Intel® MPI Library

Intel® MPI Library

Value Proposition

What	• **Intel's High Performance MPI Library**
Why	• **Scale Performance –** Tuned for Latest Intel Architectures • **Scale Forward –** Multicore and Manycore Ready • **Scale Efficiently –** Flexible Fabric Selection & Compatibility
How	• **Standards Based –** Built on Open Source MPICH Implementation • **Sustained Scalability –** Tuning for Low Latencies, High Bandwidth & Increased Processes • **Multi Fabric Support –** Supports Popular High Performance Networking Fabrics

Intel® MPI Library Overview

Streamlined product setup

- Install as root, or as standard user
- Environment variable script mpivars.(c)sh sets paths

Compilation scripts to handle details

- One set to use Intel compilers, one set for user-specified compilers

Environment variables for runtime control

- I_MPI_* variables control many factors at runtime
  - Process pinning, collective algorithms, device protocols, and more

Compiling MPI Programs

Compilation Scripts

- Automatically adds necessary links to MPI libraries and passes options to underlying compiler
- Use *mpiifort*, *mpiicpc*, or *mpiicc* to force usage of the associated Intel compiler
- Use *mpif77*, *mpicxx*, *mpicc*, or others to allow user to specify compiler (I_MPI_F77, … or –f77=, -cxx=, …)
  - Useful for makefiles portable between MPI implementations
- All compilers are found via PATH

MPI Launcher

Robust launch command

```
mpirun <mpi args> executable <program args>
```

Options available for:

- Rank distribution and pinning
- Fabric selection and control
- Environment propagation
- And more

Understanding MPI and Launcher Behavior

I_MPI_DEBUG=<level>

Debug Levels (cumulative):

- 0 – *Default*, no debug information
- 1 – Verbose error diagnostics
- 2 – Fabric selection process
- 3 – Rank, PID, node mapping
- 4 – Process pinning
- 5 – Display Intel® MPI Library environment variables
- 6 – Collective operation algorithm controls

I_MPI_HYDRA_DEBUG=1 turns on Hydra debug output

- Keep in mind that this gives a LOT of output. Only turn on if needed

Process Placement

Default placement puts one rank per core on each node

Use –ppn to control processes per node

Use a machinefile to define ranks on each node individually

Use arguments sets or configuration files for precise control for complex jobs

Fabric Selection

I_MPI_FABRICS=<intranode fabric>:<internode fabric> *or* <fabric>

Fabric options

- shm – Shared Memory (only valid for intranode)
- dapl – Direct Access Provider Library*
- ofa – Open Fabric Alliance (OFED* verbs)
- tmi – Tag Matching Interface
- tcp – Ethernet/Sockets
- ofi – OpenFabrics Interfaces*

Default behavior goes through a list to find first working fabric combination

If you specify a fabric, fallback is disabled, I_MPI_FALLBACK=1 to re-enable

Environment Propagation

Use –[g]env[*] to control environment propagation

- Adding g propagates to all ranks, otherwise only to ranks in current argument set

-env <variable> <value> Set <variable> to <value>

-envuser All user environment variables, with a few exceptions (Default)

-envall All environment variables

-envnone No environment variables

-envlist <variable list> Only the listed variables

What’s New: Intel® MPI Library 2018

- Up to 11x faster job start-up performance.
- Up to 25% reduction in job finalization time.
- Added support for the latest Intel® Xeon® Scalable processor.

Handling Heterogeneous Jobs

Global Options vs. Local Options

Global Options are applied to all ranks

- -ppn, -genv, …

Local Options are applied to a subset of ranks

- -n, -host, -env, …

WARNING: Some options can be set as local options via environment variable, but must be consistent across job

- Collective algorithms
- Fabric selection and parameters

Configuration Files and Argument Sets

Arguments Sets are used on the command line

Configuration Files are pulled from the file specified by *–configfile <configfile>*

Global arguments appear first (first line, or at beginning of first argument set)

Local arguments for each argument set next

Separated by : on command line (don’t separate globals), new line in configfile

Can be used to run heterogeneous binaries, different arguments for each binary, different environment variables, etc.

All ranks combined in order specified into one job

Examples

Configuration File

```
$ cat theconfigfile
-genv OMP_NUM_THREADS 4
-n 6 –host node1 ./exe1
-n 4 –host node2 ./exe2
# -n 4 –host dead_node3 ./exe3
-n 6 –host node4 ./exe4
$ mpirun –configfile theconfigfile
```

Argument Set

```
$ mpirun –genv OMP_NUM_THREADS 4 –n 6 –host node1 ./exe1
: -n 4 –host node2 ./exe2 : –n 6 –host node4 ./exe4
```

Intel® Trace Analyzer and Collector (ITAC)

Intel® Trace Analyzer and Collector

Value Proposition

What	• **Intel's High Performance MPI Communications Profiler & Analyzer for Scalable HPC Development**
Why	• **Scale Performance –** Perform on More Nodes • **Scale Forward –** Multicore and Manycore Ready • **Scale Efficiently –** Tune & Debug on More Nodes
How	• **Visualize -** Understand parallel application behavior • **Evaluate -** Profiling statistics and load balancing • **Analyze** – Automated analysis of common MPI issues • **Identify –** Communication hotspots

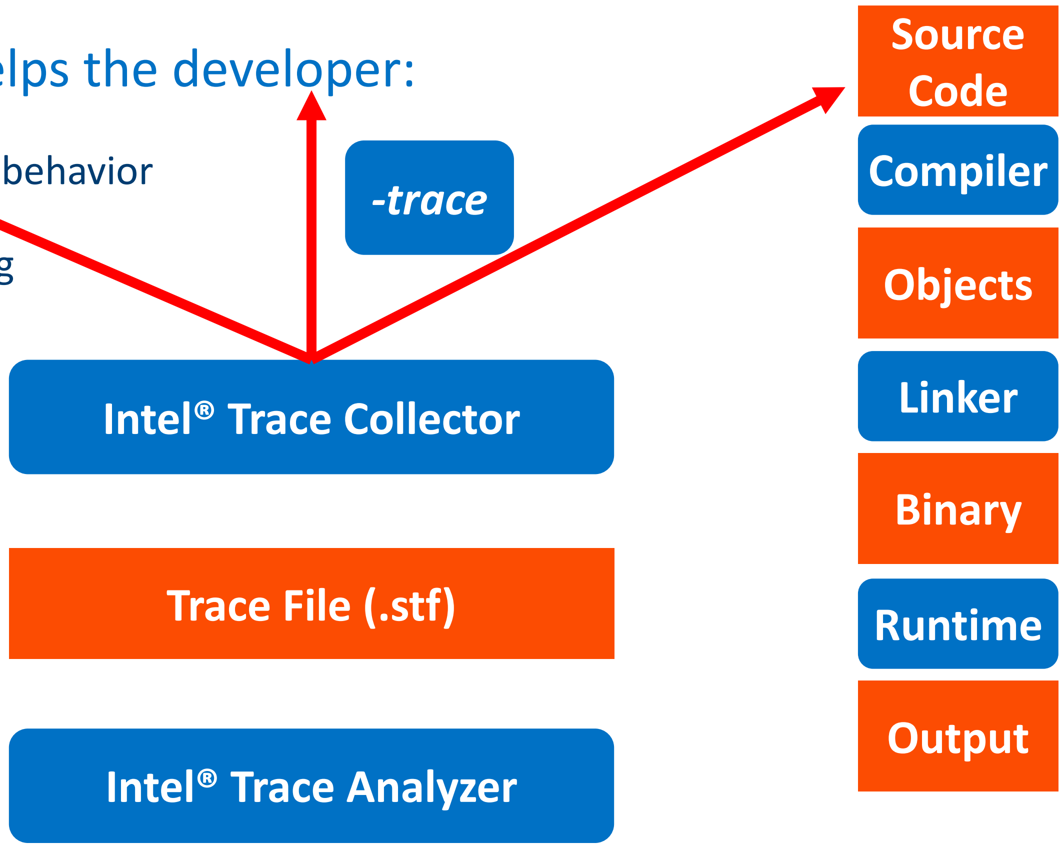
Intel® Trace Analyzer and Collector Overview

Intel® Trace Analyzer and Collector helps the developer:

- Visualize and understand parallel application behavior
- Evaluate profiling s … cing
- Identify communication hotspots

Features

- Event-based approach
- Low overhead
- Excellent scalability
- Powerful aggregation and filtering functions
- Performance Assistance and Imbalance Tuning



