  - Barrier



Qualitative validation



Dedicated machine



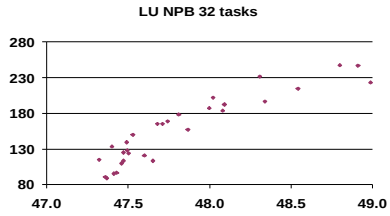
Shared environment

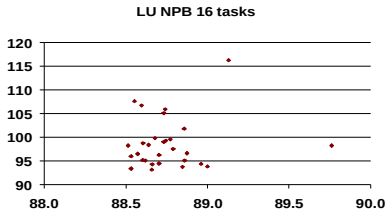
+ Dimemas

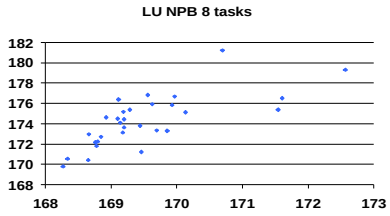
Jesus Labarta, SC,2002

Stability validation

- **On loaded system, 30 times**
  - trace & measure elapsed time
  - predict time for dedicated system



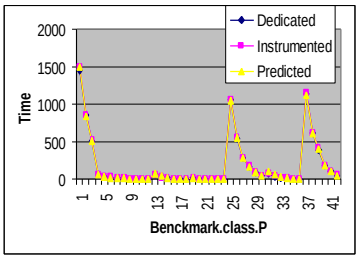




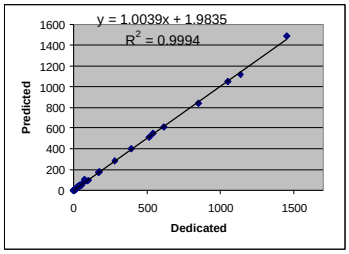
Jesus Labarta, SC,2002

NAS Benchmarks

- BT, CF, FFT, MG, IS LU, SP
- Class W and A
- P = 8..32

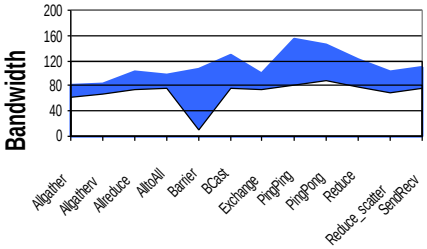


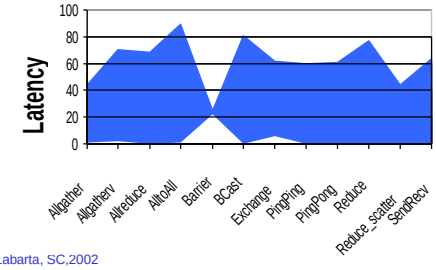
- Target machine model
  - L = 27, BW = 80, B = ∞



