Studying the behavior of parallel applications and identifying bottlenecks by using performance analysis tools

G. Markomanolis

INRIA, LIP, ENS de Lyon

Ecole IN2P3 d'informatique "Méthodologie et outils d'optimisation en développement logiciel" 9 February 2012

Outline

1. Context and motivation
2. Introduction to Performance Engineering
3. Performance Application Programming Interface
4. Scalasca
    - Scalasca hands-on
5. TAU
    - TAU hands-on
6. PerfExpert
7. Score-P
8. Talk about accuracy

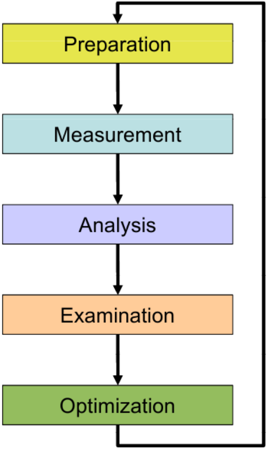
- Goals
  - Overview of the programming tools suite
  - Explain the functionality of the tools
  - Teach how to use the tools
  - Hands-on

Introduction to Performance Engineering

Performance factors of parallel applications

- “Sequential” factors
  - Computation
  - Cache and memory
  - Input/output
- “Parallel” factors
  - Partitioning/decomposition
  - Communication
  - Multithreading
  - Synchronization
  - Parallel I/O
  - Mapping

Performance engineering workflow



- Prepare application
- Collect the relevant data to the execution of the instrumented application
- Identification of performance metrics
- Presentation of results
- Modifications in order to reduce performance problems

Metrics of performance

- How often an event occurs
- The duration of some intervals, e.g. the time spent some communication calls
- The size of the messages during the communication
- Derived metrics

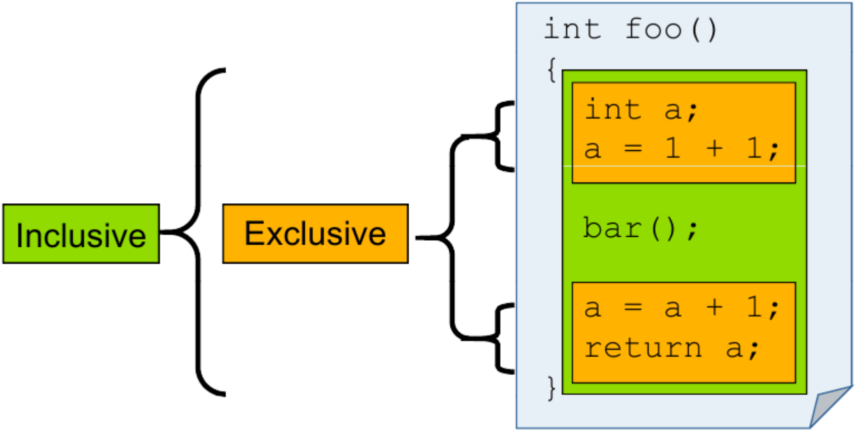
Example

- Execution time
- How many times a function is called
- IPC (Instructions per cycle)
- FLOPS

Execution time

- Wall-clock time
  - Includes waiting time
  - Includes the time consumed by other applications in time-sharing environments
- CPU time
  - Time spent by the CPU for the application
  - No measurement of the context-switched out time
- Use mean or minimum of several runs

Inclusive vs. Exclusive values

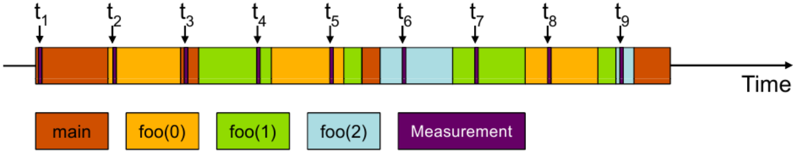


```
int foo()
{
  int a;
  a = 1 + 1;

  bar();

  a = a + 1;
  return a;
}
```

Measurement techniques

- Methods for the measurement
  - Sampling
  - Code instrumentation
- Record the data
  - Profiling
  - Tracing

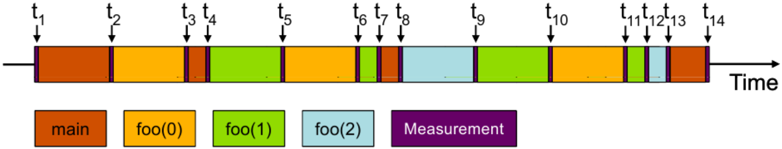
Sampling



```
int main()
{
  int i;

  for (i=0; i < 3; i++)
    foo(i);

  return 0;
}

void foo(int i)
{

  if (i > 0)
    foo(i - 1);

}
```

- Statistical inference of program behaviour
- Not very detailed information
- Only for long-running applications
- Unmodified executables

Instrumentation



```
int main()
{
  int i;
  Enter("main");
  for (i=0; i < 3; i++)
    foo(i);
  Leave("main");
  return 0;
}

void foo(int i)
{
  Enter("foo");
  if (i > 0)
    foo(i - 1);
  Leave("foo");
}
```

- Every event is captured
- Detailed information
- Processing of source-code or executable
- Overhead

Instrumentation techniques

- Static instrumentation
- Dynamic instrumentation
- Code Modification
  - Manually
  - Automatically
    - Preprocessor
    - Compiler
    - Linking against a pre-instrumented library
    - Binary-rewrite

Critical issues

- Accuracy
  - Intrusion overhead
  - Perturbation
  - Accuracy of times & counters
- Granularity
  - Number of measurements?
  - How much information?

Profiling

- Record of aggregated information
  - Total, maximum ...
- For measurements
  - Time
  - Counts
    - Function calls
    - Bytes transferred
    - Hardware counters
  - Functions, call sites
  - Processes, threads

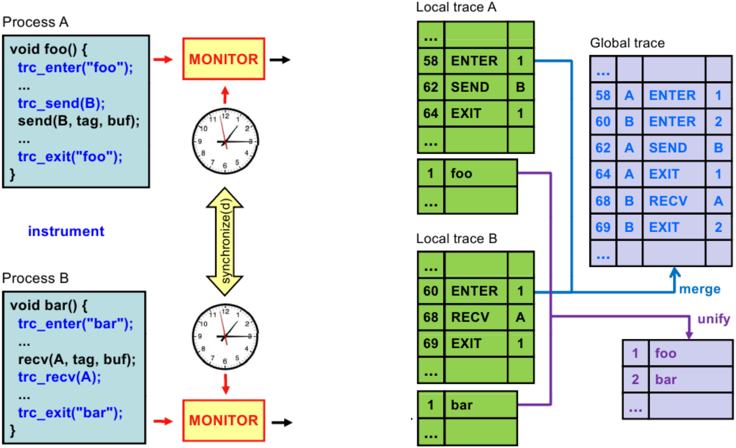
Types of profiles

- Flat profile
  - Metrics per routine for the instrumented region
  - Calling context is not taken into account
- Call-path profile
  - Metrics per executed call path
  - Distinguished by partial calling context
- Special profiles
  - Profile specific events, e.g. MPI calls
  - Comparing processes/threads

Tracing I

- Recording all the events for the demanded code
  - Enter/leave of a region
  - Send/receive a message
- Extra information in event record
  - Timestamp, location, event type
  - Event-related info (e.g.,communicator, sender/receiver)
- Chronologically ordered sequence of event records

Tracing II



Process A

```
void foo() {
  trc_enter("foo");
  ...
  trc_send(B);
  send(B, tag, buf);
  ...
  trc_exit("foo");
}
```

instrument

Process B

```
void bar() {
  trc_enter("bar");
  ...
  recv(A, tag, buf);
  trc_recv(A);
  ...
  trc_exit("bar");
}
```

Local trace A

...		
58	ENTER	1
62	SEND	B
64	EXIT	1
...		

1	foo
...	

Local trace B

...		
60	ENTER	1
68	RECV	A
69	EXIT	1
...		

1	bar
...	

Global trace

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

1	foo
2	bar
...	

Tracing vs. Profiling

- Tracing advantages
  - It is posible to reconstruct the dynamic application behaviour on any required level of abstraction
  - From the tracing it is possible to extract the profiling.
- Disadvantages
  - The traces can get really large especially when using a lot of processes or the applications is constituted by many events
  - Writing events to a file at runtime causes perturbation

Performance analysis procedure

- Performance problem?
  - Time / speedup / scalability measurements
- Key bottleneck?
  - MPI/ OpenMP / Flat profiling
- Where is the key bottleneck?
  - Call-path profiling
- Why?
  - Hardware counter analysis, selective instrumentation for better analysis
- Scalability problems?
  - Load imbalance analysis, compare profiles at various sizes function by function

Performance Application Programming (PAPI)

Middleware that provides a consistent and efficient programming interface for the performance counter hardware found in most major microprocessors
Hardware performance counters can provide insight into:

- Whole program timing
- Cache behaviors
- Branch behaviors
- Memory and resource contention and access patterns
- Pipeline stalls
- Floating point efficiency
- Instructions per cycle
- Subroutine resolution
- Process or thread attribution

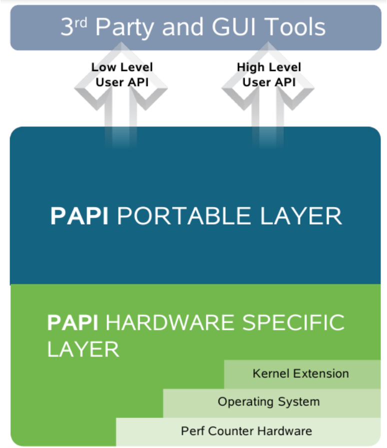
PAPI

- Events
  - Platform-neutral Present Events (e.g., PAPI_TOT_INS)
  - Platform-dependent Native Events (e.g., L3_CACHE_MISS)
- Present Events
  - Standard set of over 100 events for application performance tuning (not all of them available on every processor)
  - No standardization of the exact definition
  - Mapped to either single or linear combinations of native events on each platform
  - The papi_avail provides the available preset events on a given platform
- Native events
  - All the countable events by the CPU
  - Same interface as for preset events
  - The papi_native_avail provides the available native events on a given platform
- It is needed to use the tool papi_event_chooser in order to find out the compatible set of events that can be measured at the same moment

Intel XEON X5675

```
%  papi_avail
Available events and hardware information.
--------------------------------------------------------------------------------
PAPI Version             : 4.2.0.0
Vendor string and code   : GenuineIntel (1)
Model string and code    : Intel(R) Xeon(R) CPU  X5675  @ 3.07GHz (44)
CPU Revision             : 2.000000
CPUID Info               : Family: 6  Model: 44  Stepping: 2
CPU Megahertz            : 3066.216064
CPU Clock Megahertz      : 3066
Hdw Threads per core     : 2
Cores per Socket         : 6
NUMA Nodes               : 2
CPU's per Node           : 12
Total CPU's              : 24
Number Hardware Counters : 7
Max Multiplex Counters   : 64

The following correspond to fields in the PAPI_event_info_t structure.

    Name        Code    Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  Yes   No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
...
API_VEC_SP  0x80000069  Yes   No   Single precision vector/SIMD instructions
PAPI_VEC_DP  0x8000006a  Yes   No   Double precision vector/SIMD instructions
-------------------------------------------------------------------------
Of 107 possible events, 57 are available, of which 14 are derived.
```

PAPI Counter Interfaces



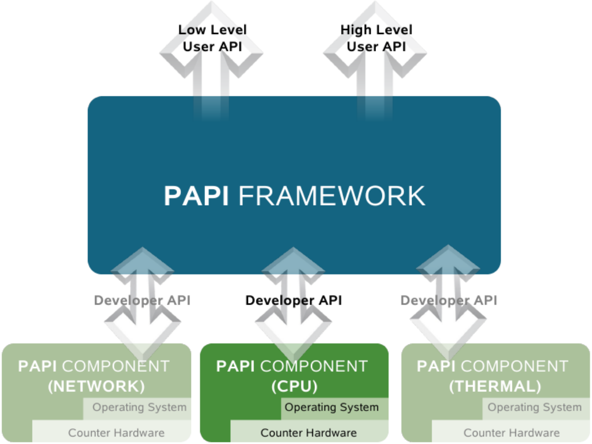
PAPI provides 3 interfaces:

- Low Level API manages hardware events in user defined groups called EventSets
- High Level API provides the ability to start, stop and read the counters for a spacific list of events
- Graphical and and-user tools provide facile data collection and visualization

Component PAPI (PAPI-C)

- Motivation:
  - Hardware counters for network counters, thermal & power measurement
  - Measure multiple counter domains at once
- Goals:
  - Isolate hardware dependent code in a separable component module
  - Add or modify API calls to support access to various components

Component PAPI



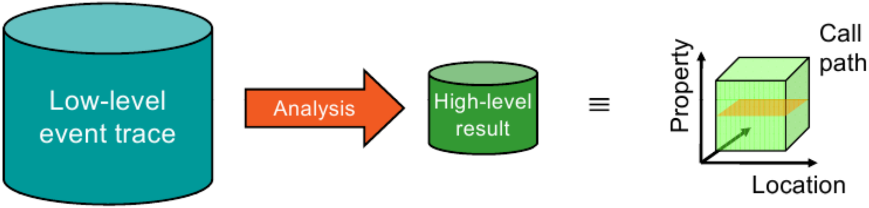
Scalable performance analysis of large-scale parallel applications

Scalasca

Techniques

- Profile analysis:
    - Summary of aggregated metrics
        - per function/call-path and/or per process/thread
    - mpiP, TAU, PerfSuite, Vampir
- Time-line analysis
    - Visual representation of the space/time sequence of events
    - An execution is demanded
- Pattern analysis
    - Search for characteristic event sequences in event traces
    - Manually: Visual time-line analysis
    - Automatically: Scalasca

Automatic trace analysis



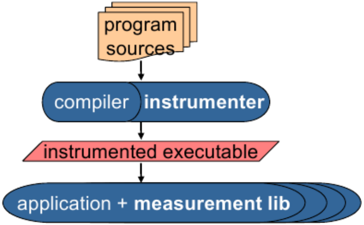
- Trace an application
- Automatic search for patterns on inefficient behaviour
- Classification of behaviour
- Much faster than manual trace analysis
- Scalability

Overview

- Supports parallel programming paradigms & languages
  - MPI, OpenMP. OpenMP/MPI
  - Fortran, C, C++
- Profiling, Tracing, Event trace analysis

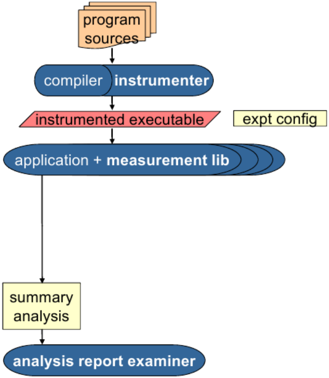
Instrumentation

- Code instrumentation
- Add instrumentation and measurement library into application executable
- MPI standard profiling interface (PMPI) to acquire MPI events



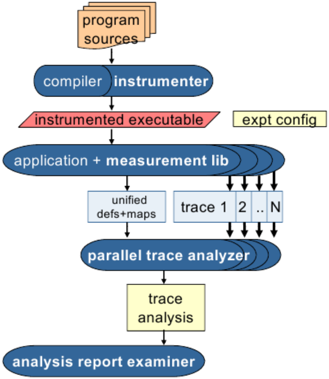
Measurement runtime summarization

- Measurements summarized by thread & call-path during execution
- Analysis report unified
- Presentation of summary analysis



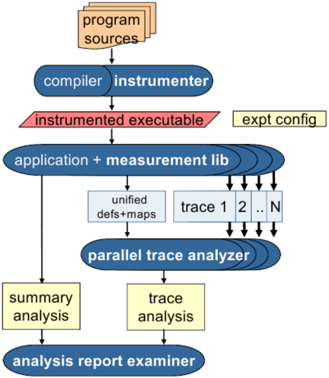
Measurement event tracing & analysis

- Time-stamped events buffered for each thread
- Flushed to files
- Trace analysis
- Presentation of analysis report



Measurement event tracing & analysis

- Scalasca instrumenter (SKIN)
- Scalasca measurement collector & analyzer (SCAN)
- Scalasca analysis report examiner (SQUARE)



EPIK

- Measurement & analysis runtime system
  - Manages runtime configuration and parallel execution
  - Configuration specified by EPIK.CONF (epik_conf)
  - An experiment archive is created (epik_<title>)
  - Optional:
    - Runtime summarization report
    - Tracing
    - Filtering of events
    - Hardware counter measurements
- Experiment archive directory
  - Contains measurement and related files
  - Contains analysis reports

Scalasca actions

Commands

- scalasca -instrument | skin [options] <compile-or-link-command>
- scalasca -analyze | scan [options] <application-launch-command>
- scalasca -examine | square [options] <experiment-archive|report>

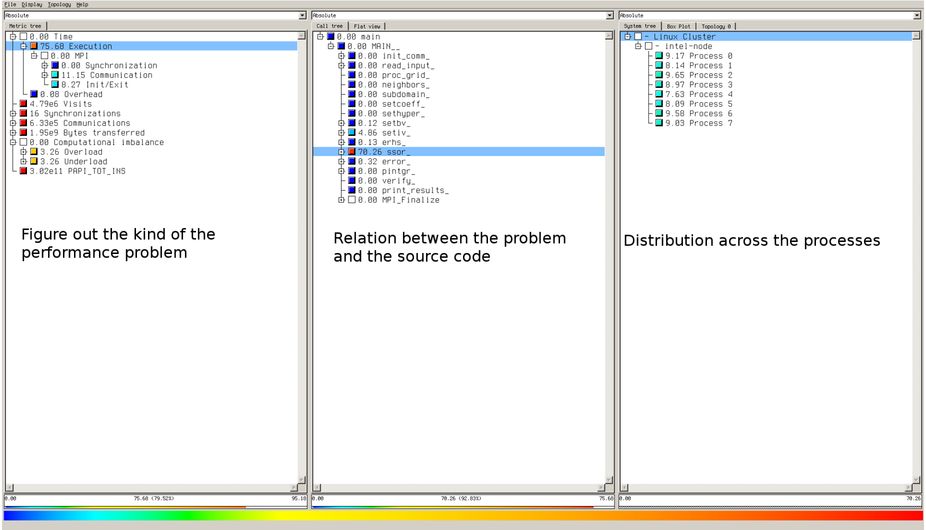
CUBE3

- Parallel program analysis report exploration tools
  - Libraries for XML report
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
- Used by Scalasca, Marmot, ompP, PerfSuite, etc.

CUBE3 Analysis

- Three coupled tree browsers
  - Performance property
  - Call-tree path
  - System location
- CUBE3 displays severities
  - Value for precise comparison
  - Colour for easy identification of hotspots
  - Inclusive value when closed and exclusive value when expanded
  - Customizable through display mode

CUBE3 - summary



```
⊟ ☐ 0.00 Time
  ⊟ ■ 75.68 Execution
    ⊟ ☐ 0.00 MPI
      ⊞ ■ 0.00 Synchronization
      ⊞ ■ 11.15 Communication
      └ ■ 8.27 Init/Exit
    └ ■ 0.08 Overhead
─ ■ 4.79e6 Visits
⊞ ■ 16 Synchronizations
⊞ ■ 6.33e5 Communications
⊞ ■ 1.95e9 Bytes transferred
⊟ ☐ 0.00 Computational imbalance
  ⊞ ■ 3.26 Overload
  ⊞ ■ 3.26 Underload
└ ■ 3.02e11 PAPI_TOT_INS
```

Figure out the kind of the performance problem

```
0.00        75.68 (79.52%)        95.18
```

```
⊟ ■ 0.00 main
  ⊟ ■ 0.00 MAIN__
    ⊞ ■ 0.00 init_comm_
    ⊞ ■ 0.00 read_input_
    ─ ■ 0.00 proc_grid_
    ─ ■ 0.00 neighbors_
    ─ ■ 0.00 subdomain_
    ─ ■ 0.00 setcoeff_
    ─ ■ 0.00 sethyper_
    ⊞ ■ 0.12 setbv_
    ⊞ ■ 4.86 setiv_
    ⊞ ■ 0.13 erhs_
    ⊞ ■ 70.26 ssor_
    ⊞ ■ 0.32 error_
    ⊞ ■ 0.00 pintgr_
    ─ ■ 0.00 verify_
    ─ ■ 0.00 print_results_
    ⊞ ☐ 0.00 MPI_Finalize
```

Relation between the problem and the source code

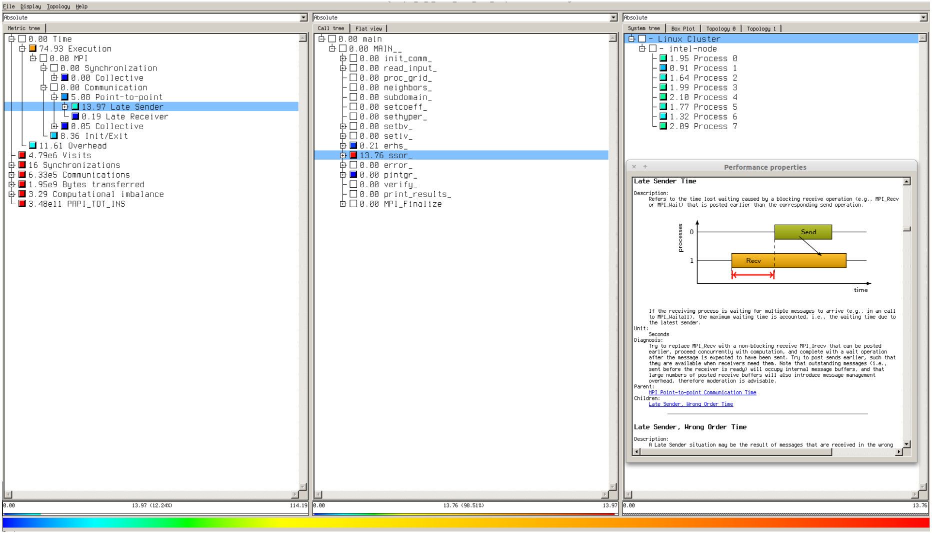
```
0.00        70.26 (92.83%)        75.68
```

```
⊟ ☐ - Linux Cluster
  ⊟ ☐ - intel-node
    ├ ■ 9.17 Process 0
    ├ ■ 8.14 Process 1
    ├ ■ 9.65 Process 2
    ├ ■ 8.97 Process 3
    ├ ■ 7.63 Process 4
    ├ ■ 8.09 Process 5
    ├ ■ 9.58 Process 6
    └ ■ 9.03 Process 7
```

Distribution across the processes

```
0.00                                  70.26
```

CUBE3 - trace



```
0.00 Time
  74.93 Execution
    0.00 MPI
      0.00 Synchronization
        0.00 Collective
      0.00 Communication
        5.08 Point-to-point
          13.97 Late Sender
            0.19 Late Receiver
        0.05 Collective
      8.36 Init/Exit
    11.61 Overhead
  4.79e6 Visits
  16 Synchronizations
  6.33e5 Communications
  1.95e9 Bytes transferred
  3.29 Computational imbalance
  3.48e11 PAPI_TOT_INS
```

```
0.00 main
  0.00 MAIN__
    0.00 init_comm_
    0.00 read_input_
    0.00 proc_grid_
    0.00 neighbors_
    0.00 subdomain_
    0.00 setcoeff_
    0.00 sethyper_
    0.00 setbv_
    0.00 setiv_
    0.21 erhs_
    13.76 ssor_
    0.00 error_
    0.00 pintgr_
    0.00 verify_
    0.00 print_results_
    0.00 MPI_Finalize
```

```
- Linux Cluster
  - intel-node
    1.95 Process 0
    0.91 Process 1
    1.64 Process 2
    1.99 Process 3
    2.10 Process 4
    1.77 Process 5
    1.32 Process 6
    2.09 Process 7
```

Performance properties

Late Sender Time

Description:
Refers to the time lost waiting caused by a blocking receive operation (e.g., MPI_Recv or MPI_Wait) that is posted earlier than the corresponding send operation.

If the receiving process is waiting for multiple messages to arrive (e.g., in an call to MPI_Waitall), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

Unit:
Seconds

Diagnosis:
Try to replace MPI_Recv with a non-blocking receive MPI_Irecv that can be posted earlier, proceed concurrently with computation, and complete with a wait operation after the message is expected to have been sent. Try to post sends earlier, such that they are available when receivers need them. Note that outstanding messages (i.e., sent before the receiver is ready) will occupy internal message buffers, and that large numbers of posted receive buffers will also introduce message management overhead, therefore moderation is advisable.

Parent:
MPI Point-to-point Communication Time

Children:
Late Sender, Wrong Order Time

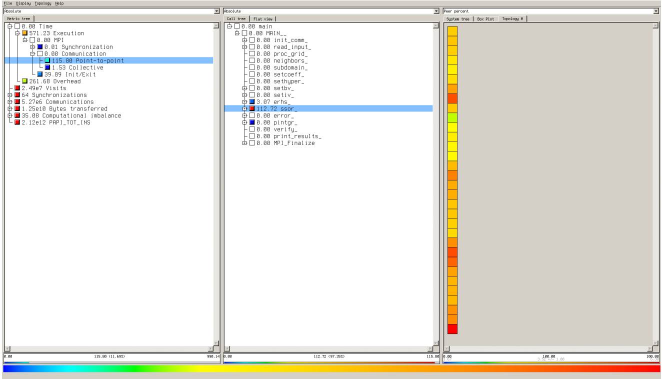
Late Sender, Wrong Order Time

Description:
A Late Sender situation may be the result of messages that are received in the wrong

```
0.00          13.97 (12.24%)          114.19
0.00          13.76 (98.51%)          13.97
0.00                                  13.76
```

Scalasca summary: LU benchmark, class B, (NPB) on AMD1 node

- 11.69% of time spent in MPI point-to-point communication
- 97.35% of which is on program callpath MAIN/SSOR
- With 17.0% std dev over 32 processes



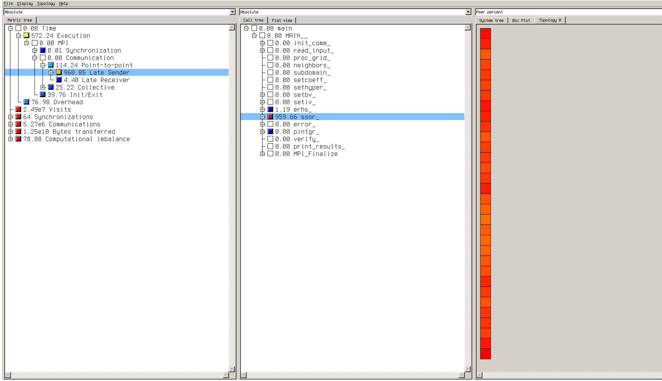
```
0.00 Time
  571.23 Execution
    0.00 MPI
      0.01 Synchronization
      0.00 Communication
        115.80 Point-to-point
        1.53 Collective
      39.89 Init/Exit
  261.68 Overhead
2.49e7 Visits
64 Synchronizations
5.27e6 Communications
1.25e10 Bytes transferred
35.08 Computational imbalance
2.12e12 PAPI_TOT_INS
```

```
0.00 main
  0.00 MAIN__
    0.00 init_comm_
    0.00 read_input_
    0.00 proc_grid_
    0.00 neighbors_
    0.00 subdomain_
    0.00 setcoeff_
    0.00 sethyper_
    0.00 setbv_
    0.00 setiv_
    3.07 erhs_
    112.72 ssor_
    0.00 error_
    0.00 pintgr_
    0.00 verify_
    0.00 print_results_
    0.00 MPI_Finalize
```

```
0.00          115.80 (11.69%)          990.14
0.00          112.72 (97.35%)          115.80
0.00          100.00                   100.00
1.67          3.52 +/- 1.08            6.38
```

Scalasca trace: LU benchmark, class B, (NPB) on AMD1 node

- We can observe that the MPI point-to-point is separated to Late Sender and Late receiver
- Late Sender is the 53.57% of the total execution
- 99.88% of this time is in SSOR



```
0.00 Time
  572.24 Execution
    0.00 MPI
      0.01 Synchronization
      0.00 Communication
        114.24 Point-to-point
          960.85 Late Sender
          4.40 Late Receiver
        25.22 Collective
      39.76 Init/Exit
    76.98 Overhead
  2.49e7 Visits
  64 Synchronizations
  5.27e6 Communications
  1.25e10 Bytes transferred
  78.88 Computational imbalance
```

```
0.00 main
  0.00 MAIN__
    0.00 init_comm_
    0.00 read_input_
    0.00 proc_grid_
    0.00 neighbors_
    0.00 subdomain_
    0.00 setcoeff_
    0.00 sethyper_
    0.00 setbv_
    0.00 setiv_
    1.19 erhs_
    959.66 ssor_
    0.00 error_
    0.00 pintgr_
    0.00 verify_
    0.00 print_results_
    0.00 MPI_Finalize
```

Scalasca 1.4

- Automatic function instrumentation and filtering
  - GCC, IBM, Intel, Pathscale & PGI compilers
  - Optional PDToolkit selective instrumentation
  - Declare which functions to exclude or include for the instrumentation
- MPI measurements & analysis
  - scalable runtime summarization & event tracing
  - Just re-link the application executable
- OpenMP measurement & analysis
  - demanded application source instrumentation
  - thread management
- Hybrid OpenMP/MPI measurement & analysis
  - combined the previous

Scalasca 1.4

- Measurement configuration of MPI events wrappers
  - P2P,COLL,ENV,CG,TOPO, ...
- MPI RMA communication analysis
- Reduced runtime overhead & lowered distortion at scale

Hands-on: NPB-MPI / LU

Scalasca

Performance analysis steps

1. Program instrumentation: skin
2. Summary measurement colelction & analysis: scan [-s]
3. Summary analysis report examination: square
4. Summary experiment scoring: square -s
5. Event trace collection & analysis: scan -t
6. Event trace analysis report examination: square

- Configuration & customization
  - Instrumentation, Measurement, Analysis, Presentation

Connection

- Connect to the nodes with enabled graphics connection

```
% ssh -X username@intelnode
% ssh -X username@amd1node
% ssh -X username@amd2node
```

NPB-MPI suite

- The NAS Parallel Benchmark suite
  - Download from `http://www.nas.nasa.gov/publications/npb.html`
  - 9 benchmarks
  - Configurable for various sizes & size of problems
- Copy the NAS to your home folder

```
% cp -r /srv/app/data/tutorial .
```

Benchmarks, Clusters

NAS Parallel Benchmarks (NPB):

- Mixed case : LU factorization (LU)
  - Instances
    - From 2 to 32 processes
    - Classes A and B
- Compile

```
% make LU NPROCS=<number> CLASS=<class>
```

  - Where <number> is the number of the processes power of two and <class> is the letter of the class, S,W,A,B,C,D or E

NPB -MPI / LU

- Studying the MPI version of the LU benchmark from the NAS Parallel Benchmarks (NPB) suite
- Summary measurement & analysis
  - Automatic instrumentation
  - Summary analysis report examination
  - PAPI hardware counter metrics
- Trace measurement collection & analysis
  - Filter determination, specification & configuration
  - Automatic trace analysis report patterns
- Manual and PDT instrumentation
- Measurement configuration
- Analysis report algebra

Scalasca usage

- Execute scalasca

```
% scalasca
Scalasca 1.4
Toolset for scalable performance analysis of large-scale parallel
applications usage: scalasca [-v][-n] {action}
    1. prepare application objects and executable for measurement:
       scalasca -instrument <compile-or-link-command> # skin
    2. run application under control of measurement system:
       scalasca -analyze <application-launch-command> # scan
    3. interactively explore measurement analysis report:
       scalasca -examine <experiment-archive|report>  # square

   -v: enable verbose commentary
   -n: show actions without taking them
   -h: show quick reference guide (only)
```

NPB instrumentation

- Go to the NAS MPI root path

```
% cd ~/tutorial/NPB3.3-MPI
```

- Add compile/link commands in Makefile (config/make.def)

```
MPIF77=scalasca -instrument mpif77
```

or

```
MPIF77=$(PREP) mpif77
```

- Clean up any previous file

```
% make clean
```

- Compile the LU benchmark for class A and 8 processors

```
% make LU CLASS=A NPROCS=8
```

or

```
% make LU CLASS=A NPROCS=8 PREP=‘‘scalasca -instrument’’
```

LU summary measurement

- Enter the folder with the executables that are instrumented by Scalasca

```
% cd bin.scalasca
```

- Execute the benchmark for 4 processes

```
% scalasca -analyze mpirun -np 8 lu.A.8
S=C=A=N: Scalasca 1.4 runtime summarization
S=C=A=N: ./epik_lu_8_sum experiment archive
S=C=A=N: Wed Jan 25 15:17:17 2012: Collect start
/usr/bin/mpirun -np 8 lu.A.8
[00000]EPIK: Created new measurement archive ./epik_lu_8_sum
[00000]EPIK: Activated ./epik_lu_8_sum [NO TRACE] (0.011s)
[... output ...]
[00000]EPIK: 69 unique paths (64 max paths, 5 max frames,
0 unknowns)
[00000]EPIK: Unifying... done (0.002s)
[00000]EPIK: Collating... done (0.002s)
[00000]EPIK: Closed experiment ./epik_lu_8_sum (0.004s)
maxHeap(*)=20.695/81.918MB
S=C=A=N: Wed Jan 25 15:17:34 2012: Collect done (status=0) 17s
S=C=A=N: ./epik_lu_8_sum complete.
```

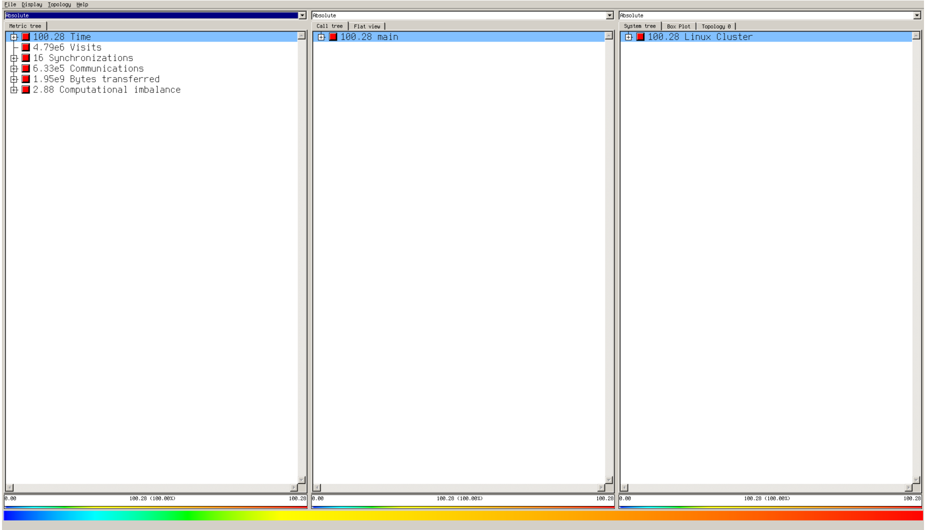
LU summary measurement

- Execute the Scalasca GUI

```
% scalasca -examine epik_lu_8_sum
```

- The measurement archive directory contains
  - a file that contains the execution output (epik.log)
  - the current configuration (epik.conf)
  - the analysis report that was collated after measurement (epitome.cube)
  - the complete analysis report produced during post-processing (summary.cube.gz)

LU summary measurement view

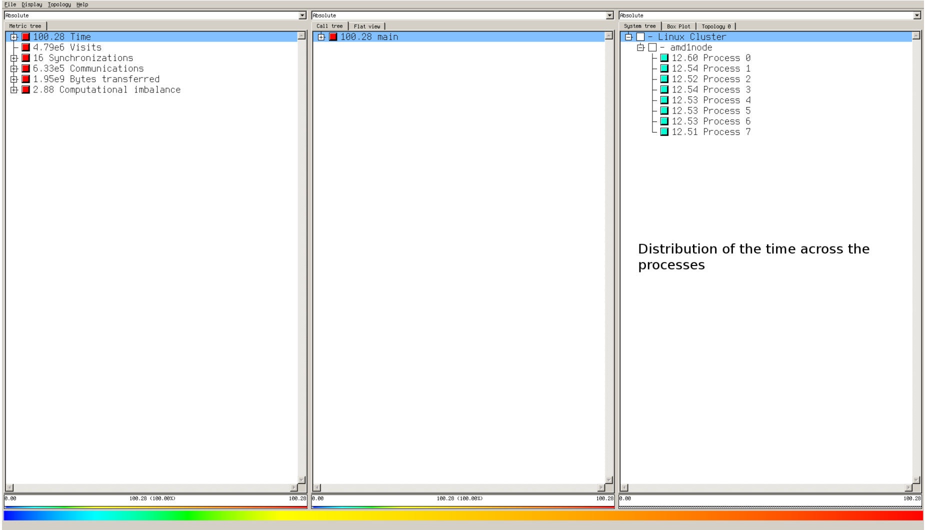


- 🟥 100.28 Time
- 🟥 4.79e6 Visits
- 🟥 16 Synchronizations
- 🟥 6.33e5 Communications
- 🟥 1.95e9 Bytes transferred
- 🟥 2.88 Computational imbalance

- 🟥 100.28 main

- 🟥 100.28 Linux Cluster

LU summary measurement, system tree



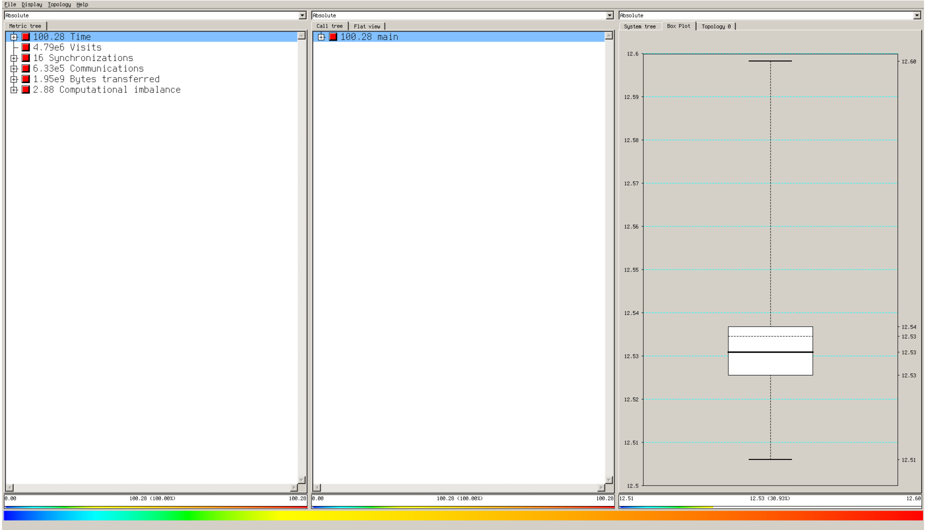
```
⊞ 🟥 100.28 Time
  🟥 4.79e6 Visits
⊞ 🟥 16 Synchronizations
⊞ 🟥 6.33e5 Communications
⊞ 🟥 1.95e9 Bytes transferred
⊞ 🟥 2.88 Computational imbalance
```

```
⊞ 🟥 100.28 main
```

```
⊟ □ - Linux Cluster
  ⊟ □ - amd1node
    ├ 🟦 12.60 Process 0
    ├ 🟦 12.54 Process 1
    ├ 🟦 12.52 Process 2
    ├ 🟦 12.54 Process 3
    ├ 🟦 12.53 Process 4
    ├ 🟦 12.53 Process 5
    ├ 🟦 12.53 Process 6
    └ 🟦 12.51 Process 7
```

Distribution of the time across the processes

LU summary measurement, box plot



100.28 Time
4.79e6 Visits
16 Synchronizations
6.33e5 Communications
1.95e9 Bytes transferred
2.88 Computational imbalance

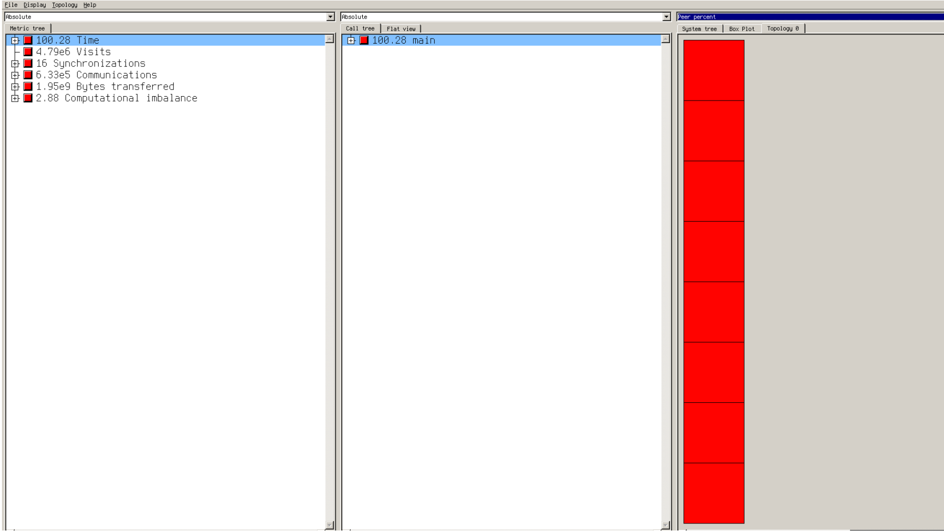
100.28 main

0.00 100.28 (100.00%) 100.28

0.00 100.28 (100.00%) 100.28

12.51 12.53 (30.93%) 12.60

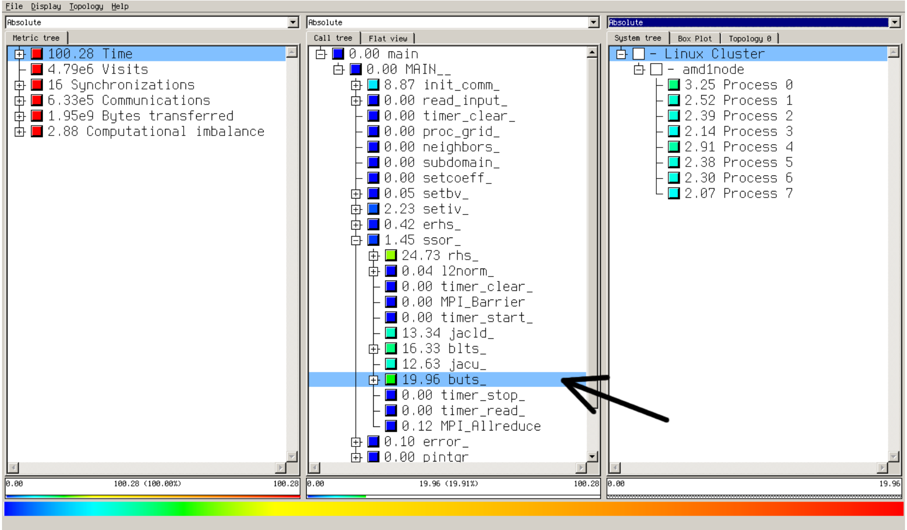
LU summary measurement, topology



- 🟥 100.28 Time
- 🟥 4.79e6 Visits
- 🟥 16 Synchronizations
- 🟥 6.33e5 Communications
- 🟥 1.95e9 Bytes transferred
- 🟥 2.88 Computational imbalance