Strengths of Event-based Tracing

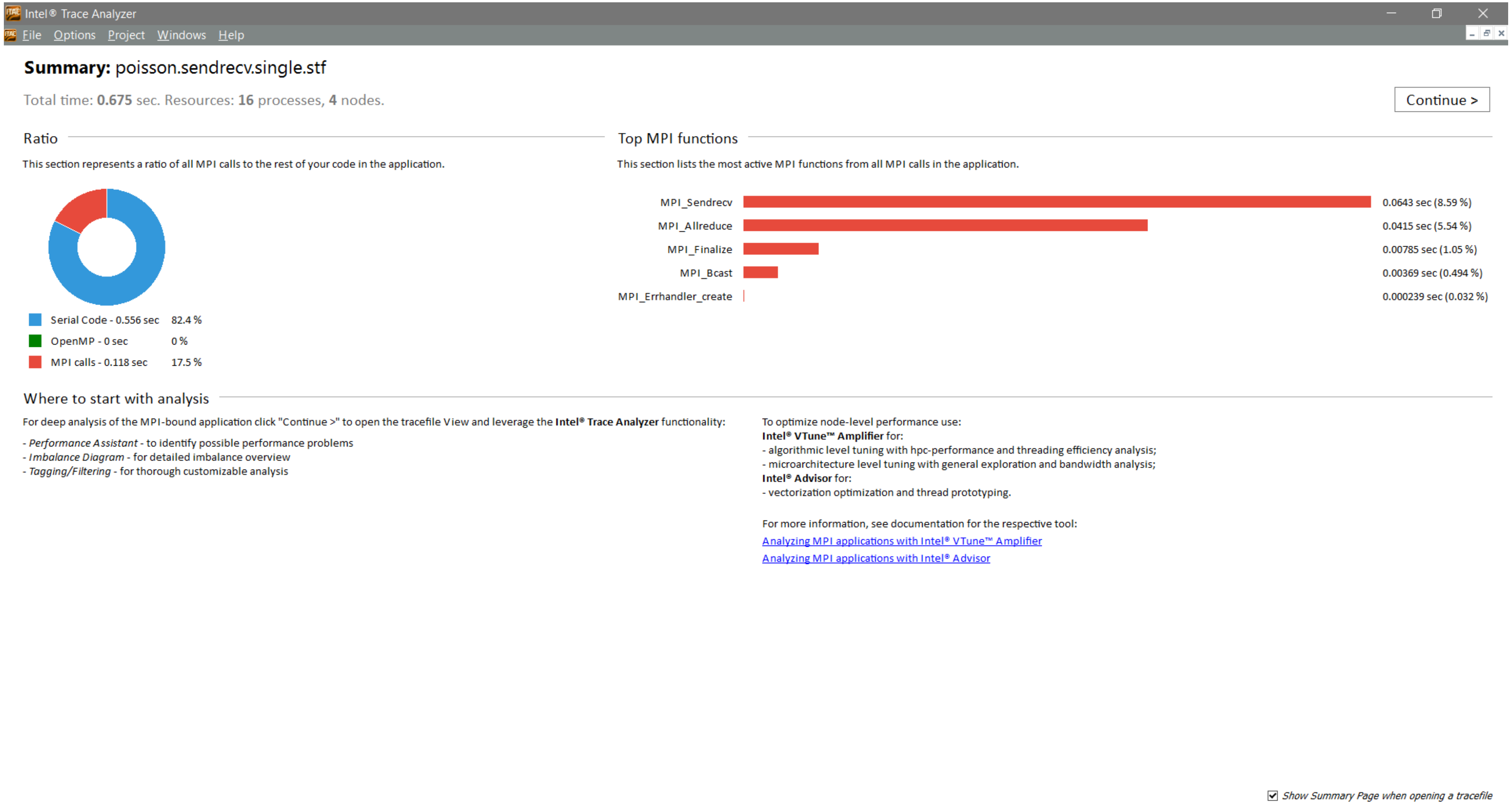
Predict	Detailed MPI program behavior
Record	Exact sequence of program states – keep timing consistent
Collect	Collect information about exchange of messages: at what times and in which order

An event-based approach is able to detect temporal dependencies!

Multiple Methods for Data Collection

Collection Mechanism	Advantages	Disadvantages
Run with –trace or preload trace collector library.	**Automatically collects all MPI calls, requires no modification to source, compile, or link.**	**No user code collection.**
Link with –trace.	Automatically collects all MPI calls.	No user code collection. Must be done at link time.
Compile with –tcollect.	Automatically instruments all function entries/exits.	Requires recompile of code.
Add API calls to source code.	Can selectively instrument desired code sections.	Requires code modification.

Intel® Trace Analyzer Summary Page



Views and Charts

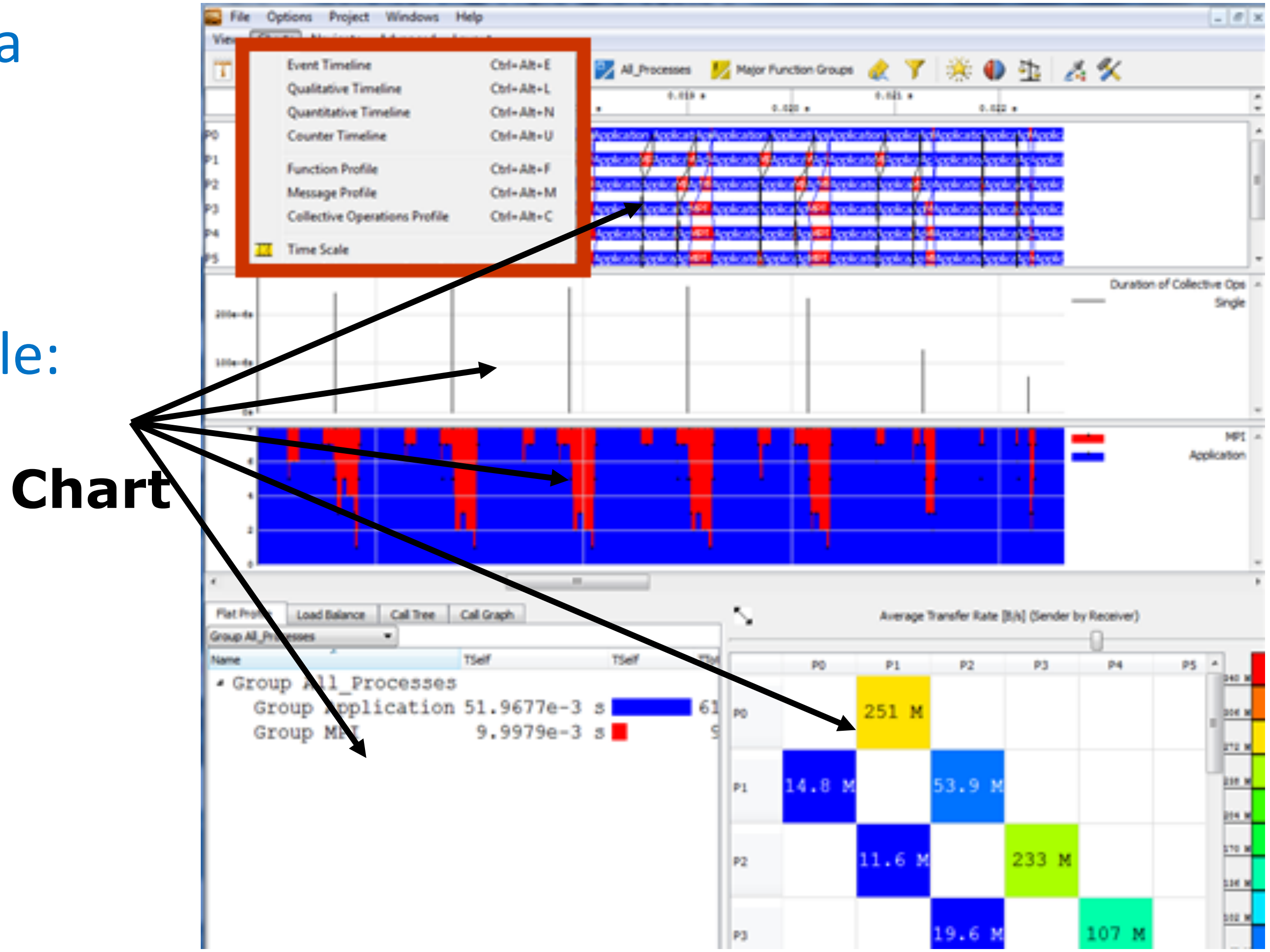
Helps navigating through the trace data and keep orientation

