USING INTEL® CLUSTER TOOLS TO OPTIMIZE MPI PROGRAMS

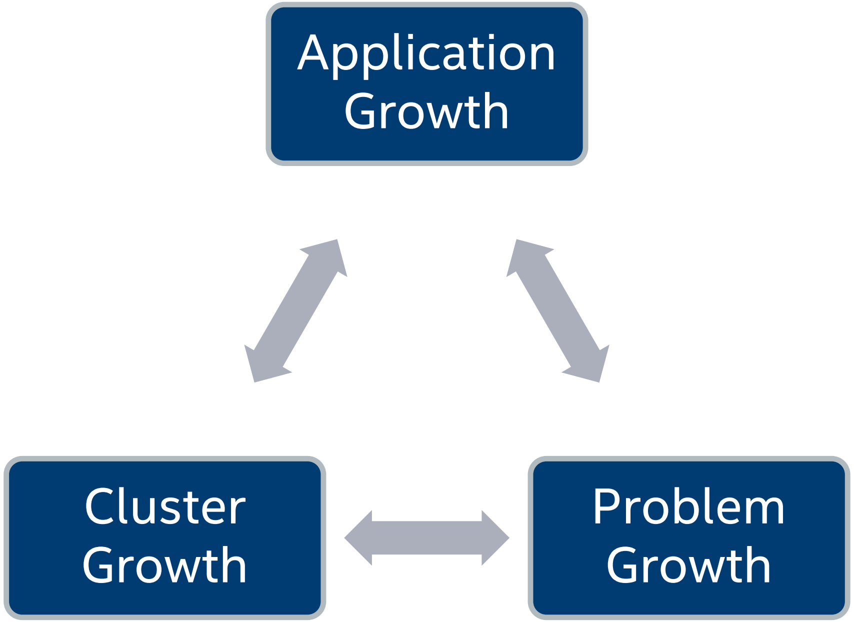
Agenda

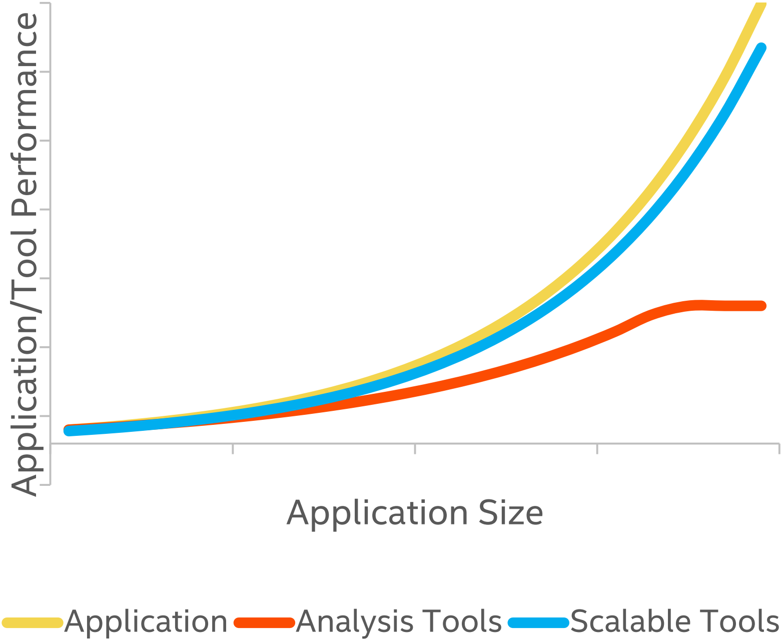
Scaling with MPI Performance Snapshot

Tuning MPI Performance with Intel® Trace Analyzer and Collector

MPI SCALING ANALYSIS CHALLENGES

MPI Scaling Analysis Challenges
To Exascale… and Beyond





MPI PERFORMANCE SNAPSHOT (MPS)

Why MPI Performance Snapshot (MPS)?