- 🟥 100.28 main

LU summary measurement, call tree

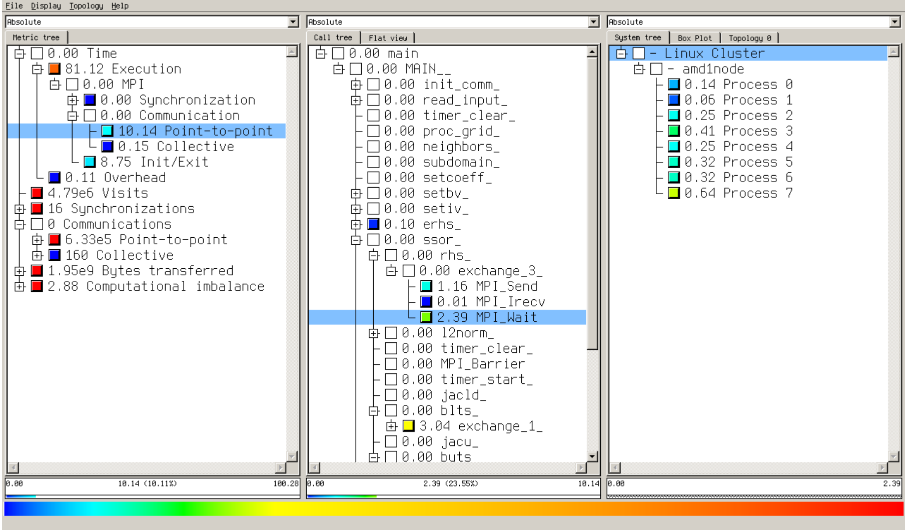


```
Absolute
Metric tree | 
⊞ 100.28 Time
  4.79e6 Visits
⊞ 16 Synchronizations
⊞ 6.33e5 Communications
⊞ 1.95e9 Bytes transferred
⊞ 2.88 Computational imbalance

0.00          100.28 (100.00%)          100.28
```

```
Absolute
Call tree | Flat view
⊟ 0.00 main
  ⊟ 0.00 MAIN__
    ⊞ 8.87 init_comm_
    ⊞ 0.00 read_input_
      0.00 timer_clear_
      0.00 proc_grid_
      0.00 neighbors_
      0.00 subdomain_
      0.00 setcoeff_
    ⊞ 0.05 setbv_
    ⊞ 2.23 setiv_
    ⊞ 0.42 erhs_
    ⊟ 1.45 ssor_
      ⊞ 24.73 rhs_
      ⊞ 0.04 l2norm_
        0.00 timer_clear_
        0.00 MPI_Barrier
        0.00 timer_start_
        13.34 jacld_
      ⊞ 16.33 blts_
        12.63 jacu_
      ⊞ 19.96 buts_
        0.00 timer_stop_
        0.00 timer_read_
        0.12 MPI_Allreduce
    ⊞ 0.10 error_
    ⊞ 0.00 pintgr

0.00          19.96 (19.91%)          100.28
```

```
Absolute
System tree | Box Plot | Topology 0
⊟ □ - Linux Cluster
  ⊟ □ - amd1node
      3.25 Process 0
      2.52 Process 1
      2.39 Process 2
      2.14 Process 3
      2.91 Process 4
      2.38 Process 5
      2.30 Process 6
      2.07 Process 7

0.00                                   19.96
```

LU summary measurement, metric tree



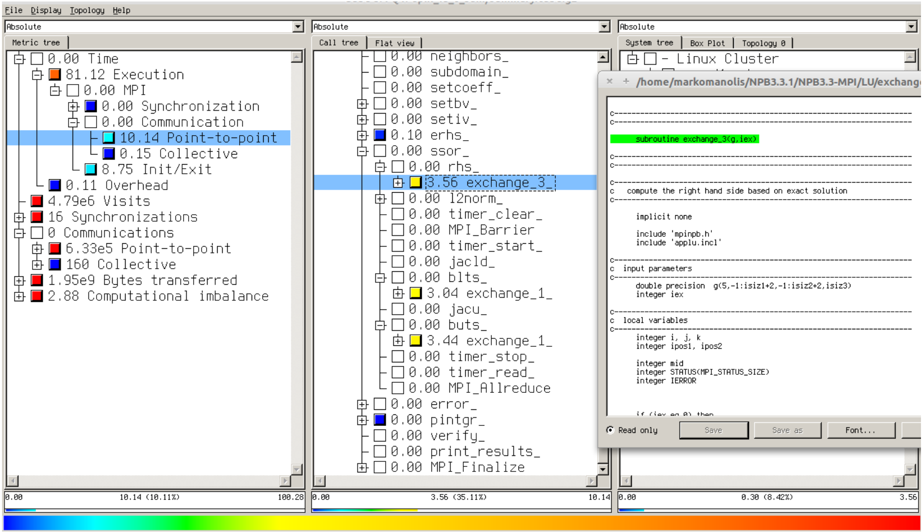
```
0.00 Time
  81.12 Execution
    0.00 MPI
      0.00 Synchronization
      0.00 Communication
        10.14 Point-to-point
        0.15 Collective
      8.75 Init/Exit
    0.11 Overhead
4.79e6 Visits
16 Synchronizations
0 Communications
  6.33e5 Point-to-point
  160 Collective
1.95e9 Bytes transferred
2.88 Computational imbalance
```

```
0.00 main
  0.00 MAIN__
    0.00 init_comm_
    0.00 read_input_
    0.00 timer_clear_
    0.00 proc_grid_
    0.00 neighbors_
    0.00 subdomain_
    0.00 setcoeff_
    0.00 setbv_
    0.00 setiv_
    0.10 erhs_
    0.00 ssor_
      0.00 rhs_
        0.00 exchange_3_
          1.16 MPI_Send
          0.01 MPI_Irecv
          2.39 MPI_Wait
      0.00 l2norm_
      0.00 timer_clear_
      0.00 MPI_Barrier
      0.00 timer_start_
      0.00 jacld_
      0.00 blts_
        3.04 exchange_1_
      0.00 jacu_
      0.00 buts
```

```
- Linux Cluster
  - amd1node
    0.14 Process 0
    0.06 Process 1
    0.25 Process 2
    0.41 Process 3
    0.25 Process 4
    0.32 Process 5
    0.32 Process 6
    0.64 Process 7
```

```
0.00        10.14 (10.11%)        100.28
0.00        2.39 (23.55%)         10.14
0.00                              2.39
```

LU summary measurement, source browser



```
⊟ □ 0.00 Time
  ⊟ ■ 81.12 Execution
    ⊟ □ 0.00 MPI
      ⊞ ■ 0.00 Synchronization
      ⊟ □ 0.00 Communication
        ├ ■ 10.14 Point-to-point
        └ ■ 0.15 Collective
      └ ■ 8.75 Init/Exit
  └ ■ 0.11 Overhead
■ 4.79e6 Visits
⊞ ■ 16 Synchronizations
⊟ □ 0 Communications
  ⊞ ■ 6.33e5 Point-to-point
  ⊞ ■ 160 Collective
⊞ ■ 1.95e9 Bytes transferred
⊞ ■ 2.88 Computational imbalance
```

```
0.00          10.14 (10.11%)          100.28
```

```
  ├ □ 0.00 neighbors_
  ├ □ 0.00 subdomain_
  ├ □ 0.00 setcoeff_
  ⊞ □ 0.00 setbv_
  ⊞ □ 0.00 setiv_
  ⊞ ■ 0.10 erhs_
  ⊟ □ 0.00 ssor_
    ⊟ □ 0.00 rhs_
      ⊞ ■ 3.56 exchange_3_
    ⊞ □ 0.00 l2norm_
    ├ □ 0.00 timer_clear_
    ├ □ 0.00 MPI_Barrier
    ├ □ 0.00 timer_start_
    ├ □ 0.00 jacld_
    ⊟ □ 0.00 blts_
      ⊞ ■ 3.04 exchange_1_
    ├ □ 0.00 jacu_
    ⊟ □ 0.00 buts_
      ⊞ ■ 3.44 exchange_1_
    ├ □ 0.00 timer_stop_
    ├ □ 0.00 timer_read_
    └ □ 0.00 MPI_Allreduce
  ⊞ □ 0.00 error_
  ⊞ ■ 0.00 pintgr_
  ├ □ 0.00 verify_
  ├ □ 0.00 print_results_
  ⊞ □ 0.00 MPI_Finalize
```

```
0.00          3.56 (35.11%)          10.14
```

```
⊟ □ - Linux Cluster
```

/home/markomanolis/NPB3.3.1/NPB3.3-MPI/LU/exchang

```
c---------------------------------------------------------------------
c---------------------------------------------------------------------

      subroutine exchange_3(g,iex)

c---------------------------------------------------------------------
c---------------------------------------------------------------------

c---------------------------------------------------------------------
c   compute the right hand side based on exact solution
c---------------------------------------------------------------------

      implicit none

      include 'mpinpb.h'
      include 'applu.incl'

c---------------------------------------------------------------------
c  input parameters
c---------------------------------------------------------------------
      double precision  g(5,-1:isiz1+2,-1:isiz2+2,isiz3)
      integer iex

c---------------------------------------------------------------------
c  local variables
c---------------------------------------------------------------------
      integer i, j, k
      integer ipos1, ipos2

      integer mid
      integer STATUS(MPI_STATUS_SIZE)
      integer IERROR



      if (iex eq 0) then
```

```
0.00          0.30 (8.42%)          3.56
```

What we did till now

- Instrument an application
- Analyze its execution with a summary measurement
- Examine it with the interactive analysis report explorer GUI
- Time metrics
- Visit counts
- MPI message statistics
- Computational imbalance

LU summary analysis result scoring

```
% scalasca -examine -s epik_lu_8_sum/
/srv/app/scalasca/bin/cube3_score -r ./epik_lu_8_sum/summary.cube.gz >
./epik_lu_8_sum/epik.score
Reading ./epik_lu_8_sum/summary.cube.gz... done.
Estimated aggregate size of event trace (total_tbc): 130098176 bytes
Estimated size of largest process trace (max_tbc):   17275190 bytes
(Hint: When tracing set ELG_BUFFER_SIZE > max_tbc to avoid
intermediate flushes or reduce requirements using file listing
names of USR regions to be filtered.)

INFO: Score report written to ./epik_lu_8_sum/epik.score
```

- The estimated size of the traces will be 130MB
- The maximum trace buffer is around to 17.3MB per process
    - If the available buffer is smaller than 17.3MB, then there will be perturbation because of flushes to the hard disk during the measurement
- Region classification
    - MPI (pure MPI library functions)
    - OMP (pure OMP functions)
    - USR (user-level source local computation)
    - COM(combined USR with OpenMP/MPI)
    - ANY/ALL (aggregate of all region types)

LU summary analysis report

```
% less epik_lu_8_sum/epik.score
 flt  type       max_tbc      time       % region
      ANY      17275190    100.28  100.00 (summary) ALL
      MPI       4574534     19.04   18.99 (summary) MPI
      COM       2259600     52.82   52.67 (summary) COM
      USR      10441032     28.30   28.22 (summary) USR

      USR       9692928      1.37    1.37 exact_
      MPI       2372550      1.84    1.84 MPI_Send
      MPI       2147556      5.85    5.83 MPI_Recv
      COM       1493952      0.46    0.46 exchange_1_
      USR        373488     12.63   12.59 jacu_
      COM        373488     13.06   13.02 blts_
      USR        373488     13.34   13.30 jacld_
      COM        373488     16.29   16.25 buts_
      MPI         35190      0.02    0.01 MPI_Irecv
      MPI         18360      2.43    2.42 MPI_Wait
      COM         12192      0.70    0.70 exchange_3_
      COM          6072     20.48   20.42 rhs_
      USR           768      0.00    0.00 timer_clear_
    ...
      USR            48          0.94    0.94 setiv_
      USR            48          0.00    0.00 timer_start_
      USR            48          0.02    0.02 setbv_
      USR            48          0.00    0.00 timer_stop_
      USR            48          0.00    0.00 timer_read_
      EPK            48          0.11    0.11 TRACING
      MPI            24          0.00    0.00 MPI_Finalize
      COM            24          0.04    0.04 error_
     ...
      USR            24          0.00    0.00 verify_
      USR            24          0.00    0.00 print_results_
      COM            24          0.00    0.00 main
      COM            24          0.32    0.32 erhs_
```

LU summary analysis report

- The estimated size of the traces will be 130MB
- The maximum trace buffer is around to 17.3MB per process
    - If the available buffer is smaller than 17.3MB, then there will be perturbation because of flushes to the hard disk during the measurement

      ```
      export ELG_BUFFER=17300000
      ```

- 28.22% of the total execution is caused by USR regions
    - We should check if there is overhead because of frequently executed small routines
- Solutions:
    - Declare the appropriate buffer
    - Declare a filter file listing (USR) regions in order not to be measured

LU summary analysis report filtering

- We choose the USR regions with small percentage of the execution time and big trace buffer in comparison with the other regions

```
% cat lu.filtering
# filtering for the LU benchmark
exact_
```

- Report scoring with the corresponding filter file

```
% scalasca -examine -s -f lu.filtering ./epik_lu_8_sum/
/srv/app/scalasca/bin/cube3_score -f lu.filtering -r
./epik_lu_8_sum/summary.cube.gz >
./epik_lu_8_sum/epik.score_lu.filtering
Reading ./epik_lu_8_sum/summary.cube.gz... done.
Applying filter "lu.filtering":
Estimated aggregate size of event trace (total_tbc): 54560192 bytes
Estimated size of largest process trace (max_tbc):    7582262 bytes
(Hint: When tracing set ELG_BUFFER_SIZE > max_tbc to avoid
intermediate flushes.)

INFO: Score report written to
./epik_lu_8_sum/epik.score_lu.filtering
```

- Now the estimated size of the traces is 54.6MB, decreased by 58% in comparision with the non filtering approach
- The maximum trace buffer is 7.6MB

LU summary analysis report with filtering

```
% less epik_lu_8_sum/epik.score_lu.filtering
flt  type       max_tbc          time        % region
 -    ANY      17275190        100.28   100.00 (summary) ALL
 -    MPI       4574534         19.04    18.99 (summary) MPI
 -    COM       2259600         52.82    52.67 (summary) COM
 -    USR      10441032         28.30    28.22 (summary) USR

 +    FLT       9692928          1.37     1.37 (summary) FLT
 *    ANY       7582262         98.91    98.63 (summary) ALL-FLT
 -    MPI       4574534         19.04    18.99 (summary) MPI-FLT
 *    COM       2259600         52.82    52.67 (summary) COM-FLT
 *    USR        748152         26.93    26.85 (summary) USR-FLT

 +    USR       9692928          1.37     1.37 exact_
 -    MPI       2372550          1.84     1.84 MPI_Send
 -    MPI       2147556          5.85     5.83 MPI_Recv
 -    COM       1493952          0.46     0.46 exchange_1_
 -    USR        373488         12.63    12.59 jacu_
 -    COM        373488         13.06    13.02 blts_
 ...
```

- The mark + indicates the filtered routines

LU filtered summary measurement

- Save the previous measurement

```
% mv epik_lu_8_sum epik_lu_8_sum_no_filter
```

- Enable the filtering and the corresponding file

```
% export EPK_FILTER=lu.filtering
```

- Execute the LU benchmark for class A and 8 processes

```
% scalasca -analyze mpirun -np 8 lu.A.8
S=C=A=N: Scalasca 1.4 runtime summarization
S=C=A=N: ./epik_lu_8_sum experiment archive
S=C=A=N: Tue Apr 12 00:06:01 2011: Collect start
/srv/app/openmpi/bin//mpirun -np 8 lu.A.8
[00000]EPIK: Created new measurement archive ./epik_lu_8_sum
[00000]EPIK: EPK_FILTER "lu.filtering" filtered 1 of 222 functions
[00000]EPIK: Activated ./epik_lu_8_sum [NO TRACE] (0.009s)
[... output ...]
[00000]EPIK: 66 unique paths (61 max paths, 5 max frames, 0 unknowns)
[00000]EPIK: Unifying... done (0.005s)
[00000]EPIK: Collating... done (0.005s)
[00000]EPIK: Closed experiment ./epik_lu_8_sum (0.011s)
maxHeap(*)=15.469/121.660MB
S=C=A=N: Tue Apr 12 00:06:14 2011: Collect done (status=0) 13s
S=C=A=N: ./epik_lu_8_sum complete.
```

LU filtered summary report

- Examine the scoring of the new measuremet

```
% scalasca -examine -s epik_lu_8_sum
INFO: Post-processing runtime summarization report...
/srv/app/scalasca/bin/cube3_score -r ./epik_lu_8_sum/summary.cube.gz >
./epik_lu_8_sum/epik.score
Reading ./epik_lu_8_sum/summary.cube.gz... done.
Estimated aggregate size of event trace (total_tbc): 54560192 bytes
Estimated size of largest process trace (max_tbc):   7582262 bytes
(Hint: When tracing set ELG_BUFFER_SIZE > max_tbc to avoid intermediate
flushes or reduce requirements using file listing names of USR regions
to be filtered.)

INFO: Score report written to ./epik_lu_8_sum/epik.score
```

LU filtered summary report

- View the score file

```
% less epik_lu_8_sum/epik.score
flt  type       max_tbc          time      % region
      ANY       7582262         98.10  100.00 (summary) ALL
      MPI       4574534         18.62   18.98 (summary) MPI
      COM       2259600         52.43   53.45 (summary) COM
      USR        748152         26.93   27.45 (summary) USR

      MPI       2372550          1.68    1.71 MPI_Send
      MPI       2147556          6.22    6.34 MPI_Recv
      COM       1493952          0.45    0.46 exchange_1_
      COM        373488         16.20   16.52 buts_
      USR        373488         12.61   12.85 jacu_
      COM        373488         12.98   13.23 blts_
...
```

- Reduction on the execution time (for bigger sizes of problems the difference is more obvious)
- Small decrease of the MPI and COM timings.

LU trace measurement collection

- Execute the application with the “-t” flag

```
% scalasca -analyze -t mpirun -np 8 lu.A.8
S=C=A=N: Scalasca 1.4 trace collection and analysis
S=C=A=N: ./epik_lu_8_trace experiment archive
S=C=A=N: Tue Apr 12 00:45:59 2011: Collect start
/srv/app/openmpi/bin//mpirun -np 8 lu.A.8
[00000]EPIK: Created new measurement archive ./epik_lu_8_trace
[00000]EPIK: EPK_FILTER "lu.filtering" filtered 1 of 222 functions
[00000]EPIK: Activated ./epik_lu_8_trace [10000000 bytes] (0.206s)
[ ... output ... ]
[00000]EPIK: Flushed 6057882 bytes to file ./epik_lu_8_trace/ELG/00000
[00000]EPIK: Unifying... done (0.012s)
[00000]EPIK: Collating... done (0.012s)
[00001]EPIK: Flushed 7582272 bytes to file ./epik_lu_8_trace/ELG/00001
[00002]EPIK: Flushed 7582272 bytes to file ./epik_lu_8_trace/ELG/00002
[...]
[00000]EPIK: 1flush=0.006GB@11.549MB/s, Pflush=0.045GB@77.119MB/s
[00000]EPIK: Closed experiment ./epik_lu_8_trace (1.125s)
maxHeap(*)=16.211/125.527MB
S=C=A=N: Tue Apr 12 00:46:18 2011: Collect done (status=0) 19s
S=C=A=N: Tue Apr 12 00:46:18 2011: Analyze start
/srv/app/openmpi/bin//mpirun -np 8 /srv/app/scalasca/bin/scout.mpi
./epik_lu_8_trace
```

- One file per MPI rank is created in the experiment directory epik_lu_8_trace

LU trace measurement analysis

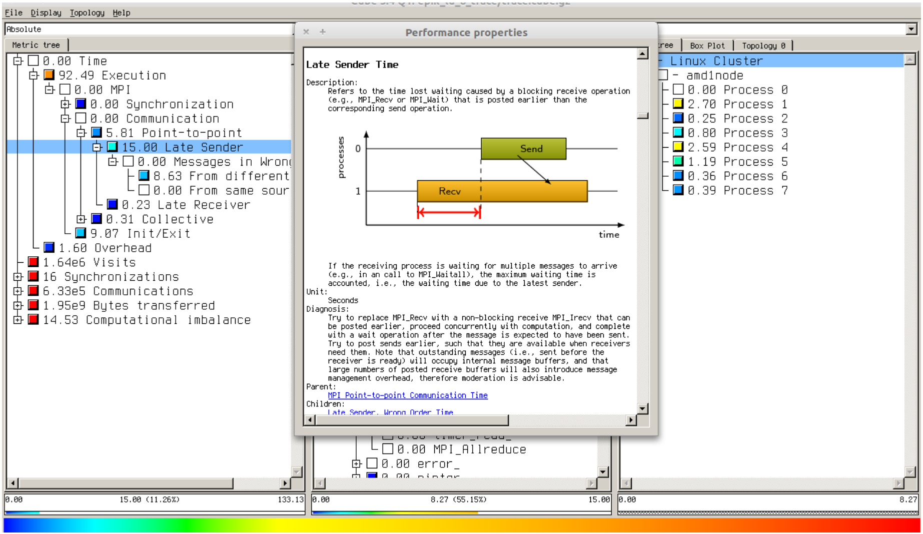
- Scalasca provides the SCOUT tool, a parallal trace analyzer which analyzes the trace files and produce an analysis report

```
S=C=A=N: Tue Apr 12 00:46:18 2011: Collect done (status=0) 19s
S=C=A=N: Tue Apr 12 00:46:18 2011: Analyze start
/srv/app/openmpi/bin/mpirun -np 8 /srv/app/scalasca/bin/scout.mpi
./epik_lu_8_trace
SCOUT   Copyright (c) 1998-2011 Forschungszentrum Juelich GmbH

Analyzing experiment archive ./epik_lu_8_trace

Reading definitions file  ... done (0.003s).
Reading event trace files ... done (0.734s).
Preprocessing             ... done (0.058s).
Analyzing trace data      ... done (0.753s).
Writing report files      ... done (0.026s).

Max. memory usage         : 26.234MB

Total processing time     : 1.593s
S=C=A=N: Tue Apr 12 00:46:21 2011: Analyze done (status=0) 3s
Warning: 19.605MB of analyzed trace data retained in ./epik_lu_8_trace/ELG!
S=C=A=N: ./epik_lu_8_trace complete.
```

- The maximum amount of memory used by any process is 26.234MB

LU trace measurement, metric tree, communication

```
% square epi_lu_8_trace
```



```
0.00 Time
  92.49 Execution
    0.00 MPI
      0.00 Synchronization
      0.00 Communication
        5.81 Point-to-point
          15.00 Late Sender
            0.00 Messages in Wrong
              8.63 From different
              0.00 From same sour
          0.23 Late Receiver
        0.31 Collective
      9.07 Init/Exit
  1.60 Overhead
1.64e6 Visits
16 Synchronizations
6.33e5 Communications
1.95e9 Bytes transferred
14.53 Computational imbalance
```

Late Sender Time

Description:
Refers to the time lost waiting caused by a blocking receive operation (e.g., MPI_Recv or MPI_Wait) that is posted earlier than the corresponding send operation.

If the receiving process is waiting for multiple messages to arrive (e.g., in an call to MPI_Waitall), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

Unit:
Seconds

Diagnosis:
Try to replace MPI_Recv with a non-blocking receive MPI_Irecv that can be posted earlier, proceed concurrently with computation, and complete with a wait operation after the message is expected to have been sent. Try to post sends earlier, such that they are available when receivers need them. Note that outstanding messages (i.e., sent before the receiver is ready) will occupy internal message buffers, and that large numbers of posted receive buffers will also introduce message management overhead, therefore moderation is advisable.

Parent:
MPI Point-to-point Communication Time

Children:
Late Sender, Wrong Order Time

```
0.00 MPI_Allreduce
0.00 error_
0.00 pintor
```

```
- Linux Cluster
  - amd1node
    0.00 Process 0
    2.70 Process 1
    0.25 Process 2
    0.80 Process 3
    2.59 Process 4
    1.19 Process 5
    0.36 Process 6
    0.39 Process 7
```

```
0.00        15.00 (11.26%)        133.13
0.00        8.27 (55.15%)         15.00
0.00                              8.27
```

EPIK user instrumentation API, Fortran

```
#include ``epik_user.inc''

subroutine foo(...)
    declarations
    EPIK_FUNC_REG("foo")
    EPIK_USER_REG(r_name,"iteration loop")
    EPIK_FUNC_START()
    ...
    EPIK_USER_START(r_name)
    do i= 1, 100
    ...
    end do
    EPIK_USER_END(r_name)
    ...
    EPIK_FUNC_END()
end subroutine foo
```

EPIK user instrumentation API, C/C++

```
#include ``epik_user.h''

void foo(...)
{
  /* declarations */
  EPIK_USER_REG(r_name,"iteration loop");
  EPIK_FUNC_START();
  ...
  EPIK_USER_START(r_name);
  for (i = 0; i < 10; ++i)
  {
   ...
  }
  EPIK_USER_END(r_name);
  ...
  EPIK_FUNC_END();
}
```

- In order to compile the source code with EPIK commands we have to use the “-user” flag

  ```
  scalasca -instrument -user mpif77
  ```

- We can mark a specific area and observe its performance during the analysis

Automatic instrumentation using PDT

- In order to enable PDT-based source-code instrumentation, the option “-pdt” is required and disable the compiler instrumentation by “-comp=none”

```
% scalasca -instrument -pdt -comp=none mpif77 ...
```

- Option for selective instrumentation file

```
% scalasca -instrument -pdt -comp=none -optTauSelectFile=lu.pdt mpif77 ...
```

- Note: For Fortran 77 most times is needed to give an extra option

```
% scalasca -instrument -pdt -comp=none -optTauSelectFile=lu.pdt mpif77 \
-ffixed-line-length-0
```

- Format of the selective instrumentation file
  - Exclude files

```
BEGIN_FILE_EXCLUDE_LIST
  test.c # Excludes file test.c
  foo*.c # Excludes all C files with prefix ’foo’
END_FILE_EXCLUDE_LIST
```

Automatic instrumentation using PDT II

- Exclude functions

```
BEGIN_EXCLUDE_LIST
  # Exclude C function matmult
  void matmult(Matrix*, Matrix*, Matrix*) C

  # Exclude C++ functions with prefix ’sort_’ and a
  # single int pointer argument
  void sort_#(int *)

  # Exclude all void functions in namespace ’foo’
  void foo::#
END_EXCLUDE_LIST
```

- The mark # is widlcard for a routine name and the mark * is a wildcard character
- Include functions for instrumentation

```
BEGIN_INCLUDE_LIST/END_INCLUDE_LIST
```

- Exclude the function EXACT from the LU benchmark

```
% cat lu.pdt
BEGIN_EXCLUDE_LIST
EXACT
END_EXCLUDE_LIST
```

Automatic instrumentation using PDT III

- Declare the appropriate compile command in the config/make.def file

```
MPIF77 = scalasca -instrument -pdt -comp=none -optTauSelectFile=/path/lu.pdt \
mpif77 -ffixed-line-length-0
```

- Compile the LU benchmark, for class A and 8 processes from the NPB root path

```
% make LU NPROCS=8 CLASS=A
```

- Enter the Scalasca folder and execute the benchmark

```
% cd bin.scalasca
% scalasca -analyze mpirun -np 8 lu.A.8
```

- Now if you apply the scoring and see the output file

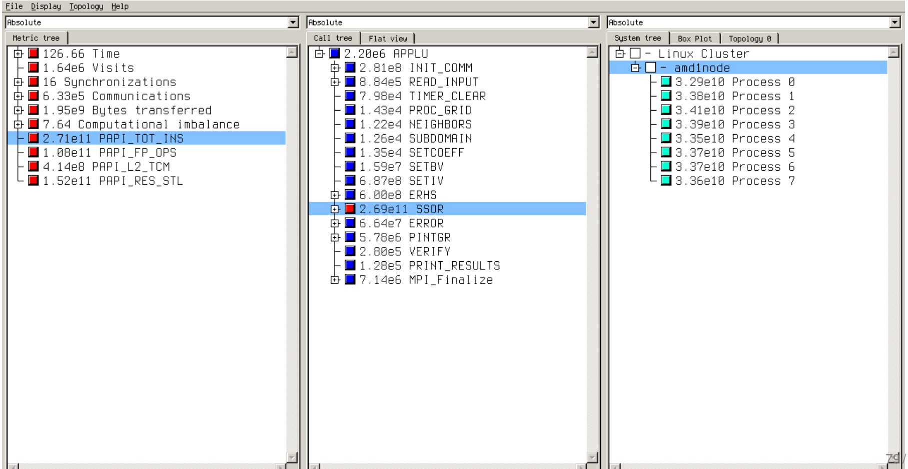
```
% scalasca -examine -s epik_lu_8_sum
% cat epik_lu_8_sum/epik.score | grep EXACT
```

Then there is no function EXACT that is traced

LU summary measurement, hardware counters

- Measure the PAPI hardware counters:
  - PAPI_TOT_INS (total instructions completed)
  - PAPI_FP_OPS (floating point operations)
  - PAPI_L2_TCM (L2 cacHe misses)
  - PAPI_RES_STL (stalled cycles on any resource)

```
% export EPK_METRICS=PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM:PAPI_RES_STL
```



```
126.66 Time
1.64e6 Visits
16 Synchronizations
6.33e5 Communications
1.95e9 Bytes transferred
7.64 Computational imbalance
2.71e11 PAPI_TOT_INS
1.08e11 PAPI_FP_OPS
4.14e8 PAPI_L2_TCM
1.52e11 PAPI_RES_STL
```

```
2.20e6 APPLU
  2.81e8 INIT_COMM
  8.84e5 READ_INPUT
  7.98e4 TIMER_CLEAR
  1.43e4 PROC_GRID
  1.22e4 NEIGHBORS
  1.26e4 SUBDOMAIN
  1.35e4 SETCOEFF
  1.59e7 SETBV
  6.87e8 SETIV
  6.00e8 ERHS
  2.69e11 SSOR
  6.64e7 ERROR
  5.78e6 PINTGR
  2.80e5 VERIFY
  1.28e5 PRINT_RESULTS
  7.14e6 MPI_Finalize
```

```
- Linux Cluster
  - amd1node
    3.29e10 Process 0
    3.38e10 Process 1
    3.41e10 Process 2
    3.39e10 Process 3
    3.35e10 Process 4
    3.37e10 Process 5
    3.37e10 Process 6
    3.36e10 Process 7
```

CUBE3 algebra utilities

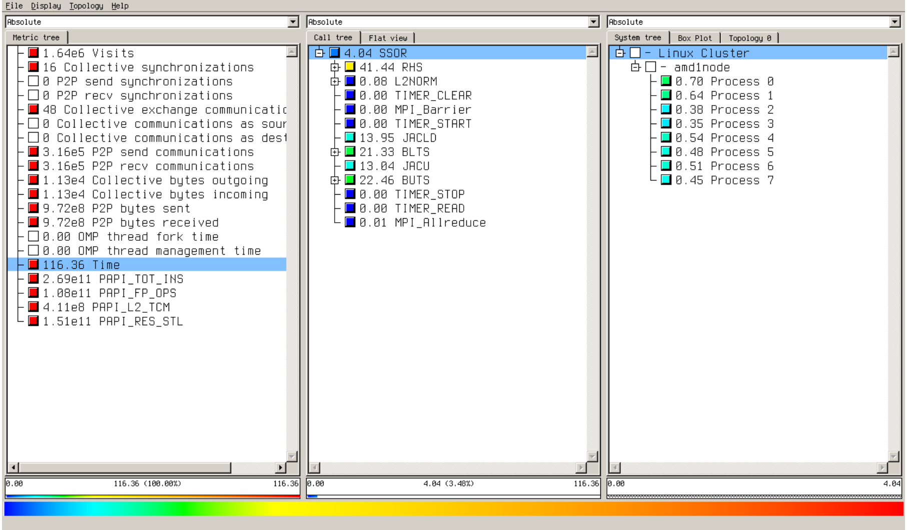
- Extract only the SSOR region with its sub-regions

```
% cube3_cut -r ’SSOR’ epik_lu_8_sum_PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM:PAPI_RES_STL/epitome.cube
Reading epik_lu_8_sum_PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM:PAPI_RES_STL/epitome.cube ... done.
++++++++++++ Cut operation begins ++++++++++++++++++++++++++
++++++++++++ Cut operation ends successfully ++++++++++++++++
Writing cut.cube.gz ... done.
```

- View the new CUBE3 file

```
% square cut.cube.gz
```

LU summary measurement, cut for SSOR



```
1.64e6 Visits
16 Collective synchronizations
0 P2P send synchronizations
0 P2P recv synchronizations
48 Collective exchange communicatic
0 Collective communications as sour
0 Collective communications as dest
3.16e5 P2P send communications
3.16e5 P2P recv communications
1.13e4 Collective bytes outgoing
1.13e4 Collective bytes incoming
9.72e8 P2P bytes sent
9.72e8 P2P bytes received
0.00 OMP thread fork time
0.00 OMP thread management time
116.36 Time
2.69e11 PAPI_TOT_INS
1.08e11 PAPI_FP_OPS
4.11e8 PAPI_L2_TCM
1.51e11 PAPI_RES_STL
```

```
4.04 SSOR
  41.44 RHS
  0.08 L2NORM
  0.00 TIMER_CLEAR
  0.00 MPI_Barrier
  0.00 TIMER_START
  13.95 JACLD
  21.33 BLTS
  13.04 JACU
  22.46 BUTS
  0.00 TIMER_STOP
  0.00 TIMER_READ
  0.01 MPI_Allreduce
```

```
- Linux Cluster
  - amd1node
    0.70 Process 0
    0.64 Process 1
    0.38 Process 2
    0.35 Process 3
    0.54 Process 4
    0.48 Process 5
    0.51 Process 6
    0.45 Process 7
```

```
0.00        116.36 (100.00%)        116.36
0.00        4.04 (3.48%)            116.36
0.00                                  4.04
```

LU summary measurement, compare two executions

- Change the name of the experiment archive directory in order to execute again the experiment with different compiler options

```
% mv epik_lu_8_sum_PAPI_TOT_INS\:PAPI_FP_OPS\:PAPI_L2_TCM\:PAPI_RES_STL/ \
epik_lu_b_8_o3_sum_PAPI_TOT_INS\:PAPI_FP_OPS\:PAPI_L2_TCM\:PAPI_RES_STL/
```

- Declare the option “-O2” in the file config/make.def

```
FFLAGS   = -O2
```

- Compile the LU benchmark, class B, 8 processes

```
% make clean
% make LU NPROCS=8 CLASS=B
```

- Enter the directory with the executables

```
% cd bin.scalasca
```

- Execute the benchmark

```
% scalasca -analyze mpirun --bind-to-core -np 8 lu.B.8
```

LU summary measurement, compare two executions II

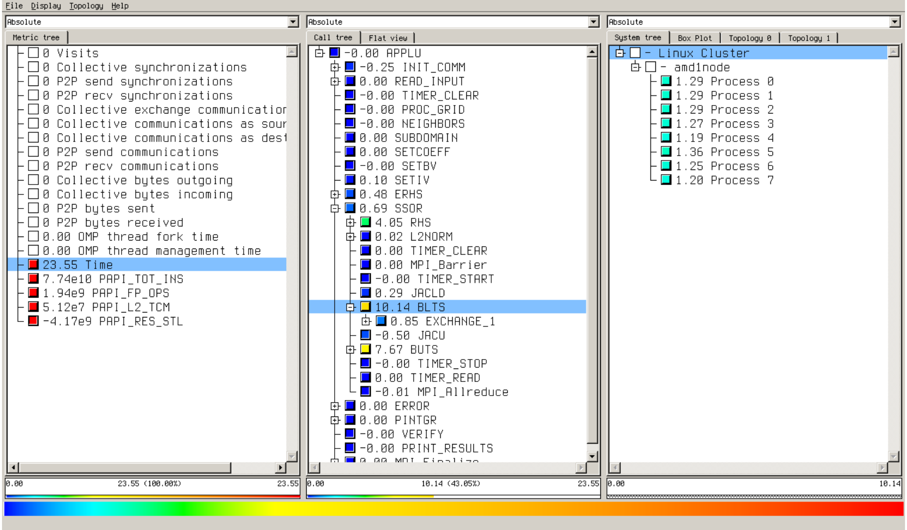
- Compare the two executions

```
% cube3_diff epik_lu_b_8_o2_sum_PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM:\
PAPI_RES_STL/epitome.cube epik_lu_8_sum_PAPI_TOT_INS:PAPI_FP_OPS: \
PAPI_L2_TCM:PAPI_RES_STL/epitome.cube
Reading epik_lu_b_8_o2_sum_PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM: \
PAPI_RES_STL/epitome.cube ... done.
Reading epik_lu_8_sum_PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM: \
PAPI_RES_STL/epitome.cube ... done.
++++++++++++ Diff operation begins ++++++++++++++++++++++++++
INFO::Merging metric dimension... done.
INFO::Merging program dimension... done.
INFO::Merging system dimension... done.
INFO::Mapping severities... done.
INFO::Merging topologies... done.
INFO::Diff operation... done.
++++++++++++ Diff operation ends successfully ++++++++++++++++
Writing diff.cube.gz ... done.
```

LU summary measurement, cut for SSOR

- View the new CUBE3 file

```
% square diff.cube.gz
```



Absolute

```
□ 0 Visits
□ 0 Collective synchronizations
□ 0 P2P send synchronizations
□ 0 P2P recv synchronizations
□ 0 Collective exchange communicatior
□ 0 Collective communications as sour
□ 0 Collective communications as dest
□ 0 P2P send communications
□ 0 P2P recv communications
□ 0 Collective bytes outgoing
□ 0 Collective bytes incoming
□ 0 P2P bytes sent
□ 0 P2P bytes received
□ 0.00 OMP thread fork time
□ 0.00 OMP thread management time
■ 23.55 Time
■ 7.74e10 PAPI_TOT_INS
■ 1.94e9 PAPI_FP_OPS
■ 5.12e7 PAPI_L2_TCM
■ -4.17e9 PAPI_RES_STL
```

0.00 | 23.55 (100.00%) | 23.55

Absolute

```
⊟ ■ -0.00 APPLU
  ⊞ ■ -0.25 INIT_COMM
  ⊞ ■ 0.00 READ_INPUT
    ■ -0.00 TIMER_CLEAR
    ■ -0.00 PROC_GRID
    ■ -0.00 NEIGHBORS
    ■ 0.00 SUBDOMAIN
    ■ 0.00 SETCOEFF
    ■ -0.00 SETBV
    ■ 0.10 SETIV
  ⊞ ■ 0.48 ERHS
  ⊟ ■ 0.69 SSOR
    ⊞ ■ 4.05 RHS
    ⊞ ■ 0.02 L2NORM
      ■ 0.00 TIMER_CLEAR
      ■ 0.00 MPI_Barrier
      ■ -0.00 TIMER_START
      ■ 0.29 JACLD
    ⊟ ■ 10.14 BLTS
      ⊞ ■ 0.85 EXCHANGE_1
      ■ -0.50 JACU
    ⊞ ■ 7.67 BUTS
      ■ -0.00 TIMER_STOP
      ■ 0.00 TIMER_READ
      ■ -0.01 MPI_Allreduce
  ⊞ ■ 0.00 ERROR
  ⊞ ■ 0.00 PINTGR
    ■ -0.00 VERIFY
    ■ -0.00 PRINT_RESULTS
  ⊞ ■ 0.00 MPI_Finalize
```

0.00 | 10.14 (43.05%) | 23.55

Absolute

```
⊟ □ - Linux Cluster
  ⊟ □ - amd1node
      ■ 1.29 Process 0
      ■ 1.29 Process 1
      ■ 1.29 Process 2
      ■ 1.27 Process 3
      ■ 1.19 Process 4
      ■ 1.36 Process 5
      ■ 1.25 Process 6
      ■ 1.20 Process 7
```

0.00 | 10.14

- Not all the parts of the code were improved by the change of the optimization option

CUBE3 utilities

There are more CUBE3 utilities:

- Difference

```
% cube3_diff first.cube second.cube -o new.cube
```

- Merge two different measurements with different metrics

```
% cube3_merge first.cube second.cube -o new.cube
```

- Calculate the mean of many measurements

```
% cube3_mean first.cube second.cube third.cube fourth.cube \
-o new.cube
```

- Compare two measurements if they are exactly the same

```
% cube3_cmp first.cube second.cube third.cube -o new.cube
```

- Cut, re-root selected sub-trees

```
% cube3_cut -r name_of_sub_tree first.cube -o new.cube
```

- There are more utilities, like cube3_clean

LU benchmark, class B

We are going to execute the LU benchmark for class B and various number of processors and observe performance issues.