Jesus Labarta, SC,2002

System characterization

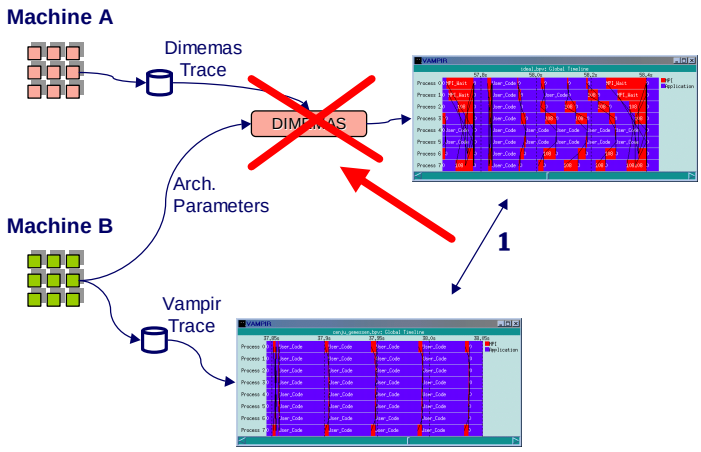




< 10% error regions

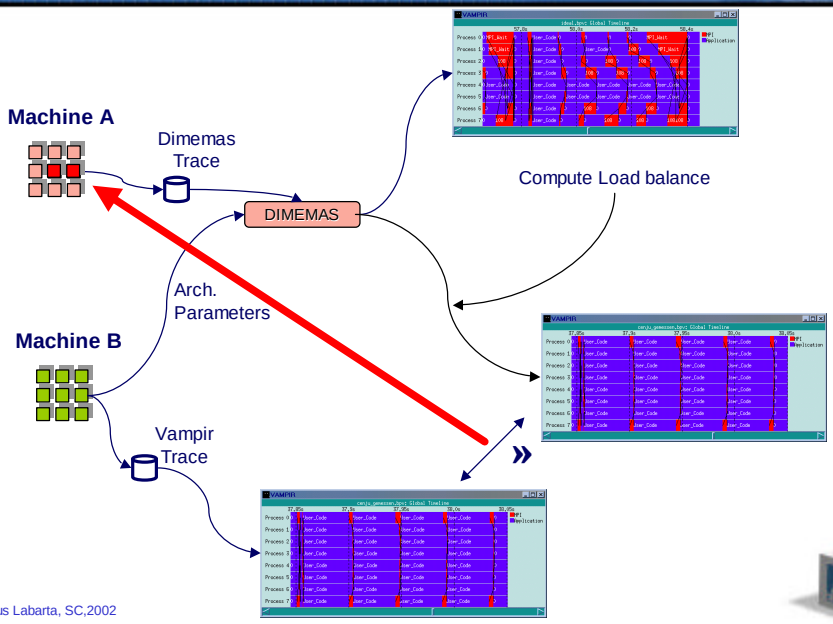
Jesus Labarta, SC,2002

Understanding architectures



Jesus Labarta, SC,2002

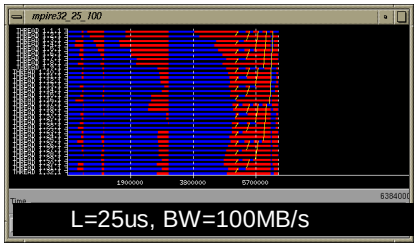
Understanding architectures

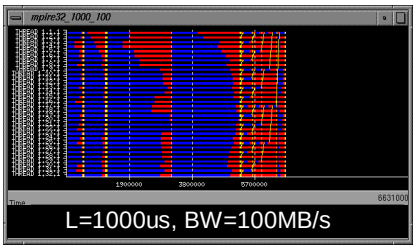


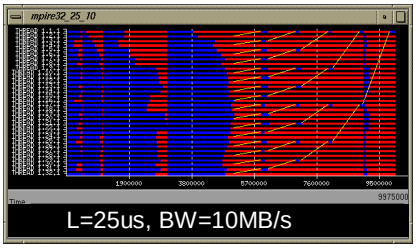
Jesus Labarta, SC,2002

Understanding applications (MPIRE)

- 32 procs (no network contention)





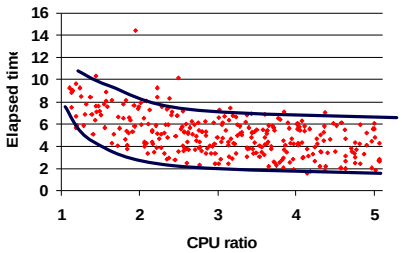


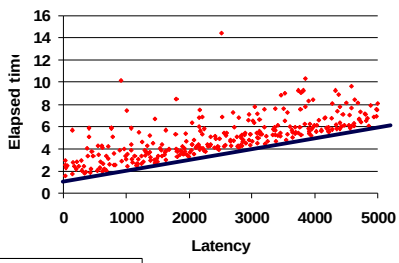
All windows same scale

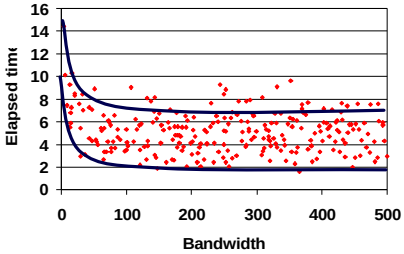
Jesus Labarta, SC,2002

Understanding applications (MPIRE)

- Wide range parametric study for 32 CPUs (no network contention)



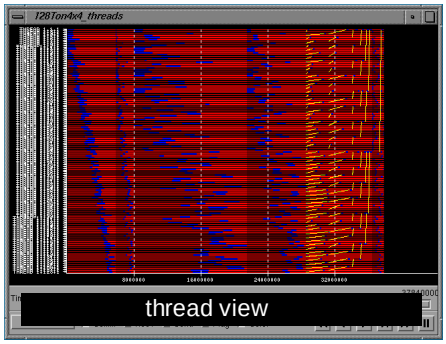


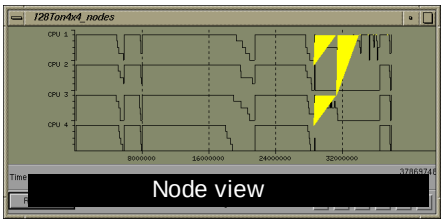


Jesus Labarta, SC,2002

Understanding applications (MPIRE)

- **Cluster of SMPs**
  - 4nodesx4, 1 link
  - 128 p, L=25, BW=100, no network contention





Conclusions

- **Paraver** and **Dimemas** are **powerful tools** to gain **deep insight** and understanding of program performance

- **Learning curve**
  - Slow start
  - Pays off

- **cepbatools@cepba.upc.es**
  - Feedback welcome
  - Support

More information

http://www.cepba.upc.es/tools

cepbatools@cepba.upc.es