Every View can contain several Charts

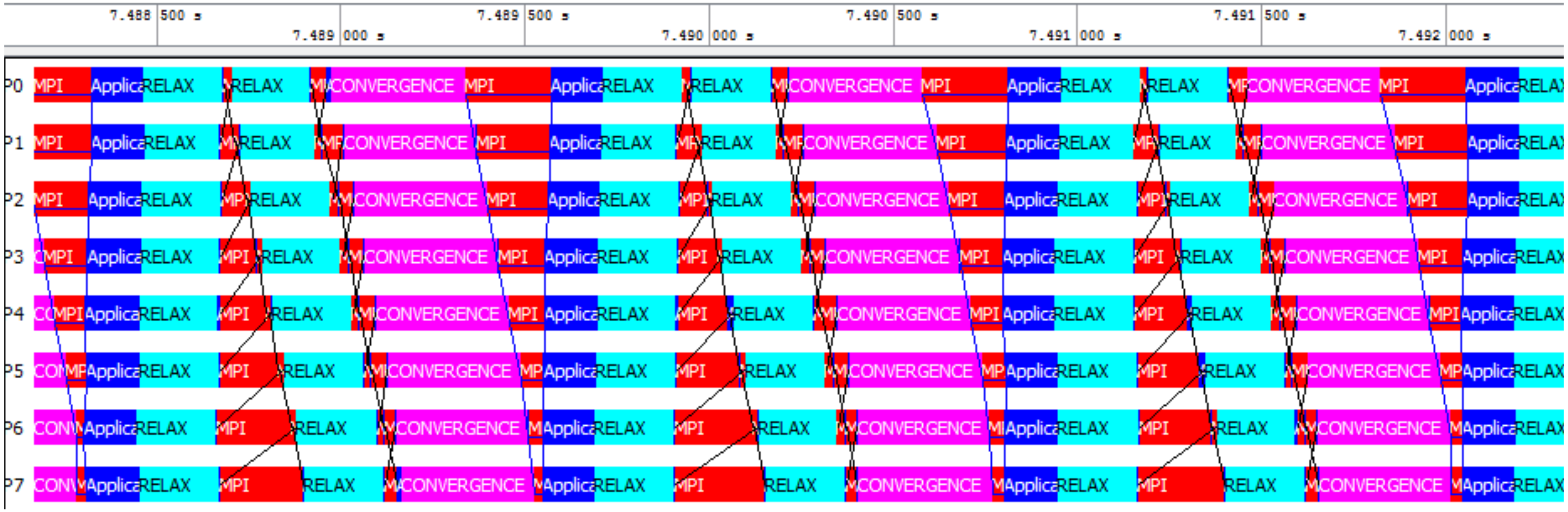
All Charts in a View are linked to a single:

- time-span
- set of threads
- set of functions

All Charts follow changes to View (e.g. zooming)



Event Timeline



Get detailed impression of program structure

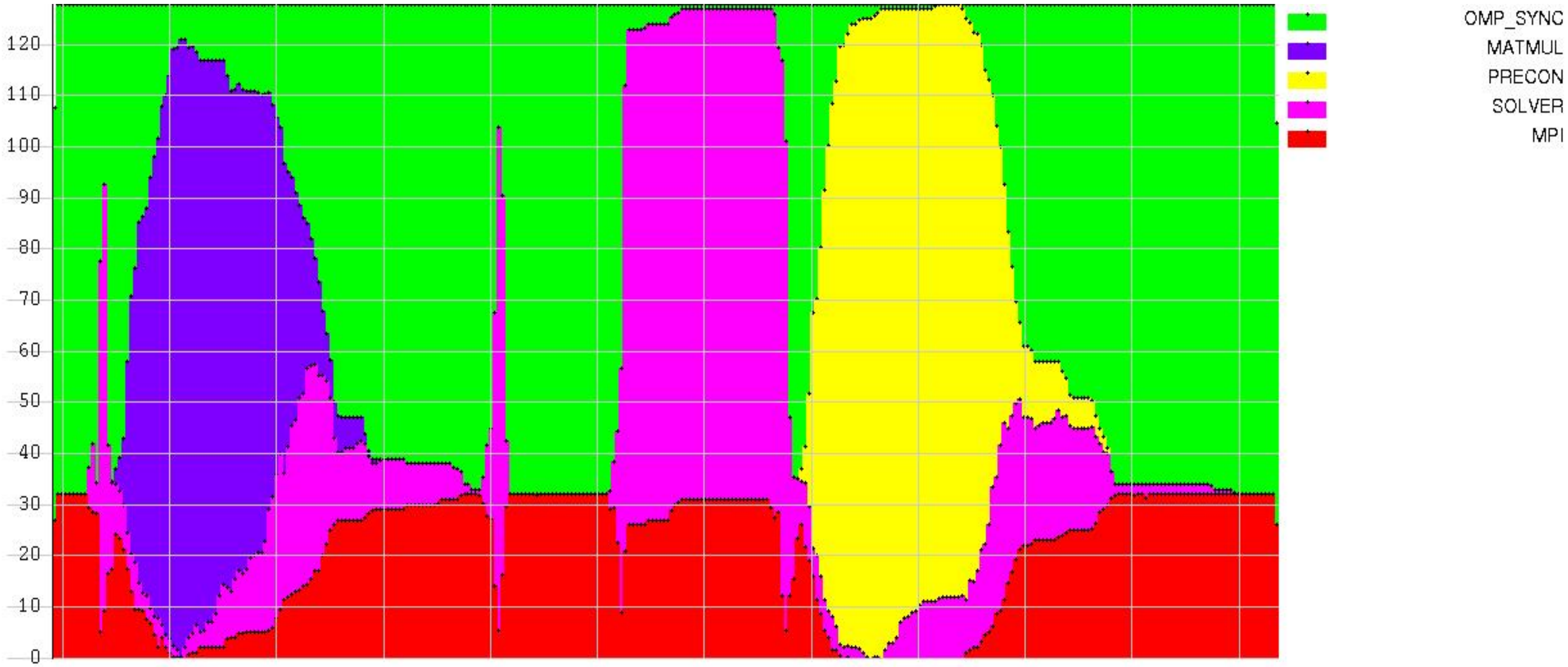
Display functions, messages, and collective operations for each rank/thread along time-axis

Retrieval of detailed event information

Quantitative Timeline

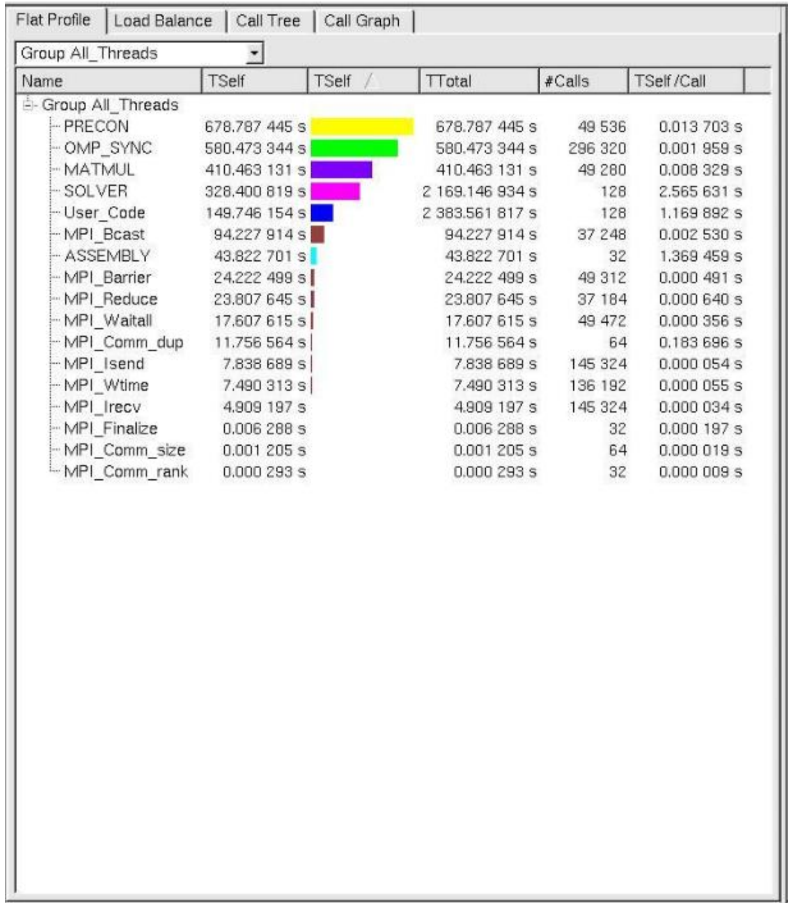
Get impression on parallelism and load balance