- Go to the root folder of the serial version of NPB and compile the LU benchmark for class B

```
% cd ~/tutorial/NPB3.3-SER
% make clean
% make LU CLASS=B
```

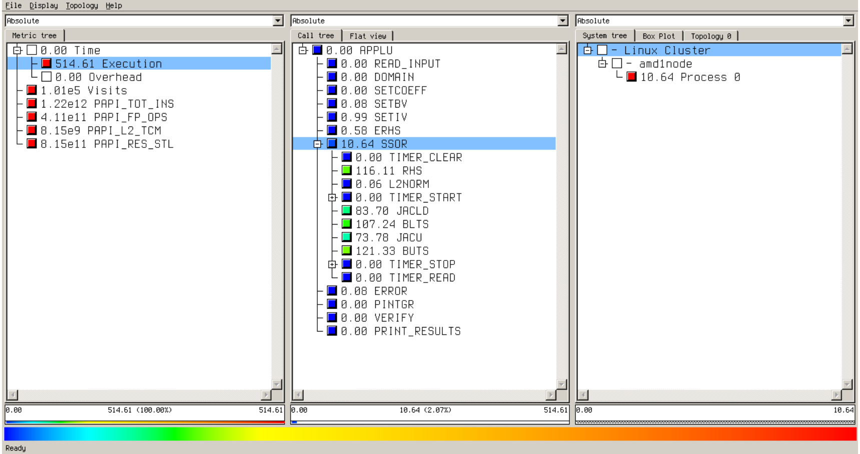
- Go to the executable directory and execute the benchmark

```
% cd bin.scalasca
% scalasca -analyze ./lu.B.x
```

- Explore the measurement analysis report

```
% square epik_lu_O_trace_PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM:\
PAPI_RES_STL
```

LU measurement summary for the serial version and class B



```
0.00 Time
  514.61 Execution
  0.00 Overhead
1.01e5 Visits
1.22e12 PAPI_TOT_INS
4.11e11 PAPI_FP_OPS
8.15e9 PAPI_L2_TCM
8.15e11 PAPI_RES_STL
```

```
0.00 APPLU
  0.00 READ_INPUT
  0.00 DOMAIN
  0.00 SETCOEFF
  0.08 SETBV
  0.99 SETIV
  0.58 ERHS
  10.64 SSOR
    0.00 TIMER_CLEAR
    116.11 RHS
    0.06 L2NORM
    0.00 TIMER_START
    83.70 JACLD
    107.24 BLTS
    73.78 JACU
    121.33 BUTS
    0.00 TIMER_STOP
    0.00 TIMER_READ
  0.08 ERROR
  0.00 PINTGR
  0.00 VERIFY
  0.00 PRINT_RESULTS
```

```
- Linux Cluster
  - amd1node
    10.64 Process 0
```

```
0.00    514.61 (100.00%)    514.61
0.00    10.64 (2.07%)    514.61
0.00    10.64
```

- The computation execution time is 514.61 seconds and there are 8.15e11 stalled cycles on any resource

- Similar for the MPI version of the LU benchmark, compile it for 2 processors and class B

```
% cd ~/tutorial/NPB3.3-MPI
% make clean
% make LU NPROCS=2 CLASS=B
```

- Compile the benchmark also for 4,8,16 and 32 processors
- Go to the executable directory

```
% cd bin.scalasca
```

- Declare the appropriate ELG buffer

```
% echo ‘‘export ELG_BUFFER_SIZE=60000000’’ >> \
~/.bashrc
```

Execute the LU benchmark

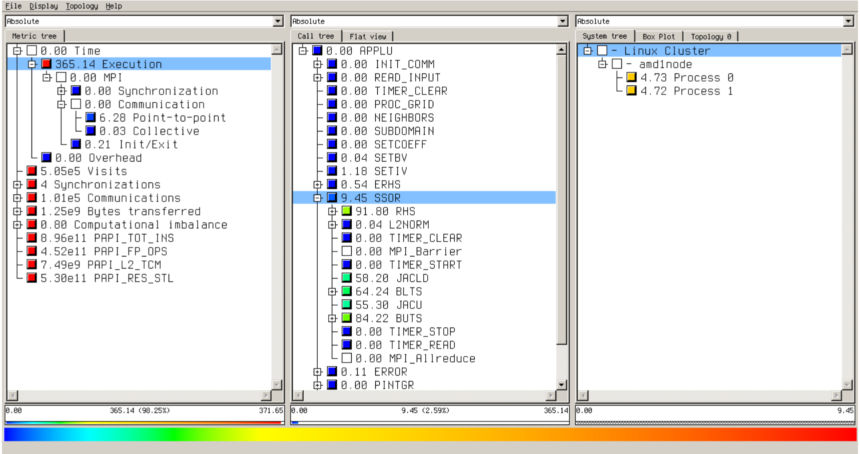
- Execute the benchmarks

```
% scalasca -analyze mpirun -np 2 --bind-to-core lu.B.2
% scalasca -analyze mpirun -np 4 --bind-to-core lu.B.4
% scalasca -analyze mpirun -np 8 --bind-to-core lu.B.8
% scalasca -analyze mpirun -np 16 --bind-to-core lu.B.16
% scalasca -analyze mpirun -np 32 --bind-to-core lu.B.32
```

- Let’s examine the measurement analysis report for the 2 processors

```
% square epik_lu_2_sum_PAPI_TOT_INS:PAPI_FP_OPS:\
PAPI_L2_TCM:PAPI_RES_STL
```

LU measurement summary for 2 processors, class B, execution time



```
0.00 Time
  365.14 Execution
    0.00 MPI
      0.00 Synchronization
      0.00 Communication
        6.28 Point-to-point
        0.03 Collective
      0.21 Init/Exit
    0.00 Overhead
5.05e5 Visits
4 Synchronizations
1.01e5 Communications
1.25e9 Bytes transferred
0.80 Computational imbalance
8.96e11 PAPI_TOT_INS
4.52e11 PAPI_FP_OPS
7.49e9 PAPI_L2_TCM
5.30e11 PAPI_RES_STL
```

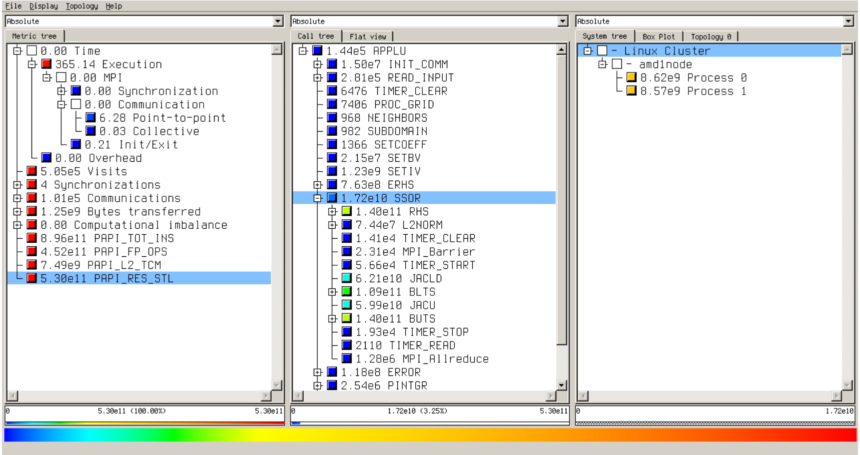
```
0.00 APPLU
  0.00 INIT_COMM
  0.00 READ_INPUT
  0.00 TIMER_CLEAR
  0.00 PROC_GRID
  0.00 NEIGHBORS
  0.00 SUBDOMAIN
  0.00 SETCOEFF
  0.04 SETBV
  1.18 SETIV
  0.54 ERHS
  9.45 SSOR
    91.80 RHS
    0.04 L2NORM
    0.00 TIMER_CLEAR
    0.00 MPI_Barrier
    0.00 TIMER_START
    58.20 JACLD
    64.24 BLTS
    55.30 JACU
    84.22 BUTS
    0.00 TIMER_STOP
    0.00 TIMER_READ
    0.00 MPI_Allreduce
  0.11 ERROR
  0.00 PINTGR
```

```
- Linux Cluster
  - amd1node
    4.73 Process 0
    4.72 Process 1
```

```
0.00          365.14 (98.25%)          371.65
0.00          9.45 (2.59%)             365.14
0.00                                   9.45
```

- The total computation execution time is 365.14 seconds for all the processors (sum) and the exclusive time for the SSOR function is 9.45 seconds
- The communication time is less than 7 seconds

LU measurement summary for 2 processors, class B, stalled cycles on any resource



```
0.00 Time
  365.14 Execution
    0.00 MPI
      0.00 Synchronization
      0.00 Communication
        6.28 Point-to-point
        0.03 Collective
      0.21 Init/Exit
  0.00 Overhead
5.05e5 Visits
4 Synchronizations
1.01e5 Communications
1.25e9 Bytes transferred
0.80 Computational imbalance
8.96e11 PAPI_TOT_INS
4.52e11 PAPI_FP_OPS
7.49e9 PAPI_L2_TCM
5.30e11 PAPI_RES_STL
```

```
0        5.30e11 (100.00%)        5.30e11
```

```
1.44e5 APPLU
  1.50e7 INIT_COMM
  2.81e5 READ_INPUT
  6476 TIMER_CLEAR
  7406 PROC_GRID
  968 NEIGHBORS
  982 SUBDOMAIN
  1366 SETCOEFF
  2.15e7 SETBV
  1.23e9 SETIV
  7.63e8 ERHS
  1.72e10 SSOR
    1.40e11 RHS
    7.44e7 L2NORM
    1.41e4 TIMER_CLEAR
    2.31e4 MPI_Barrier
    5.66e4 TIMER_START
    6.21e10 JACLD
    1.09e11 BLTS
    5.99e10 JACU
    1.40e11 BUTS
    1.93e4 TIMER_STOP
    2110 TIMER_READ
    1.28e6 MPI_Allreduce
  1.18e8 ERROR
  2.54e6 PINTGR
```

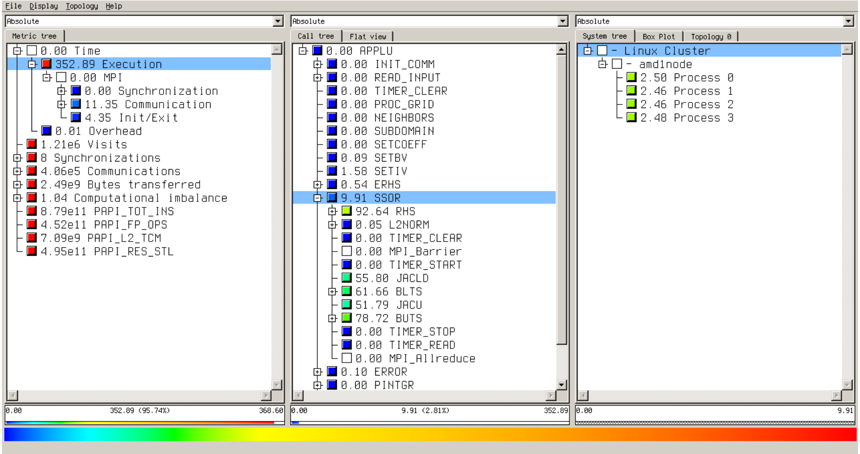
```
0        1.72e10 (3.25%)        5.30e11
```

```
- Linux Cluster
  - amd1node
    8.62e9 Process 0
    8.57e9 Process 1
```

```
0                              1.72e10
```

- There are 5.3e11 stalled cycles on any resource for all the processors (around to 8.6e9 per processor)

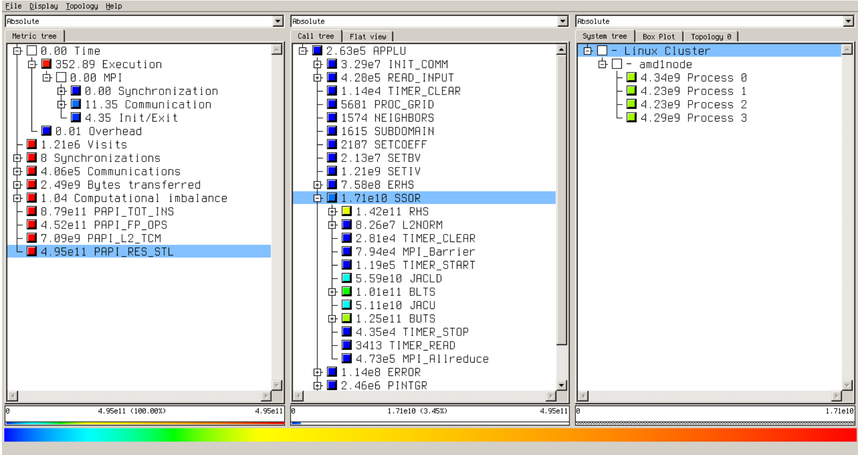
LU measurement summary for 4 processors, class B, execution time



```
File  Display  Topology  Help

Absolute                              Absolute                              Absolute
Metric tree                           Call tree | Flat view                 System tree | Box Plot | Topology 0

⊟ □ 0.00 Time                         ⊟ 🟦 0.00 APPLU                       ⊟ □ - Linux Cluster
  ⊟ 🟥 352.89 Execution                 ⊞ 🟦 0.00 INIT_COMM                   ⊟ □ - amd1node
    ⊟ □ 0.00 MPI                        ⊞ 🟦 0.00 READ_INPUT                    ├ 🟩 2.50 Process 0
      ⊞ 🟦 0.00 Synchronization         ├ 🟦 0.00 TIMER_CLEAR                   ├ 🟩 2.46 Process 1
      ⊞ 🟦 11.35 Communication          ├ 🟦 0.00 PROC_GRID                     ├ 🟩 2.46 Process 2
      └ 🟦 4.35 Init/Exit               ├ 🟦 0.00 NEIGHBORS                     └ 🟩 2.48 Process 3
    └ 🟦 0.01 Overhead                  ├ 🟦 0.00 SUBDOMAIN
  ├ 🟥 1.21e6 Visits                    ├ 🟦 0.00 SETCOEFF
  ⊞ 🟥 8 Synchronizations               ├ 🟦 0.09 SETBV
  ⊞ 🟥 4.06e5 Communications            ├ 🟦 1.58 SETIV
  ⊞ 🟥 2.49e9 Bytes transferred         ⊞ 🟦 0.54 ERHS
  ⊞ 🟥 1.04 Computational imbalance     ⊟ 🟦 9.91 SSOR
  ├ 🟥 8.79e11 PAPI_TOT_INS               ⊞ 🟩 92.64 RHS
  ├ 🟥 4.52e11 PAPI_FP_OPS                ⊞ 🟦 0.05 L2NORM
  ├ 🟥 7.09e9 PAPI_L2_TCM                 ├ 🟦 0.00 TIMER_CLEAR
  └ 🟥 4.95e11 PAPI_RES_STL               ├ □ 0.00 MPI_Barrier
                                          ├ 🟦 0.00 TIMER_START
                                          ├ 🟩 55.80 JACLD
                                          ⊞ 🟩 61.66 BLTS
                                          ├ 🟩 51.79 JACU
                                          ⊞ 🟩 78.72 BUTS
                                          ├ 🟦 0.00 TIMER_STOP
                                          ├ 🟦 0.00 TIMER_READ
                                          └ □ 0.00 MPI_Allreduce
                                        ⊞ 🟦 0.10 ERROR
                                        ⊞ 🟦 0.00 PINTGR

0.00    352.89 (95.74%)    368.60   0.00    9.91 (2.81%)    352.89   0.00                    9.91
```

- The total computation execution time is 352.89 seconds for all the processors (sum) and the exclusive time for the SSOR function is 9.91 seconds
- The communication time is less than 16 seconds

LU measurement summary for 4 processors, class B, stalled cycles on any resource



```
0.00 Time
  352.89 Execution
    0.00 MPI
      0.00 Synchronization
      11.35 Communication
      4.35 Init/Exit
    0.01 Overhead
1.21e6 Visits
8 Synchronizations
4.06e5 Communications
2.49e9 Bytes transferred
1.04 Computational imbalance
8.79e11 PAPI_TOT_INS
4.52e11 PAPI_FP_OPS
7.09e9 PAPI_L2_TCM
4.95e11 PAPI_RES_STL
```

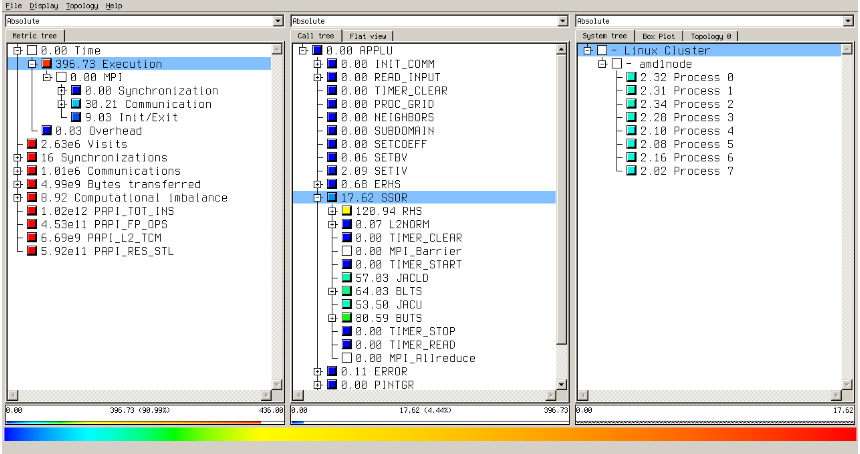
```
2.63e5 APPLU
  3.29e7 INIT_COMM
  4.28e5 READ_INPUT
  1.14e4 TIMER_CLEAR
  5681 PROC_GRID
  1574 NEIGHBORS
  1615 SUBDOMAIN
  2187 SETCOEFF
  2.13e7 SETBV
  1.21e9 SETIV
  7.58e8 ERHS
  1.71e10 SSOR
    1.42e11 RHS
    8.26e7 L2NORM
    2.81e4 TIMER_CLEAR
    7.94e4 MPI_Barrier
    1.19e5 TIMER_START
    5.59e10 JACLD
    1.01e11 BLTS
    5.11e10 JACU
    1.25e11 BUTS
    4.35e4 TIMER_STOP
    3413 TIMER_READ
    4.73e5 MPI_Allreduce
  1.14e8 ERROR
  2.46e6 PINTGR
```

```
- Linux Cluster
  - amd1node
    4.34e9 Process 0
    4.23e9 Process 1
    4.23e9 Process 2
    4.29e9 Process 3
```

```
0        4.95e11 (100.00%)        4.95e11
0        1.71e10 (3.45%)          4.95e11
0                                 1.71e10
```

- There are 4.95e11 stalled cycles on any resource for all the processors

LU measurement summary for 8 processors, class B, execution time



```
0.00 Time
  396.73 Execution
    0.00 MPI
      0.00 Synchronization
      30.21 Communication
      9.03 Init/Exit
    0.03 Overhead
  2.63e6 Visits
  16 Synchronizations
  1.01e6 Communications
  4.99e9 Bytes transferred
  8.92 Computational imbalance
  1.02e12 PAPI_TOT_INS
  4.53e11 PAPI_FP_OPS
  6.69e9 PAPI_L2_TCM
  5.92e11 PAPI_RES_STL
```

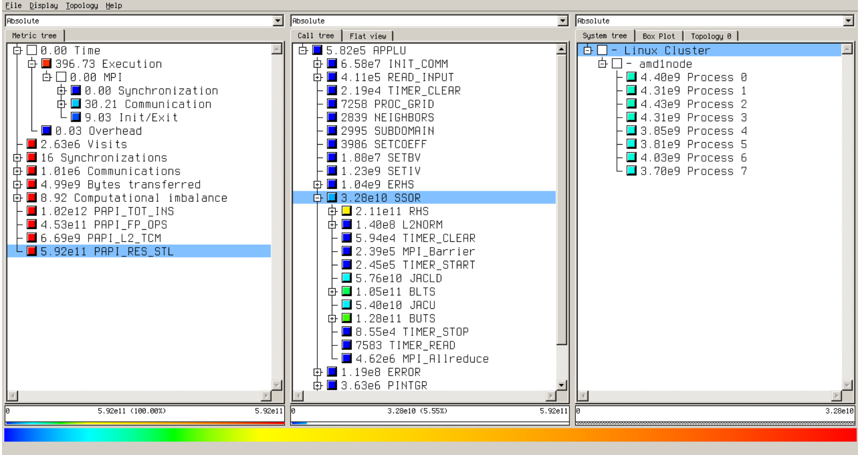
```
0.00 APPLU
  0.00 INIT_COMM
  0.00 READ_INPUT
  0.00 TIMER_CLEAR
  0.00 PROC_GRID
  0.00 NEIGHBORS
  0.00 SUBDOMAIN
  0.00 SETCOEFF
  0.06 SETBV
  2.09 SETIV
  0.68 ERHS
  17.62 SSOR
    120.94 RHS
    0.07 L2NORM
    0.00 TIMER_CLEAR
    0.00 MPI_Barrier
    0.00 TIMER_START
    57.03 JACLD
    64.03 BLTS
    53.50 JACU
    80.59 BUTS
    0.00 TIMER_STOP
    0.00 TIMER_READ
    0.00 MPI_Allreduce
  0.11 ERROR
  0.00 PINTGR
```

```
- Linux Cluster
  - amd1node
    2.32 Process 0
    2.31 Process 1
    2.34 Process 2
    2.28 Process 3
    2.10 Process 4
    2.08 Process 5
    2.16 Process 6
    2.02 Process 7
```

```
0.00          396.73 (90.99%)          436.00
0.00          17.62 (4.44%)            396.73
0.00                                   17.62
```

- The total computation execution time is 396.73 seconds for all the processors (sum) and the exclusive time for the SSOR function is 17.62 seconds
- The communication time is less than 40 seconds

LU measurement summary for 8 processors, class B, stalled cycles on any resource



```
0.00 Time
  396.73 Execution
    0.00 MPI
      0.00 Synchronization
      30.21 Communication
      9.03 Init/Exit
    0.03 Overhead
2.63e6 Visits
16 Synchronizations
1.01e6 Communications
4.99e9 Bytes transferred
8.92 Computational imbalance
1.02e12 PAPI_TOT_INS
4.53e11 PAPI_FP_OPS
6.69e9 PAPI_L2_TCM
5.92e11 PAPI_RES_STL
```

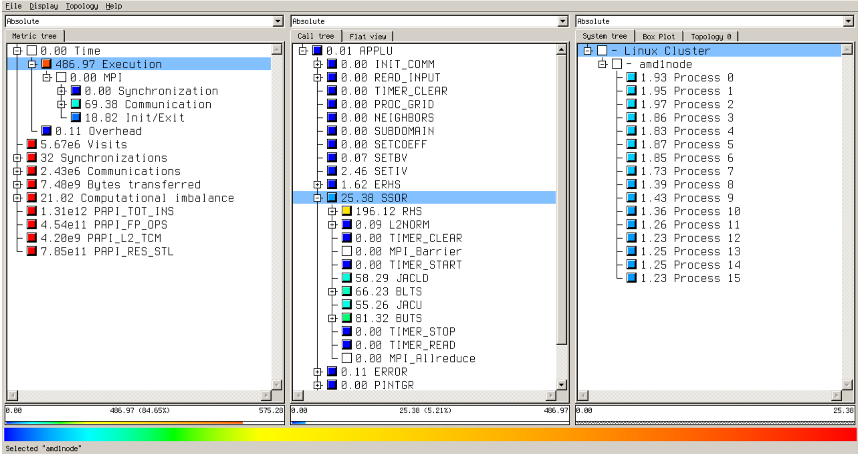
```
5.82e5 APPLU
  6.58e7 INIT_COMM
  4.11e5 READ_INPUT
  2.19e4 TIMER_CLEAR
  7258 PROC_GRID
  2839 NEIGHBORS
  2995 SUBDOMAIN
  3986 SETCOEFF
  1.88e7 SETBV
  1.23e9 SETIV
  1.04e9 ERHS
  3.28e10 SSOR
    2.11e11 RHS
    1.40e8 L2NORM
    5.94e4 TIMER_CLEAR
    2.39e5 MPI_Barrier
    2.45e5 TIMER_START
    5.76e10 JACLD
    1.05e11 BLTS
    5.40e10 JACU
    1.28e11 BUTS
    8.55e4 TIMER_STOP
    7583 TIMER_READ
    4.62e6 MPI_Allreduce
  1.19e8 ERROR
  3.63e6 PINTGR
```

```
- Linux Cluster
  - amd1node
    4.40e9 Process 0
    4.31e9 Process 1
    4.43e9 Process 2
    4.31e9 Process 3
    3.85e9 Process 4
    3.81e9 Process 5
    4.03e9 Process 6
    3.70e9 Process 7
```

```
0    5.92e11 (100.00%)    5.92e11
0    3.28e10 (5.55%)      5.92e11
0                         3.28e10
```

- There are 5.92e11 stalled cycles on any resource for all the processors

LU measurement summary for 16 processors, class B, execution time



```
0.00 Time
  486.97 Execution
    0.00 MPI
      0.00 Synchronization
      69.38 Communication
      18.82 Init/Exit
    0.11 Overhead
  5.67e6 Visits
  32 Synchronizations
  2.43e6 Communications
  7.48e9 Bytes transferred
  21.02 Computational imbalance
  1.31e12 PAPI_TOT_INS
  4.54e11 PAPI_FP_OPS
  4.20e9 PAPI_L2_TCM
  7.85e11 PAPI_RES_STL
```

```
0.00                  486.97 (84.65%)                  575.28
```

```
0.01 APPLU
  0.00 INIT_COMM
  0.00 READ_INPUT
  0.00 TIMER_CLEAR
  0.00 PROC_GRID
  0.00 NEIGHBORS
  0.00 SUBDOMAIN
  0.00 SETCOEFF
  0.07 SETBV
  2.46 SETIV
  1.62 ERHS
  25.38 SSOR
    196.12 RHS
    0.09 L2NORM
    0.00 TIMER_CLEAR
    0.00 MPI_Barrier
    0.00 TIMER_START
    58.29 JACLD
    66.23 BLTS
    55.26 JACU
    81.32 BUTS
    0.00 TIMER_STOP
    0.00 TIMER_READ
    0.00 MPI_Allreduce
  0.11 ERROR
  0.00 PINTGR
```

```
0.00                  25.38 (5.21%)                  486.97
```

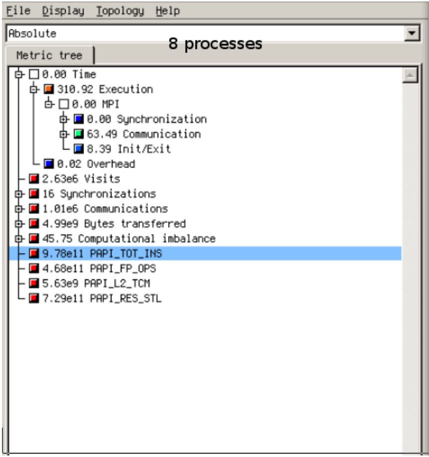
```
- Linux Cluster
  - amd1node
    1.93 Process 0
    1.95 Process 1
    1.97 Process 2
    1.86 Process 3
    1.83 Process 4
    1.87 Process 5
    1.85 Process 6
    1.73 Process 7
    1.39 Process 8
    1.43 Process 9
    1.36 Process 10
    1.26 Process 11
    1.23 Process 12
    1.25 Process 13
    1.25 Process 14
    1.23 Process 15
```

```
0.00                                                  25.38
```

Selected "amd1node"

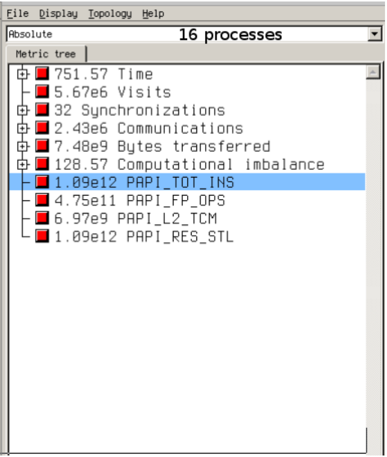
- The total computation execution time is 486.97 seconds for all the processors (sum) and the exclusive time for the SSOR function is 25.38 seconds
- The communication time is less than 89 seconds
- The value of the total instructions is increased by 28%

Comparison of the metrics for the Intel processor



8 processes

```
□ 0.00 Time
  ■ 310.92 Execution
    □ 0.00 MPI
      ■ 0.00 Synchronization
      ■ 63.49 Communication
      ■ 8.39 Init/Exit
    ■ 0.02 Overhead
■ 2.63e6 Visits
■ 16 Synchronizations
■ 1.01e6 Communications
■ 4.99e9 Bytes transferred
■ 45.75 Computational imbalance
■ 9.78e11 PAPI_TOT_INS
■ 4.68e11 PAPI_FP_OPS
■ 5.63e9 PAPI_L2_TCM
■ 7.29e11 PAPI_RES_STL
```

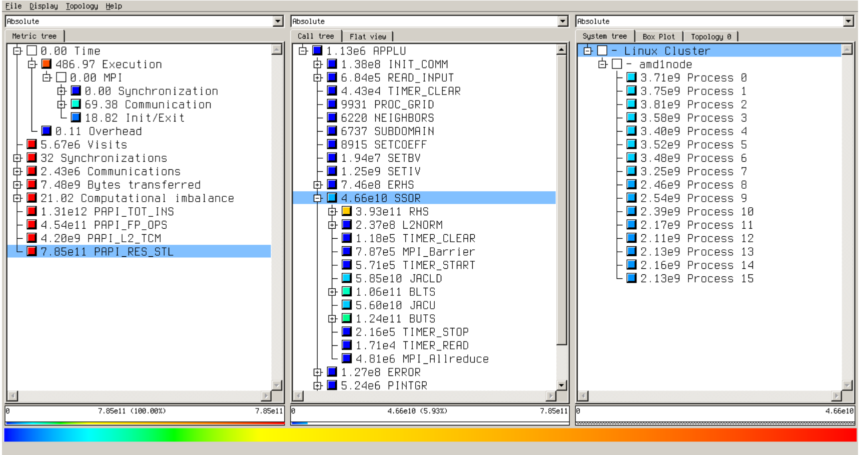


16 processes

```
■ 751.57 Time
■ 5.67e6 Visits
■ 32 Synchronizations
■ 2.43e6 Communications
■ 7.48e9 Bytes transferred
■ 128.57 Computational imbalance
■ 1.09e12 PAPI_TOT_INS
■ 4.75e11 PAPI_FP_OPS
■ 6.97e9 PAPI_L2_TCM
■ 1.09e12 PAPI_RES_STL
```

- There is no so big difference on the Intel processor and on older AMD Opteron processors (2xx and 2xxx)
- On Intel processor the difference is 11.5%

LU measurement summary for 16 processors, class B, stalled cycles on any resource



```
0.00 Time
  486.97 Execution
    0.00 MPI
      0.00 Synchronization
      69.38 Communication
      18.82 Init/Exit
  0.11 Overhead
5.67e6 Visits
32 Synchronizations
2.43e6 Communications
7.48e9 Bytes transferred
21.02 Computational imbalance
1.31e12 PAPI_TOT_INS
4.54e11 PAPI_FP_OPS
4.20e9 PAPI_L2_TCM
7.85e11 PAPI_RES_STL
```

```
0        7.85e11 (100.00%)        7.85e11
```

```
1.13e6 APPLU
  1.38e8 INIT_COMM
  6.84e5 READ_INPUT
  4.43e4 TIMER_CLEAR
  9931 PROC_GRID
  6220 NEIGHBORS
  6737 SUBDOMAIN
  8915 SETCOEFF
  1.94e7 SETBV
  1.25e9 SETIV
  7.46e8 ERHS
  4.66e10 SSOR
    3.93e11 RHS
    2.37e8 L2NORM
    1.18e5 TIMER_CLEAR
    7.87e5 MPI_Barrier
    5.71e5 TIMER_START
    5.85e10 JACLD
    1.06e11 BLTS
    5.60e10 JACU
    1.24e11 BUTS
    2.16e5 TIMER_STOP
    1.71e4 TIMER_READ
    4.81e6 MPI_Allreduce
  1.27e8 ERROR
  5.24e6 PINTGR
```

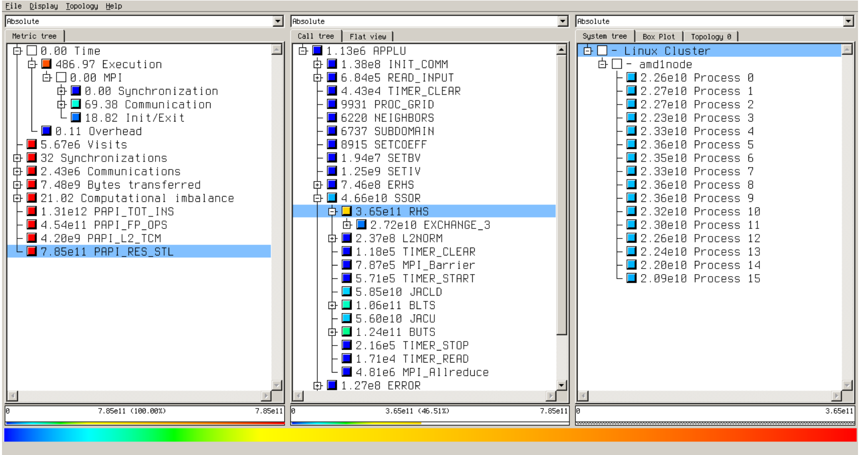
```
0        4.66e10 (5.93%)        7.85e11
```

```
- Linux Cluster
  - amd1node
    3.71e9 Process 0
    3.75e9 Process 1
    3.81e9 Process 2
    3.58e9 Process 3
    3.40e9 Process 4
    3.52e9 Process 5
    3.48e9 Process 6
    3.25e9 Process 7
    2.46e9 Process 8
    2.54e9 Process 9
    2.39e9 Process 10
    2.17e9 Process 11
    2.11e9 Process 12
    2.13e9 Process 13
    2.16e9 Process 14
    2.13e9 Process 15
```

```
0        4.66e10
```

- There are 7.85e11 stalled cycles on any resource for all the processors. Check the variation per processor on the system tree

LU measurement summary for 16 processors, class B, stalled cycles on any resource for the region RHS



```
0.00 Time
  486.97 Execution
    0.00 MPI
      0.00 Synchronization
      69.38 Communication
      18.82 Init/Exit
    0.11 Overhead
5.67e6 Visits
32 Synchronizations
2.43e6 Communications
7.48e9 Bytes transferred
21.02 Computational imbalance
1.31e12 PAPI_TOT_INS
4.54e11 PAPI_FP_OPS
4.20e9 PAPI_L2_TCM
7.85e11 PAPI_RES_STL
```

```
0    7.85e11 (100.00%)    7.85e11
```

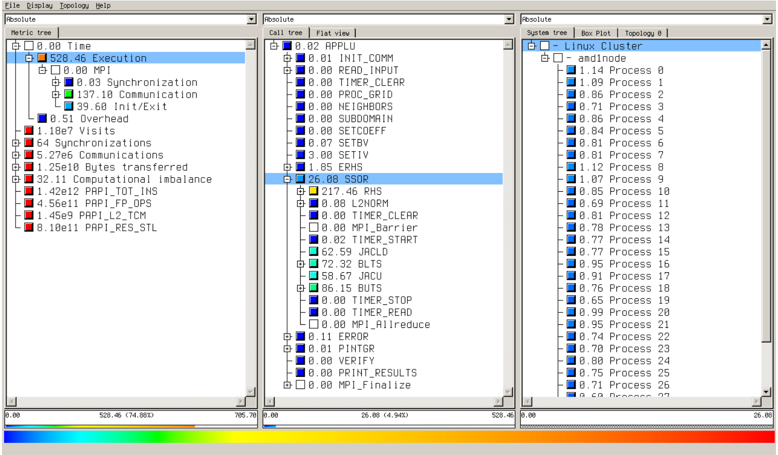
```
1.13e6 APPLU
  1.38e8 INIT_COMM
  6.84e5 READ_INPUT
  4.43e4 TIMER_CLEAR
  9931 PROC_GRID
  6220 NEIGHBORS
  6737 SUBDOMAIN
  8915 SETCOEFF
  1.94e7 SETBV
  1.25e9 SETIV
  7.46e8 ERHS
  4.66e10 SSOR
    3.65e11 RHS
      2.72e10 EXCHANGE_3
    2.37e8 L2NORM
    1.18e5 TIMER_CLEAR
    7.87e5 MPI_Barrier
    5.71e5 TIMER_START
    5.85e10 JACLD
    1.06e11 BLTS
    5.60e10 JACU
    1.24e11 BUTS
    2.16e5 TIMER_STOP
    1.71e4 TIMER_READ
    4.81e6 MPI_Allreduce
  1.27e8 ERROR
```

```
0    3.65e11 (46.51%)    7.85e11
```

```
- Linux Cluster
  - amd1node
    2.26e10 Process 0
    2.27e10 Process 1
    2.27e10 Process 2
    2.23e10 Process 3
    2.33e10 Process 4
    2.36e10 Process 5
    2.35e10 Process 6
    2.33e10 Process 7
    2.36e10 Process 8
    2.36e10 Process 9
    2.32e10 Process 10
    2.30e10 Process 11
    2.26e10 Process 12
    2.24e10 Process 13
    2.20e10 Process 14
    2.09e10 Process 15
```

```
0    3.65e11
```

LU measurement summary for 32 processors, class B, execution time



```
0.00 Time
  528.46 Execution
    0.00 MPI
      0.03 Synchronization
      137.10 Communication
      39.60 Init/Exit
    0.51 Overhead
1.18e7 Visits
64 Synchronizations
5.27e6 Communications
1.25e10 Bytes transferred
32.11 Computational imbalance
1.42e12 PAPI_TOT_INS
4.56e11 PAPI_FP_OPS
1.45e9 PAPI_L2_TCM
8.10e11 PAPI_RES_STL
```

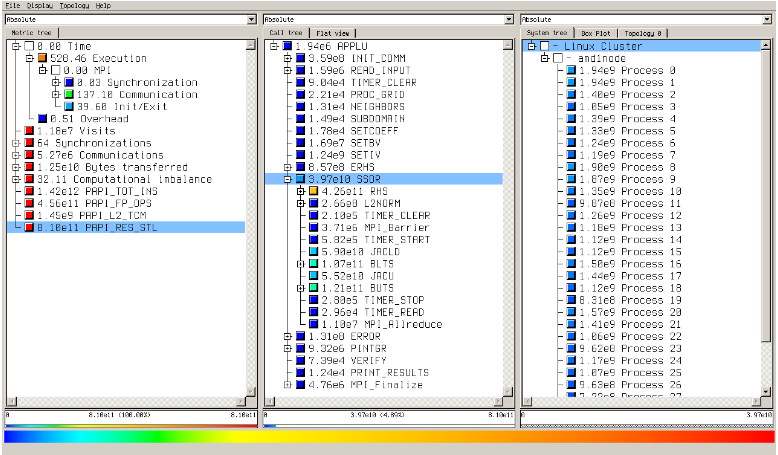
```
0.02 APPLU
  0.01 INIT_COMM
  0.00 READ_INPUT
  0.00 TIMER_CLEAR
  0.00 PROC_GRID
  0.00 NEIGHBORS
  0.00 SUBDOMAIN
  0.00 SETCOEFF
  0.07 SETBV
  3.00 SETIV
  1.85 ERHS
  26.08 SSOR
    217.46 RHS
    0.08 L2NORM
    0.00 TIMER_CLEAR
    0.00 MPI_Barrier
    0.02 TIMER_START
    62.59 JACLD
    72.32 BLTS
    58.67 JACU
    86.15 BUTS
    0.00 TIMER_STOP
    0.00 TIMER_READ
    0.00 MPI_Allreduce
  0.11 ERROR
  0.01 PINTGR
  0.00 VERIFY
  0.00 PRINT_RESULTS
  0.00 MPI_Finalize
```

```
- Linux Cluster
  - amd1node
    1.14 Process 0
    1.09 Process 1
    0.86 Process 2
    0.71 Process 3
    0.86 Process 4
    0.84 Process 5
    0.81 Process 6
    0.81 Process 7
    1.12 Process 8
    1.07 Process 9
    0.85 Process 10
    0.69 Process 11
    0.81 Process 12
    0.78 Process 13
    0.77 Process 14
    0.77 Process 15
    0.95 Process 16
    0.91 Process 17
    0.76 Process 18
    0.65 Process 19
    0.99 Process 20
    0.95 Process 21
    0.74 Process 22
    0.70 Process 23
    0.80 Process 24
    0.75 Process 25
    0.71 Process 26
    0.60 Process 27
```

```
0.00        528.46 (74.88%)        705.70
0.00        26.08 (4.94%)          528.46
0.00                               26.08
```

- The total computation execution time is 528.46 seconds for all the processors (sum) and the exclusive time for the SSOR function is 26.08 seconds
- The communication time is less than 177 seconds

LU measurement summary for 32 processors, class B, stalled cycles on any resource



```
0.00 Time
  528.46 Execution
    0.00 MPI
      0.03 Synchronization
      137.10 Communication
      39.60 Init/Exit
    0.51 Overhead
  1.18e7 Visits
  64 Synchronizations
  5.27e6 Communications
  1.25e10 Bytes transferred
  32.11 Computational imbalance
  1.42e12 PAPI_TOT_INS
  4.56e11 PAPI_FP_OPS
  1.45e9 PAPI_L2_TCM
  8.10e11 PAPI_RES_STL
```

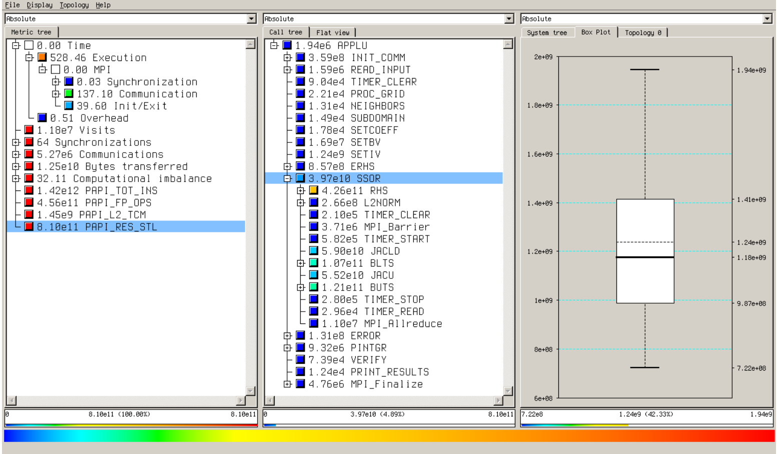
```
1.94e6 APPLU
  3.59e8 INIT_COMM
  1.59e6 READ_INPUT
  9.04e4 TIMER_CLEAR
  2.21e4 PROC_GRID
  1.31e4 NEIGHBORS
  1.49e4 SUBDOMAIN
  1.78e4 SETCOEFF
  1.69e7 SETBV
  1.24e9 SETIV
  8.57e8 ERHS
  3.97e10 SSOR
    4.26e11 RHS
    2.66e8 L2NORM
    2.10e5 TIMER_CLEAR
    3.71e6 MPI_Barrier
    5.82e5 TIMER_START
    5.90e10 JACLD
    1.07e11 BLTS
    5.52e10 JACU
    1.21e11 BUTS
    2.80e5 TIMER_STOP
    2.96e4 TIMER_READ
    1.10e7 MPI_Allreduce
  1.31e8 ERROR
  9.32e6 PINTGR
  7.39e4 VERIFY
  1.24e4 PRINT_RESULTS
  4.76e6 MPI_Finalize
```

```
- Linux Cluster
  - amd1node
    1.94e9 Process 0
    1.94e9 Process 1
    1.40e9 Process 2
    1.05e9 Process 3
    1.39e9 Process 4
    1.33e9 Process 5
    1.24e9 Process 6
    1.19e9 Process 7
    1.90e9 Process 8
    1.87e9 Process 9
    1.35e9 Process 10
    9.87e8 Process 11
    1.26e9 Process 12
    1.18e9 Process 13
    1.12e9 Process 14
    1.12e9 Process 15
    1.50e9 Process 16
    1.44e9 Process 17
    1.12e9 Process 18
    8.31e8 Process 19
    1.57e9 Process 20
    1.41e9 Process 21
    1.06e9 Process 22
    9.62e8 Process 23
    1.17e9 Process 24
    1.07e9 Process 25
    9.63e8 Process 26
```

```
0        8.10e11 (100.00%)        8.10e11
0        3.97e10 (4.89%)          8.10e11
0                                 3.97e10
```