- Advantages
  - Get an initial profile of the application very quickly
  - Performance variation at scale can be detected and triaged quickly
  - Provides development recommendations to developers based on analysis
    - Intel® Trace Analyzer and Collector or Intel® VTune™ Amplifier XE for deeper analysis
  - Easy to use out of the box functionality
- Benefits
  - Difficult performance issues are easier to spot
  - Application performance guidance is obtained easily
  - Experienced & non-experienced developers can adopt quickly

2 MPI Performance Snapshots

- Bundled version:
  - is a part Parallel Studio Cluster Edition
  - relies on Intel VTune Amplifier
  - Intel MPI supports MPS (Hydra knows some environemt variables)
- Standalone version:
  - Can be downloaded with no charge.
  - Contains Application Perfomance Snapshot (APS) to collect Hardware counters
  - Has got a launcher script

What's new

Absolutely new collector which can collect information from all MPI functions.

Collector can work with any MPICH-based MPI implementation (OpenMPI is not supported so far)

Only one library to collect statistics and all other metrics (except Gflops, CPI, Memory bound)

Collector uses only MPI standard calls to support compatibility

New mechanism of MPI imbalance collection based on MPI_T_ mechanism – iMPI only

rdpmc timer for really low intrusion

Binary file format for MPI statistics

Each file writes to its own file

Different levels of statistics (MPS_STAT_LEVEL=1…5) 5 is default now.

New parser for binary statistics. (Still supports native iMPI statistics)

Absolutely new HTML report. All metrics on one page.

HTML report now shows top 5 MPI functions

What's not supported

MPS doesn't print statistics on finalization

PAPI library – we cannot rely on this library

MPI_Pcontrol() is not supported yet.

No MPI imbalance in other MPI implementations (might be added later)

OpenMP imbalance can be caught from Intel OpenMP library only.

MPI functions should not be called from OpenMP regions.

VTune Amplifier requires RedHat 6 and later and `perf` utility supported by kernel

MPS USAGE

How to run

“Bundle”

```
$ mpirun –mps –n N app_name
```

Standalone

```
$ source mpsvars.sh --vtune
$ mpirun –n N mpsrun.sh app_name
```

\`mpsrun.sh\` is a script which sets needed environment variables including LD_PRELOAD.

In VTune mode it runs \`amplxe-cl\` for each process

MPS Output

Summary

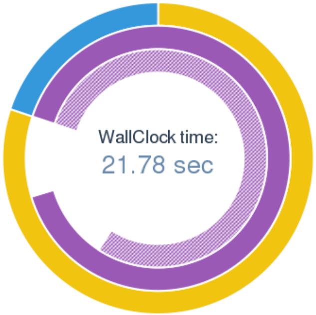
Files and folders:

- stats.txt
  - MPI statistics
- app_stat.txt
  - MPS collector statistics
- _mps/results.<node>/
  - VTune results

```
===================== GENERAL STATISTICS =====================
Total time:    448.391 sec (All ranks)
       MPI:      40.73%
   NON_MPI:      59.27%

WallClock :
      MIN :             89.594 sec (rank 1)
      MAX :             89.975 sec (rank 4)

=================== MEMORY USAGE STATISTICS ==================
All ranks:     226.969 MB
      MIN:      24.172 MB (rank 2)
      MAX:      96.465 MB (rank 0)

================== MPI IMBALANCE STATISTICS ==================
MPI Imbalance:            31.798 sec          7.092% (All ranks)
          MIN:             2.219 sec          2.467% (rank 4)
          MAX:             9.157 sec         10.219% (rank 0)
```

HTML Reporting

MPI Performance Snapshot Summary



MPI Time: 17.28 sec 80.10%
MPI Imbalance: 7.44 sec 34.47%
Computation Time: 4.29 sec 19.90%
OpenMP Time: 19.46 sec 90.19%
OpenMP Imbalance: 17.05 sec 79.03%
Serial Time: 0.00 sec 0.00%

WallClock time: 21.78 sec
Total application lifetime. The time is elapsed time for the slowest process. This metric includes the MPI Time and the Computation time below.

MPI Time: 17.28 sec 80.10%
Time spent inside the MPI library. High values are usually bad.
This value is HIGH. The application is Communication-bound. More details...