Show for every function how many threads/ranks are currently executing it



Flat Function Profile

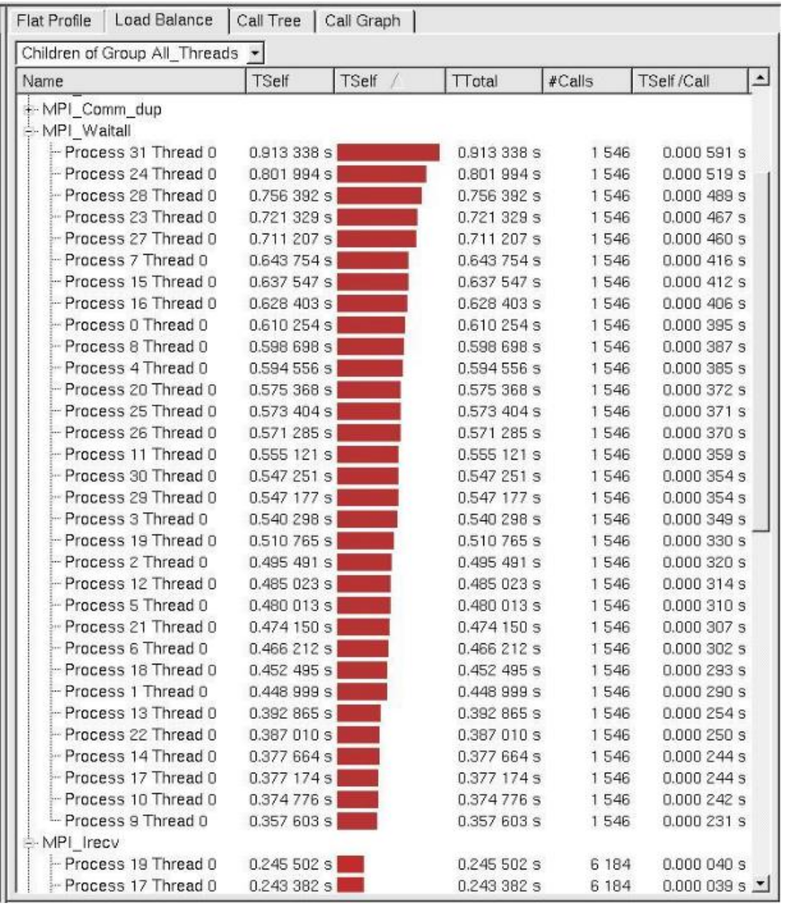
Statistics about functions



Flat Profile | Load Balance | Call Tree | Call Graph

Group All_Threads

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Threads					
PRECON	678.787 445 s		678.787 445 s	49 536	0.013 703 s
OMP_SYNC	580.473 344 s		580.473 344 s	296 320	0.001 959 s
MATMUL	410.463 131 s		410.463 131 s	49 280	0.008 329 s
SOLVER	328.400 819 s		2 169.146 934 s	128	2.565 631 s
User_Code	149.746 154 s		2 383.561 817 s	128	1.169 892 s
MPI_Bcast	94.227 914 s		94.227 914 s	37 248	0.002 530 s
ASSEMBLY	43.822 701 s		43.822 701 s	32	1.369 459 s
MPI_Barrier	24.222 499 s		24.222 499 s	49 312	0.000 491 s
MPI_Reduce	23.807 645 s		23.807 645 s	37 184	0.000 640 s
MPI_Waitall	17.607 615 s		17.607 615 s	49 472	0.000 356 s
MPI_Comm_dup	11.756 564 s		11.756 564 s	64	0.183 696 s
MPI_Isend	7.838 689 s		7.838 689 s	145 324	0.000 054 s
MPI_Wtime	7.490 313 s		7.490 313 s	136 192	0.000 055 s
MPI_Irecv	4.909 197 s		4.909 197 s	145 324	0.000 034 s
MPI_Finalize	0.006 288 s		0.006 288 s	32	0.000 197 s
MPI_Comm_size	0.001 205 s		0.001 205 s	64	0.000 019 s
MPI_Comm_rank	0.000 293 s		0.000 293 s	32	0.000 009 s



Flat Profile | Load Balance | Call Tree | Call Graph

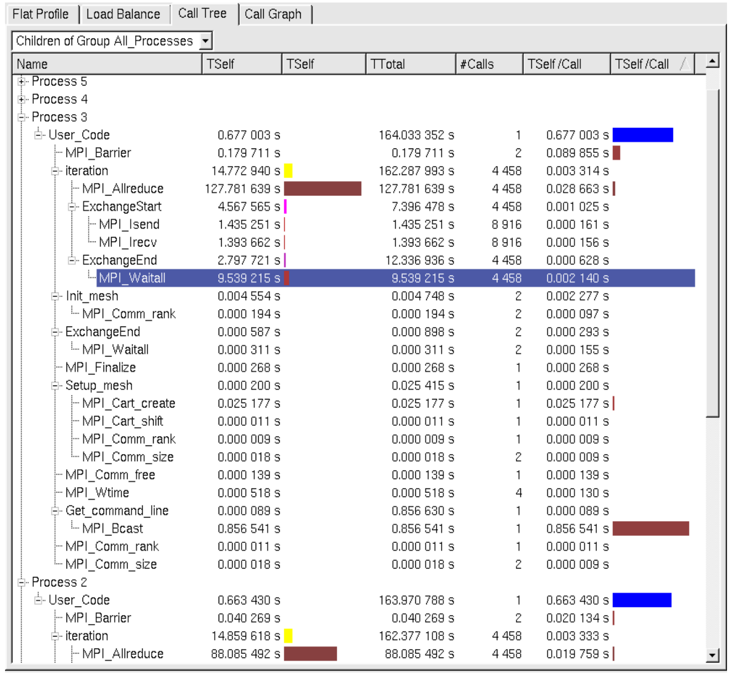
Children of Group All_Threads

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
MPI_Comm_dup					
MPI_Waitall					
Process 31 Thread 0	0.913 338 s		0.913 338 s	1 546	0.000 591 s
Process 24 Thread 0	0.801 994 s		0.801 994 s	1 546	0.000 519 s
Process 28 Thread 0	0.756 392 s		0.756 392 s	1 546	0.000 489 s
Process 23 Thread 0	0.721 329 s		0.721 329 s	1 546	0.000 467 s
Process 27 Thread 0	0.711 207 s		0.711 207 s	1 546	0.000 460 s
Process 7 Thread 0	0.643 754 s		0.643 754 s	1 546	0.000 416 s
Process 15 Thread 0	0.637 547 s		0.637 547 s	1 546	0.000 412 s
Process 16 Thread 0	0.628 403 s		0.628 403 s	1 546	0.000 406 s
Process 0 Thread 0	0.610 254 s		0.610 254 s	1 546	0.000 395 s
Process 8 Thread 0	0.598 698 s		0.598 698 s	1 546	0.000 387 s
Process 4 Thread 0	0.594 556 s		0.594 556 s	1 546	0.000 385 s
Process 20 Thread 0	0.575 368 s		0.575 368 s	1 546	0.000 372 s
Process 25 Thread 0	0.573 404 s		0.573 404 s	1 546	0.000 371 s
Process 26 Thread 0	0.571 285 s		0.571 285 s	1 546	0.000 370 s
Process 11 Thread 0	0.555 121 s		0.555 121 s	1 546	0.000 359 s
Process 30 Thread 0	0.547 251 s		0.547 251 s	1 546	0.000 354 s
Process 29 Thread 0	0.547 177 s		0.547 177 s	1 546	0.000 354 s
Process 3 Thread 0	0.540 298 s		0.540 298 s	1 546	0.000 349 s
Process 19 Thread 0	0.510 765 s		0.510 765 s	1 546	0.000 330 s
Process 2 Thread 0	0.495 491 s		0.495 491 s	1 546	0.000 320 s
Process 12 Thread 0	0.485 023 s		0.485 023 s	1 546	0.000 314 s
Process 5 Thread 0	0.480 013 s		0.480 013 s	1 546	0.000 310 s
Process 21 Thread 0	0.474 150 s		0.474 150 s	1 546	0.000 307 s
Process 6 Thread 0	0.466 212 s		0.466 212 s	1 546	0.000 302 s
Process 18 Thread 0	0.452 495 s		0.452 495 s	1 546	0.000 293 s
Process 1 Thread 0	0.448 999 s		0.448 999 s	1 546	0.000 290 s
Process 13 Thread 0	0.392 865 s		0.392 865 s	1 546	0.000 254 s
Process 22 Thread 0	0.387 010 s		0.387 010 s	1 546	0.000 250 s
Process 14 Thread 0	0.377 664 s		0.377 664 s	1 546	0.000 244 s
Process 17 Thread 0	0.377 174 s		0.377 174 s	1 546	0.000 244 s
Process 10 Thread 0	0.374 776 s		0.374 776 s	1 546	0.000 242 s
Process 9 Thread 0	0.357 603 s		0.357 603 s	1 546	0.000 231 s
MPI_Irecv					
Process 19 Thread 0	0.245 502 s		0.245 502 s	6 184	0.000 040 s
Process 17 Thread 0	0.243 382 s		0.243 382 s	6 184	0.000 039 s