- There are 8.10e11 stalled cycles on any resource for all the processors. Check the variation per processor on the system tree

LU measurement summary for 32 processors, class B, stalled cycles on any resource, box plot



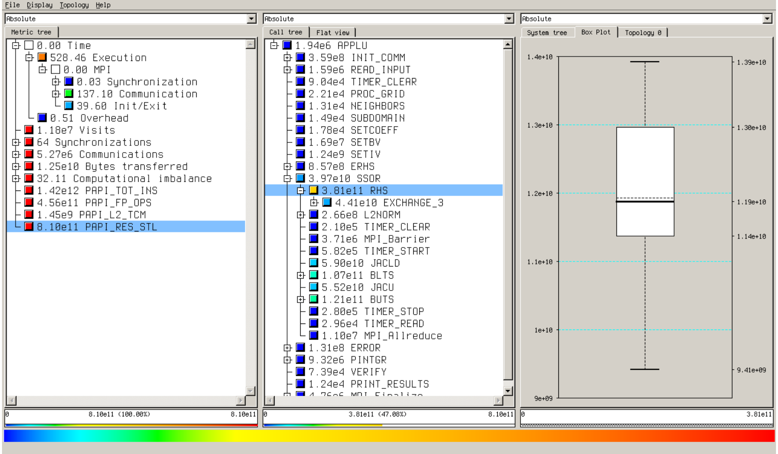
```
0.00 Time
  528.46 Execution
    0.00 MPI
      0.03 Synchronization
      137.10 Communication
      39.60 Init/Exit
  0.51 Overhead
1.18e7 Visits
64 Synchronizations
5.27e6 Communications
1.25e10 Bytes transferred
32.11 Computational imbalance
1.42e12 PAPI_TOT_INS
4.56e11 PAPI_FP_OPS
1.45e9 PAPI_L2_TCM
8.10e11 PAPI_RES_STL
```

```
1.94e6 APPLU
  3.59e8 INIT_COMM
  1.59e6 READ_INPUT
  9.04e4 TIMER_CLEAR
  2.21e4 PROC_GRID
  1.31e4 NEIGHBORS
  1.49e4 SUBDOMAIN
  1.78e4 SETCOEFF
  1.69e7 SETBV
  1.24e9 SETIV
  8.57e8 ERHS
  3.97e10 SSOR
    4.26e11 RHS
    2.66e8 L2NORM
    2.10e5 TIMER_CLEAR
    3.71e6 MPI_Barrier
    5.82e5 TIMER_START
    5.90e10 JACLD
    1.07e11 BLTS
    5.52e10 JACU
    1.21e11 BUTS
    2.80e5 TIMER_STOP
    2.96e4 TIMER_READ
    1.10e7 MPI_Allreduce
  1.31e8 ERROR
  9.32e6 PINTGR
  7.39e4 VERIFY
  1.24e4 PRINT_RESULTS
  4.76e6 MPI_Finalize
```

```
0    8.10e11 (100.00%)    8.10e11
0    3.97e10 (4.89%)    8.10e11
7.22e8    1.24e9 (42.33%)    1.94e9
```

- The minimum value is 7.22e08 and the maximum 1.94e09 where the mean is 1.24e09

LU measurement summary for 32 processors, class B, stalled cycles on any resource, box plot for RHS region



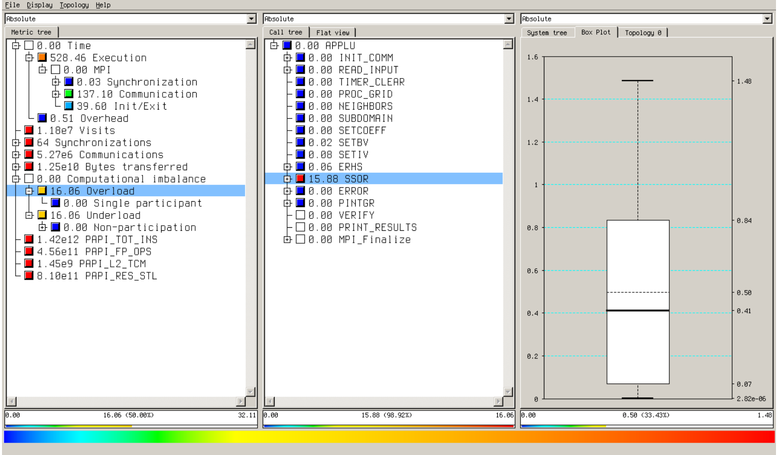
```
0.00 Time
  528.46 Execution
    0.00 MPI
      0.03 Synchronization
      137.10 Communication
      39.60 Init/Exit
    0.51 Overhead
  1.18e7 Visits
  64 Synchronizations
  5.27e6 Communications
  1.25e10 Bytes transferred
  32.11 Computational imbalance
  1.42e12 PAPI_TOT_INS
  4.56e11 PAPI_FP_OPS
  1.45e9 PAPI_L2_TCM
  8.10e11 PAPI_RES_STL
```

```
1.94e6 APPLU
  3.59e8 INIT_COMM
  1.59e6 READ_INPUT
  9.04e4 TIMER_CLEAR
  2.21e4 PROC_GRID
  1.31e4 NEIGHBORS
  1.49e4 SUBDOMAIN
  1.78e4 SETCOEFF
  1.69e7 SETBV
  1.24e9 SETIV
  8.57e8 ERHS
  3.97e10 SSOR
    3.81e11 RHS
      4.41e10 EXCHANGE_3
    2.66e8 L2NORM
    2.10e5 TIMER_CLEAR
    3.71e6 MPI_Barrier
    5.82e5 TIMER_START
    5.90e10 JACLD
    1.07e11 BLTS
    5.52e10 JACU
    1.21e11 BUTS
    2.80e5 TIMER_STOP
    2.96e4 TIMER_READ
    1.10e7 MPI_Allreduce
  1.31e8 ERROR
  9.32e6 PINTGR
  7.39e4 VERIFY
  1.24e4 PRINT_RESULTS
  4.76e6 MPI_Finalize
```

```
0        8.10e11 (100.00%)        8.10e11
0        3.81e11 (47.08%)         8.10e11
0                                 3.81e11
```

- The minimum value is 9.41e09 and the maximum 1.39e10 where the mean is 1.19e10

LU measurement summary for 32 processors, class B, computational imbalance, overload, box plot



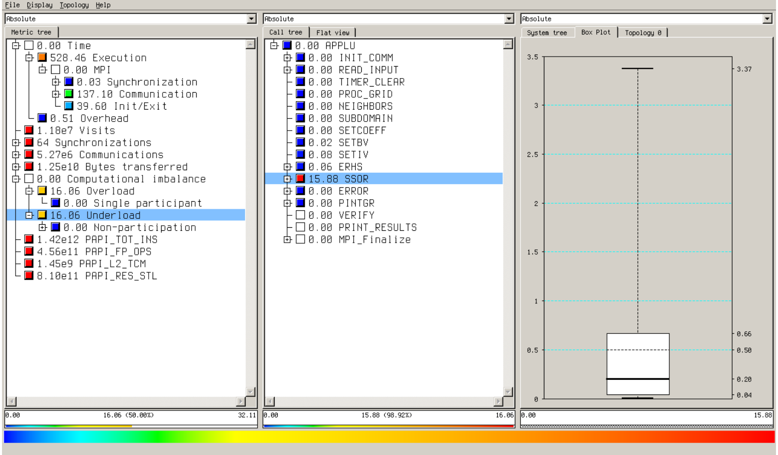
```
0.00 Time
  528.46 Execution
    0.00 MPI
      0.03 Synchronization
      137.10 Communication
      39.60 Init/Exit
    0.51 Overhead
1.18e7 Visits
64 Synchronizations
5.27e6 Communications
1.25e10 Bytes transferred
0.00 Computational imbalance
  16.06 Overload
    0.00 Single participant
  16.06 Underload
    0.00 Non-participation
1.42e12 PAPI_TOT_INS
4.56e11 PAPI_FP_OPS
1.45e9 PAPI_L2_TCM
8.10e11 PAPI_RES_STL
```

```
0.00 APPLU
  0.00 INIT_COMM
  0.00 READ_INPUT
  0.00 TIMER_CLEAR
  0.00 PROC_GRID
  0.00 NEIGHBORS
  0.00 SUBDOMAIN
  0.00 SETCOEFF
  0.02 SETBV
  0.08 SETIV
  0.06 ERHS
  15.88 SSOR
  0.00 ERROR
  0.00 PINTGR
  0.00 VERIFY
  0.00 PRINT_RESULTS
  0.00 MPI_Finalize
```

```
0.00        16.06 (50.00%)        32.11
0.00        15.88 (98.92%)        16.06
0.00        0.50 (33.43%)         1.48
```

- Overload means that the execution time is bigger than the average value of all the processes. There is no single participant, so this overload is caused by more than one process

LU measurement summary for 32 processors, class B, computational imbalance, underload, box plot



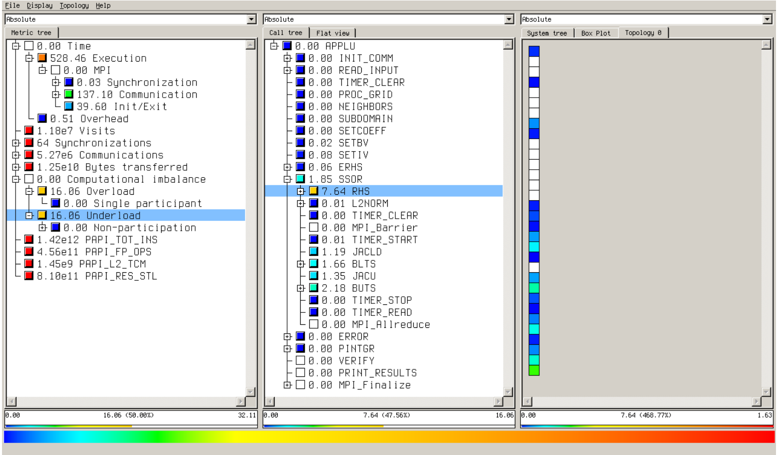
```
0.00 Time
  528.46 Execution
    0.00 MPI
      0.03 Synchronization
      137.10 Communication
      39.60 Init/Exit
    0.51 Overhead
1.18e7 Visits
64 Synchronizations
5.27e6 Communications
1.25e10 Bytes transferred
0.00 Computational imbalance
  16.06 Overload
    0.00 Single participant
  16.06 Underload
    0.00 Non-participation
1.42e12 PAPI_TOT_INS
4.56e11 PAPI_FP_OPS
1.45e9 PAPI_L2_TCM
8.10e11 PAPI_RES_STL
```

```
0.00 APPLU
  0.00 INIT_COMM
  0.00 READ_INPUT
  0.00 TIMER_CLEAR
  0.00 PROC_GRID
  0.00 NEIGHBORS
  0.00 SUBDOMAIN
  0.00 SETCOEFF
  0.02 SETBV
  0.08 SETIV
  0.06 ERHS
  15.88 SSOR
  0.00 ERROR
  0.00 PINTGR
  0.00 VERIFY
  0.00 PRINT_RESULTS
  0.00 MPI_Finalize
```

```
0.00        16.06 (50.00%)        32.11
0.00        15.88 (98.92%)        16.06
0.00                              15.88
```

- Underload means that the execution time is less than the average value of all the processes. There is no non-participant, so all the processes execute the underloaded call-path

LU measurement summary for 32 processors, class B, computational imbalance, box plot for RHS region



```
0.00 Time
  528.46 Execution
    0.00 MPI
      0.03 Synchronization
      137.10 Communication
      39.60 Init/Exit
  0.51 Overhead
1.18e7 Visits
64 Synchronizations
5.27e6 Communications
1.25e10 Bytes transferred
0.00 Computational imbalance
  16.06 Overload
    0.00 Single participant
  16.06 Underload
    0.00 Non-participation
1.42e12 PAPI_TOT_INS
4.56e11 PAPI_FP_OPS
1.45e9 PAPI_L2_TCM
8.10e11 PAPI_RES_STL
```

```
0.00 APPLU
  0.00 INIT_COMM
  0.00 READ_INPUT
  0.00 TIMER_CLEAR
  0.00 PROC_GRID
  0.00 NEIGHBORS
  0.00 SUBDOMAIN
  0.00 SETCOEFF
  0.02 SETBV
  0.08 SETIV
  0.06 ERHS
  1.85 SSOR
    7.64 RHS
    0.01 L2NORM
    0.00 TIMER_CLEAR
    0.00 MPI_Barrier
    0.01 TIMER_START
    1.19 JACLD
    1.66 BLTS
    1.35 JACU
    2.18 BUTS
    0.00 TIMER_STOP
    0.00 TIMER_READ
    0.00 MPI_Allreduce
  0.00 ERROR
  0.00 PINTGR
  0.00 VERIFY
  0.00 PRINT_RESULTS
  0.00 MPI_Finalize
```

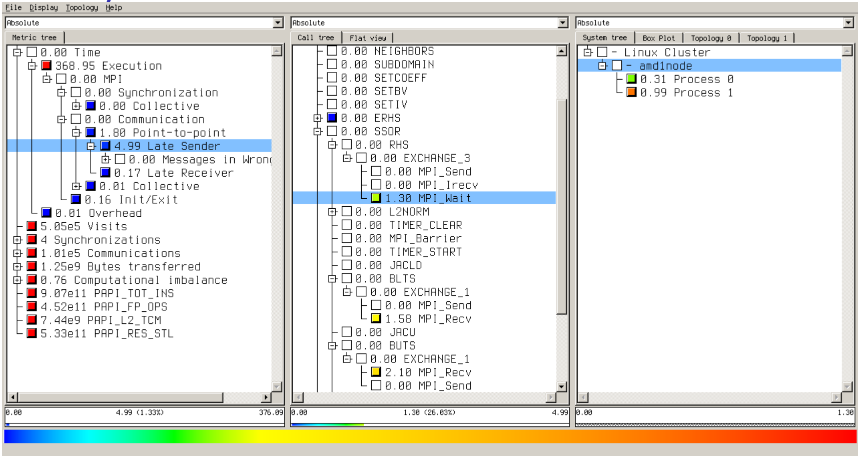
```
0.00            16.06 (50.00%)            32.11
0.00            7.64 (47.56%)             16.06
0.00            7.64 (468.77%)            1.63
```

- We can observe that the underload value for the last process is big enough in comparison with the rest ones

Overal conclusions from the summary measurements

- The floating operations remain stable but not the total completed instructions. The variation on Intel processor or older AMD Opterons is not so big as on this AMD Opteron
- The role of the communication is important while we increase the number of the processes
- The computation time is increasing while we increase the number of the processes

LU measurement trace for 2 processors, class B, identify communication issues



```
□ 0.00 Time
  🟥 368.95 Execution
    □ 0.00 MPI
      □ 0.00 Synchronization
        🟦 0.00 Collective
      □ 0.00 Communication
        🟦 1.80 Point-to-point
          🟦 4.99 Late Sender
            □ 0.00 Messages in Wrong
          🟦 0.17 Late Receiver
        🟦 0.01 Collective
      🟦 0.16 Init/Exit
    🟦 0.01 Overhead
🟥 5.05e5 Visits
🟥 4 Synchronizations
🟥 1.01e5 Communications
🟥 1.25e9 Bytes transferred
🟥 0.76 Computational imbalance
🟥 9.07e11 PAPI_TOT_INS
🟥 4.52e11 PAPI_FP_OPS
🟥 7.44e9 PAPI_L2_TCM
🟥 5.33e11 PAPI_RES_STL
```

```
0.00          4.99 (1.33%)          376.09
```

```
□ 0.00 NEIGHBORS
□ 0.00 SUBDOMAIN
□ 0.00 SETCOEFF
□ 0.00 SETBV
□ 0.00 SETIV
🟦 0.00 ERHS
□ 0.00 SSOR
  □ 0.00 RHS
    □ 0.00 EXCHANGE_3
      □ 0.00 MPI_Send
      □ 0.00 MPI_Irecv
      🟩 1.30 MPI_Wait
  □ 0.00 L2NORM
  □ 0.00 TIMER_CLEAR
  □ 0.00 MPI_Barrier
  □ 0.00 TIMER_START
  □ 0.00 JACLD
  □ 0.00 BLTS
    □ 0.00 EXCHANGE_1
      □ 0.00 MPI_Send
      🟨 1.58 MPI_Recv
  □ 0.00 JACU
  □ 0.00 BUTS
    □ 0.00 EXCHANGE_1
      🟨 2.10 MPI_Recv
      □ 0.00 MPI_Send
```

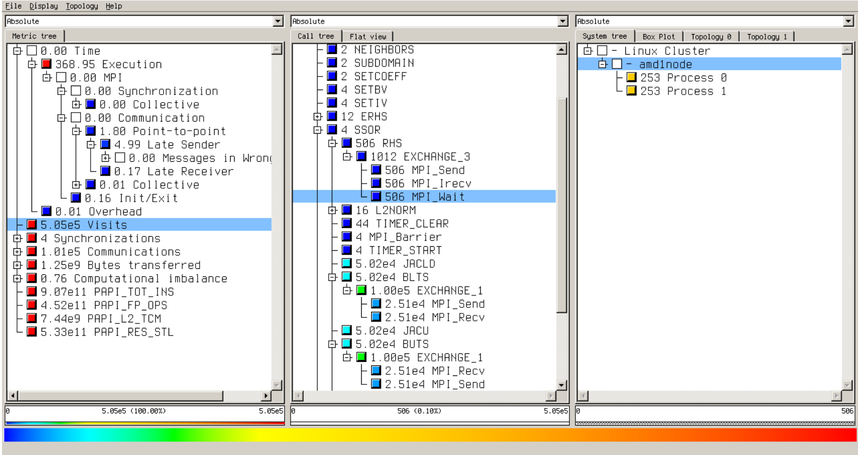
```
0.00          1.30 (26.03%)          4.99
```

```
□ - Linux Cluster
  □ - amd1node
    🟩 0.31 Process 0
    🟧 0.99 Process 1
```

```
0.00                                 1.30
```

- The late sender measures the lost time which is caused by a blocking receive operation which is posted earlier than the corresponding send operation
- The process 0 delays the execution for 0.31 seconds because of the MPI_Wait and the process 1 for 0.99 seconds

LU measurement trace for 2 processors, class B, visits



Absolute

Metric tree

```
□ 0.00 Time
  🟥 368.95 Execution
    □ 0.00 MPI
      □ 0.00 Synchronization
        🟦 0.00 Collective
      □ 0.00 Communication
        🟦 1.80 Point-to-point
          🟦 4.99 Late Sender
            □ 0.00 Messages in Wrong
          🟦 0.17 Late Receiver
        🟦 0.01 Collective
      🟦 0.16 Init/Exit
    🟦 0.01 Overhead
🟥 5.05e5 Visits
🟥 4 Synchronizations
🟥 1.01e5 Communications
🟥 1.25e9 Bytes transferred
🟥 0.76 Computational imbalance
🟥 9.07e11 PAPI_TOT_INS
🟥 4.52e11 PAPI_FP_OPS
🟥 7.44e9 PAPI_L2_TCM
🟥 5.33e11 PAPI_RES_STL
```

0 5.05e5 (100.00%) 5.05e5

Absolute

Call tree | Flat view

```
🟦 2 NEIGHBORS
🟦 2 SUBDOMAIN
🟦 2 SETCOEFF
🟦 4 SETBV
🟦 4 SETIV
🟦 12 ERHS
🟦 4 SSOR
  🟦 506 RHS
    🟦 1012 EXCHANGE_3
      🟦 506 MPI_Send
      🟦 506 MPI_Irecv
      🟦 506 MPI_Wait
  🟦 16 L2NORM
  🟦 44 TIMER_CLEAR
  🟦 4 MPI_Barrier
  🟦 4 TIMER_START
  🟦 5.02e4 JACLD
  🟦 5.02e4 BLTS
    🟩 1.00e5 EXCHANGE_1
      🟦 2.51e4 MPI_Send
      🟦 2.51e4 MPI_Recv
  🟦 5.02e4 JACU
  🟦 5.02e4 BUTS
    🟩 1.00e5 EXCHANGE_1
      🟦 2.51e4 MPI_Recv
      🟦 2.51e4 MPI_Send
```

0 506 (0.10%) 5.05e5

Absolute

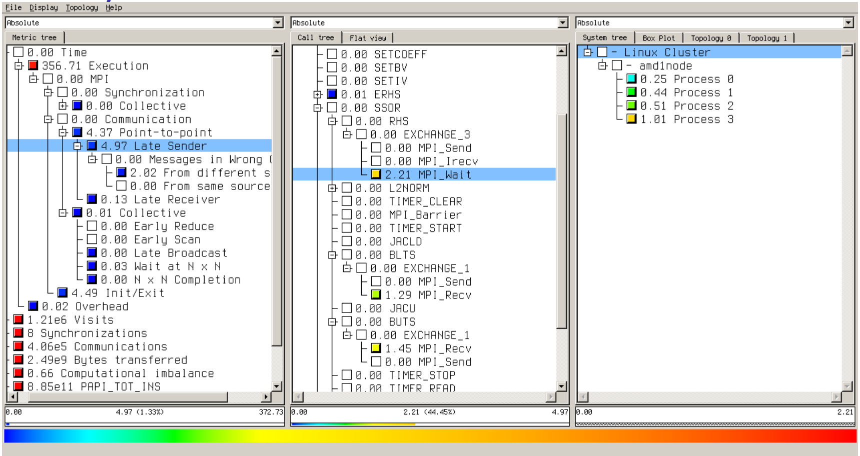
System tree | Box Plot | Topology 0 | Topology 1

```
□ - Linux Cluster
  □ - amd1node
    🟨 253 Process 0
    🟨 253 Process 1
```

0 506

- We can observe the number of the visits for each function, for example there are 506 calls to MPI_Wait for the call-path RHS - EXCHANGE_3

LU measurement trace for 4 processors, class B, identify communication issues



```
⊟ □ 0.00 Time
  ⊟ ■ 356.71 Execution
    ⊟ □ 0.00 MPI
      ⊟ □ 0.00 Synchronization
        ⊞ ■ 0.00 Collective
      ⊟ □ 0.00 Communication
        ⊟ ■ 4.37 Point-to-point
          ⊟ ■ 4.97 Late Sender
            ⊟ □ 0.00 Messages in Wrong (
              ├ ■ 2.02 From different s
              └ □ 0.00 From same source
          └ ■ 0.13 Late Receiver
        ⊟ ■ 0.01 Collective
          ├ □ 0.00 Early Reduce
          ├ □ 0.00 Early Scan
          ├ ■ 0.00 Late Broadcast
          ├ ■ 0.03 Wait at N x N
          └ ■ 0.00 N x N Completion
      └ ■ 4.49 Init/Exit
    └ ■ 0.02 Overhead
■ 1.21e6 Visits
■ 8 Synchronizations
■ 4.06e5 Communications
■ 2.49e9 Bytes transferred
■ 0.66 Computational imbalance
■ 8.85e11 PAPI_TOT_INS
```

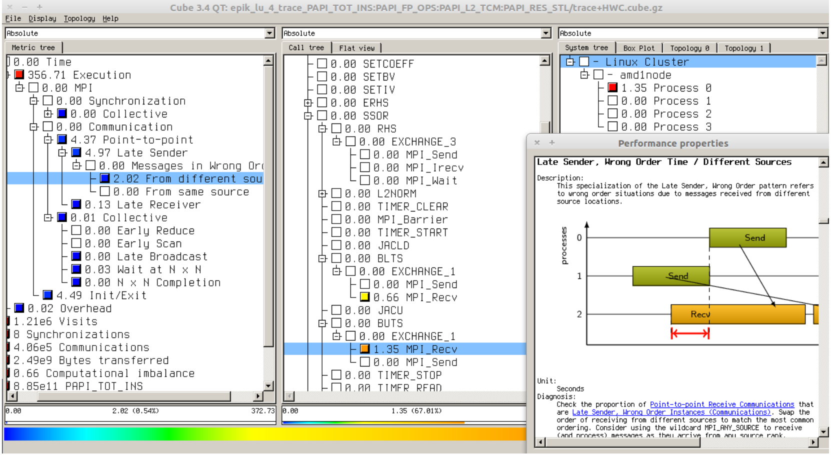
```
├ □ 0.00 SETCOEFF
├ □ 0.00 SETBV
├ □ 0.00 SETIV
⊞ ■ 0.01 ERHS
⊟ □ 0.00 SSOR
  ⊟ □ 0.00 RHS
    ⊟ □ 0.00 EXCHANGE_3
      ├ □ 0.00 MPI_Send
      ├ □ 0.00 MPI_Irecv
      └ ■ 2.21 MPI_Wait
  ⊞ □ 0.00 L2NORM
  ├ □ 0.00 TIMER_CLEAR
  ├ □ 0.00 MPI_Barrier
  ├ □ 0.00 TIMER_START
  ├ □ 0.00 JACLD
  ⊟ □ 0.00 BLTS
    ⊟ □ 0.00 EXCHANGE_1
      ├ □ 0.00 MPI_Send
      └ ■ 1.29 MPI_Recv
  ├ □ 0.00 JACU
  ⊟ □ 0.00 BUTS
    ⊟ □ 0.00 EXCHANGE_1
      ├ ■ 1.45 MPI_Recv
      └ □ 0.00 MPI_Send
  ├ □ 0.00 TIMER_STOP
  ├ □ 0.00 TIMER_READ
```

```
⊟ □ - Linux Cluster
  ⊟ □ - amd1node
    ├ ■ 0.25 Process 0
    ├ ■ 0.44 Process 1
    ├ ■ 0.51 Process 2
    └ ■ 1.01 Process 3
```

```
0.00        4.97 (1.33%)        372.73
0.00        2.21 (44.45%)       4.97
0.00                            2.21
```

- The lost time because of the Late Sender is almost the same as in the previous case
- However we have a case of different sources which its duration is 2.02 seconds

LU measurement trace for 4 processors, class B, identify communication issues, late sender from different sources



```
0.00 Time
356.71 Execution
  0.00 MPI
    0.00 Synchronization
      0.00 Collective
    0.00 Communication
      4.37 Point-to-point
        4.97 Late Sender
          0.00 Messages in Wrong Or
            2.02 From different sou
            0.00 From same source
        0.13 Late Receiver
      0.01 Collective
        0.00 Early Reduce
        0.00 Early Scan
        0.00 Late Broadcast
        0.03 Wait at N x N
        0.00 N x N Completion
    4.49 Init/Exit
0.02 Overhead
1.21e6 Visits
8 Synchronizations
4.06e5 Communications
2.49e9 Bytes transferred
0.66 Computational imbalance
8.85e11 PAPI_TOT_INS
```

```
0.00                 2.02 (0.54%)               372.73
```

```
0.00 SETCOEFF
0.00 SETBV
0.00 SETIV
0.00 ERHS
0.00 SSOR
  0.00 RHS
    0.00 EXCHANGE_3
      0.00 MPI_Send
      0.00 MPI_Irecv
      0.00 MPI_Wait
  0.00 L2NORM
  0.00 TIMER_CLEAR
  0.00 MPI_Barrier
  0.00 TIMER_START
  0.00 JACLD
  0.00 BLTS
    0.00 EXCHANGE_1
      0.00 MPI_Send
      0.66 MPI_Recv
  0.00 JACU
  0.00 BUTS
    0.00 EXCHANGE_1
      1.35 MPI_Recv
      0.00 MPI_Send
  0.00 TIMER_STOP
  0.00 TIMER_READ
```

```
0.00                 1.35 (67.01%)
```

```
- Linux Cluster
  - amd1node
    1.35 Process 0
    0.00 Process 1
    0.00 Process 2
    0.00 Process 3
```

Late Sender, Wrong Order Time / Different Sources

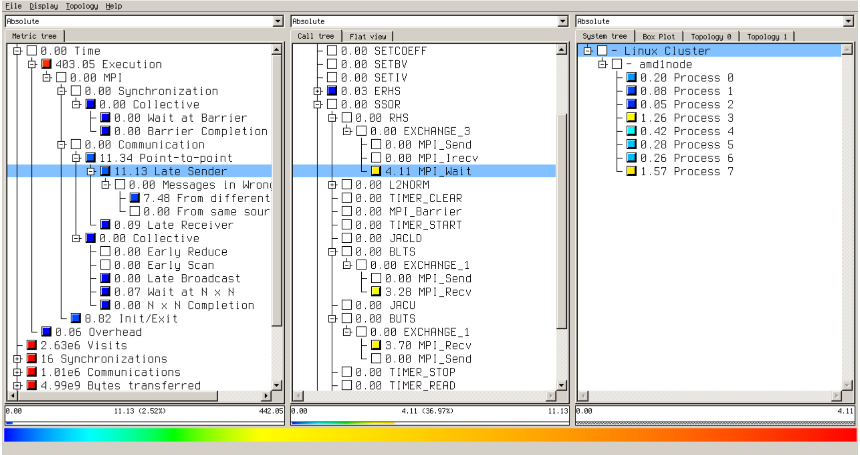
Description:
This specialization of the Late Sender, Wrong Order pattern refers to wrong order situations due to messages received from different source locations.

Unit:
Seconds

Diagnosis:
Check the proportion of Point-to-point Receive Communications that are Late Sender, Wrong Order Instances (Communications). Swap the order of receiving from different sources to match the most common ordering. Consider using the wildcard MPI_ANY_SOURCE to receive (and process) messages as they arrive from any source rank.

- In this case or we should reverse the sequence of the MPI_Recv calls in order to avoid this phenomenon or to use the MPI_ANY_SOURCE tag

LU measurement trace for 8 processors, class B, identify communication issues



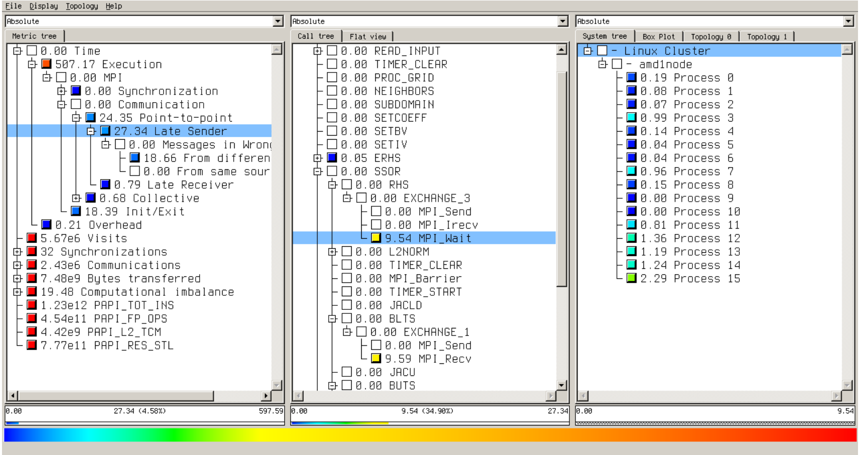
```
Absolute
⊟ □ 0.00 Time
  ⊟ ■ 403.05 Execution
    ⊟ □ 0.00 MPI
      ⊟ □ 0.00 Synchronization
        ⊟ ■ 0.00 Collective
          ├ ■ 0.00 Wait at Barrier
          └ ■ 0.00 Barrier Completion
      ⊟ □ 0.00 Communication
        ⊟ ■ 11.34 Point-to-point
          ⊟ ■ 11.13 Late Sender
            ⊟ □ 0.00 Messages in Wrong
              ├ ■ 7.48 From different
              └ □ 0.00 From same sour
          └ ■ 0.09 Late Receiver
        ⊟ ■ 0.00 Collective
          ├ □ 0.00 Early Reduce
          ├ □ 0.00 Early Scan
          ├ ■ 0.00 Late Broadcast
          ├ ■ 0.07 Wait at N x N
          └ ■ 0.00 N x N Completion
      └ ■ 8.82 Init/Exit
    └ ■ 0.06 Overhead
  ■ 2.63e6 Visits
⊞ ■ 16 Synchronizations
⊞ ■ 1.01e6 Communications
⊞ ■ 4.99e9 Bytes transferred

0.00              11.13 (2.52%)              442.05
```

```
Absolute
├ □ 0.00 SETCOEFF
├ □ 0.00 SETBV
├ □ 0.00 SETIV
⊞ ■ 0.03 ERHS
⊟ □ 0.00 SSOR
  ⊟ □ 0.00 RHS
    ⊟ □ 0.00 EXCHANGE_3
      ├ □ 0.00 MPI_Send
      ├ □ 0.00 MPI_Irecv
      └ ■ 4.11 MPI_Wait
  ⊞ □ 0.00 L2NORM
  ├ □ 0.00 TIMER_CLEAR
  ├ □ 0.00 MPI_Barrier
  ├ □ 0.00 TIMER_START
  ├ □ 0.00 JACLD
  ⊟ □ 0.00 BLTS
    ⊟ □ 0.00 EXCHANGE_1
      ├ □ 0.00 MPI_Send
      └ ■ 3.28 MPI_Recv
  ├ □ 0.00 JACU
  ⊟ □ 0.00 BUTS
    ⊟ □ 0.00 EXCHANGE_1
      ├ ■ 3.70 MPI_Recv
      └ □ 0.00 MPI_Send
  ├ □ 0.00 TIMER_STOP
  ├ □ 0.00 TIMER_READ

0.00              4.11 (36.97%)              11.13
```

```
Absolute
⊟ □ - Linux Cluster
  ⊟ □ - amd1node
    ├ ■ 0.20 Process 0
    ├ ■ 0.08 Process 1
    ├ ■ 0.05 Process 2
    ├ ■ 1.26 Process 3
    ├ ■ 0.42 Process 4
    ├ ■ 0.28 Process 5
    ├ ■ 0.26 Process 6
    └ ■ 1.57 Process 7

0.00                                          4.11
```

- The maximum duration of the Late Sender starts to increase (1.57 seconds for process 7)

LU measurement trace for 16 processors, class B, identify communication issues



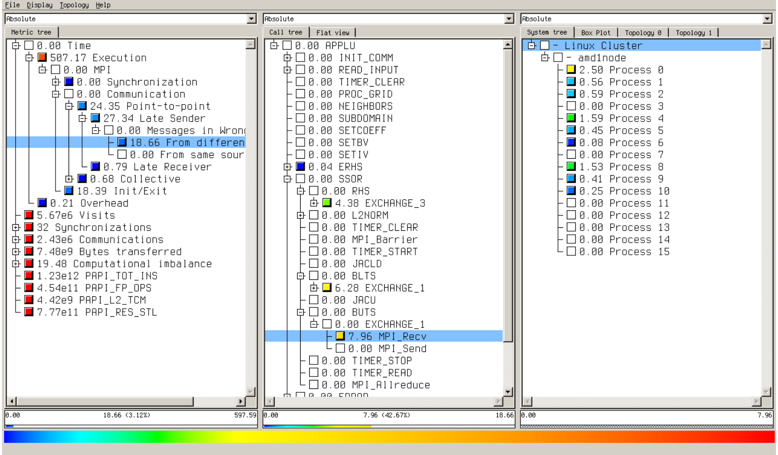
```
Absolute
Metric tree
□ 0.00 Time
  ■ 507.17 Execution
    □ 0.00 MPI
      ■ 0.00 Synchronization
      □ 0.00 Communication
        ■ 24.35 Point-to-point
          ■ 27.34 Late Sender
            □ 0.00 Messages in Wrong
              ■ 18.66 From differen
              □ 0.00 From same sour
          ■ 0.79 Late Receiver
        ■ 0.68 Collective
      ■ 18.39 Init/Exit
  ■ 0.21 Overhead
■ 5.67e6 Visits
■ 32 Synchronizations
■ 2.43e6 Communications
■ 7.48e9 Bytes transferred
■ 19.48 Computational imbalance
■ 1.23e12 PAPI_TOT_INS
■ 4.54e11 PAPI_FP_OPS
■ 4.42e9 PAPI_L2_TCM
■ 7.77e11 PAPI_RES_STL
0.00          27.34 (4.58%)          597.59
```

```
Absolute
Call tree | Flat view
□ 0.00 READ_INPUT
□ 0.00 TIMER_CLEAR
□ 0.00 PROC_GRID
□ 0.00 NEIGHBORS
□ 0.00 SUBDOMAIN
□ 0.00 SETCOEFF
□ 0.00 SETBV
□ 0.00 SETIV
■ 0.05 ERHS
□ 0.00 SSOR
  □ 0.00 RHS
    □ 0.00 EXCHANGE_3
      □ 0.00 MPI_Send
      □ 0.00 MPI_Irecv
      ■ 9.54 MPI_Wait
  □ 0.00 L2NORM
  □ 0.00 TIMER_CLEAR
  □ 0.00 MPI_Barrier
  □ 0.00 TIMER_START
  □ 0.00 JACLD
  □ 0.00 BLTS
    □ 0.00 EXCHANGE_1
      □ 0.00 MPI_Send
      ■ 9.59 MPI_Recv
  □ 0.00 JACU
  □ 0.00 BUTS
0.00          9.54 (34.90%)          27.34
```

```
Absolute
System tree | Box Plot | Topology 0 | Topology 1
□ - Linux Cluster
  □ - amd1node
    ■ 0.19 Process 0
    ■ 0.08 Process 1
    ■ 0.07 Process 2
    ■ 0.99 Process 3
    ■ 0.14 Process 4
    ■ 0.04 Process 5
    ■ 0.04 Process 6
    ■ 0.96 Process 7
    ■ 0.15 Process 8
    ■ 0.00 Process 9
    ■ 0.00 Process 10
    ■ 0.81 Process 11
    ■ 1.36 Process 12
    ■ 1.19 Process 13
    ■ 1.24 Process 14
    ■ 2.29 Process 15
0.00                                 9.54
```

- Similar, the maximum duration of the Late Sender increases (2.29 seconds for process 15)

LU measurement trace for 16 processors, class B, identify communication issues, late sender from different sources



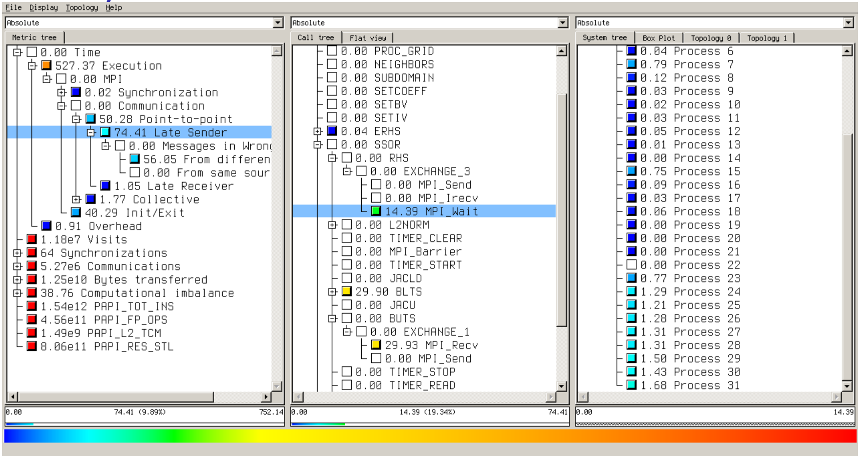
```
□ 0.00 Time
  ■ 507.17 Execution
    □ 0.00 MPI
      ■ 0.00 Synchronization
      □ 0.00 Communication
        ■ 24.35 Point-to-point
          ■ 27.34 Late Sender
            □ 0.00 Messages in Wrong
              ■ 18.66 From differen
              □ 0.00 From same sour
          ■ 0.79 Late Receiver
        ■ 0.68 Collective
      ■ 18.39 Init/Exit
    ■ 0.21 Overhead
  ■ 5.67e6 Visits
  ■ 32 Synchronizations
  ■ 2.43e6 Communications
  ■ 7.48e9 Bytes transferred
  ■ 19.48 Computational imbalance
  ■ 1.23e12 PAPI_TOT_INS
  ■ 4.54e11 PAPI_FP_OPS
  ■ 4.42e9 PAPI_L2_TCM
  ■ 7.77e11 PAPI_RES_STL

0.00          18.66 (3.12%)          597.59
```

```
□ 0.00 APPLU
  □ 0.00 INIT_COMM
  □ 0.00 READ_INPUT
  □ 0.00 TIMER_CLEAR
  □ 0.00 PROC_GRID
  □ 0.00 NEIGHBORS
  □ 0.00 SUBDOMAIN
  □ 0.00 SETCOEFF
  □ 0.00 SETBV
  □ 0.00 SETIV
  ■ 0.04 ERHS
  □ 0.00 SSOR
    □ 0.00 RHS
      ■ 4.38 EXCHANGE_3
    □ 0.00 L2NORM
    □ 0.00 TIMER_CLEAR
    □ 0.00 MPI_Barrier
    □ 0.00 TIMER_START
    □ 0.00 JACLD
    □ 0.00 BLTS
      ■ 6.28 EXCHANGE_1
    □ 0.00 JACU
    □ 0.00 BUTS
      □ 0.00 EXCHANGE_1
        ■ 7.96 MPI_Recv
        □ 0.00 MPI_Send
    □ 0.00 TIMER_STOP
    □ 0.00 TIMER_READ
    □ 0.00 MPI_Allreduce
  □ 0.00 ERROR

0.00          7.96 (42.67%)          18.66
```

```
□ - Linux Cluster
  □ - amd1node
    ■ 2.50 Process 0
    ■ 0.56 Process 1
    ■ 0.59 Process 2
    □ 0.00 Process 3
    ■ 1.59 Process 4
    ■ 0.45 Process 5
    ■ 0.08 Process 6
    □ 0.00 Process 7
    ■ 1.53 Process 8
    ■ 0.41 Process 9
    ■ 0.25 Process 10
    □ 0.00 Process 11
    □ 0.00 Process 12
    □ 0.00 Process 13
    □ 0.00 Process 14
    □ 0.00 Process 15

0.00                                   7.96
```

- The maximum delay because of the wrong sequence of the MPI_Recv calls is 2.5 seconds for process 0