MPI Imbalance: 7.44 sec 34.47%
Mean unproductive wait time per process spent in the MPI library calls when a process is waiting for data. This time is part of the MPI time above. High values are usually bad.
This value is HIGH. The application workload is NOT well balanced between MPI ranks. More details...

Computation Time: 4.29 sec 19.90%
Mean time per process spent in the application code. This is the sum of the OpenMP Time and the Serial time. High values are usually good.
This value is LOW.

OpenMP Time: 19.46 sec 90.19%
Mean time per process spent in the OpenMP parallel regions. High values are usually good and indicate that the application is well-threaded.
This value is HIGH.

OpenMP Imbalance: 17.05 sec 79.03%
Mean unproductive wait time per process spent in OpenMP parallel regions (normally at synchronization barriers). High values are usually bad.
This value is HIGH. The application's OpenMP work sharing is NOT well load-balanced. More details...

Serial Time: 0.00 sec 0.00%
Mean application time per process spent outside OpenMP parallel regions. High values may be good or bad depending on the application algorithm.
This value is NEGLIGIBLE. This application is well parallelized via OpenMP directives.

Application: build/heart_demo
Number of ranks: 17
Used statistics: app_stat_20160310-035458.txt, stats_20160310-035458.txt
Creation date: 2016-03-10 03:55:21

MPS HTML Report Breakdown – MPI Time

MPI Time – Time spent in MPI calls

MPI Imbalance – MPI time spent waiting

Lower is better

If MPI Time or MPI Imbalance are high, use Intel® Trace Analyzer and Collector to investigate and optimize MPI usage

MPI Time: 17.28 sec 80.10%

Time spent inside the MPI library. High values are usually bad.
This value is HIGH. The application is Communication-bound. More details...

MPI Imbalance: 7.44 sec 34.47%

Mean unproductive wait time per process spent in the MPI library calls when a process is waiting for data. This time is part of the MPI time above. High values are usually bad.
This value is HIGH. The application workload is NOT well balanced between MPI ranks. More details...

Optimization Notice

MPS HTML Report Breakdown – OpenMP Time

OpenMP Time – Computation time spent in OpenMP parallel regions – higher is better

OpenMP Imbalance – OpenMP Time spent waiting – lower is better

If OpenMP Imbalance is high – recommend using Intel® VTune™ Amplifier XE

If OpenMP Time is low – Intel® Advisor to find opportunities to add more threading

OpenMP Time: 19.46 sec 90.19%

Mean time per process spent in the OpenMP parallel regions. High values are usually good and indicate that the application is well-threaded.

This value is HIGH.

OpenMP Imbalance: 17.05 sec 79.03%

Mean unproductive wait time per process spent in OpenMP parallel regions (normally at synchronization barriers). High values are usually bad.

This value is HIGH. The application's OpenMP work sharing is NOT well load-balanced. More details...