Call Tree and Call Graph

Function statistics including calling hierarchy

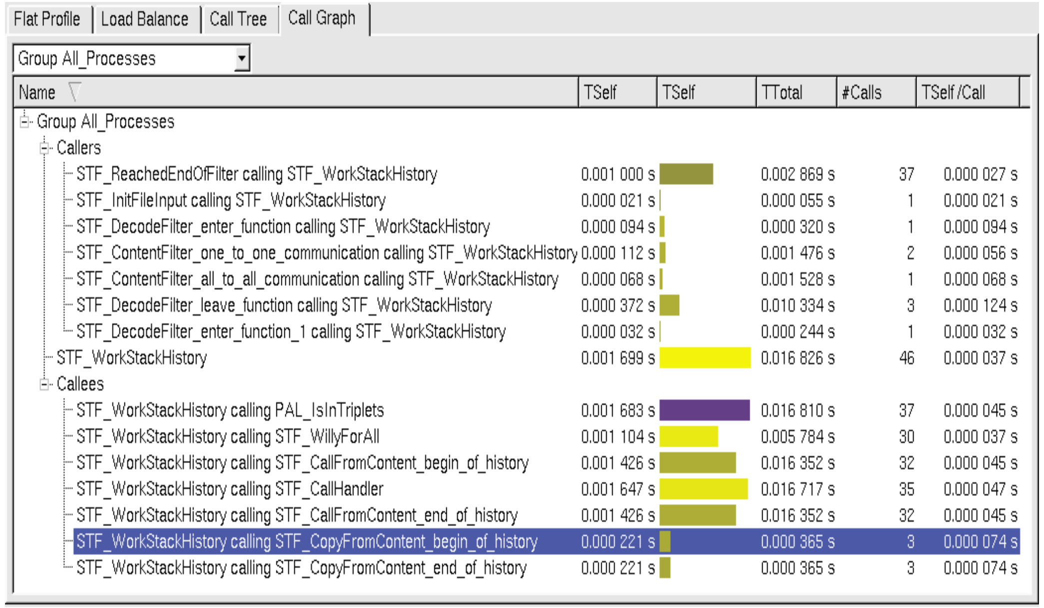
- Call Tree shows call stack
- Call Graph shows calling dependencies



Flat Profile | Load Balance | Call Tree | Call Graph

Children of Group All_Processes

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call	TSelf /Call
Process 5						
Process 4						
Process 3						
User_Code	0.677 003 s		164.033 352 s	1	0.677 003 s	
MPI_Barrier	0.179 711 s		0.179 711 s	2	0.089 855 s	
iteration	14.772 940 s		162.287 993 s	4 458	0.003 314 s	
MPI_Allreduce	127.781 639 s		127.781 639 s	4 458	0.028 663 s	
ExchangeStart	4.567 565 s		7.396 478 s	4 458	0.001 025 s	
MPI_Isend	1.435 251 s		1.435 251 s	8 916	0.000 161 s	
MPI_Irecv	1.393 662 s		1.393 662 s	8 916	0.000 156 s	
ExchangeEnd	2.797 721 s		12.336 936 s	4 458	0.000 628 s	
MPI_Waitall	9.539 215 s		9.539 215 s	4 458	0.002 140 s	
Init_mesh	0.004 554 s		0.004 748 s	2	0.002 277 s	
MPI_Comm_rank	0.000 194 s		0.000 194 s	2	0.000 097 s	
ExchangeEnd	0.000 587 s		0.000 898 s	2	0.000 293 s	
MPI_Waitall	0.000 311 s		0.000 311 s	2	0.000 155 s	
MPI_Finalize	0.000 268 s		0.000 268 s	1	0.000 268 s	
Setup_mesh	0.000 200 s		0.025 415 s	1	0.000 200 s	
MPI_Cart_create	0.025 177 s		0.025 177 s	1	0.025 177 s	
MPI_Cart_shift	0.000 011 s		0.000 011 s	1	0.000 011 s	
MPI_Comm_rank	0.000 009 s		0.000 009 s	1	0.000 009 s	
MPI_Comm_size	0.000 018 s		0.000 018 s	2	0.000 009 s	
MPI_Comm_free	0.000 139 s		0.000 139 s	1	0.000 139 s	
MPI_Wtime	0.000 518 s		0.000 518 s	4	0.000 130 s	
Get_command_line	0.000 089 s		0.856 630 s	1	0.000 089 s	
MPI_Bcast	0.856 541 s		0.856 541 s	1	0.856 541 s	
MPI_Comm_rank	0.000 011 s		0.000 011 s	1	0.000 011 s	
MPI_Comm_size	0.000 018 s		0.000 018 s	2	0.000 009 s	
Process 2						
User_Code	0.663 430 s		163.970 788 s	1	0.663 430 s	
MPI_Barrier	0.040 269 s		0.040 269 s	2	0.020 134 s	
iteration	14.859 618 s		162.377 108 s	4 458	0.003 333 s	
MPI_Allreduce	88.085 492 s		88.085 492 s	4 458	0.019 759 s	



Flat Profile | Load Balance | Call Tree | Call Graph

Group All_Processes

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes					
Callers					
STF_ReachedEndOfFilter calling STF_WorkStackHistory	0.001 000 s		0.002 869 s	37	0.000 027 s
STF_InitFileInput calling STF_WorkStackHistory	0.000 021 s		0.000 055 s	1	0.000 021 s
STF_DecodeFilter_enter_function calling STF_WorkStackHistory	0.000 094 s		0.000 320 s	1	0.000 094 s
STF_ContentFilter_one_to_one_communication calling STF_WorkStackHistory	0.000 112 s		0.001 476 s	2	0.000 056 s
STF_ContentFilter_all_to_all_communication calling STF_WorkStackHistory	0.000 068 s		0.001 528 s	1	0.000 068 s
STF_DecodeFilter_leave_function calling STF_WorkStackHistory	0.000 372 s		0.010 334 s	3	0.000 124 s
STF_DecodeFilter_enter_function_1 calling STF_WorkStackHistory	0.000 032 s		0.000 244 s	1	0.000 032 s
STF_WorkStackHistory	0.001 699 s		0.016 826 s	46	0.000 037 s
Callees					
STF_WorkStackHistory calling PAL_IsInTriplets	0.001 683 s		0.016 810 s	37	0.000 045 s
STF_WorkStackHistory calling STF_WillyForAll	0.001 104 s		0.005 784 s	30	0.000 037 s
STF_WorkStackHistory calling STF_CallFromContent_begin_of_history	0.001 426 s		0.016 352 s	32	0.000 045 s
STF_WorkStackHistory calling STF_CallHandler	0.001 647 s		0.016 717 s	35	0.000 047 s
STF_WorkStackHistory calling STF_CallFromContent_end_of_history	0.001 426 s		0.016 352 s	32	0.000 045 s
STF_WorkStackHistory calling STF_CopyFromContent_begin_of_history	0.000 221 s		0.000 365 s	3	0.000 074 s
STF_WorkStackHistory calling STF_CopyFromContent_end_of_history	0.000 221 s		0.000 365 s	3	0.000 074 s

Communication Profiles

Statistics about point-to-point or collective communication

Matrix supports grouping by attributes in each dimension

- Sender, Receiver, Data volume per msg, Tag, Communicator, Type

Available attributes

- Count, Bytes transferred, Time, Transfer rate

Total Time [s] (Collective Operation by Process)

	P0	P1	P2	P3	P4	P5	P6	P7	Sum	Mean	StdDev
MPI_Barrier	0.063	0.052	0.040	0.180	0.258	0.066	0.079	0.215	0.952	0.119	0.080
MPI_Bcast	0.000	0.860	0.865	0.857	0.853	0.855	0.860	0.861	6.010	0.751	0.284
MPI_Allreduce	87.299	120.679	88.085	127.782	89.071	124.266	109.330	137.064	883.576	110.447	18.704
Sum	87.362	121.590	88.990	128.818	90.182	125.187	110.268	138.141	890.538		
Mean	29.121	40.530	29.663	42.939	30.061	41.729	36.756	46.047		37.106	
StdDev	41.139	56.675	41.312	59.993	41.727	58.363	51.318	64.359			52.973