LU measurement trace for 32 processors, class B, identify communication issues



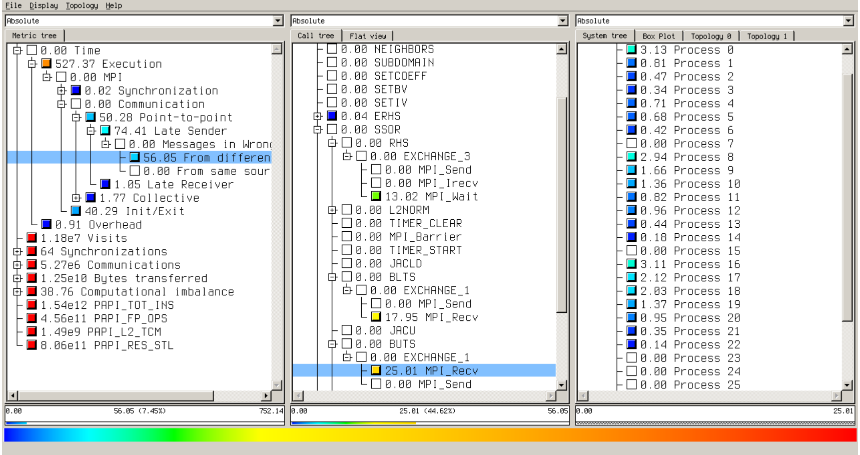
```
Absolute
Metric tree
□ 0.00 Time
  ■ 527.37 Execution
    □ 0.00 MPI
      ■ 0.02 Synchronization
      □ 0.00 Communication
        ■ 50.28 Point-to-point
          ■ 74.41 Late Sender
            □ 0.00 Messages in Wrong
              ■ 56.05 From differen
              □ 0.00 From same sour
          ■ 1.05 Late Receiver
        ■ 1.77 Collective
      ■ 40.29 Init/Exit
  ■ 0.91 Overhead
■ 1.18e7 Visits
■ 64 Synchronizations
■ 5.27e6 Communications
■ 1.25e10 Bytes transferred
■ 38.76 Computational imbalance
■ 1.54e12 PAPI_TOT_INS
■ 4.56e11 PAPI_FP_OPS
■ 1.49e9 PAPI_L2_TCM
■ 8.06e11 PAPI_RES_STL

0.00          74.41 (9.89%)          752.14
```

```
Absolute
Call tree | Flat view
□ 0.00 PROC_GRID
□ 0.00 NEIGHBORS
□ 0.00 SUBDOMAIN
□ 0.00 SETCOEFF
□ 0.00 SETBV
□ 0.00 SETIV
■ 0.04 ERHS
□ 0.00 SSOR
  □ 0.00 RHS
    □ 0.00 EXCHANGE_3
      □ 0.00 MPI_Send
      □ 0.00 MPI_Irecv
      ■ 14.39 MPI_Wait
  □ 0.00 L2NORM
  □ 0.00 TIMER_CLEAR
  □ 0.00 MPI_Barrier
  □ 0.00 TIMER_START
  □ 0.00 JACLD
  ■ 29.90 BLTS
  □ 0.00 JACU
  □ 0.00 BUTS
    □ 0.00 EXCHANGE_1
      ■ 29.93 MPI_Recv
      □ 0.00 MPI_Send
  □ 0.00 TIMER_STOP
  □ 0.00 TIMER_READ

0.00          14.39 (19.34%)          74.41
```

```
Absolute
System tree | Box Plot | Topology 0 | Topology 1
■ 0.04 Process 6
■ 0.79 Process 7
■ 0.12 Process 8
■ 0.03 Process 9
■ 0.02 Process 10
■ 0.03 Process 11
■ 0.05 Process 12
■ 0.01 Process 13
■ 0.00 Process 14
■ 0.75 Process 15
■ 0.09 Process 16
■ 0.03 Process 17
■ 0.06 Process 18
■ 0.00 Process 19
■ 0.00 Process 20
■ 0.00 Process 21
□ 0.00 Process 22
■ 0.77 Process 23
■ 1.29 Process 24
■ 1.21 Process 25
■ 1.28 Process 26
■ 1.31 Process 27
■ 1.31 Process 28
■ 1.50 Process 29
■ 1.43 Process 30
■ 1.68 Process 31

0.00                                  14.39
```

- While in the previous cases the delay of the MPI_Wait was bigger than the other MPI calls now it is not. It is crucial to study the other call paths as the delay is 29.93 seconds for the MPI_Recv of the EXCHANGE_1 region

LU measurement trace for 32 processors, class B, identify communication issues, late sender from different sources



```
⊟ ☐ 0.00 Time
   ⊟ 🟧 527.37 Execution
      ⊟ ☐ 0.00 MPI
         ⊞ 🟦 0.02 Synchronization
         ⊟ ☐ 0.00 Communication
            ⊟ 🟦 50.28 Point-to-point
               ⊟ 🟦 74.41 Late Sender
                  ⊟ ☐ 0.00 Messages in Wrong
                     🟦 56.05 From differen
                     ☐ 0.00 From same sour
               🟦 1.05 Late Receiver
            ⊞ 🟦 1.77 Collective
         🟦 40.29 Init/Exit
      🟦 0.91 Overhead
🟥 1.18e7 Visits
⊞ 🟥 64 Synchronizations
⊞ 🟥 5.27e6 Communications
⊞ 🟥 1.25e10 Bytes transferred
⊞ 🟥 38.76 Computational imbalance
🟥 1.54e12 PAPI_TOT_INS
🟥 4.56e11 PAPI_FP_OPS
🟥 1.49e9 PAPI_L2_TCM
🟥 8.06e11 PAPI_RES_STL
```

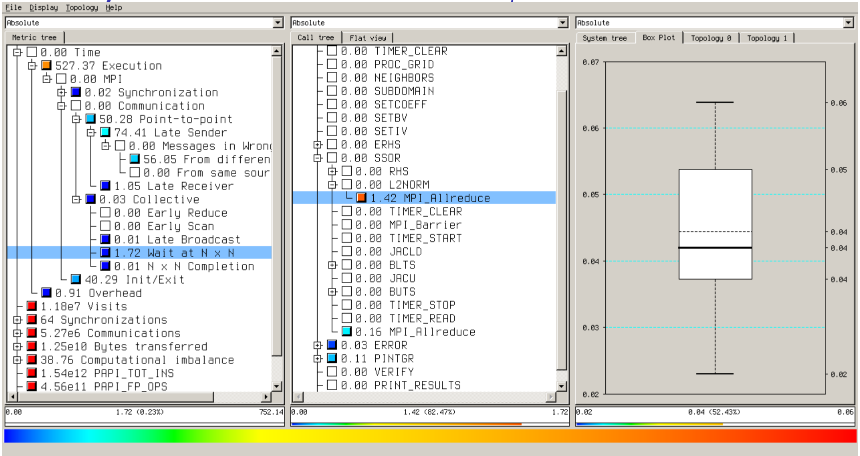
```
☐ 0.00 NEIGHBORS
☐ 0.00 SUBDOMAIN
☐ 0.00 SETCOEFF
☐ 0.00 SETBV
☐ 0.00 SETIV
⊞ 🟦 0.04 ERHS
⊟ ☐ 0.00 SSOR
   ⊟ ☐ 0.00 RHS
      ⊟ ☐ 0.00 EXCHANGE_3
         ☐ 0.00 MPI_Send
         ☐ 0.00 MPI_Irecv
         🟩 13.02 MPI_Wait
   ⊞ ☐ 0.00 L2NORM
   ☐ 0.00 TIMER_CLEAR
   ☐ 0.00 MPI_Barrier
   ☐ 0.00 TIMER_START
   ☐ 0.00 JACLD
   ⊟ ☐ 0.00 BLTS
      ⊟ ☐ 0.00 EXCHANGE_1
         ☐ 0.00 MPI_Send
         🟨 17.95 MPI_Recv
   ☐ 0.00 JACU
   ⊟ ☐ 0.00 BUTS
      ⊟ ☐ 0.00 EXCHANGE_1
         🟨 25.01 MPI_Recv
         ☐ 0.00 MPI_Send
```

```
🟦 3.13 Process 0
🟦 0.81 Process 1
🟦 0.47 Process 2
🟦 0.34 Process 3
🟦 0.71 Process 4
🟦 0.68 Process 5
🟦 0.42 Process 6
☐ 0.00 Process 7
🟦 2.94 Process 8
🟦 1.66 Process 9
🟦 1.36 Process 10
🟦 0.82 Process 11
🟦 0.96 Process 12
🟦 0.44 Process 13
🟦 0.18 Process 14
☐ 0.00 Process 15
🟦 3.11 Process 16
🟦 2.12 Process 17
🟦 2.03 Process 18
🟦 1.37 Process 19
🟦 0.95 Process 20
🟦 0.35 Process 21
🟦 0.14 Process 22
☐ 0.00 Process 23
☐ 0.00 Process 24
☐ 0.00 Process 25
```

```
0.00          56.05 (7.45%)          752.14
0.00          25.01 (44.62%)          56.05
0.00                                  25.01
```

- For this case the duration of the Late Sender is increased and its propotional to the total communication time is increased

LU measurement trace for 32 processors, class B, identify communication issues, wait at NxN



```
0.00 Time
  527.37 Execution
    0.00 MPI
      0.02 Synchronization
      0.00 Communication
        50.28 Point-to-point
          74.41 Late Sender
            0.00 Messages in Wrong
              56.05 From differen
              0.00 From same sour
          1.05 Late Receiver
        0.03 Collective
          0.00 Early Reduce
          0.00 Early Scan
          0.01 Late Broadcast
          1.72 Wait at N x N
          0.01 N x N Completion
      40.29 Init/Exit
    0.91 Overhead
1.18e7 Visits
64 Synchronizations
5.27e6 Communications
1.25e10 Bytes transferred
38.76 Computational imbalance
1.54e12 PAPI_TOT_INS
4.56e11 PAPI_FP_OPS
```

```
0.00 TIMER_CLEAR
0.00 PROC_GRID
0.00 NEIGHBORS
0.00 SUBDOMAIN
0.00 SETCOEFF
0.00 SETBV
0.00 SETIV
0.00 ERHS
0.00 SSOR
  0.00 RHS
  0.00 L2NORM
    1.42 MPI_Allreduce
  0.00 TIMER_CLEAR
  0.00 MPI_Barrier
  0.00 TIMER_START
  0.00 JACLD
  0.00 BLTS
  0.00 JACU
  0.00 BUTS
  0.00 TIMER_STOP
  0.00 TIMER_READ
  0.16 MPI_Allreduce
0.03 ERROR
0.11 PINTGR
0.00 VERIFY
0.00 PRINT_RESULTS
```

```
0.00        1.72 (0.23%)        752.14
0.00        1.42 (82.47%)       1.72
0.02        0.04 (52.43%)       0.06
```

- The total delay time caused by the MPI_Allreduce is 1.42 seconds where the minimum delay is 0.02 seconds, the maximum 0.06 seconds and the mean time is 0.04 seconds.
- The boxplot provides useful information for a lot of processors with an easy way

Conclusions

- As we increase the number of the processors that participate to the execution, the Late Sender delay is becoming bigger and should be fixed by applying a better load balancing on the computation part as some processors finish faster than the others
- Moreover the delay because of the difference of sources is increasing and the proposed ways to be fixed are by changing the sequence of the MPI_Recv calls or use the MPI_ANY_SOURCE

TAU Performance System

TAU

TAU Performance System

- Tuning and Analysis Utilities
- Performance profiling and tracing
- Instrumentation, measurement, analysis, visualization
- Performance data management and data mining
- Easy to integrate in application frameworks

TAU Performance System

- TAU is a performance evaluation tool
- Parallel profiling and tracing
- TAU can automatically instrument your source code through PDT for routines, loops, I/O, memory, phases, etc.
- TAU provides various analysis tools

Simplest Case

- Uninstrumented code:

```
% mpirun -np 4 lu.B.4
```

- With TAU:

```
% mpirun -np 4 tau_exec ./lu.B.4
% paraprof
```

How does TAU work?

- Instrumentation:
  - Adds probes to perform measurements
  - Source code instrumentation
  - Wrapping external libraries (I/O, CUDA, OpenCL)
  - Rewriting the binary executable
- Measurement:
  - Profiling or Tracing
  - Direct instrumentation
  - Indirect instrumentation (sampling)
  - Throttling
  - Per-thread storage of performance data
  - Interface with external packages (PAPI, Scalasca, Score-P, VampirTrace)
- Analysis:
  - Visualization of profiles and traces
  - 3D visualization with paraprof, perfexplorer tools
  - Trace conversion tools

Using TAU: Introduction

- TAU supports several measurement and thread option
- Each measurement configuration of TAU corresponds to a unique stub makefile and library that is generated during the configuration of the tool
- Instrumenting source code automatically using PDT
  - Choose the appropriate TAU stub makefile

    ```
    % export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-pdt
    ```

  - Use tau_f90.sh, tau_cxx.sh, tau_cc.sh as F90, C++ and C compilers

    ```
    mpif90 test.f90 -> tau_f90.sh test.f90
    ```

- Set runtime environment variables, execute application and analyze the data

  ```
  % pprof (text based profile display)
  % paraprof (GUI)
  ```

- **Important**: For calling pprof just execute pprof_tau for avoiding conflict with the pprof tool

TAU Instrumentation Approach

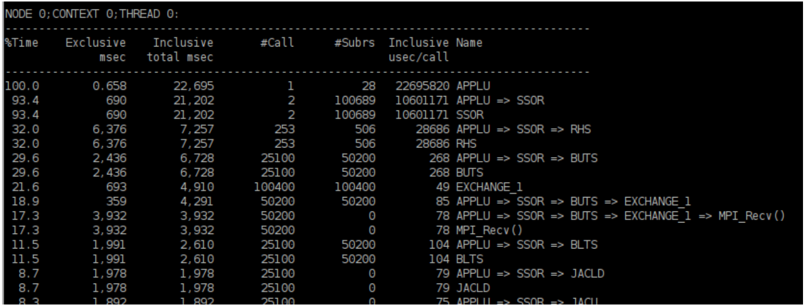
- Supports both firect and indirect performance observation
  - Direct instrumentation of program code
  - Instrumentation invokes performance measurement
  - Event measurement
  - Indirect mode: sampling, hardware performance counter overflow
- User-defined events
  - Interval (Start/stop)
  - Atomic, trigger at a single point with data
  - Context events, atomic events with executing context

Direct Observation: Events

- Event types
  - Interval events
    - Measures exclusive & inclusive duration between events
    - Metrics monotonically increase
  - Atomic events
    - Capture performance data state
    - Shows extent variation of triggered values
- Code events
  - Routines, classes, templates
  - Statement-level blocks, loops

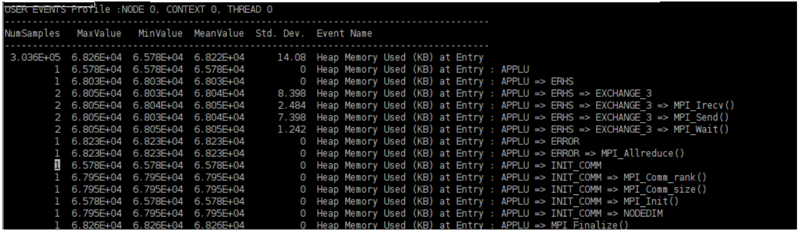
Interval and Atomic events

- Interval events



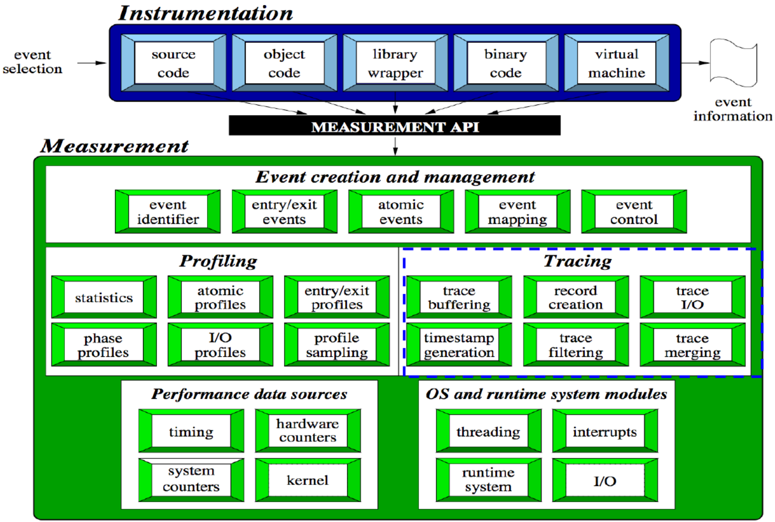
```
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.658       22,695           1          28   22695820 APPLU
 93.4          690       21,202           2      100689   10601171 APPLU => SSOR
 93.4          690       21,202           2      100689   10601171 SSOR
 32.0        6,376        7,257         253         506      28686 APPLU => SSOR => RHS
 32.0        6,376        7,257         253         506      28686 RHS
 29.6        2,436        6,728       25100       50200        268 APPLU => SSOR => BUTS
 29.6        2,436        6,728       25100       50200        268 BUTS
 21.6          693        4,910      100400      100400         49 EXCHANGE_1
 18.9          359        4,291       50200       50200         85 APPLU => SSOR => BUTS => EXCHANGE_1
 17.3        3,932        3,932       50200           0         78 APPLU => SSOR => BUTS => EXCHANGE_1 => MPI_Recv()
 17.3        3,932        3,932       50200           0         78 MPI_Recv()
 11.5        1,991        2,610       25100       50200        104 APPLU => SSOR => BLTS
 11.5        1,991        2,610       25100       50200        104 BLTS
  8.7        1,978        1,978       25100           0         79 APPLU => SSOR => JACLD
  8.7        1,978        1,978       25100           0         79 JACLD
  8.3        1,892        1,892       25100           0         75 APPLU => SSOR => JACU
```

- Atomic events



```
USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
 3.036E+05  6.826E+04  6.578E+04  6.822E+04      14.08  Heap Memory Used (KB) at Entry
         1  6.578E+04  6.578E+04  6.578E+04          0  Heap Memory Used (KB) at Entry : APPLU
         1  6.803E+04  6.803E+04  6.803E+04          0  Heap Memory Used (KB) at Entry : APPLU => ERHS
         2  6.805E+04  6.803E+04  6.804E+04      8.398  Heap Memory Used (KB) at Entry : APPLU => ERHS => EXCHANGE_3
         2  6.805E+04  6.804E+04  6.805E+04      2.484  Heap Memory Used (KB) at Entry : APPLU => ERHS => EXCHANGE_3 => MPI_Irecv()
         2  6.805E+04  6.803E+04  6.804E+04      7.398  Heap Memory Used (KB) at Entry : APPLU => ERHS => EXCHANGE_3 => MPI_Send()
         2  6.805E+04  6.805E+04  6.805E+04      1.242  Heap Memory Used (KB) at Entry : APPLU => ERHS => EXCHANGE_3 => MPI_Wait()
         1  6.823E+04  6.823E+04  6.823E+04          0  Heap Memory Used (KB) at Entry : APPLU => ERROR
         1  6.823E+04  6.823E+04  6.823E+04          0  Heap Memory Used (KB) at Entry : APPLU => ERROR => MPI_Allreduce()
         1  6.578E+04  6.578E+04  6.578E+04          0  Heap Memory Used (KB) at Entry : APPLU => INIT_COMM
         1  6.795E+04  6.795E+04  6.795E+04          0  Heap Memory Used (KB) at Entry : APPLU => INIT_COMM => MPI_Comm_rank()
         1  6.795E+04  6.795E+04  6.795E+04          0  Heap Memory Used (KB) at Entry : APPLU => INIT_COMM => MPI_Comm_size()
         1  6.578E+04  6.578E+04  6.578E+04          0  Heap Memory Used (KB) at Entry : APPLU => INIT_COMM => MPI_Init()
         1  6.795E+04  6.795E+04  6.795E+04          0  Heap Memory Used (KB) at Entry : APPLU => INIT_COMM => NODEDIM
         1  6.826E+04  6.826E+04  6.826E+04          0  Heap Memory Used (KB) at Entry : APPLU => MPI_Finalize()
```

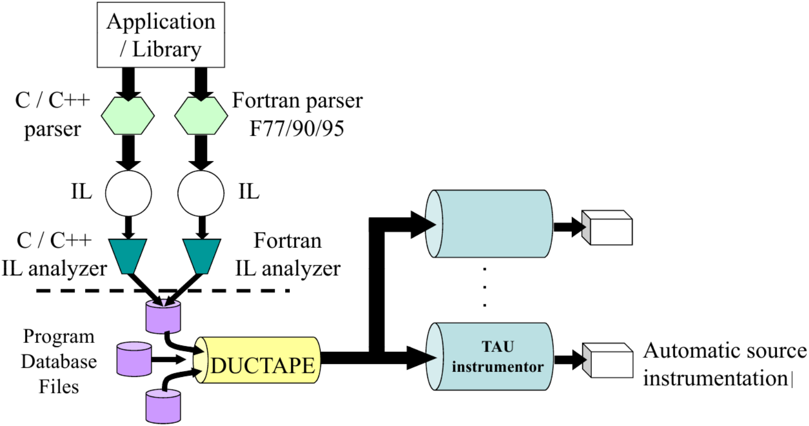
TAU Instrumentation/Measurement



Direct Instrumentation Options in TAU

- Source code Instrumentation
  - Manual instrumentation
  - Automatic instrumentation (PDT)
  - compiler generates instrumented object code
- Library level instrumentation
- Runtime pre-loading and interception of library calls
- Binary code instrumentation
  - Rewrite the binary, runtime instrumentation

TAU Instrumentation/Measurement

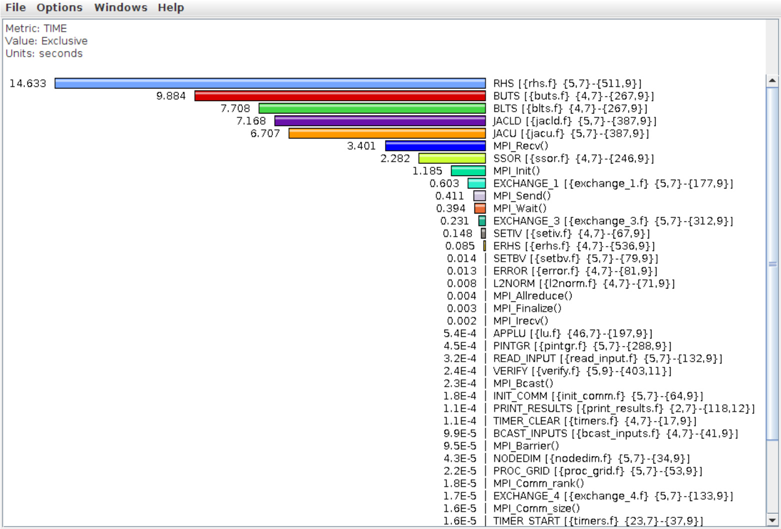


PDT: Automatic source code instrumentation

- Instrument source code using PDT and PAPI
  - Choose the appropriate TAU stub makefile

```
% export TAU_MAKEFILE=/srv/app/tau/x86_64/lib/Makefile.\
tau-papi-mpi-pdt
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
```

- Time spent in each routine



Metric: TIME
Value: Exclusive
Units: seconds

14.633	RHS [{rhs.f} {5,7}-{511,9}]
9.884	BUTS [{buts.f} {4,7}-{267,9}]
7.708	BLTS [{blts.f} {4,7}-{267,9}]
7.168	JACLD [{jacld.f} {5,7}-{387,9}]
6.707	JACU [{jacu.f} {5,7}-{387,9}]
3.401	MPI_Recv()
2.282	SSOR [{ssor.f} {4,7}-{246,9}]
1.185	MPI_Init()
0.603	EXCHANGE_1 [{exchange_1.f} {5,7}-{177,9}]
0.411	MPI_Send()
0.394	MPI_Wait()
0.231	EXCHANGE_3 [{exchange_3.f} {5,7}-{312,9}]
0.148	SETIV [{setiv.f} {4,7}-{67,9}]
0.085	ERHS [{erhs.f} {4,7}-{536,9}]
0.014	SETBV [{setbv.f} {5,7}-{79,9}]
0.013	ERROR [{error.f} {4,7}-{81,9}]
0.008	L2NORM [{l2norm.f} {4,7}-{71,9}]
0.004	MPI_Allreduce()
0.003	MPI_Finalize()
0.002	MPI_Irecv()
5.4E-4	APPLU [{lu.f} {46,7}-{197,9}]
4.5E-4	PINTGR [{pintgr.f} {5,7}-{288,9}]
3.2E-4	READ_INPUT [{read_input.f} {5,7}-{132,9}]
2.4E-4	VERIFY [{verify.f} {5,9}-{403,11}]
2.3E-4	MPI_Bcast()
1.8E-4	INIT_COMM [{init_comm.f} {5,7}-{64,9}]
1.1E-4	PRINT_RESULTS [{print_results.f} {2,7}-{118,12}]
1.1E-4	TIMER_CLEAR [{timers.f} {4,7}-{17,9}]
9.9E-5	BCAST_INPUTS [{bcast_inputs.f} {4,7}-{41,9}]
9.5E-5	MPI_Barrier()
4.3E-5	NODEDIM [{nodedim.f} {5,7}-{34,9}]
2.2E-5	PROC_GRID [{proc_grid.f} {5,7}-{53,9}]
1.8E-5	MPI_Comm_rank()
1.7E-5	EXCHANGE_4 [{exchange_4.f} {5,7}-{133,9}]
1.6E-5	MPI_Comm_size()
1.6E-5	TIMER_START [{timers.f} {23,7}-{37,9}]

Generating a flat profile with MPI

- Declare the appropriate environment variables

```
% export TAU_MAKEFILE= /srv/app/tau/x86_64/lib/Makefile.tau-papi-mpi-pdt
```

- Declare the compiler (config/make.def)

```
MPIF77=tau_f90.sh
```

- Compile the LU benchmark, class A, 4 processors

```
% make clean
% make LU NPROCS=4 CLASS=A
```

- Execute the benchmark

```
% cd bin.tau
% mpirun -np 4 lu.A.4
```

- Pack the profile data

```
% paraprof --pack app.ppk
% paraprof app.ppk
```

- Click on “node 0”

Automatic Instrumentation

- Wrapper scripts
  - Replace F77 (gfortran) with tau_f90.sh
  - Automatically instruments Fortran source code and links with TAU MPI Wrapper library
  - Use tau_cc.sh and tau_cxx.sh for C and C++

```
CC = mpicc -> CC = tau_cc.sh
CXX = mpicxx -> CXX = tau_cxx.sh
F90 = mpif90 -> F90 = tau_f90.sh
```

Instrumentation, re-writing Binaries with MAQAO (beta)

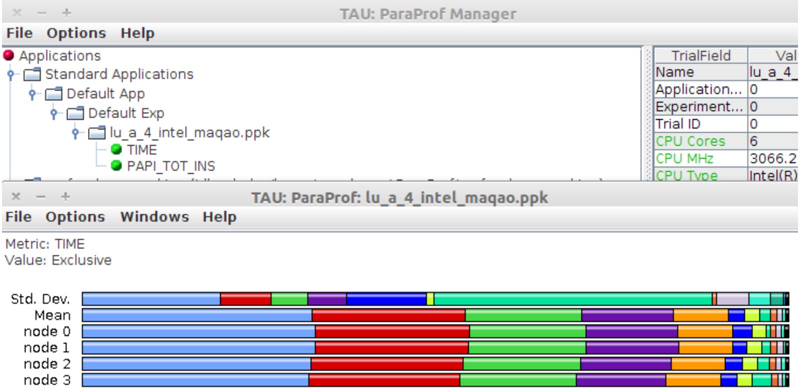
- Instrument:

```
% tau_rewrite lu.A.4 -T papi,pdt -o lu.A.4.inst
```

- Perform measurement and execute it:

```
% mpirun --bind-to-core -np 4 lu.A.4.inst
```

Paraprof with binary instrumentation through MAQAO



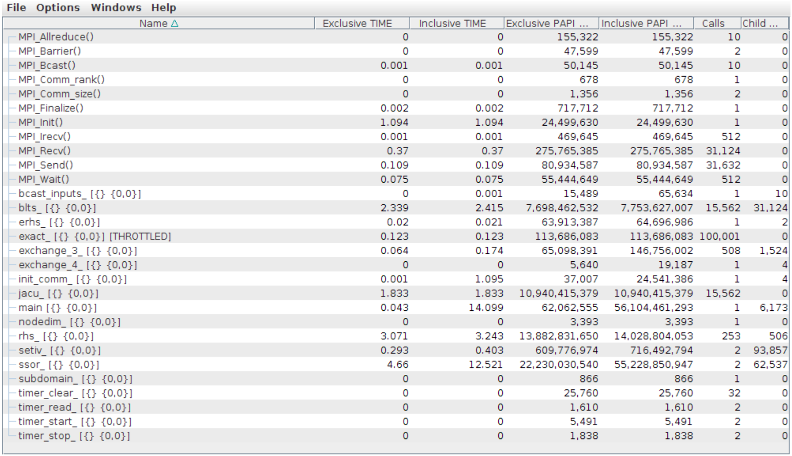
- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - lu_a_4_intel_maqao.ppk
          - TIME
          - PAPI_TOT_INS

TrialField	Val
Name	lu_a_4_
Application...	0
Experiment...	0
Trial ID	0
CPU Cores	6
CPU MHz	3066.2
CPU Type	Intel(R)

TAU: ParaProf: lu_a_4_intel_maqao.ppk
File Options Windows Help

Metric: TIME
Value: Exclusive

Paraprof with binary instrumentation through MAQAO



Name △	Exclusive TIME	Inclusive TIME	Exclusive PAPI ...	Inclusive PAPI ...	Calls	Child ...
MPI_Allreduce()	0	0	155,322	155,322	10	0
MPI_Barrier()	0	0	47,599	47,599	2	0
MPI_Bcast()	0.001	0.001	50,145	50,145	10	0
MPI_Comm_rank()	0	0	678	678	1	0
MPI_Comm_size()	0	0	1,356	1,356	2	0
MPI_Finalize()	0.002	0.002	717,712	717,712	1	0
MPI_Init()	1.094	1.094	24,499,630	24,499,630	1	0
MPI_Irecv()	0.001	0.001	469,645	469,645	512	0
MPI_Recv()	0.37	0.37	275,765,385	275,765,385	31,124	0
MPI_Send()	0.109	0.109	80,934,587	80,934,587	31,632	0
MPI_Wait()	0.075	0.075	55,444,649	55,444,649	512	0
bcast_inputs_ [{} {0,0}]	0	0.001	15,489	65,634	1	10
blts_ [{} {0,0}]	2.339	2.415	7,698,462,532	7,753,627,007	15,562	31,124
erhs_ [{} {0,0}]	0.02	0.021	63,913,387	64,696,986	1	2
exact_ [{} {0,0}] [THROTTLED]	0.123	0.123	113,686,083	113,686,083	100,001	0
exchange_3_ [{} {0,0}]	0.064	0.174	65,098,391	146,756,002	508	1,524
exchange_4_ [{} {0,0}]	0	0	5,640	19,187	1	4
init_comm_ [{} {0,0}]	0.001	1.095	37,007	24,541,386	1	4
jacu_ [{} {0,0}]	1.833	1.833	10,940,415,379	10,940,415,379	15,562	0
main [{} {0,0}]	0.043	14.099	62,062,555	56,104,461,293	1	6,173
nodedim_ [{} {0,0}]	0	0	3,393	3,393	1	0
rhs_ [{} {0,0}]	3.071	3.243	13,882,831,650	14,028,804,053	253	506
setiv_ [{} {0,0}]	0.293	0.403	609,776,974	716,492,794	2	93,857
ssor_ [{} {0,0}]	4.66	12.521	22,230,030,540	55,228,850,947	2	62,537
subdomain_ [{} {0,0}]	0	0	866	866	1	0
timer_clear_ [{} {0,0}]	0	0	25,760	25,760	32	0
timer_read_ [{} {0,0}]	0	0	1,610	1,610	2	0
timer_start_ [{} {0,0}]	0	0	5,491	5,491	2	0
timer_stop_ [{} {0,0}]	0	0	1,838	1,838	2	0

Hands-on: NPB-MPI / LU

TAU

Declare compiler wrappers

- Enter the hands_on directory

```
% cd ~/tutorial/NPB3.3-MPI
```

- Activate the TAU compiler wrappers

```
% vim config/make.def
  #MPIF77 = mpif77
   MPIF77 = tau_f90.sh
```

- Re-compile

```
% make clean
% make LU CLASS=A NPROCS=4
```

- Execute the benchmark

```
% cd bin.tau
% mpirun -np 4 lu.A.4
% paraprof &
```

Compile-Time Environment Variables

-optVerbose	Turn on verbose debugging messages
-optCompInst	Use compiler based instrumentation
-optNoCompInst	Do not revert to compiler instrumentation if source instrumentation fails
-optTrackIO	Wrap POSIX I/O call and calculates vol/bw of I/O operations
-optKeepFiles	Does not remove .pdb and .inst.* files
-optPreProcess	Preprocess Fortran sources before instrumentation
-optTauSelectFile=“<file”	Specify selective instrumentation file for tau_instrumentor
-optTauWrapFile=“<file>”	Specify path to link_options.tau generated by tau_gen_wrapper
-optHeaderInst	Enable instrumentation of headers
-optLinking=“”	Options passed to the linker
-optCompile=“”	Options passed to the compiler
-optPdtF95Opts=“”	Add options for Fortran parser in PDT
-optPdtF95Reset=“”	Reset options for Fortran parser in PDT
-optPdtCOpts=“”	Options for C parser in PDT
-optPdtCxxOpts=“”	Options for C++ parser in PDT

Compiling Fortran Codes with TAU

- For using free format in .f files, use:

```
% export TAU_OPTIONS=’-optPdtF95Opts=‘‘-R free’’’
```

- Use compiler based instrumentation instead of PDT:

```
% export TAU_OPTIONS=’-optCompInst’
```

- Use C preprocessor directives in Fortran code:

```
% export TAU_OPTIONS=’-optPreProcess -optDetectMemoryLeaks’
```

- Use an instrumentation specification file:

```
% export TAU_OPTIONS=’-optTauSelectFile=select.tau’
```

Runtime Environment Variables in TAU

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_LEAKS	0	Setting to 1 turns on leak detection
TAU_TRACK_HEAP	0	Setting to 1 turns on heap memory/headroom at routine entry & exit
TAU_CALLPATH_DEPTH	2	Specifies depth of callpath
TAU_TRACK_IO_PARAMS	0	Setting to 1 with -optTrackIO
TAU_SAMPLING	1	Generates sample based profiles
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time
TAU_COMPENSATE	0	Setting to 1 enables runtime compensation of instrumentation overhead
TAU_PROFILE_FORMAT	Profile	Setting to “merged” generates a single file, “snapshot” generates snapshot per thread
TAU_METRICS	TIME	Setting to a comma separated list (TIME:PAPI_TOT_INS:PAPI_FP_OPS)

Loop level profile

- Declare the options for TAU

```
% export TAU_PROFILE=1
% export TAU_PROFILE_FORMAT=Profile
% export TAU_OPTIONS=’-optTauSelectFile=select.tau’

% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine=‘‘#’’
  END_INSTRUMENT_SECTION
```

- Compile the benchmark

```
% make clean
% make LU NPROCS=4 CLASS=A
```

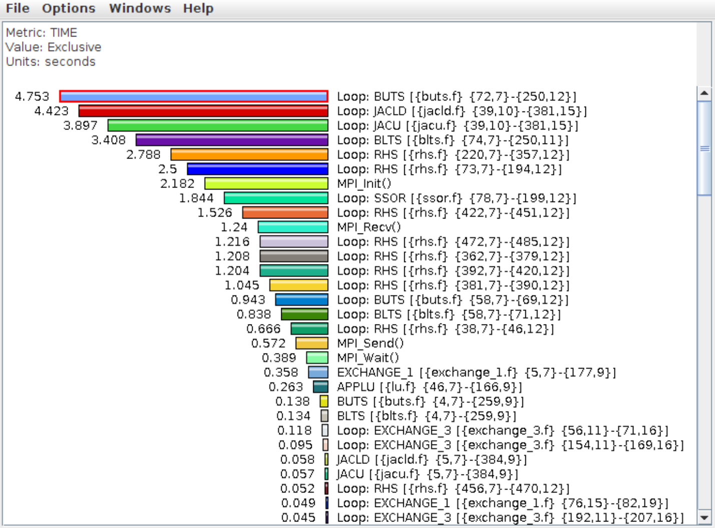
- Execute the benchmark

```
% cd bin.tau
% mpirun -np 4 lu.A.4
```

- Analyze the profiling data

```
% paraprof --pack lu_a_4.ppk
% paraprof lu_a_4.ppk
```

LU benchmark, loop profile



Metric: TIME
Value: Exclusive
Units: seconds

4.753	Loop: BUTS [{buts.f} {72,7}-{250,12}]
4.423	Loop: JACLD [{jacld.f} {39,10}-{381,15}]
3.897	Loop: JACU [{jacu.f} {39,10}-{381,15}]
3.408	Loop: BLTS [{blts.f} {74,7}-{250,11}]
2.788	Loop: RHS [{rhs.f} {220,7}-{357,12}]
2.5	Loop: RHS [{rhs.f} {73,7}-{194,12}]
2.182	MPI_Init()
1.844	Loop: SSOR [{ssor.f} {78,7}-{199,12}]
1.526	Loop: RHS [{rhs.f} {422,7}-{451,12}]
1.24	MPI_Recv()
1.216	Loop: RHS [{rhs.f} {472,7}-{485,12}]
1.208	Loop: RHS [{rhs.f} {362,7}-{379,12}]
1.204	Loop: RHS [{rhs.f} {392,7}-{420,12}]
1.045	Loop: RHS [{rhs.f} {381,7}-{390,12}]
0.943	Loop: BUTS [{buts.f} {58,7}-{69,12}]
0.838	Loop: BLTS [{blts.f} {58,7}-{71,12}]
0.666	Loop: RHS [{rhs.f} {38,7}-{46,12}]
0.572	MPI_Send()
0.389	MPI_Wait()
0.358	EXCHANGE_1 [{exchange_1.f} {5,7}-{177,9}]
0.263	APPLU [{lu.f} {46,7}-{166,9}]
0.138	BUTS [{buts.f} {4,7}-{259,9}]
0.134	BLTS [{blts.f} {4,7}-{259,9}]
0.118	Loop: EXCHANGE_3 [{exchange_3.f} {56,11}-{71,16}]
0.095	Loop: EXCHANGE_3 [{exchange_3.f} {154,11}-{169,16}]
0.058	JACLD [{jacld.f} {5,7}-{384,9}]
0.057	JACU [{jacu.f} {5,7}-{384,9}]
0.052	Loop: RHS [{rhs.f} {456,7}-{470,12}]
0.049	Loop: EXCHANGE_1 [{exchange_1.f} {76,15}-{82,19}]
0.045	Loop: EXCHANGE_3 [{exchange_3.f} {192,11}-{207,16}]

PAPI profile with 2 or more metrics

- Declare the environment variable TAU_METRICS

```
% export TAU_METRICS=TIME:PAPI_FP_OPS:PAPI_TOT_INS
```

- Execute the benchmark

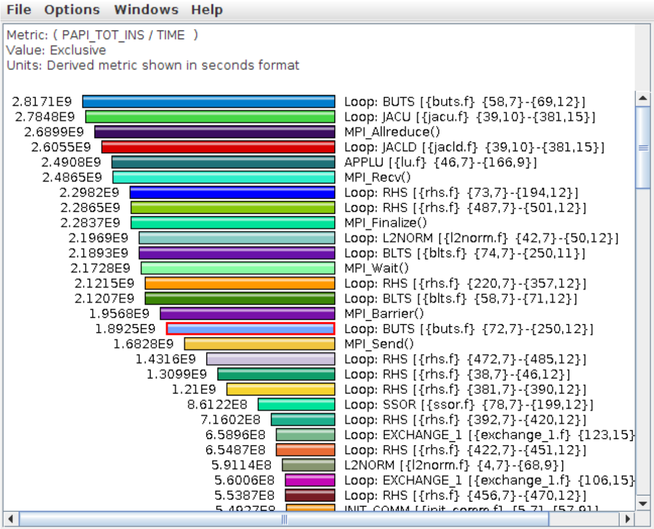
```
% mpirun -np 4 lu.A.4
```

- Analyze the profiling data

```
% paraprof --pack lu_a_4_papi.ppk
% paraprof lu_a_4_papi.ppk
```

- Click Options -> Show Derived Metric Panel -> click PAPI_TOT_INS, click “/”, click TIME, Apply, choose the new metric by double clicking

LU benchmark, loop profile, instructions per second



Metric: (PAPI_TOT_INS / TIME)
Value: Exclusive
Units: Derived metric shown in seconds format

2.8171E9	Loop: BUTS [{buts.f} {58,7}-{69,12}]
2.7848E9	Loop: JACU [{jacu.f} {39,10}-{381,15}]
2.6899E9	MPI_Allreduce()
2.6055E9	Loop: JACLD [{jacld.f} {39,10}-{381,15}]
2.4908E9	APPLU [{lu.f} {46,7}-{166,9}]
2.4865E9	MPI_Recv()
2.2982E9	Loop: RHS [{rhs.f} {73,7}-{194,12}]
2.2865E9	Loop: RHS [{rhs.f} {487,7}-{501,12}]
2.2837E9	MPI_Finalize()
2.1969E9	Loop: L2NORM [{l2norm.f} {42,7}-{50,12}]
2.1893E9	Loop: BLTS [{blts.f} {74,7}-{250,11}]
2.1728E9	MPI_Wait()
2.1215E9	Loop: RHS [{rhs.f} {220,7}-{357,12}]
2.1207E9	Loop: BLTS [{blts.f} {58,7}-{71,12}]
1.9568E9	MPI_Barrier()
1.8925E9	Loop: BUTS [{buts.f} {72,7}-{250,12}]
1.6828E9	MPI_Send()
1.4316E9	Loop: RHS [{rhs.f} {472,7}-{485,12}]
1.3099E9	Loop: RHS [{rhs.f} {38,7}-{46,12}]
1.21E9	Loop: RHS [{rhs.f} {381,7}-{390,12}]
8.6122E8	Loop: SSOR [{ssor.f} {78,7}-{199,12}]
7.1602E8	Loop: RHS [{rhs.f} {392,7}-{420,12}]
6.5896E8	Loop: EXCHANGE_1 [{exchange_1.f} {123,15}
6.5487E8	Loop: RHS [{rhs.f} {422,7}-{451,12}]
5.9114E8	L2NORM [{l2norm.f} {4,7}-{68,9}]
5.6006E8	Loop: EXCHANGE_1 [{exchange_1.f} {106,15}
5.5387E8	Loop: RHS [{rhs.f} {456,7}-{470,12}]
5.4927E8	INIT_COMM [{init_comm.f} {5,7}-{57,9}]

Callpath Profile

- Enable the Callpath Profile

```
% export TAU_CALLPATH=1
% export TAU_CALLPATH_DEPTH=10
```