TUNING MPI APPLICATION PERFORMANCE WITH INTEL® TRACE ANALYZER AND COLLECTOR

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

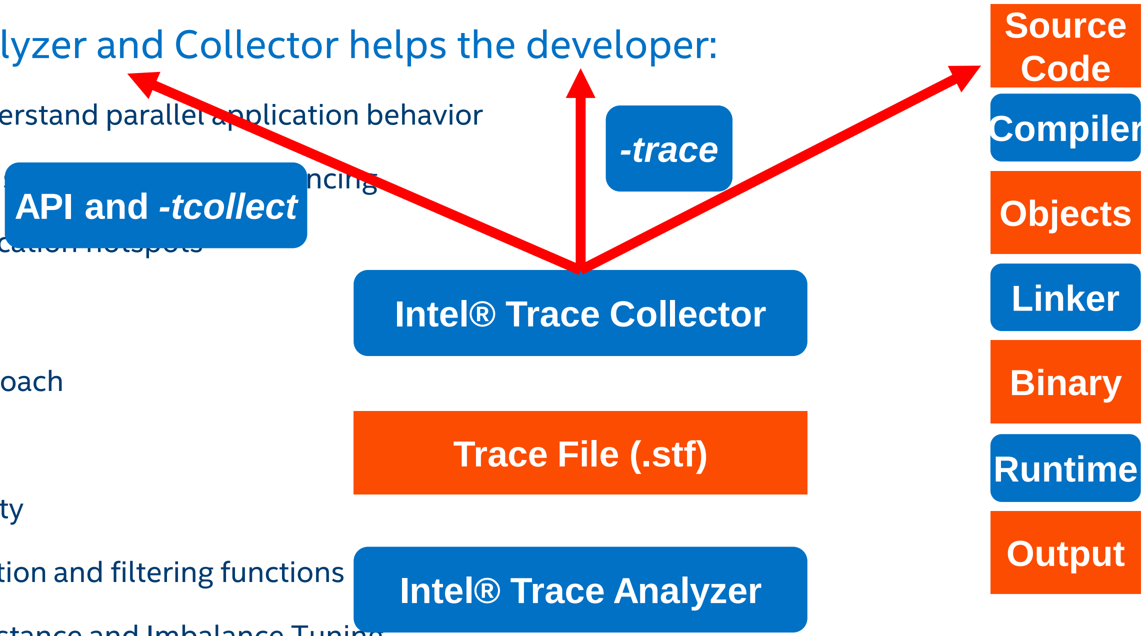
Intel® Trace Analyzer and Collector Overview

Intel® Trace Analyzer and Collector helps the developer:

- Visualize and understand parallel application behavior
- Evaluate profiling [illegible]ncing
- Identify communication hotspots

Features

- Event-based approach
- Low overhead
- Excellent scalability
- Powerful aggregation and filtering functions
- Performance Assistance and Imbalance Tuning
- NEW in 9.1: MPI Performance Snapshot



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.

Views and Charts

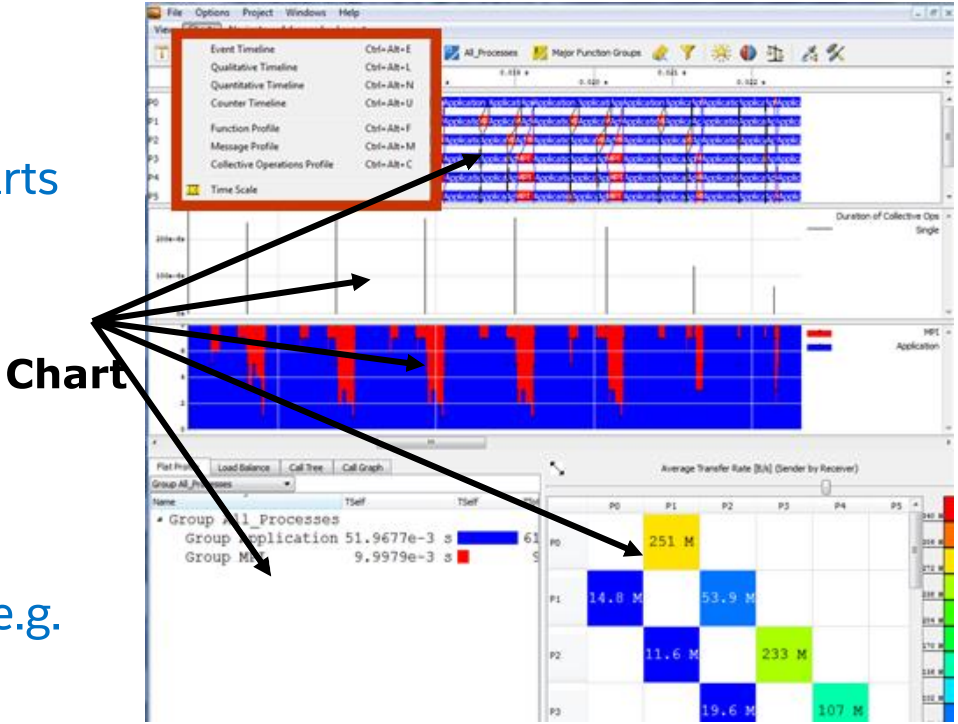
Helps navigating through the trace data and keep orientation