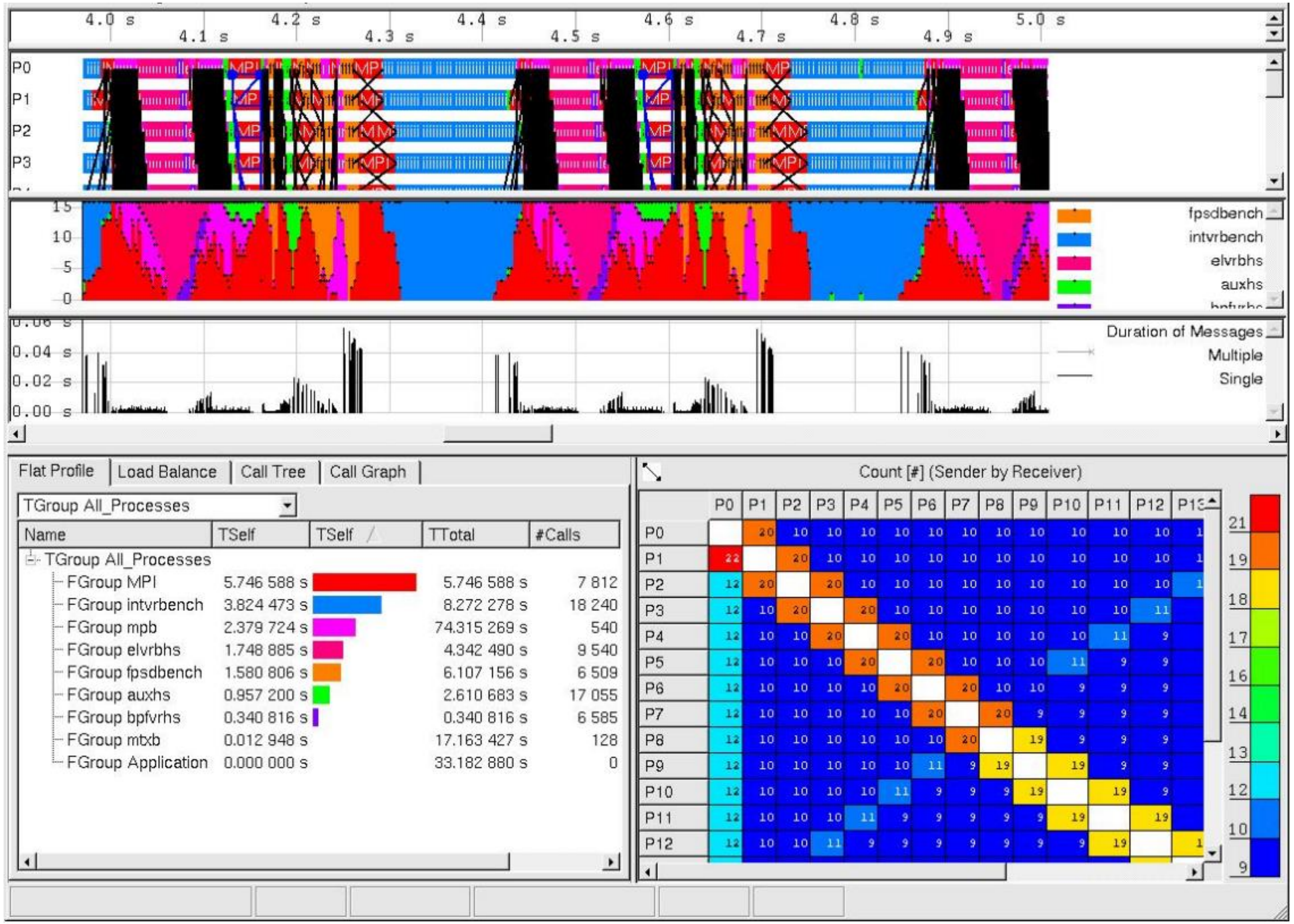
123 110 96 82 69 55 41 27 14 0

Total Time [s] (Sender by Receiver)

	P0	P1	P2	P3	P4	P5	P6	P7	Sum	Mean	StdDev
P0		74.641							74.641	74.641	0.000
P1	23.903		45.249						69.152	34.576	10.673
P2		51.590		47.961					99.551	49.776	1.814
P3			41.605		36.904				78.509	39.254	2.351
P4				51.558		54.114			105.672	52.836	1.278
P5					37.884		34.262		72.146	36.073	1.811
P6						37.619		35.861	73.480	36.740	0.879
P7							24.384		24.384	24.384	0.000
Sum	23.903	126.231	86.854	99.519	74.788	91.733	58.646	35.861	597.535		
Mean	23.903	63.116	43.427	49.759	37.394	45.866	29.323	35.861		42.681	
StdDev	0.000	11.526	1.822	1.798	0.490	8.248	4.939	0.000			12.629

70 64 59 54 49 44 39 34 29 24

Zooming



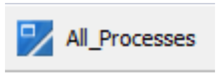
4.0 s
4.1 s
4.2 s
4.3 s
4.4 s
4.5 s
4.6 s
4.7 s
4.8 s
4.9 s
5.0 s
P0
P1
P2
P3
fpsdbench
intvrbench
elvrbhs
auxhs
0.04 s
0.02 s
0.00 s
Duration of Messages
Multiple
Single
Flat Profile
Load Balance
Call Tree
Call Graph
TGroup All_Processes
Name
TSelf
TSelf
TTotal
#Calls
TGroup All_Processes
FGroup MPI 5.746 588 s 5.746 588 s 7 812
FGroup intvrbench 3.824 473 s 8.272 278 s 18 240
FGroup mpb 2.379 724 s 74.315 269 s 540
FGroup elvrbhs 1.748 885 s 4.342 490 s 9 540
FGroup fpsdbench 1.580 806 s 6.107 156 s 6 509
FGroup auxhs 0.957 200 s 2.610 683 s 17 055
FGroup bpfvrhs 0.340 816 s 0.340 816 s 6 585
FGroup mtxb 0.012 948 s 17.163 427 s 128
FGroup Application 0.000 000 s 33.182 880 s 0
Count [#] (Sender by Receiver)
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
21
19
18
17
16
14
13
12
10
9

Grouping and Aggregation

Allow analysis on different levels of detail by aggregating data upon group-definitions

Functions and threads can be grouped hierarchically

- Process Groups and Function Groups



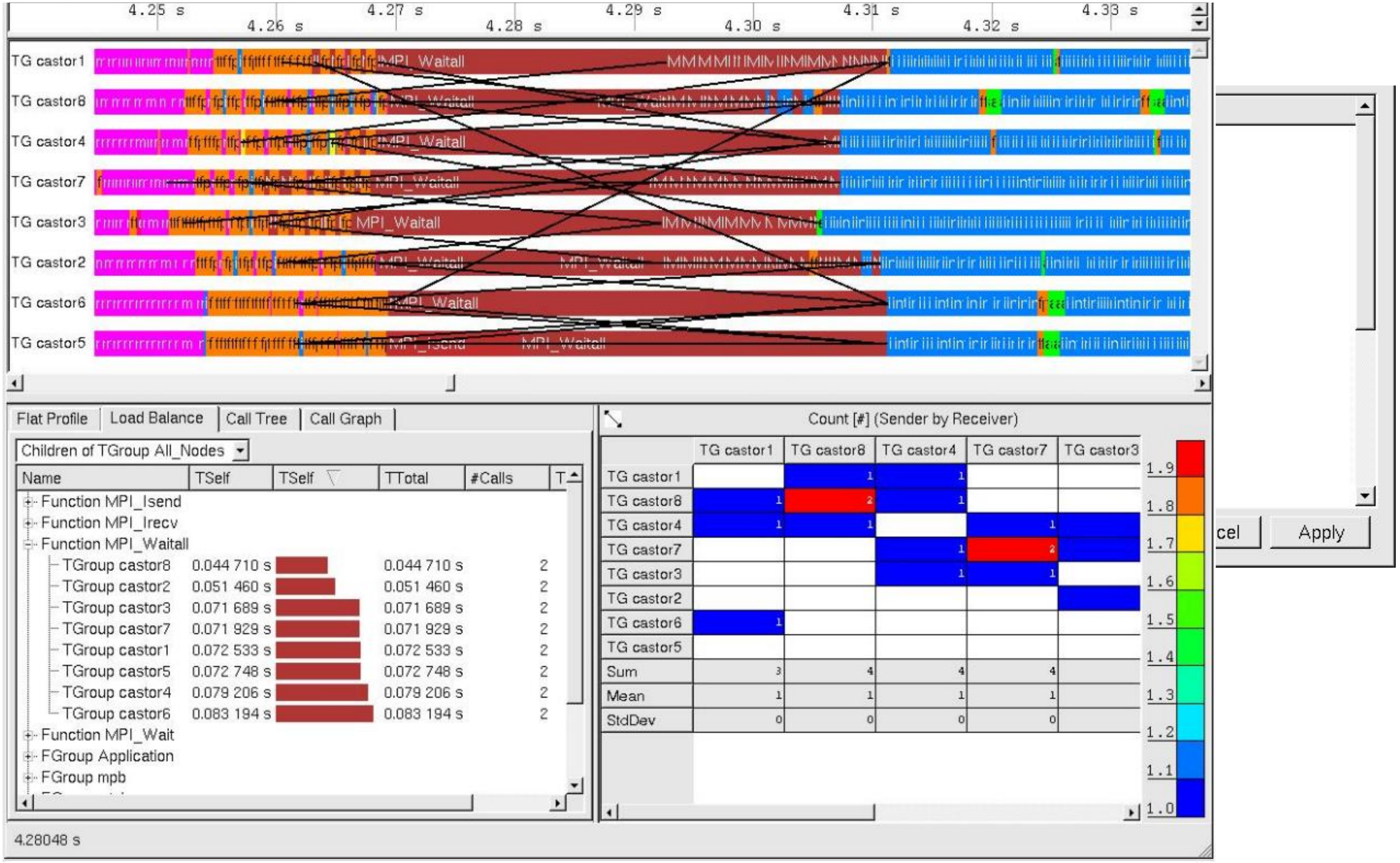
Arbitrary nesting is supported

- Functions/threads on the same level as groups
- User can define his/her own groups