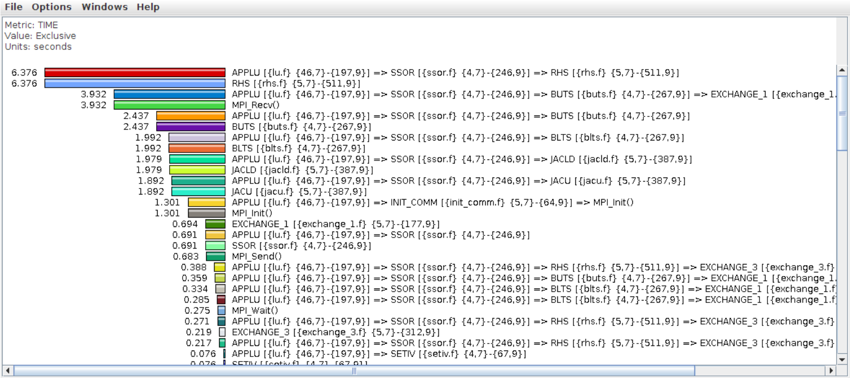
- Execute the benchmark

```
% mpirun -np 4 lu.A.4
```

- Analyze the profiling data

```
% paraprof --pack lu_a_4_papi_callpath.ppk
% paraprof lu_a_4_papi_callpath.ppk
```

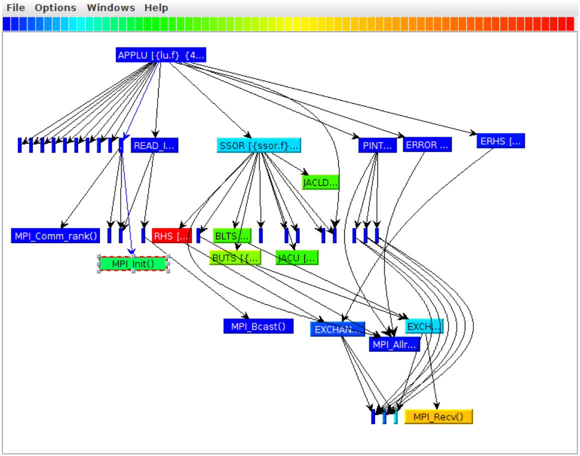
Callpath Profile



```
Metric: TIME
Value: Exclusive
Units: seconds

6.376  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => RHS [{rhs.f} {5,7}-{511,9}]
6.376  RHS [{rhs.f} {5,7}-{511,9}]
3.932  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => BUTS [{buts.f} {4,7}-{267,9}] => EXCHANGE_1 [{exchange_1.
3.932  MPI_Recv()
2.437  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => BUTS [{buts.f} {4,7}-{267,9}]
2.437  BUTS [{buts.f} {4,7}-{267,9}]
1.992  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => BLTS [{blts.f} {4,7}-{267,9}]
1.992  BLTS [{blts.f} {4,7}-{267,9}]
1.979  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => JACLD [{jacld.f} {5,7}-{387,9}]
1.979  JACLD [{jacld.f} {5,7}-{387,9}]
1.892  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => JACU [{jacu.f} {5,7}-{387,9}]
1.892  JACU [{jacu.f} {5,7}-{387,9}]
1.301  APPLU [{lu.f} {46,7}-{197,9}] => INIT_COMM [{init_comm.f} {5,7}-{64,9}] => MPI_Init()
1.301  MPI_Init()
0.694  EXCHANGE_1 [{exchange_1.f} {5,7}-{177,9}]
0.691  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}]
0.691  SSOR [{ssor.f} {4,7}-{246,9}]
0.683  MPI_Send()
0.388  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => RHS [{rhs.f} {5,7}-{511,9}] => EXCHANGE_3 [{exchange_3.f}
0.359  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => BUTS [{buts.f} {4,7}-{267,9}] => EXCHANGE_1 [{exchange_1.
0.334  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => BLTS [{blts.f} {4,7}-{267,9}] => EXCHANGE_1 [{exchange_1.f}
0.285  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => BLTS [{blts.f} {4,7}-{267,9}] => EXCHANGE_1 [{exchange_1.f}
0.275  MPI_Wait()
0.271  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => RHS [{rhs.f} {5,7}-{511,9}] => EXCHANGE_3 [{exchange_3.f}
0.219  EXCHANGE_3 [{exchange_3.f} {5,7}-{312,9}]
0.217  APPLU [{lu.f} {46,7}-{197,9}] => SSOR [{ssor.f} {4,7}-{246,9}] => RHS [{rhs.f} {5,7}-{511,9}] => EXCHANGE_3 [{exchange_3.f}
0.076  APPLU [{lu.f} {46,7}-{197,9}] => SETIV [{setiv.f} {4,7}-{67,9}]
0.076  SETIV [{setiv.f} {4,7}-{67,9}]
```

Call graph

- From the ParaProf window, Click Windows -> Click Thread -> Click Call Graph, select for which process you want to see the call graph



Communication Matrix Display

- Enable the communication matrix

```
% export TAU_COMM_MATRIX=1
```

- Execute the benchmark

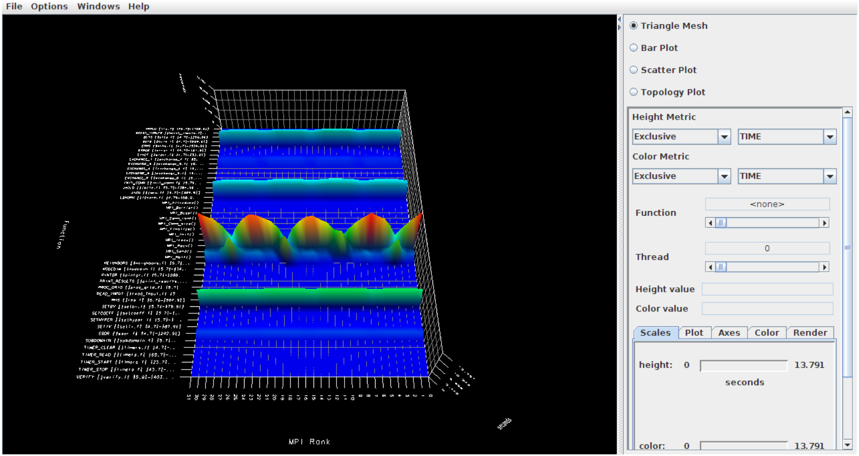
```
% mpirun -np 4 lu.A.4
```

- Analyze the profiling data

```
% paraprof --pack lu_a_4_papi_comm.ppk
% paraprof lu_a_4_papi_comm.ppk
```

- Click Windows -> Click 3D Communication Matrix

Communication Matrix Display, exclusive time



● Triangle Mesh

○ Bar Plot

○ Scatter Plot

○ Topology Plot

Height Metric

Exclusive ▼ TIME ▼

Color Metric

Exclusive ▼ TIME ▼

Function <none>

Thread 0

Height value

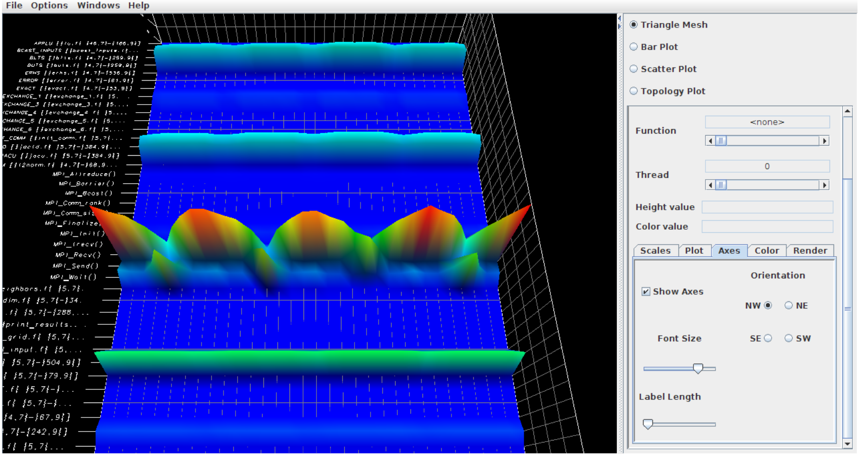
Color value

height: 0 13.791

seconds

color: 0 13.791

Communication Matrix Display, zoom in communication


File
Options
Windows
Help
APPLU [{lu.f} {46,7}-{166,9}]
BCAST_INPUTS [{bcast_inputs.f}...
BLTS [{blts.f} {4,7}-{259,9}]
BUTS [{buts.f} {4,7}-{259,9}]
ERHS [{erhs.f} {4,7}-{536,9}]
ERROR [{error.f} {4,7}-{81,9}]
EXACT [{exact.f} {4,7}-{53,9}]
EXCHANGE_1 [{exchange_1.f} {5...
XCHANGE_3 [{exchange_3.f} {5....
CHANGE_4 [{exchange_4.f} {5....
CHANGE_5 [{exchange_5.f} {5....
HANGE_6 [{exchange_6.f} {5....
T_COMM [{init_comm.f} {5,7}...
D [{jacld.f} {5,7}-{384,9}...
ACU [{jacu.f} {5,7}-{384,9}]
X [{l2norm.f} {4,7}-{68,9...
MPI_Allreduce()
MPI_Barrier()
MPI_Bcast()
MPI_Comm_rank()
MPI_Comm_siz
MPI_Finalize
MPI_Init()
MPI_Irecv()
MPI_Recv()
MPI_Send()
MPI_Wait()
eighbors.f} {5,7}.
dim.f} {5,7}-{34.
.f} {5,7}-{288,...
{print_results...
_grid.f} {5,7}...
_input.f} {5,...
} {5,7}-{504,9}]
} {5,7}-{79,9}]
.f} {5,7}-}...
.f} {5,7}-}...
{4,7}-{67,9}]
,7}-{242,9}]
.f} {5,7}...
Scales
Plot
Axes
Color
Render

● Triangle Mesh
○ Bar Plot
○ Scatter Plot
○ Topology Plot

Function <none>

Thread 0

Height value

Color value

Orientation

☑ Show Axes

NW ● ○ NE

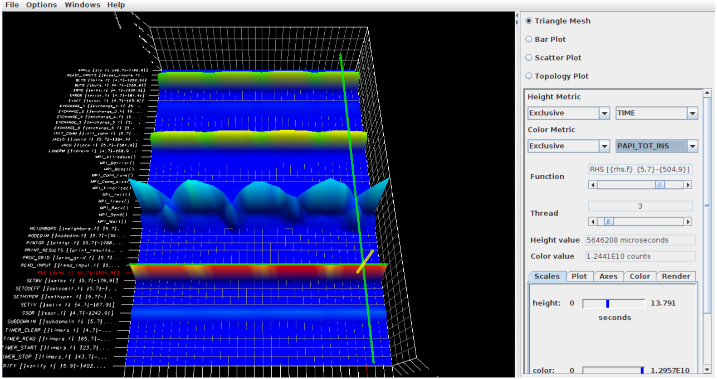
SE ○ ○ SW

Font Size

Label Length

- Inspect the duration of the MPI calls

Communication Matrix Display, time, total instructions


File
Options
Windows
Help
APPLU [{lu.f} {46,7}-{166,9}]
BCAST_INPUTS [{bcast_inputs.f}..
BLTS [{blts.f} {4,7}-{252,9}]
BUTS [{buts.f} {4,7}-{259,9}]
ERHS [{erhs.f} {4,7}-{536,9}]
ERROR [{error.f} {4,7}-{81,9}]
EXACT [{exact.f} {4,7}-{53,9}]
EXCHANGE_1 [{exchange_1.f} {5..
EXCHANGE_3 [{exchange_3.f} {5...
EXCHANGE_4 [{exchange_4.f} {5...
EXCHANGE_5 [{exchange_5.f} {5..
EXCHANGE_6 [{exchange_6.f} {5..
INIT_COMM [{init_comm.f} {5,7}..
JACLD [{jacld.f} {5,7}-{384,9}..
JACU [{jacu.f} {5,7}-{384,9}]
L2NORM [{l2norm.f} {4,7}-{68,9..
MPI_Allreduce()
MPI_Barrier()
MPI_Bcast()
MPI_Comm_rank()
MPI_Comm_size()
MPI_Finalize()
MPI_Init()
MPI_Irecv()
MPI_Recv()
MPI_Send()
MPI_Wait()
NEIGHBORS [{neighbors.f} {5,7}.
NODEDIM [{nodedim.f} {5,7}-{34...
PINTGR [{pintgr.f} {5,7}-{288,...
PRINT_RESULTS [{print_results...
PROC_GRID [{proc_grid.f} {5,7}...
READ_INPUT [{read_input.f} {5,...
RHS [{rhs.f} {5,7}-{504,9}]
SETBV [{setbv.f} {5,7}-{79,9}]
SETCOEFF [{setcoeff.f} {5,7}-{..
SETHYPER [{sethyper.f} {5,7}-{..
SETIV [{setiv.f} {4,7}-{67,9}]
SSOR [{ssor.f} {4,7}-{242,9}]
SUBDOMAIN [{subdomain.f} {5,7}...
TIMER_CLEAR [{timers.f} {4,7}-...
TIMER_READ [{timers.f} {65,7}-...
TIMER_START [{timers.f} {23,7}...
IMER_STOP [{timers.f} {43,7}-...
RIFY [{verify.f} {5,9}-{403....
Scales
Plot
Axes
Color
Render

● Triangle Mesh

○ Bar Plot

○ Scatter Plot

○ Topology Plot

Height Metric

Exclusive ▾ TIME ▾

Color Metric

Exclusive ▾ PAPI_TOT_INS ▾

Function: RHS [{rhs.f} {5,7}-{504,9}]

Thread: 3

Height value: 5646208 microseconds

Color value: 1.2441E10 counts

height: 0 13.791 seconds

color: 0 1.2957E10

- Study the total instructions per function

Trace the LU benchmark and prepare them for the JumpShot

- Enable the tracing feature

```
% export TAU_TRACE=1
```

- Execute the benchmark

```
% mpirun -np 4 lu.A.4
```

- Merge the tracefiles

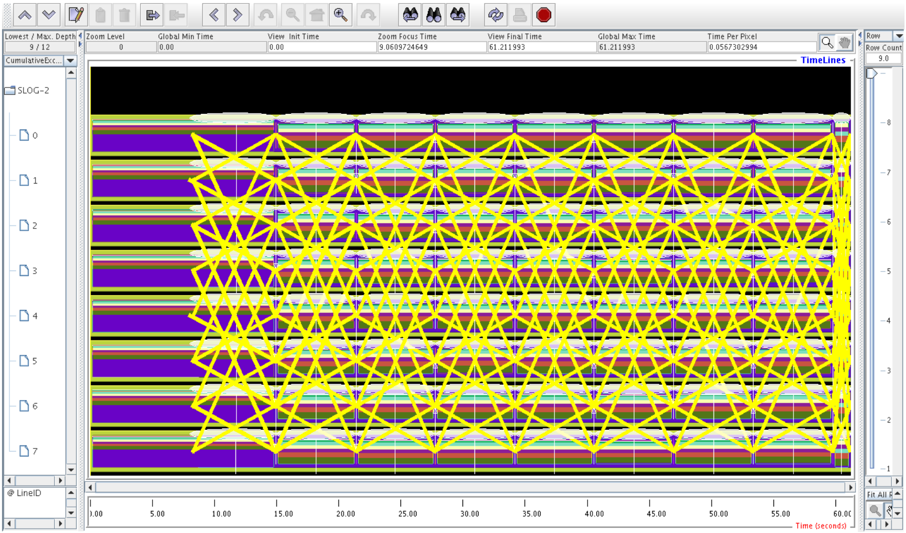
```
% tau_treemerge.pl
```

- Convert the traces to SLOG2 format

```
% tau2log2 tau.trc tau.edf -o app.slog2
% jumpshot app.slog2
```

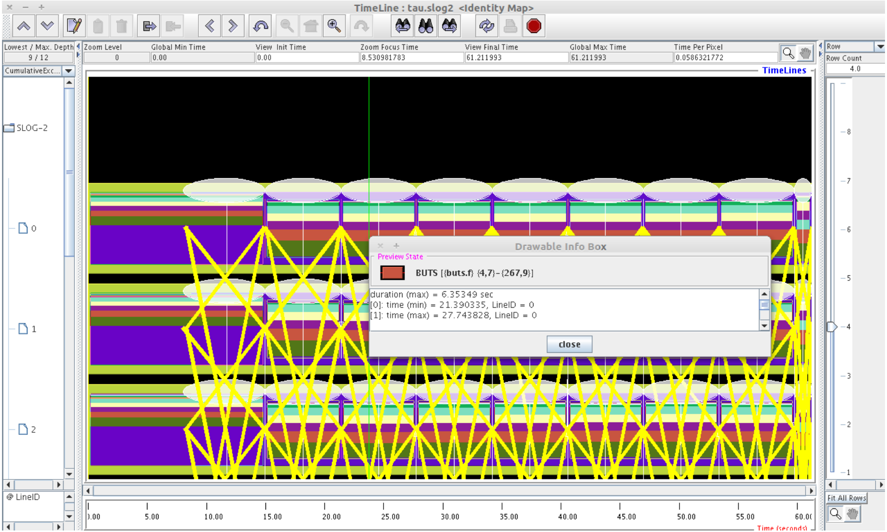
- The following example is for the LU benchmark, class B and 8 processes

View traces from the Jumpshot tool



Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Per Pixel	Row Count
9 / 12	0	0.00	0.00	9.0609724649	61.211993	61.211993	0.0567302994	9.0

View traces from the Jumpshot tool



TimeLine : tau.slog2 <Identity Map>

Lowest / Max. Depth: 9 / 12
Zoom Level: 0
Global Min Time: 0.00
View Init Time: 0.00
Zoom Focus Time: 8.530981783
View Final Time: 61.211993
Global Max Time: 61.211993
Time Per Pixel: 0.0586321772
Row Count: 4.0

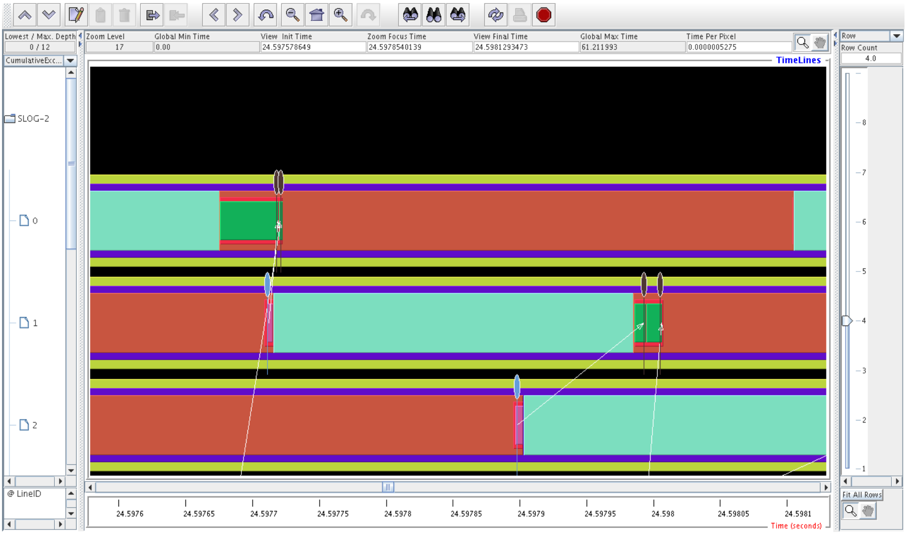
Drawable Info Box

Preview State

🟥 **BUTS [{buts.f} {4,7}-{267,9}]**

```
duration (max) = 6.35349 sec
[0]: time (min) = 21.390335, LineID = 0
[1]: time (max) = 27.743828, LineID = 0
```

View traces from the Jumpshot tool



Lowest / Max. Depth: 0 / 12

Zoom Level: 17

Global Min Time: 0.00

View Init Time: 24.597578649

Zoom Focus Time: 24.5978540139

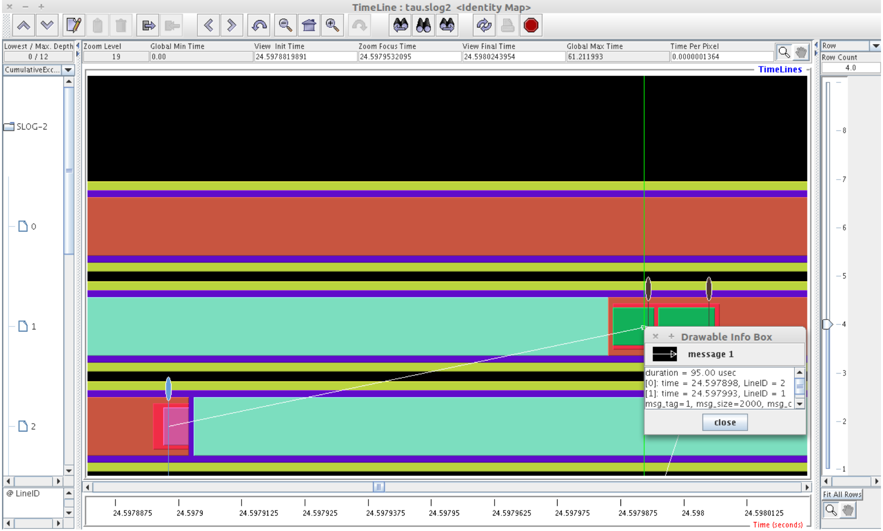
View Final Time: 24.5981293473

Global Max Time: 61.211993

Time Per Pixel: 0.0000005275

Row Count: 4.0

View traces from the Jumpshot tool



TimeLine : tau.slog2 <Identity Map>

Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Per Pixel
0 / 12	19	0.00	24.5978819891	24.5979532095	24.5980243954	61.211993	0.0000001364

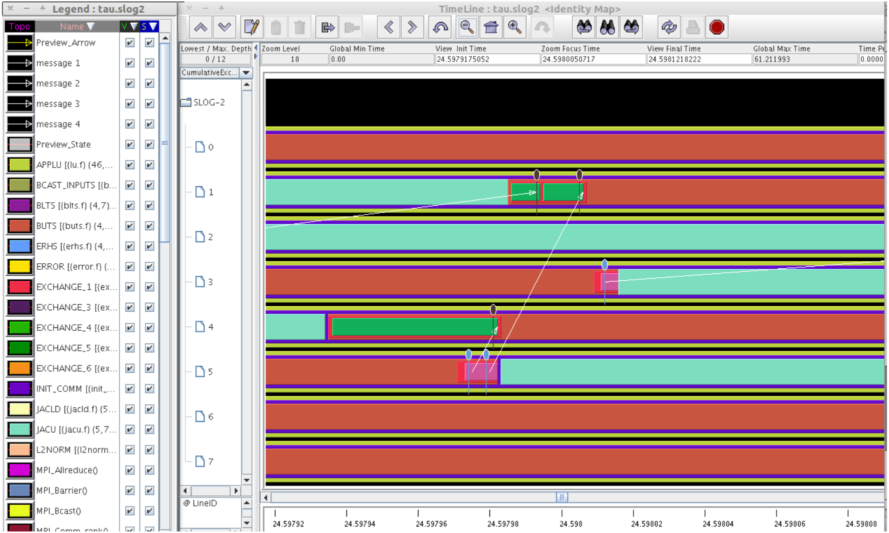
Row Count: 4.0

Drawable Info Box

message 1

```
duration = 95.00 usec
[0]: time = 24.597898, LineID = 2
[1]: time = 24.597993, LineID = 1
msg_tag=1, msg_size=2000, msg_c
```

View traces from the Jumpshot tool



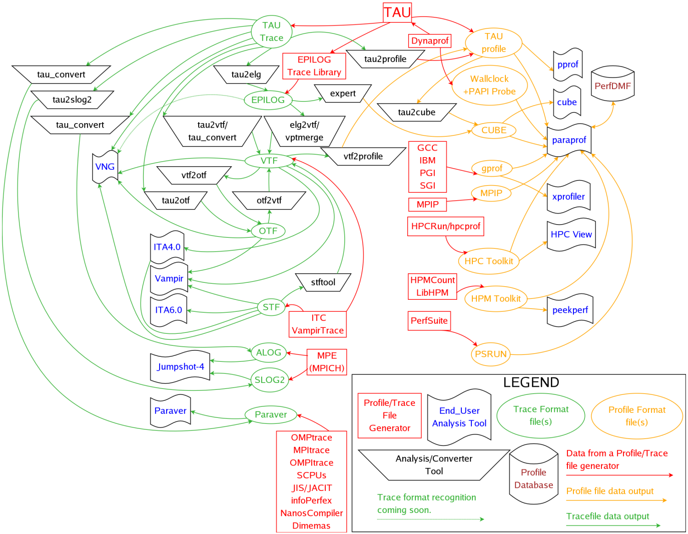
Legend : tau.slog2

Topo	Name	V	S
→	Preview_Arrow	☑	☑
→	message 1	☑	☑
→	message 2	☑	☑
→	message 3	☑	☑
→	message 4	☑	☑
▨	Preview_State	☑	☑
▨	APPLU [(lu.f) {46,...	☑	☑
▨	BCAST_INPUTS [(b...	☑	☑
▨	BLTS [(blts.f) {4,7}...	☑	☑
▨	BUTS [(buts.f) {4,...	☑	☑
▨	ERHS [(erhs.f) {4,...	☑	☑
▨	ERROR [(error.f) {...	☑	☑
▨	EXCHANGE_1 [(ex...	☑	☑
▨	EXCHANGE_3 [(ex...	☑	☑
▨	EXCHANGE_4 [(ex...	☑	☑
▨	EXCHANGE_5 [(ex...	☑	☑
▨	EXCHANGE_6 [(ex...	☑	☑
▨	INIT_COMM [(init_...	☑	☑
▨	JACLD [(jacld.f) {5...	☑	☑
▨	JACU [(jacu.f) {5,7...	☑	☑
▨	L2NORM [(l2norm...	☑	☑
▨	MPI_Allreduce()	☑	☑
▨	MPI_Barrier()	☑	☑
▨	MPI_Bcast()	☑	☑
▨	MPI_Comm_rank()	☑	☑

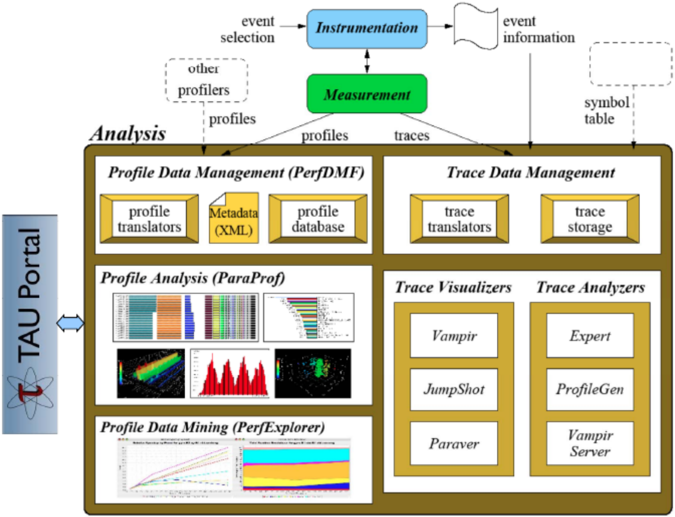
TimeLine : tau.slog2 <Identity Map>

Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Pe
0 / 12	18	0.00	24.5979175052	24.5980050717	24.5981218222	61.211993	0.0000

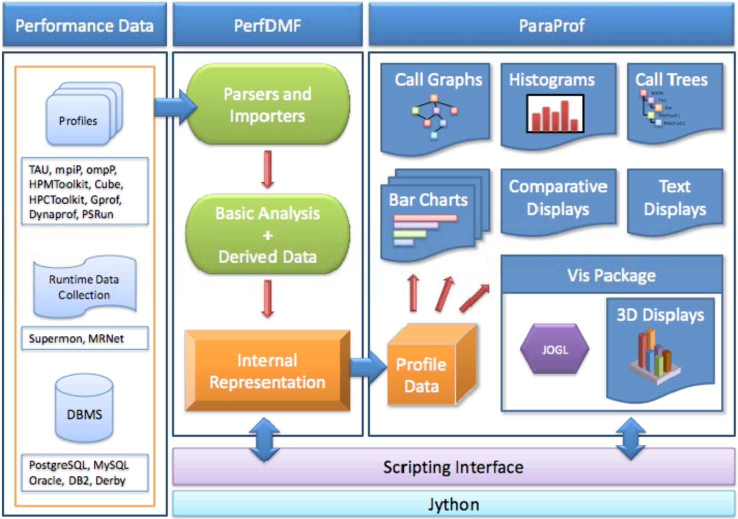
Connection between various tools



TAU Analysis



Framework


Performance Data
PerfDMF
ParaProf
Profiles
TAU, mpiP, ompP, HPMToolkit, Cube, HPCToolkit, Gprof, Dynaprof, PSRun
Runtime Data Collection
Supermon, MRNet
DBMS
PostgreSQL, MySQL Oracle, DB2, Derby
Parsers and Importers
Basic Analysis + Derived Data
Internal Representation
Call Graphs
Histograms
Call Trees
Bar Charts
Comparative Displays
Text Displays
Vis Package
Profile Data
JOGL
3D Displays
Scripting Interface
Jython

Prepare and execute the experiments

```
% export TAU_METRICS=TIME:PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM:\
PAPI_RES_STL
% export TAU_CALLPATH=0
% export TAU_PROFILE_FORMAT=profile
```

- Compile the LU benchmark for classes A,B and 2-32 processes

```
% make clean; make LU NPROCS=2 CLASS=A
% make clean; make LU NPROCS=4 CLASS=A
...
% make clean; make LU NPROCS=2 CLASS=B
% make clean; make LU NPROCS=4 CLASS=B
...
```

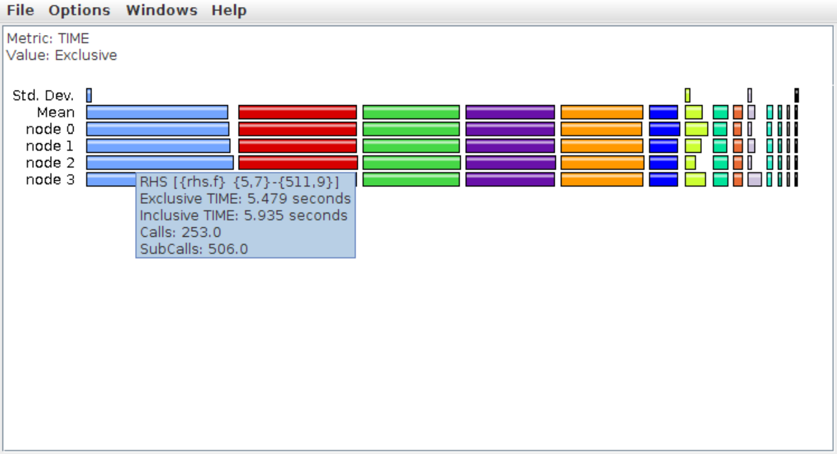
- Execute the experiments for class A and 4,8,16,32 processes (example for class A and 4-8 processes)

```
% cd bin.tau
% rm -r MULTI*  // if there are data from previous experiments
% mpirun -np 4 --bind-to-core lu.A.4
% paraprof --pack lu_a_4.ppk
% rm -r MULTI*
% mpirun -np 8 --bind-to-core lu.A.8
% paraprof --pack lu_a_8.ppk
% rm -r MULTI*
```

Paraprof

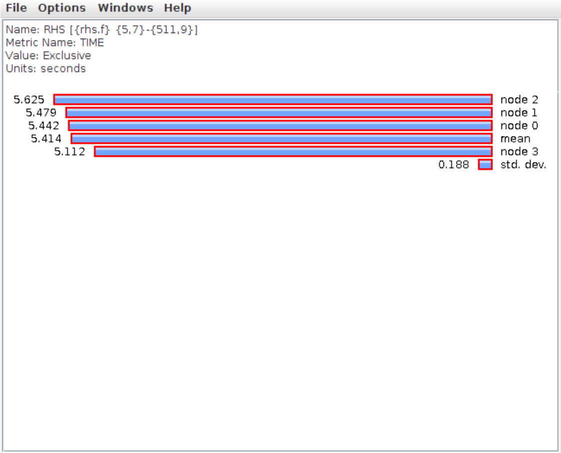
```
% paraprof lu_a_4.ppk
```

- Click Options -> Uncheck Stack Bars Together



RHS [{rhs.f} {5,7}-{511,9}]
Exclusive TIME: 5.479 seconds
Inclusive TIME: 5.935 seconds
Calls: 253.0
SubCalls: 506.0

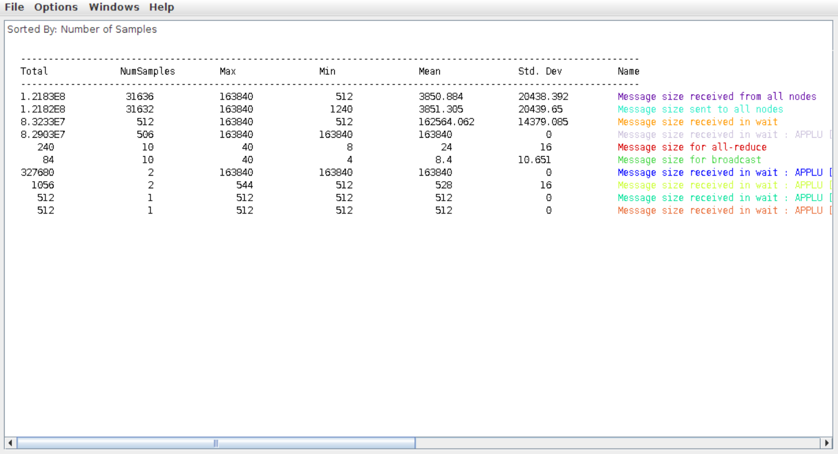
Paraprof



Name: RHS [{rhs.f} {5,7}-{511,9}]
Metric Name: TIME
Value: Exclusive
Units: seconds

Paraprof

- Click Windows -> Click Thread -> Click User Events Statistics -> Select a thread

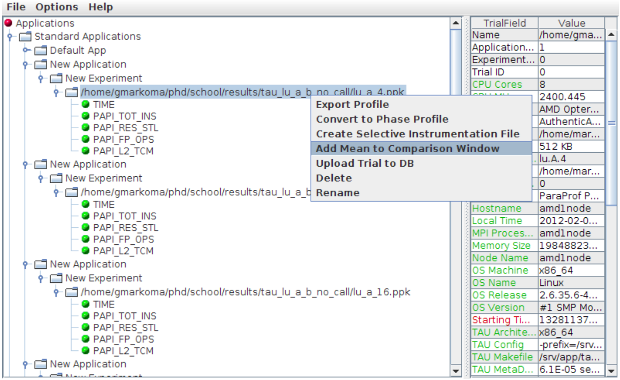


Sorted By: Number of Samples

```
 ------------------------------------------------------------------------------------------------------------------------------------------
 Total              NumSamples          Max                 Min                 Mean                Std. Dev            Name
 ------------------------------------------------------------------------------------------------------------------------------------------
 1.2183E8           31636               163840              512                 3850.884            20438.392           Message size received from all nodes
 1.2182E8           31632               163840              1240                3851.305            20439.65            Message size sent to all nodes
 8.3233E7           512                 163840              512                 162564.062          14379.085           Message size received in wait
 8.2903E7           506                 163840              163840              163840              0                   Message size received in wait : APPLU [
    240             10                  40                  8                   24                  16                  Message size for all-reduce
     84             10                  40                  4                   8.4                 10.651              Message size for broadcast
 327680             2                   163840              163840              163840              0                   Message size received in wait : APPLU [
   1056             2                   544                 512                 528                 16                  Message size received in wait : APPLU [
    512             1                   512                 512                 512                 0                   Message size received in wait : APPLU [
    512             1                   512                 512                 512                 0                   Message size received in wait : APPLU [
```

Compare different executions per function for LU benchmark, class A

- Add all the data to paraprof
  Click File -> CLick Open... -> Click Select File(s) and select your file
- Repeat the previous procedure for all the experiments for classes A and B
- Select the name of the first experiment for class A, do right click on it and select the option “Add Mean to Comparison Window”. A new window pops up, do not close it

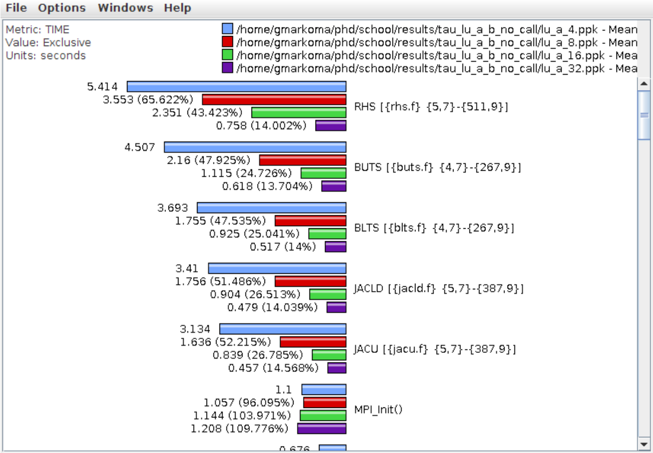


```
Applications
  Standard Applications
    Default App
    New Application
      New Experiment
        /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_4.ppk
          TIME
          PAPI_TOT_INS
          PAPI_RES_STL
          PAPI_FP_OPS
          PAPI_L2_TCM
    New Application
      New Experiment
        /home/gmarkoma/phd/school/results/tau_lu_a_b
          TIME
          PAPI_TOT_INS
          PAPI_RES_STL
          PAPI_FP_OPS
          PAPI_L2_TCM
    New Application
      New Experiment
        /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_16.ppk
          TIME
          PAPI_TOT_INS
          PAPI_RES_STL
          PAPI_FP_OPS
          PAPI_L2_TCM
    New Application
      New Experiment
```

TrialField	Value
Name	/home/gma...
Application...	1
Experiment...	0
Trial ID	0
CPU Cores	8
CPU MH...	2400.445
	AMD Opter...
	AuthenticA...
	/home/mar...
	512 KB
	lu.A.4
	/home/mar...
	0
	ParaProf P...
Hostname	amd1node
Local Time	2012-02-0...
MPI Proces...	amd1node
Memory Size	19848823...
Node Name	amd1node
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.35.6-4...
OS Version	#1 SMP Mo...
Starting Ti...	13281137...
TAU Archite...	x86_64
TAU Config	-prefix=/srv...
TAU Makefile	/srv/app/ta...
TAU MetaD...	6.1E-05 se...