Every View can contain several Charts

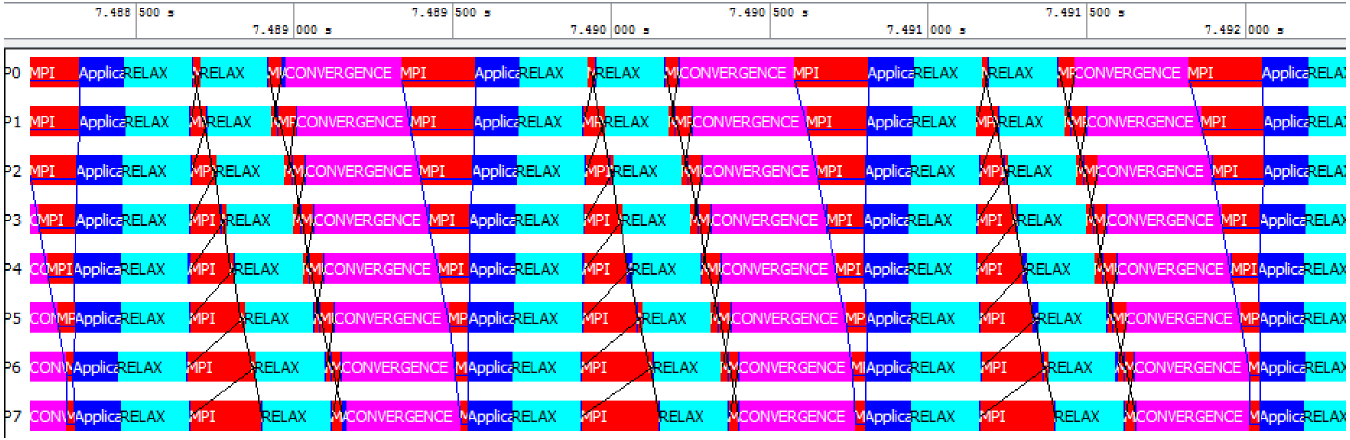
All Charts in a View are linked to a single:

- 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

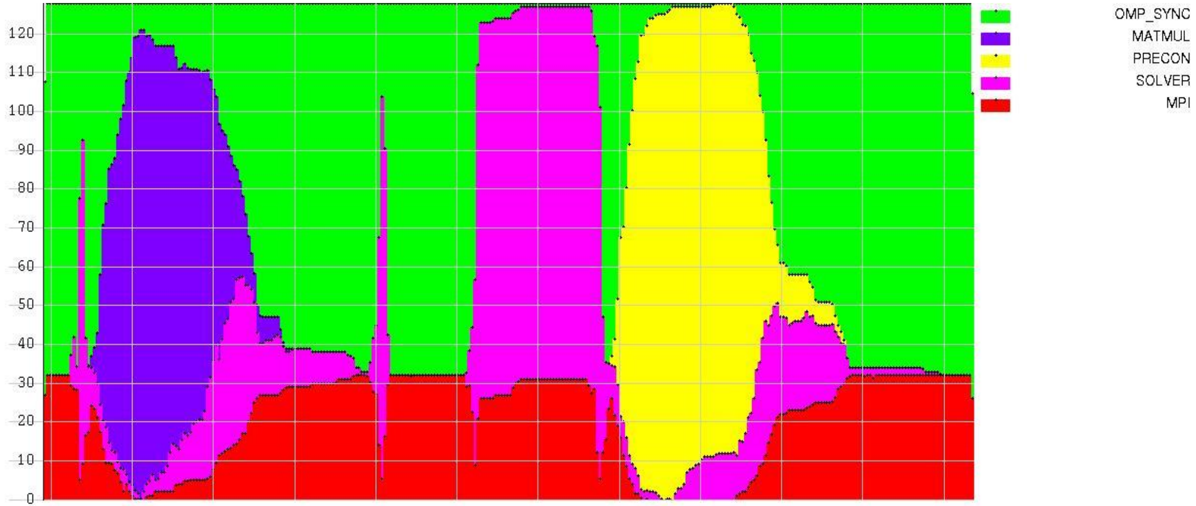
Display functions, messages, and collective operations for each rank/thread along time-axis