Aggregation is part of View-definition

- All charts in a View adapt to requested grouping
- All charts support aggregation

Aggregation Example



Tagging and Filtering

Help concentrating on relevant parts

Avoid getting lost in huge amounts of trace data

Define a set of interesting data

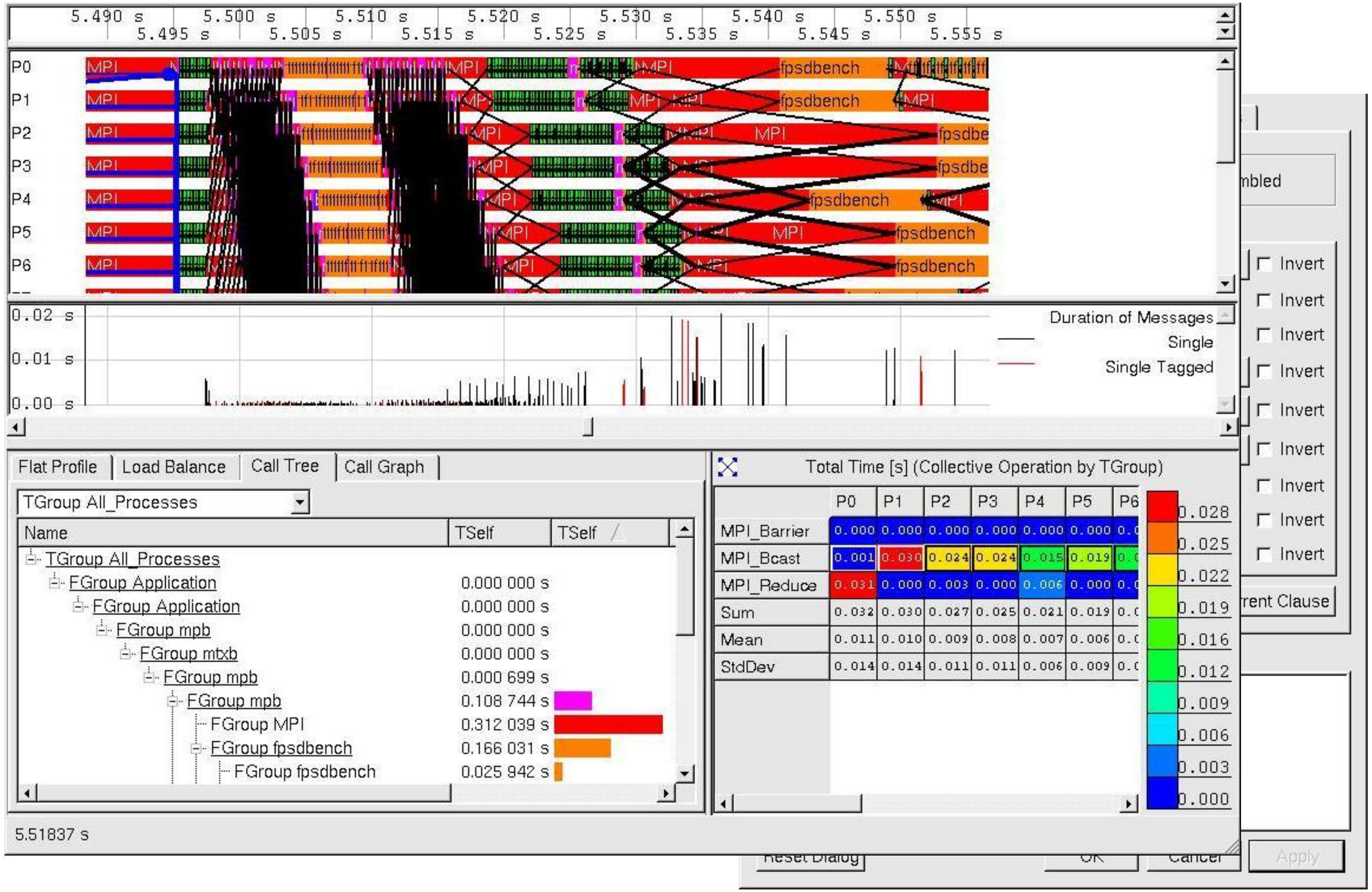
- E.g. all occurrences of function x
- E.g. all messages with tag y on communicator z