- Repeat the procedure for all the experiments of class A

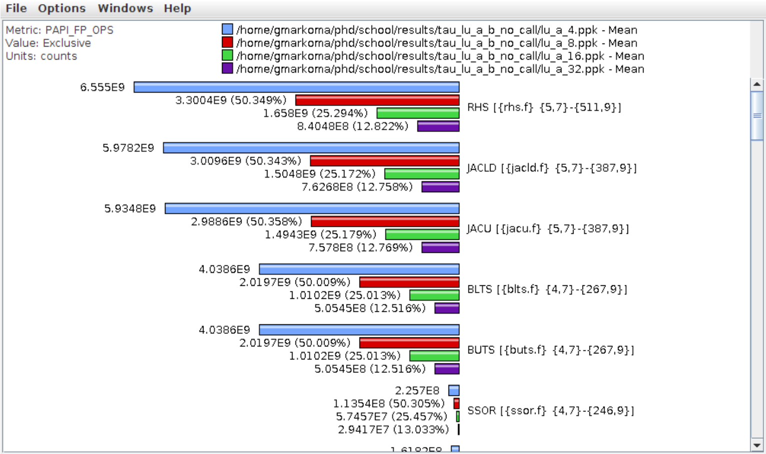
Compare the duration of the functions while we increase the number of the processes



- While we double the number of the processes the duration of the RHS function is decreased by around to 35%

Compare the duration of the functions, studying the floating operations

- Click Options -> Click Select Metric -> Click Select Exclusive -> Click PAPI_FP_OPS



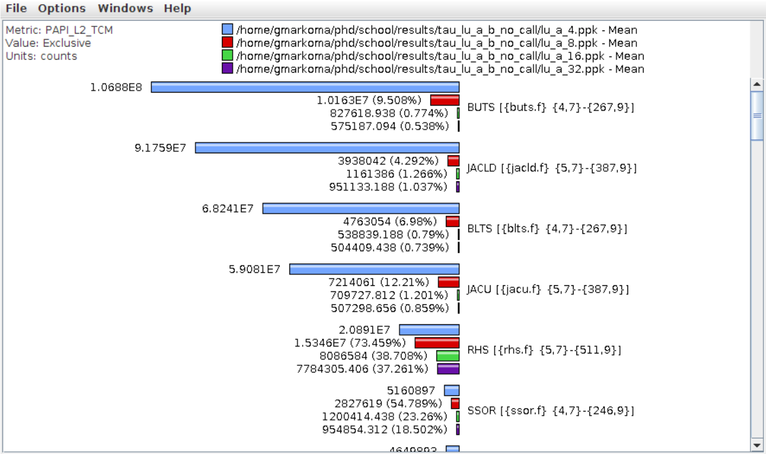
Metric: PAPI_FP_OPS
Value: Exclusive
Units: counts

🟦 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_4.ppk - Mean
🟥 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_8.ppk - Mean
🟩 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_16.ppk - Mean
🟪 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_32.ppk - Mean

- The value of the floating operations do not justify the execution time of the function RHS

Compare the duration of the functions, studying L2 cache misses

- Click Options -> Click Select Metric -> Click Select Exclusive -> Click PAPI_L2_TCM



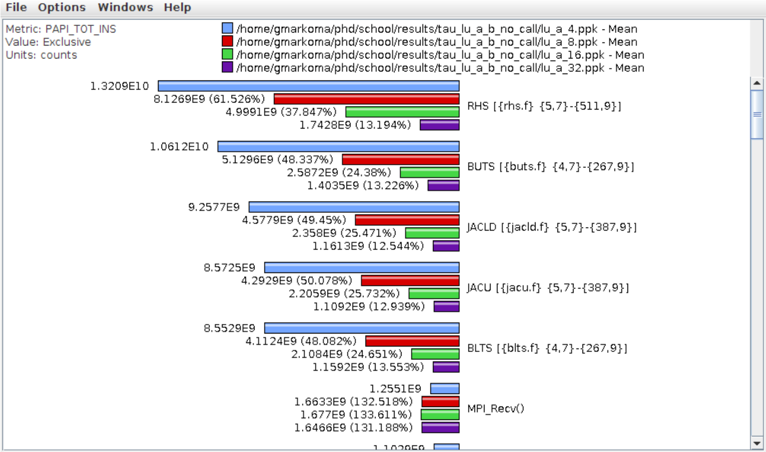
Metric: PAPI_L2_TCM
Value: Exclusive
Units: counts

🟦 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_4.ppk - Mean
🟥 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_8.ppk - Mean
🟩 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_16.ppk - Mean
🟪 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_32.ppk - Mean

- Neither the L2 cache misses justify the execution time of the function RHS

Compare the duration of the functions, studying the total instructions

- Click Options -> Click Select Metric -> Click Select Exclusive -> Click PAPI_TOT_INS



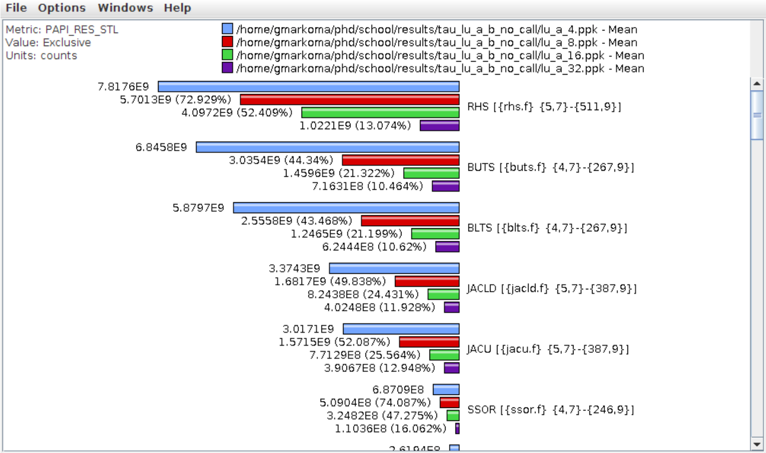
Metric: PAPI_TOT_INS
Value: Exclusive
Units: counts

File
Options
Windows
Help
🟦 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_4.ppk - Mean
🟥 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_8.ppk - Mean
🟩 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_16.ppk - Mean
🟪 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_32.ppk - Mean
1.3209E10
8.1269E9 (61.526%)
4.9991E9 (37.847%)
1.7428E9 (13.194%)
RHS [{rhs.f} {5,7}-{511,9}]
1.0612E10
5.1296E9 (48.337%)
2.5872E9 (24.38%)
1.4035E9 (13.226%)
BUTS [{buts.f} {4,7}-{267,9}]
9.2577E9
4.5779E9 (49.45%)
2.358E9 (25.471%)
1.1613E9 (12.544%)
JACLD [{jacld.f} {5,7}-{387,9}]
8.5725E9
4.2929E9 (50.078%)
2.2059E9 (25.732%)
1.1092E9 (12.939%)
JACU [{jacu.f} {5,7}-{387,9}]
8.5529E9
4.1124E9 (48.082%)
2.1084E9 (24.651%)
1.1592E9 (13.553%)
BLTS [{blts.f} {4,7}-{267,9}]
1.2551E9
1.6633E9 (132.518%)
1.677E9 (133.611%)
1.6466E9 (131.188%)
MPI_Recv()
1.1029E9

- The value of the total completed instructions can justify a part of the mentioned difference

Compare the duration of the functions, studying the stalled cycles on any resource

- Click Options -> Click Select Metric -> Click Select Exclusive -> Click PAPI_RES_STL



Metric: PAPI_RES_STL
Value: Exclusive
Units: counts

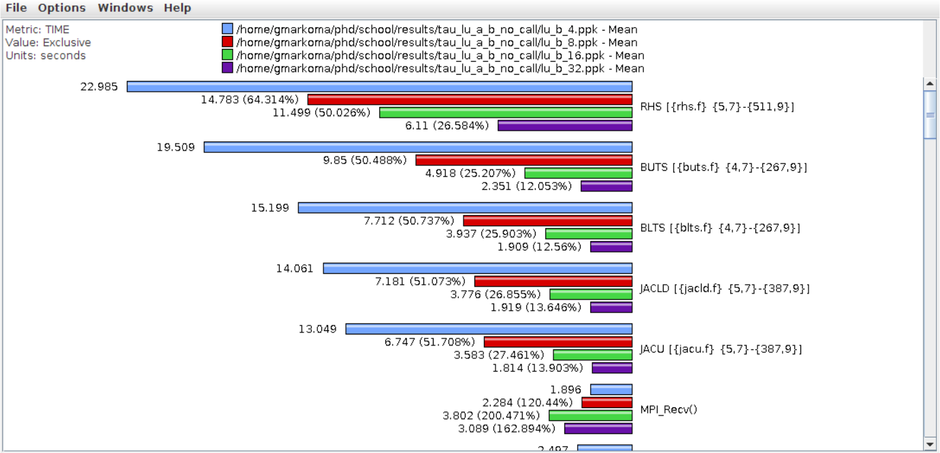
🟦 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_4.ppk - Mean
🟥 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_8.ppk - Mean
🟩 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_16.ppk - Mean
🟪 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_32.ppk - Mean

- Moreover the stalled cycles did not decrease as expected, thus the difference can be caused by this reason also.

Compare different executions per function for LU benchmark, class B

- Select the name of the first experiment for class B, do right click on it and select the option “Add Mean to Comparison Window”. A new window pops up, do not close it
- Repeat the procedure for all the experiments of class B

Compare the duration of the functions while we increase the number of the processes



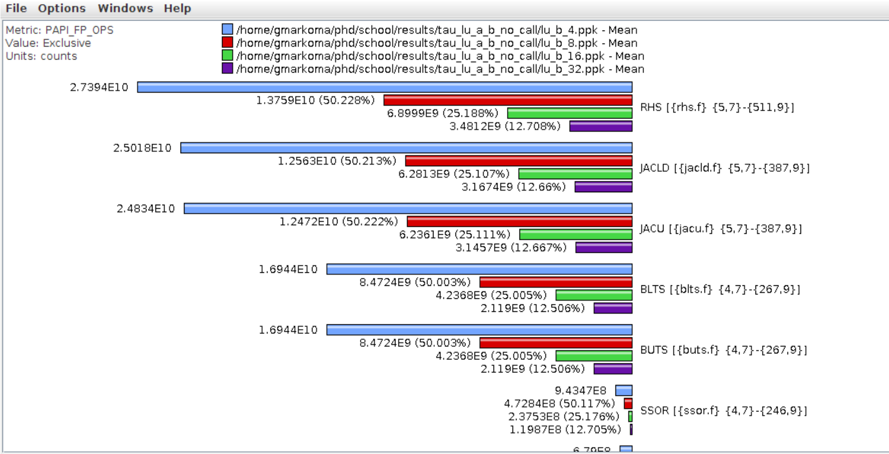
Metric: TIME
Value: Exclusive
Units: seconds

🟦 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_4.ppk - Mean
🟥 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_8.ppk - Mean
🟩 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_16.ppk - Mean
🟪 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_32.ppk - Mean

- While we double the number of the processes the duration of the RHS function is decreased by around to 36%

Compare the duration of the functions, studying the floating operations

- Click Options -> Click Select Metric -> Click Select Exclusive -> Click PAPI_FP_OPS



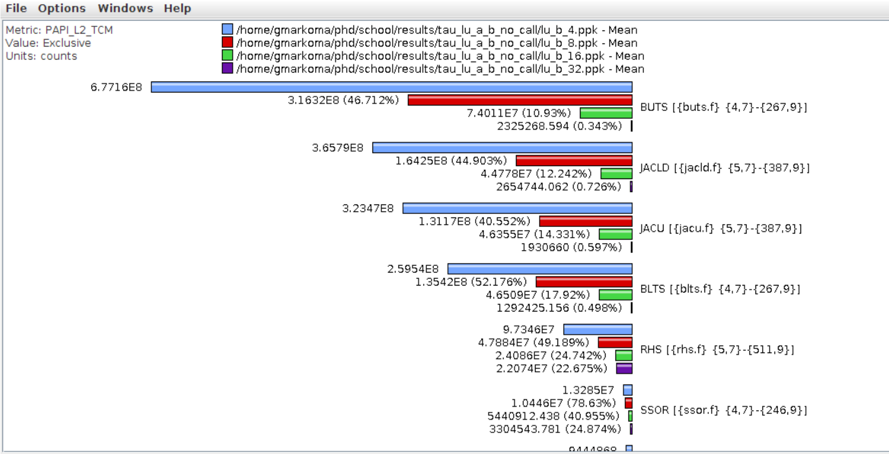
Metric: PAPI_FP_OPS
Value: Exclusive
Units: counts

File Options Windows Help
🟦 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_4.ppk - Mean
🟥 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_8.ppk - Mean
🟩 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_16.ppk - Mean
🟪 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_32.ppk - Mean
2.7394E10
1.3759E10 (50.228%)
6.8999E9 (25.188%)
3.4812E9 (12.708%)
RHS [{rhs.f} {5,7}-{511,9}]
2.5018E10
1.2563E10 (50.213%)
6.2813E9 (25.107%)
3.1674E9 (12.66%)
JACLD [{jacld.f} {5,7}-{387,9}]
2.4834E10
1.2472E10 (50.222%)
6.2361E9 (25.111%)
3.1457E9 (12.667%)
JACU [{jacu.f} {5,7}-{387,9}]
1.6944E10
8.4724E9 (50.003%)
4.2368E9 (25.005%)
2.119E9 (12.506%)
BLTS [{blts.f} {4,7}-{267,9}]
1.6944E10
8.4724E9 (50.003%)
4.2368E9 (25.005%)
2.119E9 (12.506%)
BUTS [{buts.f} {4,7}-{267,9}]
9.4347E8
4.7284E8 (50.117%)
2.3753E8 (25.176%)
1.1987E8 (12.705%)
SSOR [{ssor.f} {4,7}-{246,9}]

- The value of the floating operations do not justify the execution time of the function RHS

Compare the duration of the functions, studying L2 cache misses

- Click Options -> Click Select Metric -> Click Select Exclusive -> Click PAPI_L2_TCM



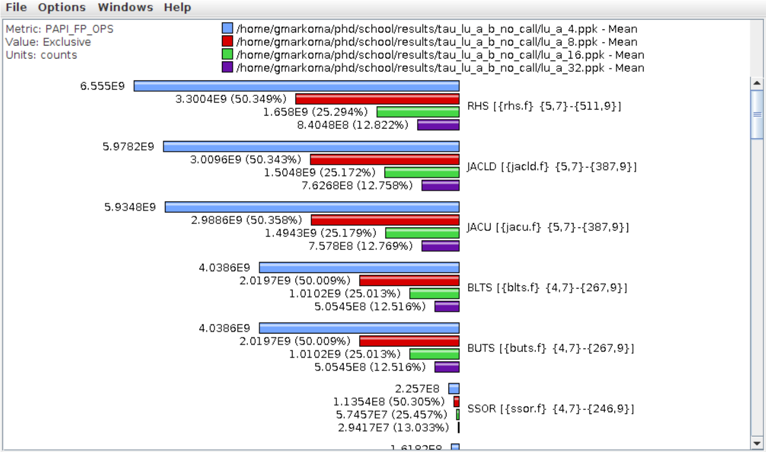
Metric: PAPI_L2_TCM
Value: Exclusive
Units: counts

File Options Windows Help
/home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_4.ppk - Mean
/home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_8.ppk - Mean
/home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_16.ppk - Mean
/home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_b_32.ppk - Mean
6.7716E8
3.1632E8 (46.712%)
7.4011E7 (10.93%)
2325268.594 (0.343%)
BUTS [{buts.f} {4,7}-{267,9}]
3.6579E8
1.6425E8 (44.903%)
4.4778E7 (12.242%)
2654744.062 (0.726%)
JACLD [{jacld.f} {5,7}-{387,9}]
3.2347E8
1.3117E8 (40.552%)
4.6355E7 (14.331%)
1930660 (0.597%)
JACU [{jacu.f} {5,7}-{387,9}]
2.5954E8
1.3542E8 (52.176%)
4.6509E7 (17.92%)
1292425.156 (0.498%)
BLTS [{blts.f} {4,7}-{267,9}]
9.7346E7
4.7884E7 (49.189%)
2.4086E7 (24.742%)
2.2074E7 (22.675%)
RHS [{rhs.f} {5,7}-{511,9}]
1.3285E7
1.0446E7 (78.63%)
5440912.438 (40.955%)
3304543.781 (24.874%)
SSOR [{ssor.f} {4,7}-{246,9}]
9444968

- Neither the L2 cache misses justify the execution time of the function RHS

Compare the duration of the functions, studying the total instructions

- Click Options -> Click Select Metric -> Click Select Exclusive -> Click PAPI_TOT_INS



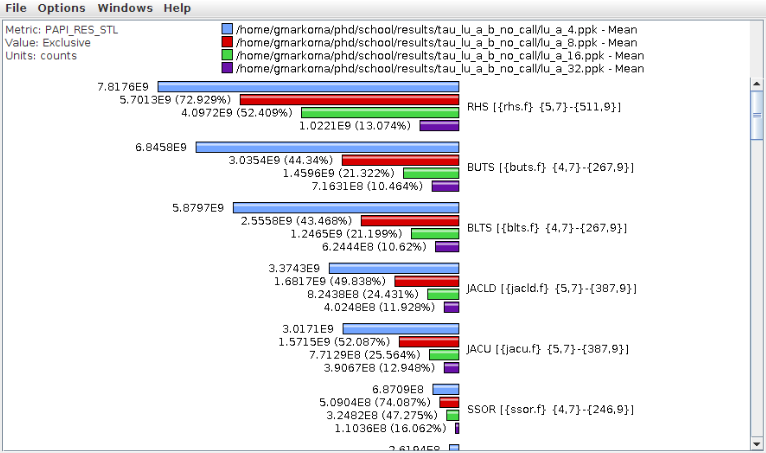
Metric: PAPI_FP_OPS
Value: Exclusive
Units: counts

🟦 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_4.ppk - Mean
🟥 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_8.ppk - Mean
🟩 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_16.ppk - Mean
🟪 /home/gmarkoma/phd/school/results/tau_lu_a_b_no_call/lu_a_32.ppk - Mean

- The value of the total completed instructions can justify a part of the mentioned difference

Compare the duration of the functions, studying the stalled cycles on any resource

- Click Options -> Click Select Metric -> Click Select Exclusive -> Click PAPI_RES_STL



- Moreover the stalled cycles did not decrease as expected, thus the difference can be caused by this reason also.

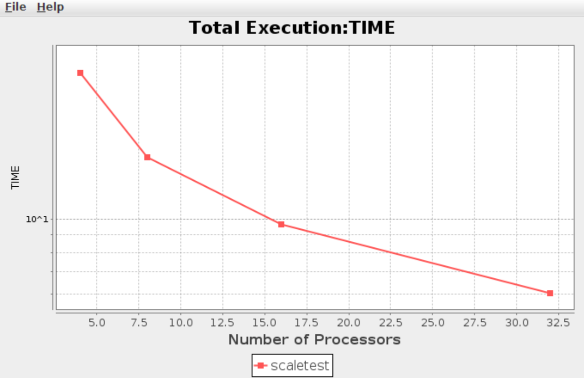
Add the experiments on the database perfdmf

- Add the experiments on the default database

```
% perfdmf_loadtrial -a sc_lu_a -x scaletest -n 4 lu_a_4.ppk
% perfdmf_loadtrial -a sc_lu_a -x scaletest -n 8 lu_a_8.ppk
% perfdmf_loadtrial -a sc_lu_a -x scaletest -n 16 lu_a_16.ppk
% perfdmf_loadtrial -a sc_lu_a -x scaletest -n 32 lu_a_32.ppk

% perfdmf_loadtrial -a sc_lu_b -x scaletest -n 4 lu_b_4.ppk
% perfdmf_loadtrial -a sc_lu_b -x scaletest -n 8 lu_b_8.ppk
% perfdmf_loadtrial -a sc_lu_b -x scaletest -n 16 lu_b_16.ppk
% perfdmf_loadtrial -a sc_lu_b -x scaletest -n 32 lu_b_32.ppk
```

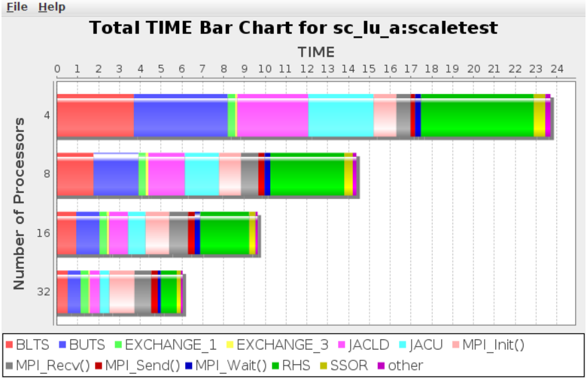
PerfExplorer, Total Execution Time for class A

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Total Execution Time -> Select the metric TIME -> Click OK



PerfExplorer, Stacked Bar Chart for class A and TIME

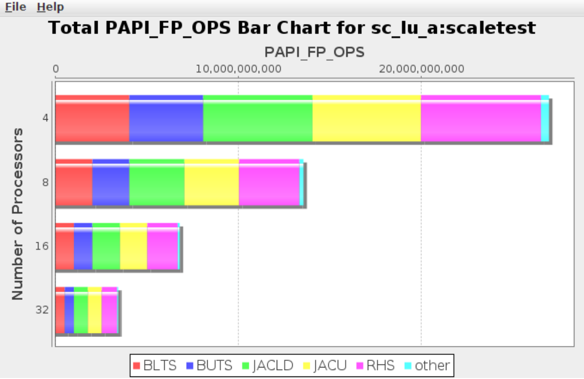
- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Stacked Bar Chart -> Select the metric TIME ->



Click OK

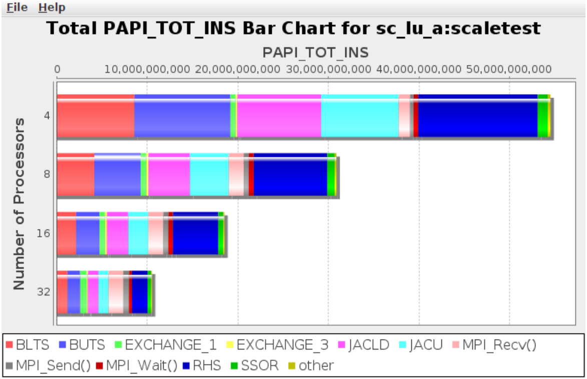
PerfExplorer, Stacked Bar Chart for class A and PAPI_FP_OPS

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Stacked Bar Chart -> Select the metric PAPI_FP_OPS -> Click OK


File Help
Total PAPI_FP_OPS Bar Chart for sc_lu_a:scaletest
PAPI_FP_OPS
0
10,000,000,000
20,000,000,000
Number of Processors
4
8
16
32
🟥 BLTS 🟦 BUTS 🟩 JACLD 🟨 JACU 🟪 RHS ■ other

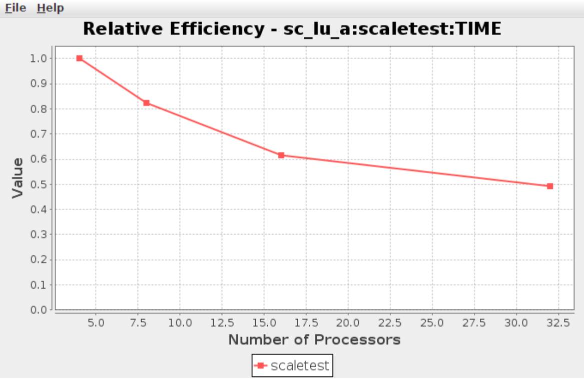
PerfExplorer, Stacked Bar Chart for class A and PAPI_TOT_INS

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Stacked Bar Chart -> Select the metric PAPI_TOT_INS -> Click OK



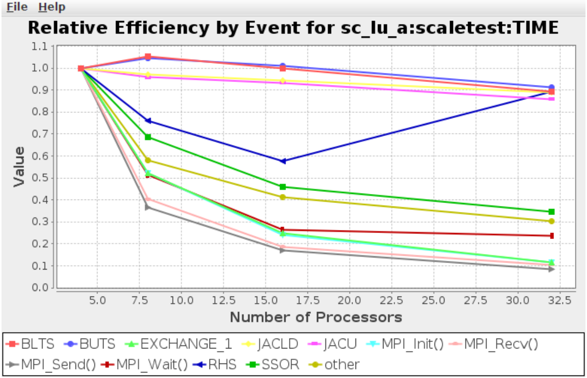
PerfExplorer, Relative Efficiency for class A

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Relative Efficiency -> Select the “The problem size remains constant“-> Click OK



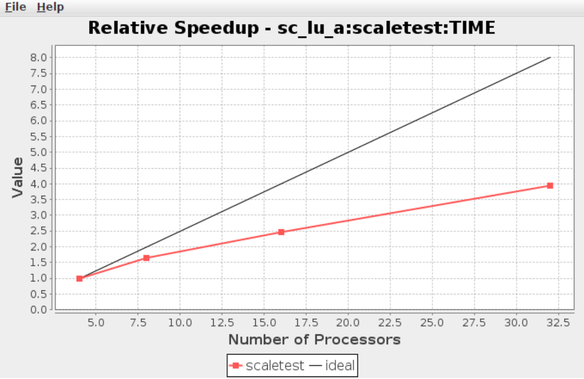
PerfExplorer, Relative Efficiency for class A per event

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Relative Efficiency by event -> Select the metric TIME -> Click OK -> Select the “The problem size remains constant“-> Click OK



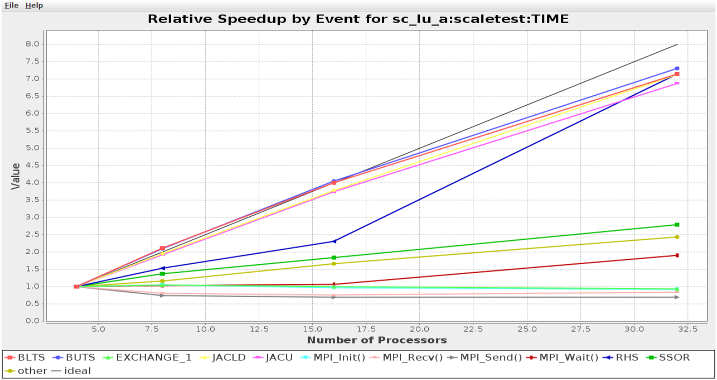
PerfExplorer, Relative Speedup for class A

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Relative Speedup -> Select the metric TIME -> Click OK -> Select the “The problem size remains constant“-> Click OK



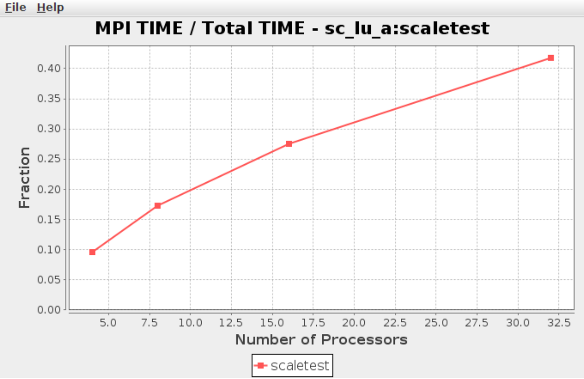
PerfExplorer, Relative Speedup by event for class A

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Relative Speedup -> Select the metric TIME -> Click OK -> Select the “The problem size remains constant“-> Click OK



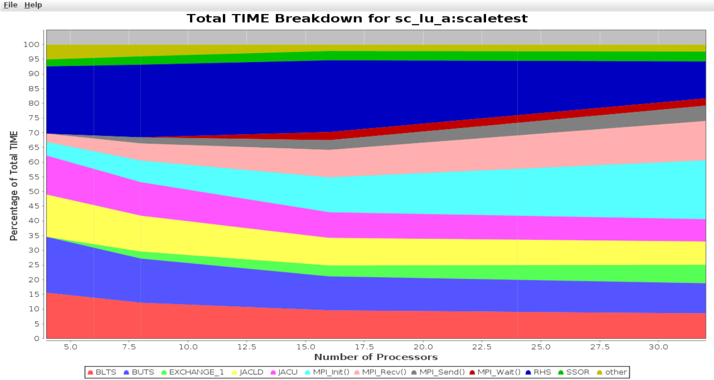
PerfExplorer, MPI Time for class A

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Group % of Total Runtime -> Select the metric TIME -> Click OK -> Select MPI group -> Click OK



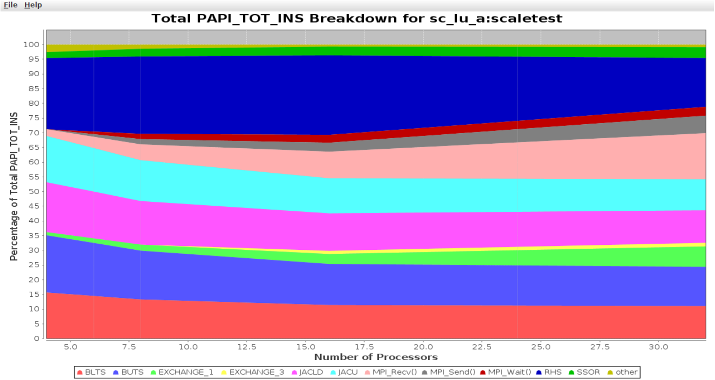
PerfExplorer, Runtime Breakdown for class A

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Runtime Breakdown -> Select the metric TIME -> Click OK



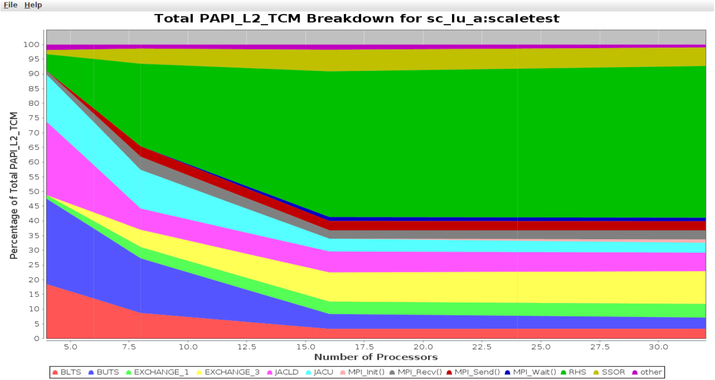
PerfExplorer, Runtime Breakdown PAPI_TOT_INS, for class A

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Runtime Breakdown -> Select the metric PAPI_TOT_INS -> Click OK



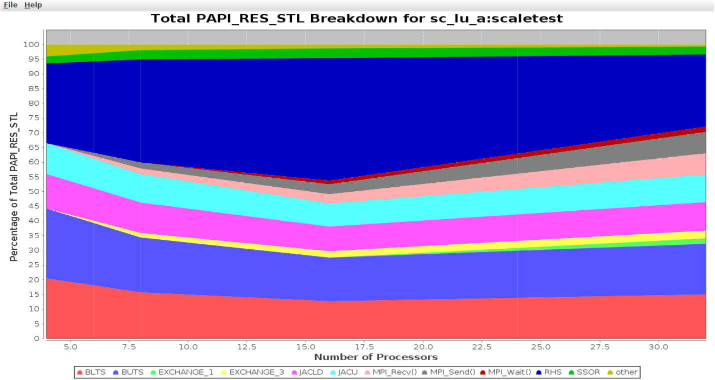
PerfExplorer, Runtime Breakdown PAPI_L2_TCM, for class A

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Runtime Breakdown -> Select the metric PAPI_L2_TCM -> Click OK



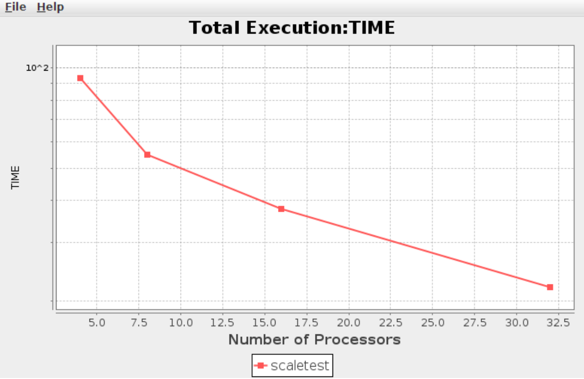
PerfExplorer, Runtime Breakdown PAPI_RES_STL, for class A

- Expand the database perfdmf -> Expland the Application name sc_lu_a -> Select the experiment name scaletest
- Click Charts -> Click Runtime Breakdown -> Select the metric PAPI_RES_STL -> Click OK



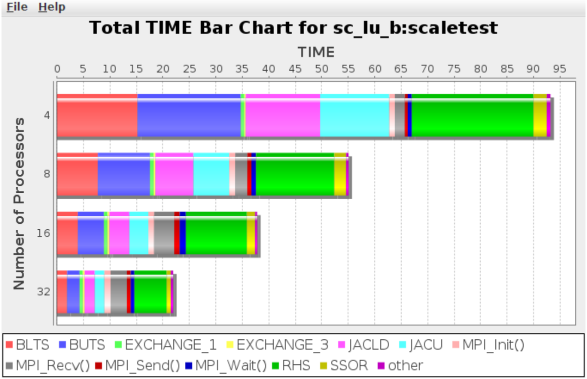
PerfExplorer, Total Execution Time for class B

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Total Execution Time -> Select the metric TIME -> Click OK



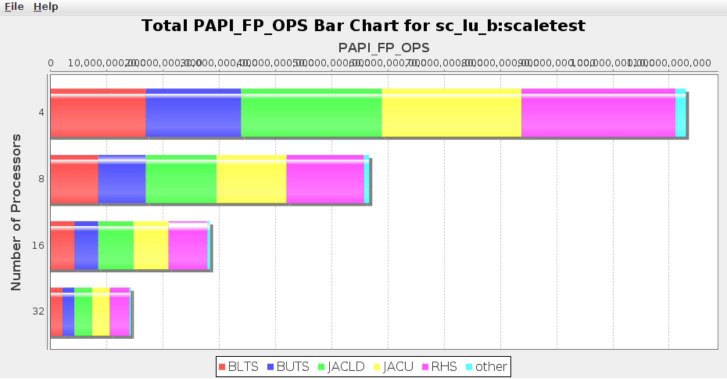
PerfExplorer, Stacked Bar Chart for class B

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Stacked Bar Chart -> Select the metric TIME -> Click OK



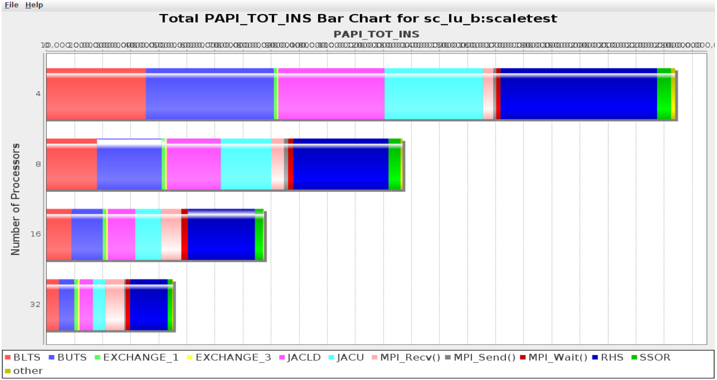
PerfExplorer, Stacked Bar Chart for class B and PAPI_FP_OPS

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Stacked Bar Chart -> Select the metric PAPI_FP_OPS -> Click OK



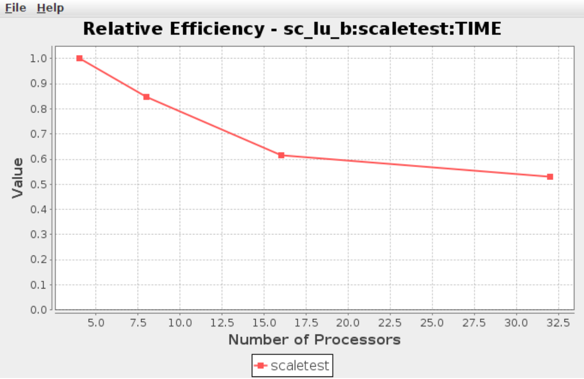
PerfExplorer, Stacked Bar Chart for class B and PAPI_TOT_INS

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Stacked Bar Chart -> Select the metric PAPI_TOT_INS -> Click OK



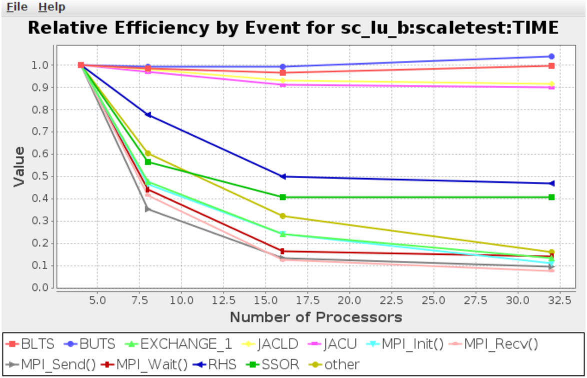
PerfExplorer, Relative Efficiency for class B

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Relative Efficiency -> Select the ”The problem size remains constant“ -> Click OK



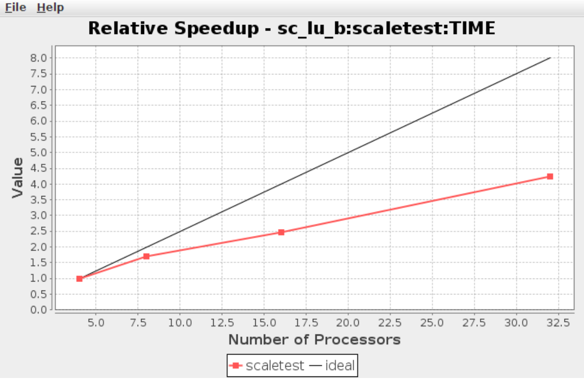
PerfExplorer, Relative Efficiency for class B per event

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Relative Efficiency by event -> Select the metric TIME -> Click OK -> Select the ”The problem size remains constant“-> Click OK



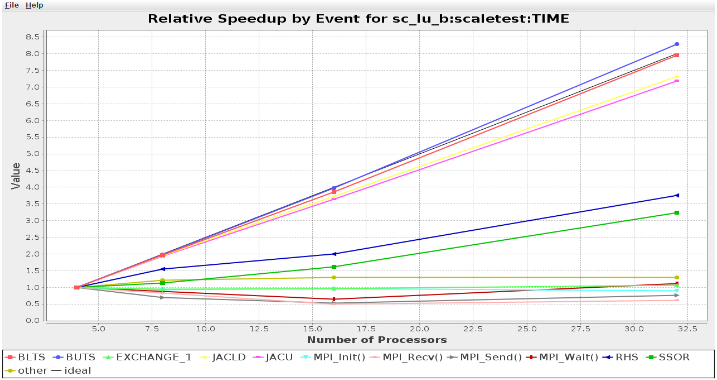
PerfExplorer, Relative Speedup for class B

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Relative Speedup -> Select the metric TIME -> Click OK -> Select the “The problem size remains constant“-> Click OK



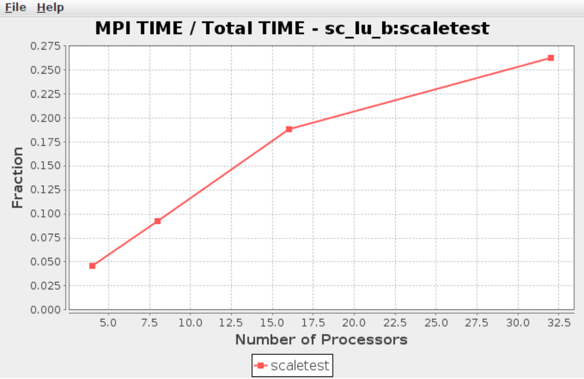
PerfExplorer, Relative Speedup by event for class B

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Relative Speedup -> Select the metric TIME -> Click OK -> Select the “The problem size remains constant“-> Click OK



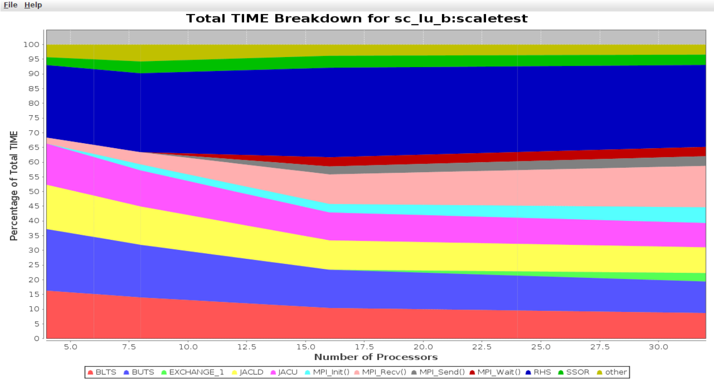
PerfExplorer, MPI Time for class B

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Group % of Total Runtime -> Select the metric TIME -> Click OK -> Select MPI group -> Click OK



PerfExplorer, Runtime Breakdown for class B

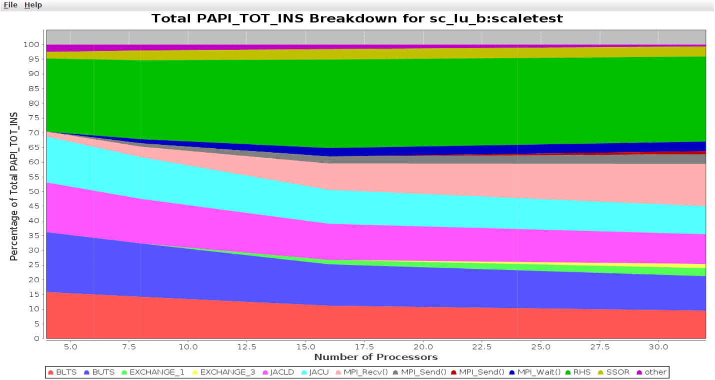
- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Runtime Breakdown -> Select the metric TIME ->



Click OK

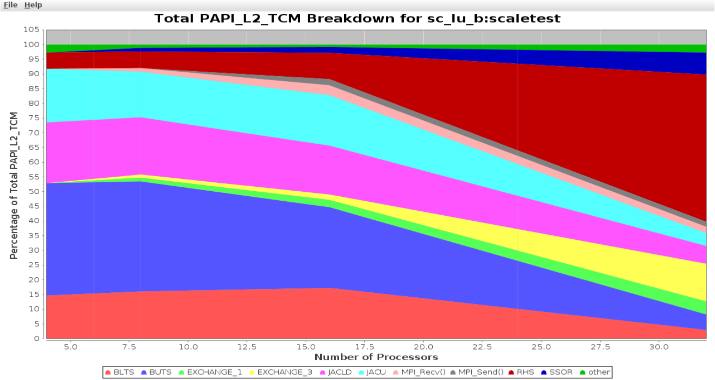
PerfExplorer, Runtime Breakdown PAPI_TOT_INS, for class B

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Runtime Breakdown -> Select the metric PAPI_TOT_INS -> Click OK



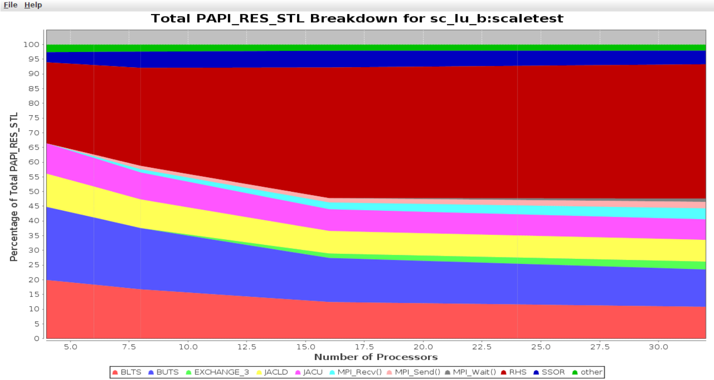
PerfExplorer, Runtime Breakdown PAPI_L2_TCM, for class B

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Runtime Breakdown -> Select the metric PAPI_L2_TCM -> Click OK



PerfExplorer, Runtime Breakdown PAPI_RES_STL, for class B

- Expand the database perfdmf -> Expland the Application name sc_lu_b -> Select the experiment name scaletest
- Click Charts -> Click Runtime Breakdown -> Select the metric PAPI_RES_STL -> Click OK



Apply dynamic phases on SSOR

- Declare the appropriate TAU stub Makefile

```
% vim ~/.bashrc
% export TAU_MAKEFILE=/srv/app/tau/x86_64/lib/Makefile.tau\
-phase-papi-mpi-pdt
```

- Declare where the phase starts and ends. The phase starts at the line 83 of the file ssor.f and ends at line 202. Moreover it is called ”iteration“

```
% vim dyn_phase.pdt
BEGIN_INSTRUMENT_SECTION
dynamic phase name="iteration" file="ssor.f" line=83 to line=202
END_INSTRUMENT_SECTION
```

- Declare the appropriate options

```
% vim ~/.bashrc
export TAU_OPTIONS='-optPDTInst -optTauSelectFile=/path/ \
dyn_phase.pdt'
```

- Compile the instances for 4-32 processes from the roof folder of NAS

```
% make clean; make LU NPROCS=4 CLASS=A
% make clean; make LU NPROCS=8 CLASS=A
...
% make clean; make LU NPROCS=4 CLASS=B
% make clean; make LU NPROCS=8 CLASS=B
...
```

Apply dynamic phases on SSOR

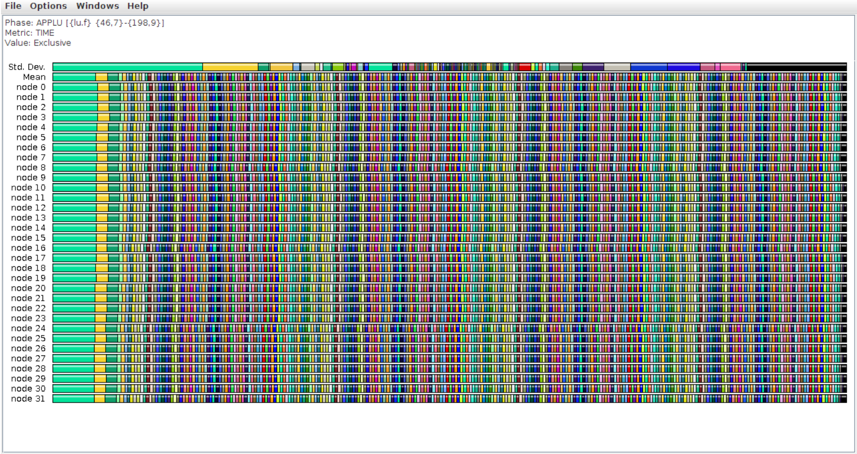
- Execute the benchmarks and pack the performance data

```
% cd bin.tau
% rm -r MULTI*
% mpirun --bind-to-core -np 4 lu.A.4
% paraprof --pack lu_a_4_phases.ppk
% rm -r MULTI*
% mpirun --bind-to-core -np 8 lu.A.8
% paraprof --pack lu_a_8_phases.ppk
...
```