Combine several filters:
Intersection, Union, Complement

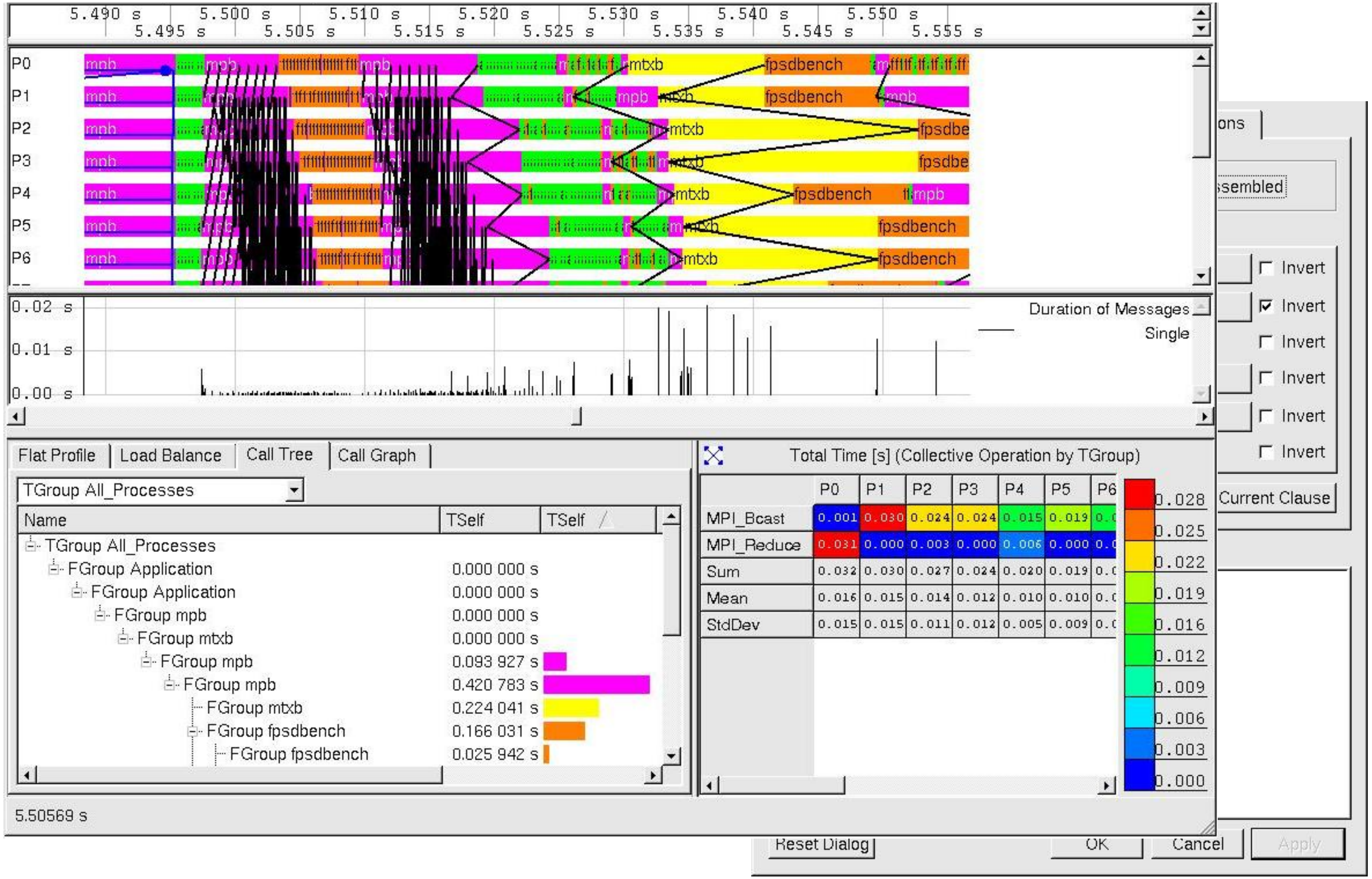
Apply it

- Tagging: Highlight messages
- Filtering: Suppress all non-matching events

Tagging Example



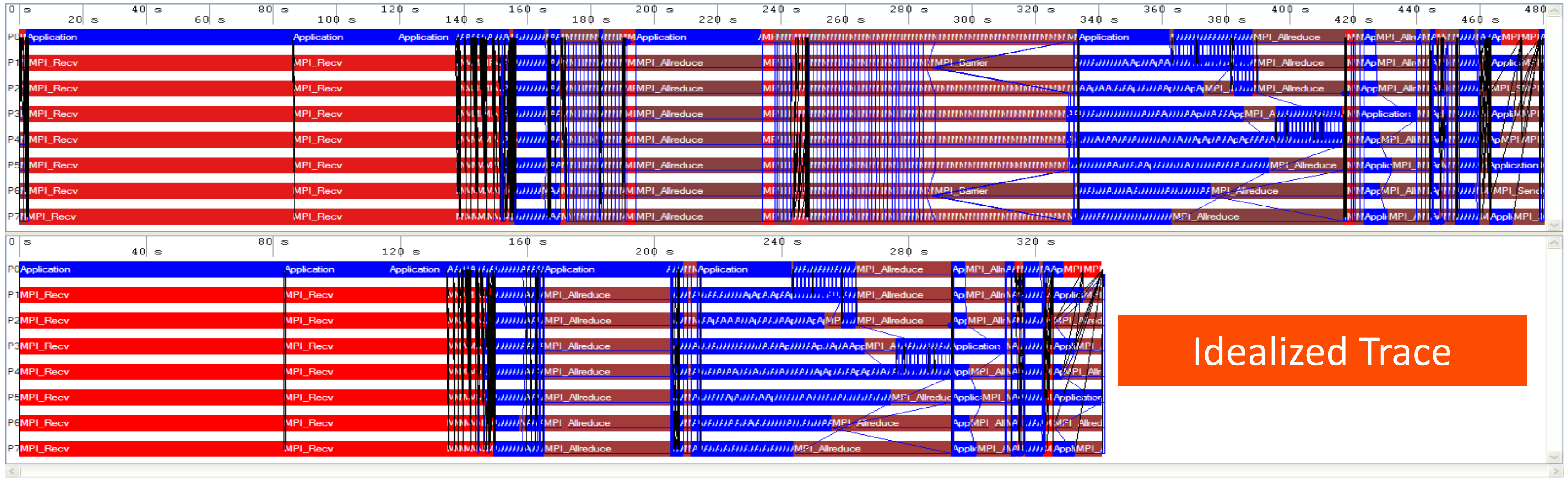
Filtering Example



Ideal Interconnect Simulator (Idealizer)

Helps to figure out application's imbalance simulating its behavior in the "ideal communication environment"

Actual trace



Idealized Trace

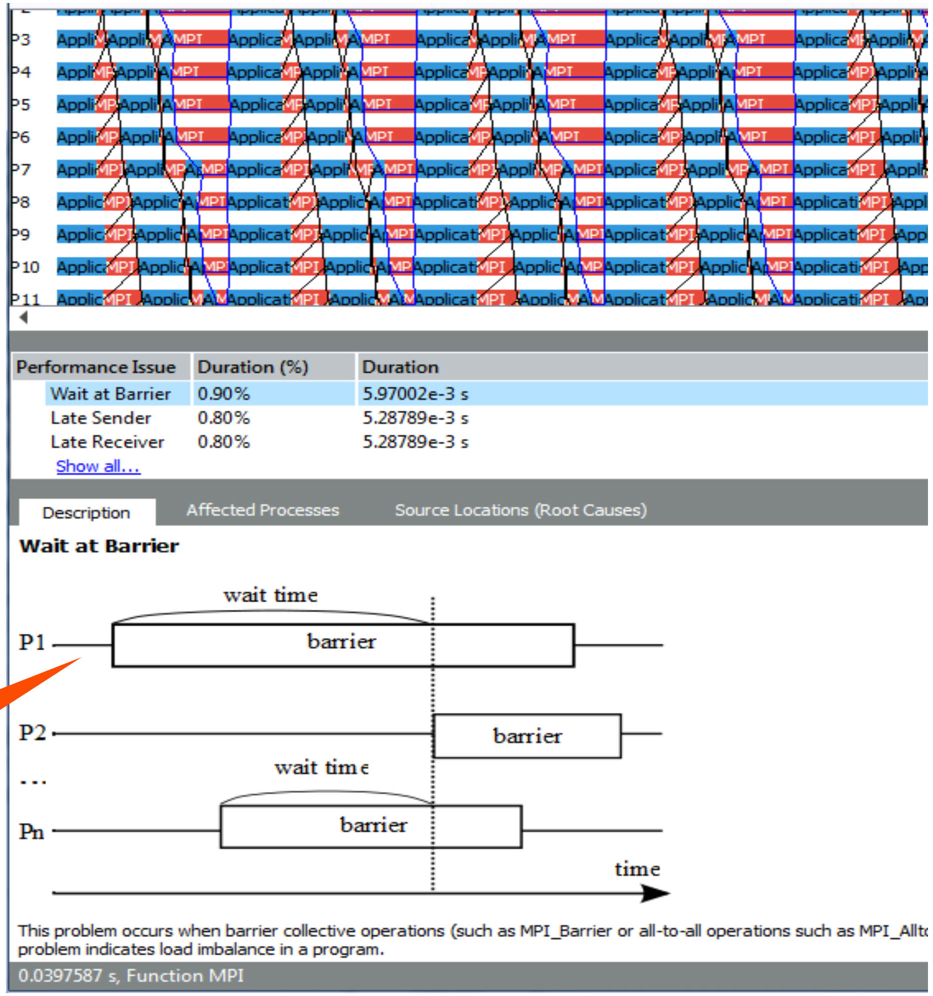
Easy way to identify application bottlenecks

MPI Performance Assistance

Automatic Performance Assistant

Detect common MPI performance issues

Automated tips on potential solutions

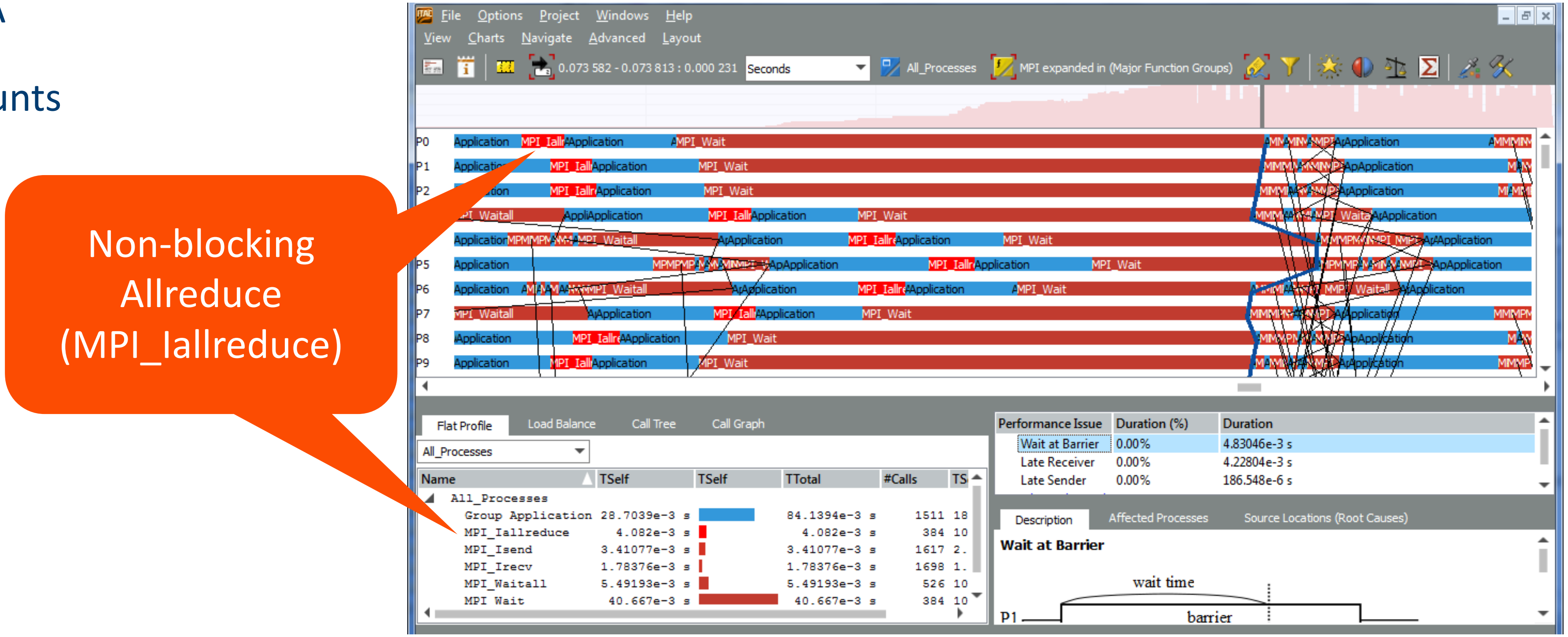


Automatically detect performance issues and their impact on runtime

MPI-3.0 Support

Support for major MPI-3.0 features

- Non-blocking collectives
- Fast RMA
- Large counts



What’s New in Intel® Trace Analyzer and Collector 2018

Support for OpenSHMEM* applications.

Support for the latest Intel® Xeon® Scalable and Intel® Xeon Phi™ processors.