- View the data with the Paraprof tool

```
% paraprof lu_a_4_phases.ppk
```

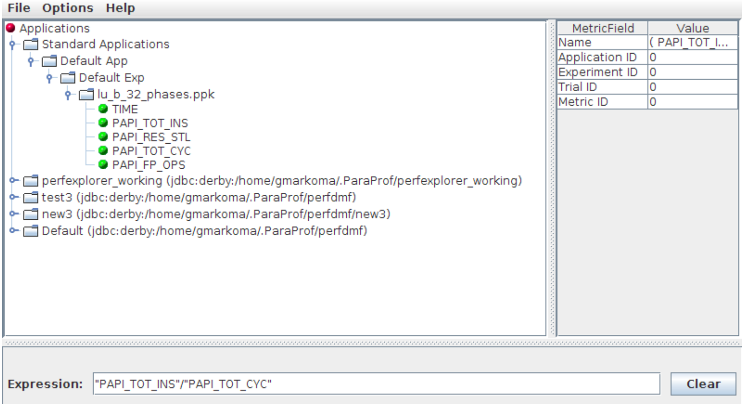
Paraprof and dynamic phases for the LU benchmark, class B, 32 processes



Phase: APPLU [{lu.f} {46,7}-{198,9}]
Metric: TIME
Value: Exclusive

Create an expression of a new metric

- Close the Paraprof sub-window (the one on top of the Paraprof Manager)
- From the Paraprof Manager, Click Options -> Click Show Derived Metric Panel
- Expand the lu_a_4_phases experiment (example in the screenshot for the lu_b_32_phases)
- Select the PAPI_TOT_INS metric, after click the symvol ”/“ from the Derived Metric Panel and select the metric PAPI_TOT_CYC
- Click Apply



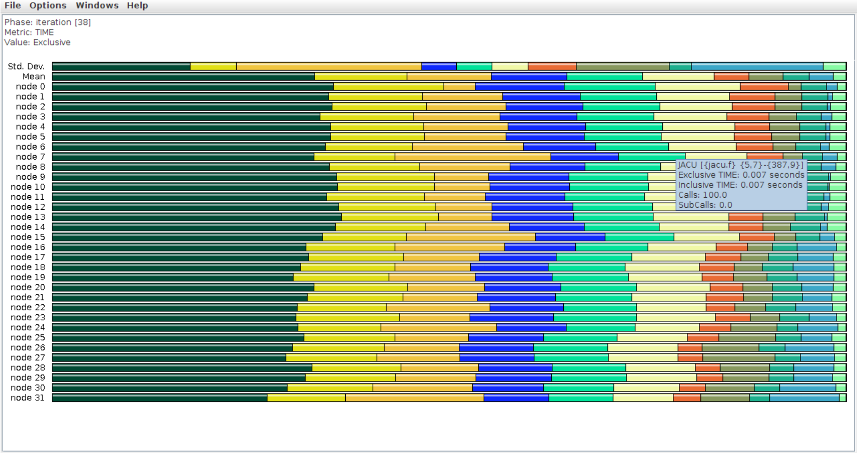
- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - lu_b_32_phases.ppk
          - TIME
          - PAPI_TOT_INS
          - PAPI_RES_STL
          - PAPI_TOT_CYC
          - PAPI_FP_OPS
  - perfexplorer_working (jdbc:derby:/home/gmarkoma/.ParaProf/perfexplorer_working)
  - test3 (jdbc:derby:/home/gmarkoma/.ParaProf/perfdmf)
  - new3 (jdbc:derby:/home/gmarkoma/.ParaProf/perfdmf/new3)
  - Default (jdbc:derby:/home/gmarkoma/.ParaProf/perfdmf)

MetricField	Value
Name	(PAPI_TOT_I...
Application ID	0
Experiment ID	0
Trial ID	0
Metric ID	0

Expression: "PAPI_TOT_INS"/"PAPI_TOT_CYC"

Profile of a phase

- Double click on the metric TIME of the lu_a_4_phases.ppk
- Right click on any iteration (small continuous areas) and select Open Profile for this Phase



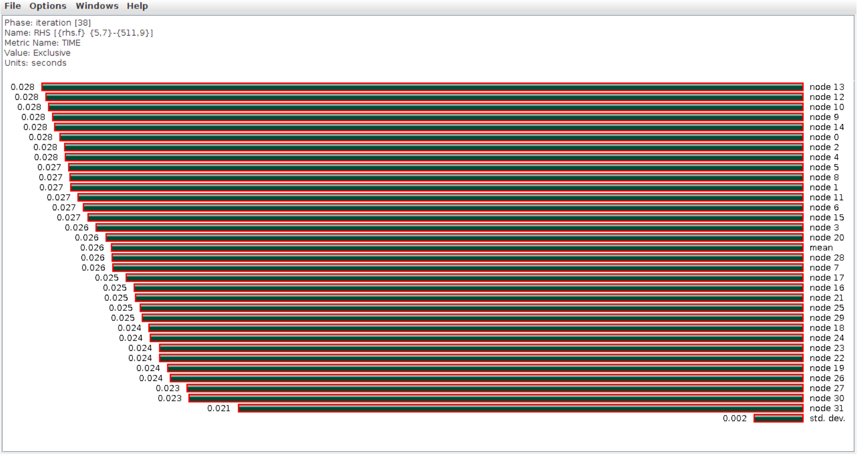
Phase: iteration [38]
Metric: TIME
Value: Exclusive

JACU [{jacu.f} {5,7}-{387,9}]
Exclusive TIME: 0.007 seconds
Inclusive TIME: 0.007 seconds
Calls: 100.0
SubCalls: 0.0

- We chose randomly the 38th iteration

Study the phase

- Select the first group of bar charts, and you can see the following

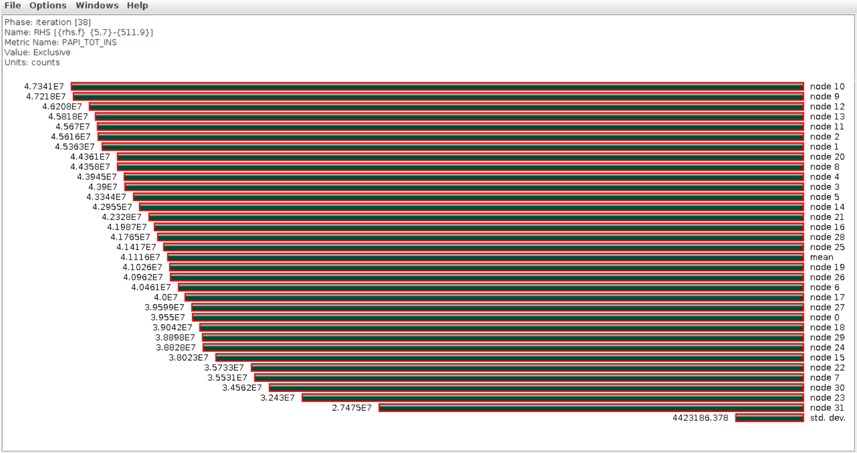


Phase: iteration [38]
Name: RHS [{rhs.f} {5,7}-{511,9}]
Metric Name: TIME
Value: Exclusive
Units: seconds

- We can observe that for the 38th iteration the duration of the function RHS varies from 0.021 to 0.028 seconds (33%)

Study the instructions of a specific iteration

- Click Options -> Select Metric -> Exclusive -> PAPI_TOT_INS

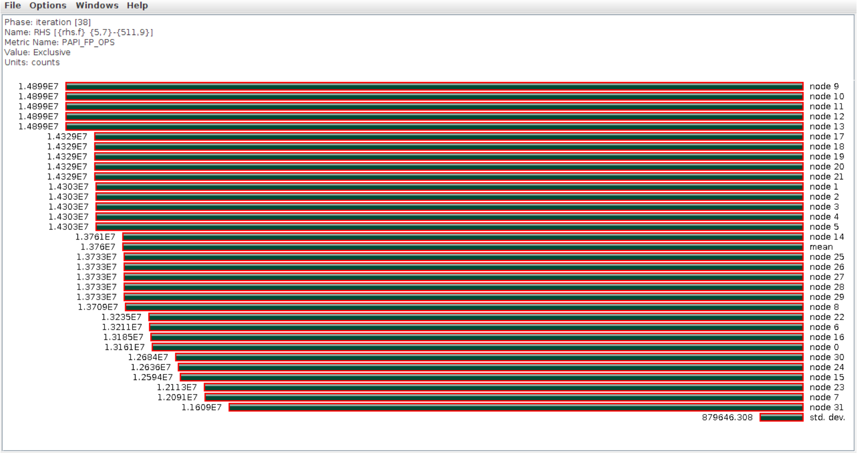


Phase: iteration [38]
Name: RHS [{rhs.f} {5,7}-{511,9}]
Metric Name: PAPI_TOT_INS
Value: Exclusive
Units: counts

- We can observe that for the 38th iteration the value of the total completed instructions for the function RHS varies from 2.7475E7 to 4.7341E7 (72.3%)

Study the floating operations of a specific iteration

- Click Options -> Select Metric -> Exclusive -> PAPI_FP_OPS

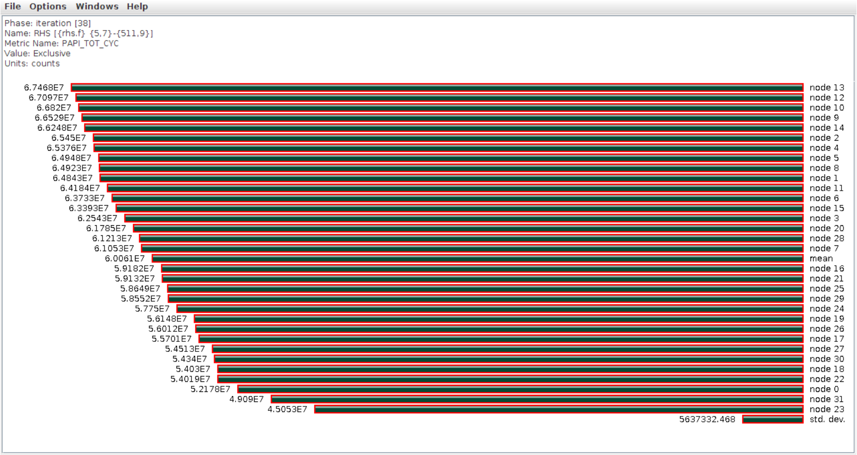


Phase: iteration [38]
Name: RHS [{rhs.f} {5,7}-{511,9}]
Metric Name: PAPI_FP_OPS
Value: Exclusive
Units: counts

- We can observe that for the 38th iteration the value of the total completed instructions for the function RHS varies from 1.16E7 to 1.49E7 (28.4%)
- This indicates a computational imbalance at least for this iteration

Study the cycles of a specific iteration

- Click Options -> Select Metric -> Exclusive -> PAPI_TOT_CYC

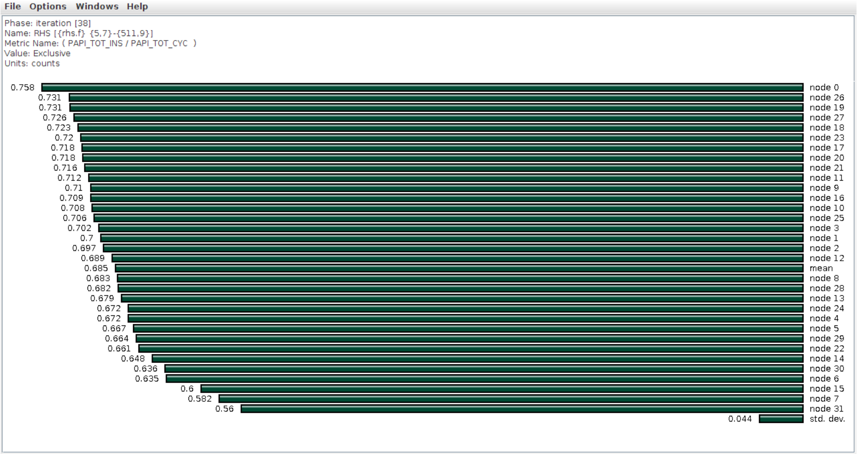


Phase: iteration [38]
Name: RHS [{rhs.f} {5,7}-{511,9}]
Metric Name: PAPI_TOT_CYC
Value: Exclusive
Units: counts

- We can observe that for the 38th iteration the value of the cycles for the function RHS varies from 4.5E7 to 6.746E7 (49.9%)

Study the instructions per cycle of a specific iteration

- Click Options -> Select Metric -> Exclusive -> (PAPI_TOT_INS / PAPI_TOT_CYC)

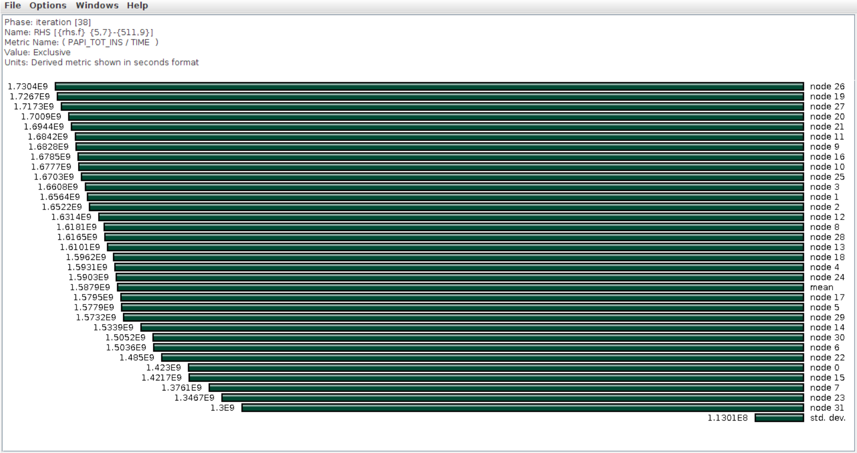


Phase: iteration [38]
Name: RHS [{rhs.f} {5,7}-{511,9}]
Metric Name: (PAPI_TOT_INS / PAPI_TOT_CYC)
Value: Exclusive
Units: counts

- We can observe that for the 38th iteration the value of the IPC for the function RHS varies from 0.56 to 0.758 (35.3%)

Study the instructions per second of a specific iteration

- Click Options -> Select Metric -> Exclusive -> (PAPI_TOT_INS / TIME)

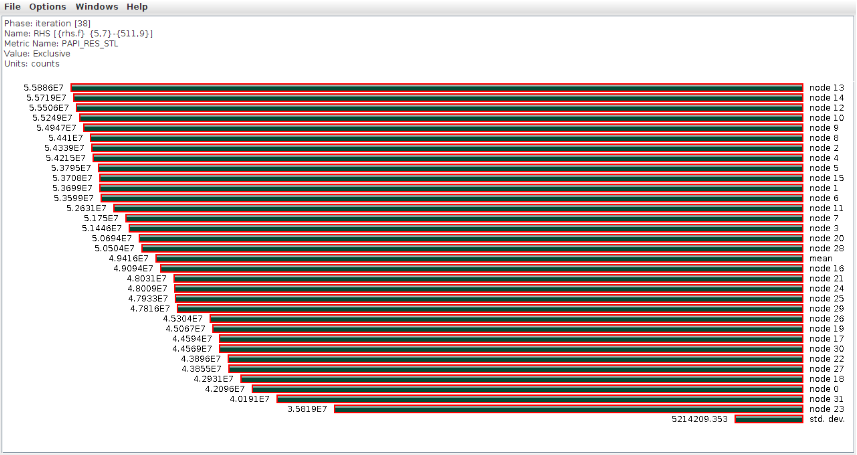


Phase: iteration [38]
Name: RHS [{rhs.f} {5,7}-{511,9}]
Metric Name: (PAPI_TOT_INS / TIME)
Value: Exclusive
Units: Derived metric shown in seconds format

- We can observe that for the 38th iteration the value of the IPC for the function RHS varies from 1.3E9 to 1.73E9 (33%)

Study the stalled cycles of a specific iteration

- Click Options -> Select Metric -> Exclusive -> PAPI_RES_STL

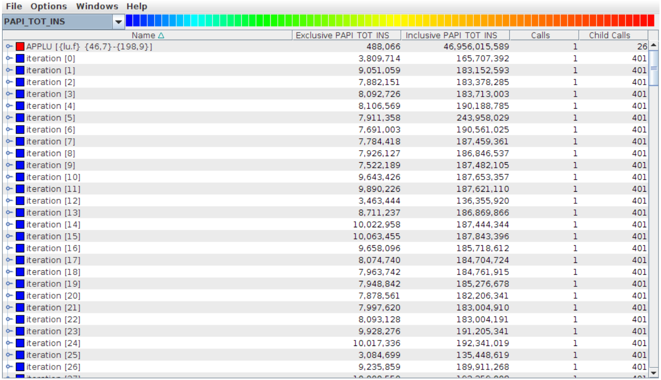


Phase: iteration [38]
Name: RHS [{rhs.f} {5,7}-{511,9}]
Metric Name: PAPI_RES_STL
Value: Exclusive
Units: counts

- We can observe that for the 38th iteration the value of the cycles for the function RHS varies from 3.58E7 to 5.59E7 (56%)

Compare specific iterations

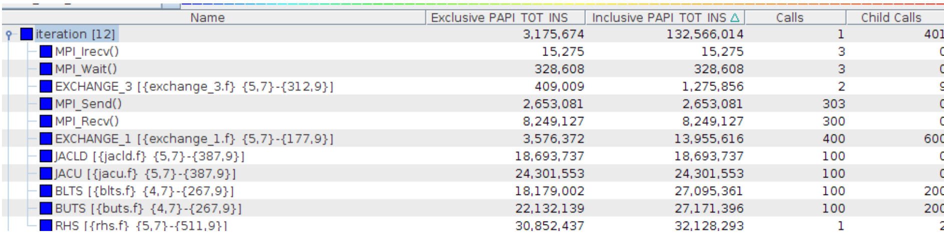
- Double click the metric PAPI_TOT_INS from the Paraprof Manager window
- On the new window right click for example on node 3 and select ”Show Thread Statistics Table“



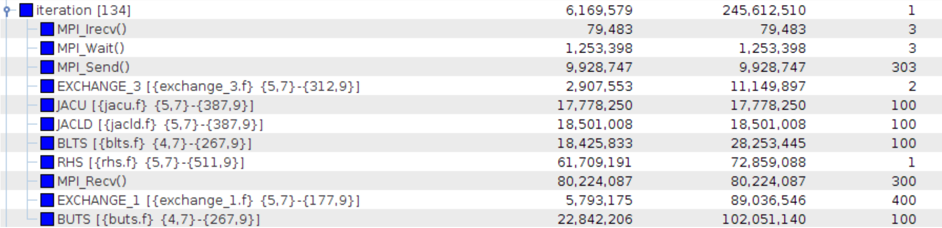
Name △	Exclusive PAPI_TOT_INS	Inclusive PAPI_TOT_INS	Calls	Child Calls
APPLU [{lu.f} {46,7}-{198,9}]	488,066	46,956,015,589	1	26
iteration [0]	3,809,714	165,707,392	1	401
iteration [1]	9,051,059	183,152,593	1	401
iteration [2]	7,882,151	183,378,285	1	401
iteration [3]	8,092,726	183,713,003	1	401
iteration [4]	8,106,569	190,188,785	1	401
iteration [5]	7,911,358	243,958,029	1	401
iteration [6]	7,691,003	190,561,025	1	401
iteration [7]	7,784,418	187,459,361	1	401
iteration [8]	7,926,127	186,846,537	1	401
iteration [9]	7,522,189	187,482,105	1	401
iteration [10]	9,643,426	187,653,357	1	401
iteration [11]	9,890,226	187,621,110	1	401
iteration [12]	3,463,444	136,355,920	1	401
iteration [13]	8,711,237	186,869,866	1	401
iteration [14]	10,022,958	187,444,344	1	401
iteration [15]	10,063,455	187,843,396	1	401
iteration [16]	9,658,096	185,718,612	1	401
iteration [17]	8,074,740	184,704,724	1	401
iteration [18]	7,963,742	184,761,915	1	401
iteration [19]	7,948,842	185,276,678	1	401
iteration [20]	7,878,561	182,206,341	1	401
iteration [21]	7,997,620	183,004,910	1	401
iteration [22]	8,093,128	183,004,191	1	401
iteration [23]	9,928,276	191,205,341	1	401
iteration [24]	10,017,336	192,341,019	1	401
iteration [25]	3,084,699	135,448,619	1	401
iteration [26]	9,235,859	189,911,268	1	401
iteration [27]	[illegible]	[illegible]	1	401

- Now we can choose an iteration and expand it
- Sort by the ”Inclusive PAPI_TOT_INS“ by clicking on the head of the column. The minimum value is 1.32E8 and the maximum 2.45E8 (85.6%) so thereis computational imbalance

Which functions cause the previous difference



Name	Exclusive PAPI_TOT_INS	Inclusive PAPI_TOT_INS △	Calls	Child Calls
iteration [12]	3,175,674	132,566,014	1	401
MPI_Irecv()	15,275	15,275	3	0
MPI_Wait()	328,608	328,608	3	0
EXCHANGE_3 [{exchange_3.f} {5,7}-{312,9}]	409,009	1,275,856	2	9
MPI_Send()	2,653,081	2,653,081	303	0
MPI_Recv()	8,249,127	8,249,127	300	0
EXCHANGE_1 [{exchange_1.f} {5,7}-{177,9}]	3,576,372	13,955,616	400	600
JACLD [{jacld.f} {5,7}-{387,9}]	18,693,737	18,693,737	100	0
JACU [{jacu.f} {5,7}-{387,9}]	24,301,553	24,301,553	100	0
BLTS [{blts.f} {4,7}-{267,9}]	18,179,002	27,095,361	100	200
BUTS [{buts.f} {4,7}-{267,9}]	22,132,139	27,171,396	100	200
RHS [{rhs.f} {5,7}-{511,9}]	30,852,437	32,128,293	1	2



iteration [134]	6,169,579	245,612,510	1
MPI_Irecv()	79,483	79,483	3
MPI_Wait()	1,253,398	1,253,398	3
MPI_Send()	9,928,747	9,928,747	303
EXCHANGE_3 [{exchange_3.f} {5,7}-{312,9}]	2,907,553	11,149,897	2
JACU [{jacu.f} {5,7}-{387,9}]	17,778,250	17,778,250	100
JACLD [{jacld.f} {5,7}-{387,9}]	18,501,008	18,501,008	100
BLTS [{blts.f} {4,7}-{267,9}]	18,425,833	28,253,445	100
RHS [{rhs.f} {5,7}-{511,9}]	61,709,191	72,859,088	1
MPI_Recv()	80,224,087	80,224,087	300
EXCHANGE_1 [{exchange_1.f} {5,7}-{177,9}]	5,793,175	89,036,546	400
BUTS [{buts.f} {4,7}-{267,9}]	22,842,206	102,051,140	100

Conclusions

- In general be sure that you are trusting the hardware
- Be careful about your measurements. Identify any strange result that is obvious
- Plotting the characteristics of a function can be different related to each loop
- Create multiple dynamic phases for identifying strange behavior on iterative procedures (be careful about the overhead)
- The metric of the stalled cycles on any resource is a good one for investigating if there is any overhead

PerfExpert

PerfExpert tool

- Not only measures but also analyses performance
  - Tell us where the slow code sections are as well why they perform poorly
  - Suggests source-code changes (unfortunately only for icc compiler for now)
  - Simple to use

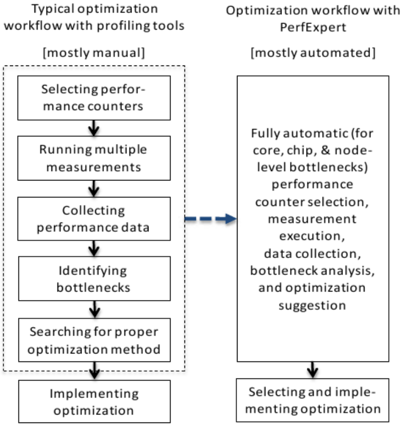
PerfExpert tool

- Identification of potential causes for slow speed
  - We can find a lot of information through various tools
- How can we decide if a value is big or not?
  - There are 25,578,391 L2 cache misses in a loop, is it good?
  - How can we reduce it?

PerfExpert tool

- It uses the HPCToolkit
- It executes the application many times for measuring various metrics
- In every execution the total completed instructions are measured in order to be able to compare the different execution in the case of any variation
- It identifies and characterizes the causes of each bottleneck in each code segment
- Local Cycles Per Instruction (LCPI) introduced

PerfExpert



Typical optimization workflow with profiling tools

[mostly manual]

Selecting performance counters

Running multiple measurements

Collecting performance data

Identifying bottlenecks

Searching for proper optimization method

Implementing optimization

Optimization workflow with PerfExpert

[mostly automated]

Fully automatic (for core, chip, & node-level bottlenecks) performance counter selection, measurement execution, data collection, bottleneck analysis, and optimization suggestion

Selecting and implementing optimization

PerfExpert tool

- During the installation, PerfExpert measures various architecture parameters, L1 data access latency etc.
- The LCPI values are a combination of PAPI metrics and architecture parameters

Local Cycles Per Instruction

- Data Accesses, L1 data hits

  ```
   (PAPI_LD_INS * L1_dlat) / PAPI_TOT_INS
  ```
- Data Accesses, L2 data misses

  ```
  ((PAPI_L2_TCM - PAPI_L2_ICM) * mem_lat) / PAPI_TOT_INS
  ```
- Instruction Accesses, L2 instruction misses

  ```
  PAPI_L2_ICM * mem_lat / PAPI_TOT_INS
  ```

PerfExpert tool

- Easy to use, no need to re-compile
- Compile your application as you already do
- Execute the LU benchmark with PerfExpert

```
% mpirun --bind-to-core -np 4 perfexpert_run_exp ./lu.A.4
```

- An XML file (experiment.xml) is created and we can see the output of the functions which consume at least the 10% of the total execution with the following command

```
 perfexpert 0.1 experiment.xml
```

Output

```
Function rhs_() (19.4% of the total runtime)
===============================================================================
ratio to total instrns       %  0.........25...........50.........75........100
   - floating point      :   50 ************************
   - data accesses       :   40 *******************
* GFLOPS (% max)         :   13 ******
-------------------------------------------------------------------------------
performance assessment     LCPI good......okay......fair......poor......bad....
* overall                :  0.9 >>>>>>>>>>>>>>>>>>>
upper bound estimates
* data accesses          :  1.2 >>>>>>>>>>>>>>>>>>>>>>>
   - L1d hits            :  0.4 >>>>>>>>
   - L2d hits            :  0.4 >>>>>>>
   - L2d misses          :  0.4 >>>>>>>>
* instruction accesses   :  1.2 >>>>>>>>>>>>>>>>>>>>>>>>
   - L1i hits            :  0.2 >>>>
   - L2i hits            :  0.0 >
   - L2i misses          :  1.0 >>>>>>>>>>>>>>>>>>>>
* data TLB               :  0.0 >
* instruction TLB        :  0.0 >
* branch instructions    :  0.1 >
   - correctly predicted :  0.0 >
   - mispredicted        :  0.0 >
* floating-point instr   :  1.6 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
   - fast FP instr       :  1.2 >>>>>>>>>>>>>>>>>>>>>>>>
   - slow FP instr       :  0.4 >>>>>>>
```

Explanation

- In general if the LCPI value is less than 0.5 then it is considered as a good value
- Compared to the value of the total completed instructions there are 50% floating point operations and 40% data accesses
- The value GFLOPS represent the percentage of the maximum possible GFLOP value for the specific machine
- The overal performance is the cycles per instruction which is not very good in this example
- It seems that the overhead is distributed on L1 data hits, L2 data hits and L2 cache misses but is not too much each one
- However the L2 instructions misses cost is really big
- The fast FP instr includes the floating point multiply and add instructions
- the slow FP instr includes the floating point divide instructions

Now we know where we should look for identifying the reason of the overhead

Compare two executions

- Rename your previous experiment file

```
% mv experiment.xml perf_lu_a_4.xml
```

- Execute the LU benchmark for class A and 8 processes

```
% mpirun --bind-to-core -np 8 perfexpert_run_exp ./lu.A.8
```

- Compare your data

```
% perfexpert 0.1 perf_lu_a_4.xml experiment.xml
```

Output of the comparison

```
Function rhs_() (runtimes are 5.438s and 3.326s)
=================================================================================
ratio to total instrns        %  0.........25...........50.........75........100
   - floating point       :      ***********************
   - data accesses        :      *******************
* GFLOPS (% max)          :      *****
---------------------------------------------------------------------------------
performance assessment      LCPI good......okay......fair......poor......bad....
* overall                 :      >>>>>>>>>>>>>>>>>>>222
upper bound estimates
* data accesses           :      >>>>>>>>>>>>>>>>>>>>>>>22
   - L1d hits             :      >>>>>>>>
   - L2d hits             :      >>>>>>>
   - L2d misses           :      >>>>>>>>22
* instruction accesses    :      >>>>>>>>>>>>>>>>>>>>>11
   - L1i hits             :      >>>>
   - L2i hits             :      >
   - L2i misses           :      >>>>>>>>>>>>>>>>>11
* data TLB                :      >
* instruction TLB         :      >
* branch instructions     :      >
   - correctly predicted  :      >
   - mispredicted         :      >
* floating-point instr    :      >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
   - fast FP instr        :      >>>>>>>>>>>>>>>>>>>>>>>>
   - slow FP instr        :      >>>>>>>
```

- The value 1 or 2 at the end of the evaluation means which application has bigger value on this metric

AutoSCOPE

- Status
  - Know that there is a performance problem
  - Know why it performs poorly
  - Do not know how to improve the performance
- AutoSCOPE
  - Suggests remedies based on analysis results
    - Including code examples and compiler flags
    - For the moment only for Intel compiler (soon for gcc?)

Use AutoSCOPE

- Save the output of the *perfexpert* call

```
% perfexpert 0.1 perf_lu_a_4.xml > output_lu_a_4
```

- Call the autoscope

```
% autoscope output_lu_a_4
Function rhs_() (19.4% of the total runtime)
=================================================================
* eliminate floating-point operations through distributivity
- example: d[i] = a[i] * b[i] + a[i] * c[i]; ->
             d[i] = a[i] * (b[i] + c[i]);

* eliminate floating-point operations through associativity
- example:d[i]=(a[i] * b[i]) * c[i]; y[i] = (x[i] * a[i]) * b[i];->
    temp = a[i] * b[i]; d[i] = temp * c[i]; y[i] = x[i] * temp;

* use trace scheduling to reduce the branch taken frequency
- example: if (likely_condition) f(); else g(); h(); ->
  void s() {g(); h();} ... if (!likely_condition) {s();} f(); h();
```

AutoSCOPE

```
* factor out common code into subroutines
  - example: ... same_code ... same_code ... ->
   void f() {same_code;} ... f() ... f() ...;

* allow inlining only for subroutines with one call site or very short
  bodies
  - compiler flag: use the "-nolib-inline", "-fno-inline",
 "-fno-inline-functions", or "-finline-limit=" (with a small ) compiler
   flags

* make subroutines more general and use them more
  - example: void f() {statements1; same_code;}
    void g() {statements2; same_code;} ->
   void fg(int flag) {if (flag) {statements1;} else {statements2;}
    same_code;}

* split off cold code into separate subroutines and place them at the
   end of the source file
  - example: if (unlikely_condition) {lots_of_code} ->
  void f() {lots_of_code} ... if (unlikely_condition) f();

* reduce the code size
  - compiler flag: use the "-Os" or "-O1" compiler flag
```

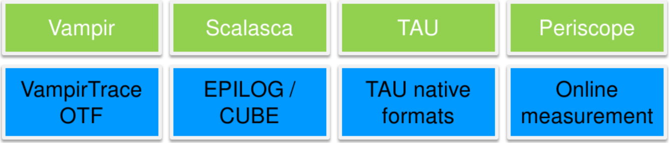
AutoSCOPE for the loop of RHS function

```
Loop in function rhs_() (19.4% of the total runtime)
======================================================================
* move loop invariant computations out of loop
  - example: loop i {x = x + a * b * c[i];} ->
   temp = a * b; loop i {x = x + temp * c[i];}

* lower the loop unroll factor
  - example: loop i step 4 {code_i; code_i+1; code_i+2; code_i+3;} ->
    loop i step 2 {code_i; code_i+1;}
  - compiler flag: use the "-no-unroll-aggressive" compiler flag
```

Score-P - A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU and Vampir

Why a new tool?

- Several performance tools co-exist
- Different measurement systems and output format
- Complementary features and overlapping functionality
- Redundant effort for development and maintenance
- Limited or expensive interoperability
- Complications for user experience, support, training

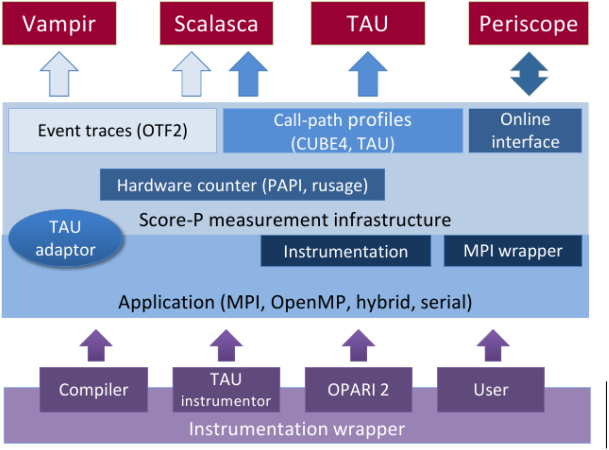


Vampir	Scalasca	TAU	Periscope
VampirTrace OTF	EPILOG / CUBE	TAU native formats	Online measurement

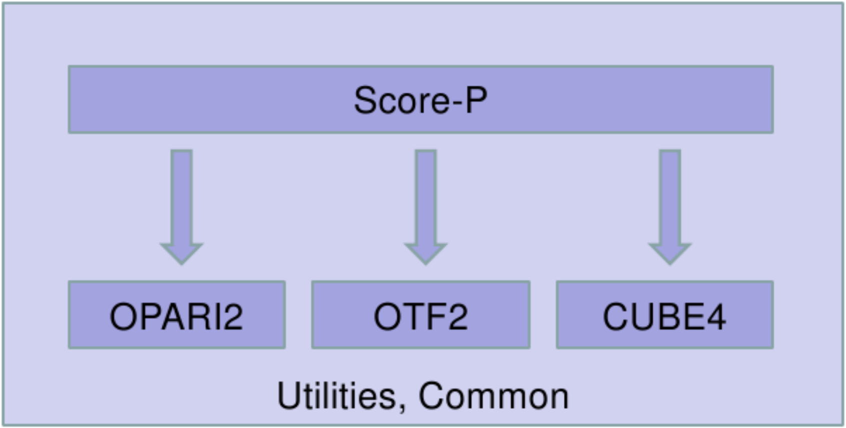
Idea

- Common infrastructure and effort
- Common data formats OTF2 and CUBE4
- Sharing ideas and implement faster
- No effort for maintenance, testing etc for various tools
- Single learning curve

Score-P Architecture



Components



- Separate, stand-alone packages
- Common functionality factored out
- Automated builds and tests

Score-P

- Instrumenter *scorep*
- Links application to measurement library *libscorep_(serial|omp|mpi|mpi_omp)*
- Records time, visits, communication metrics, hardware counters
  - Efficient utilization of available memory
  - Minimize perturbation/overhead
  - Useful for unification
  - Access data during runtime
- Switch modes (tracing, profiling, online) without recompilation

Score-P Instrumentation

- Instrument

```
mpicc -c foo.c -> scorep mpicc -c foo.c
```

- Help

```
 % scorep --help
...
--user       Enables manual user instrumentation.
--nouser     Disables manual user instrumentation. Is
             disabled by default.
--pdt        Enables source code instrumentation with PDT
             using the TAU instrumentor.
             It will automatically enable the user
             instrumentation and disable compiler
             instrumentation.
---
```

- Instrument with PDT

```
scorep --pdt mpicc -c foo.c
```

- Automatic detect serial/OpenMP/MPI/hybrid

Score-P Run-Time Recording

- Uncomment the appropriate MPIF77 command in config/make.def of NAS benchmarks

```
MPIF77 = scorep mpif77
```

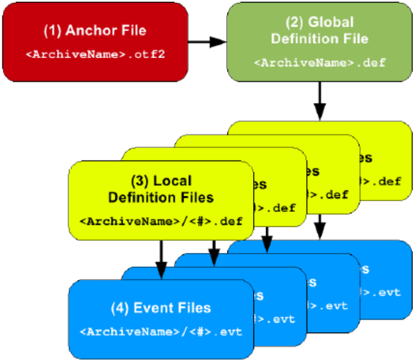
- Customize via environment variables

Environment Variable	Default	Description
SCOREP_ENABLE_PROFILING	1	Setting to 0 turns off profiling
SCOREP_ENABLE_TRACING	1	Setting to 0 turns off tracing
SCOREP_TOTAL_MEMORY	1200k	Total memory in bytes for the measurement system excluding trace memory
SCOREP_EXPERIMENT_DIRECTORY	”“	Name of the experiment directory
SCOREP_MPI_ENABLE_GROUPS	DEFAULT	The names of the function groups which are measured (CG, P2P...)
SCOREP_SELECTIVE_CONFIG_FILE	”“	A file name which configures selective tracing
SCOREP_FILTER_FILE	”“	A file name which contain the filter rules
SCOREP_PROFILING_MAX_CALLPATH_DEPTH	30	Maximum depth of the calltree
SCOREP_PROFILING_FORMAT	DEFAULT	Profile output format (NONE, TAU_SNAPSHOT, CUBE4, DEFAULT)
SCOREP_METRIC_PAPI	”“	PAPI metric names

- It supports selective tracing

The Open Trace Format Version 2 (OTF2)

- Event trace data format
  - Event record types + definition record types
- Multi-file format
  - Anchor file
  - Global and local definitions + mappings
  - Event files
- OTF2 API



Re-design OTF2

- One process/thread per file
- Memory event trace buffer becomes part of trace format
- No re-write for unification, mapping tables
- Forward/Backward reading

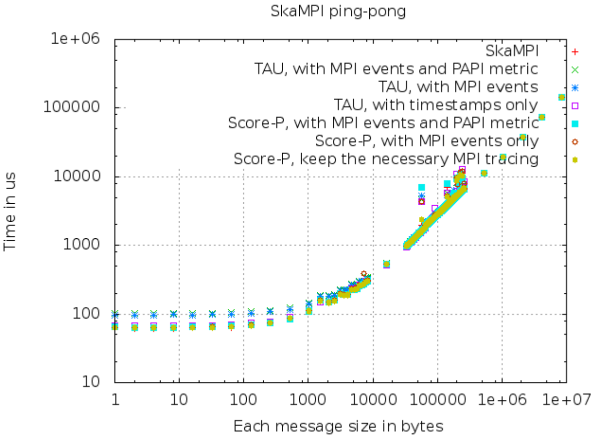
Selective Tracing

- Score-P allows to disable the instrumentation on specific parts of the code (SCOREP_RECORDING_OFF/ON)
- It allows online access for handling the data on the fly for profiling mode
- Parameters profiling, we can split-up the callpath for executions of different parameter values (INT64, UINT64, String)

Future features

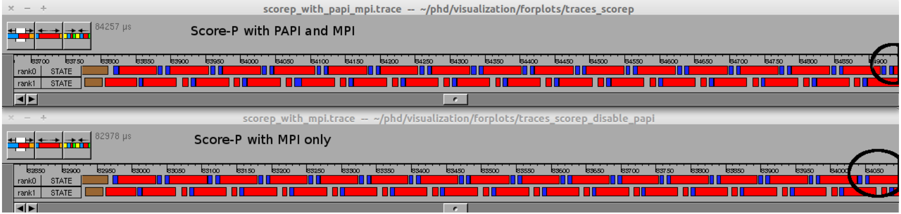
- Scalability to maximum available CPU core count
- Support for sampling, binary instrumentation
- Support for new architectures
- Allow experimental versions of new features or research
- Future integration in Open MPI releases

Accuracy: SkaMPI vs TAU vs Score-P



- Score-P provides less overhead compared to TAU

Accuracy: Score-P and PAPI



scorep_with_papi_mpi.trace -- ~/phd/visualization/forplots/traces_scorep

84257 µs

Score-P with PAPI and MPI

scorep_with_mpi.trace -- ~/phd/visualization/forplots/traces_scorep_disable_papi

82978 µs

Score-P with MPI only

- Comparing the tracing of Score-P with and without PAPI and visualize the traces through Paje format. PAPI measurement adds some overhead

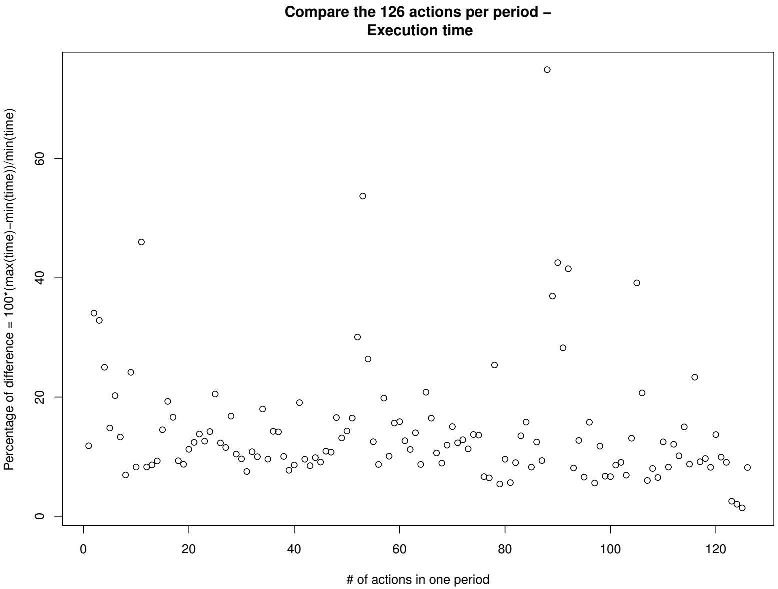
Accuracy: Hardware counters

- Apply selective instrumentation for capturing only MPI events with PAPI without any info for the computation

```
BEGIN_FILE_EXCLUDE_LIST
*
END_FILE_EXCLUDE_LIST
```

- Execution of the LU benchmark, class A and 4 processes on the cluster bordereau (Grid'5000): 36.24 seconds, 82.36 billions instructions
- Without the exclusion: 46.4 seconds, 92.9 billions instructions
- 12.79% of the instructions caused by the instrumentation tool!

Accuracy: Scalasca and periodicity data



- Time variation of the executed code which maps only to computation,

Thank you!
Questions?