Retrieval of detailed event information

Quantitative Timeline

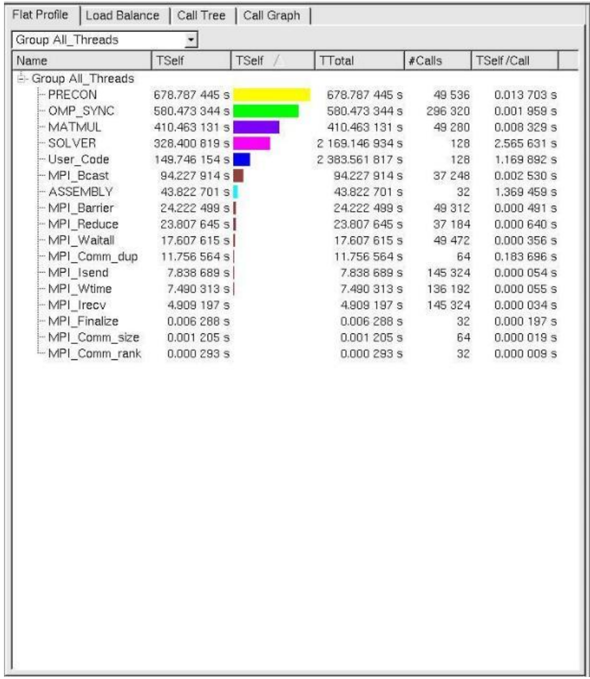
Get impression on parallelism and load balance

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



Flat Function Profile

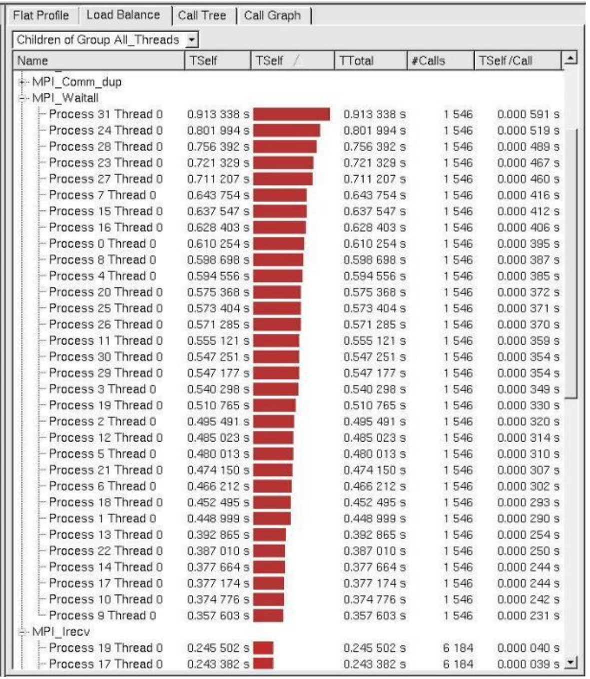
Statistics about functions



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

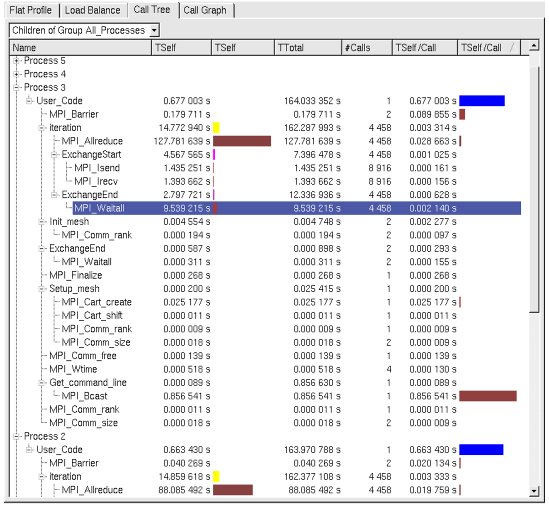
Children of Group All_Threads

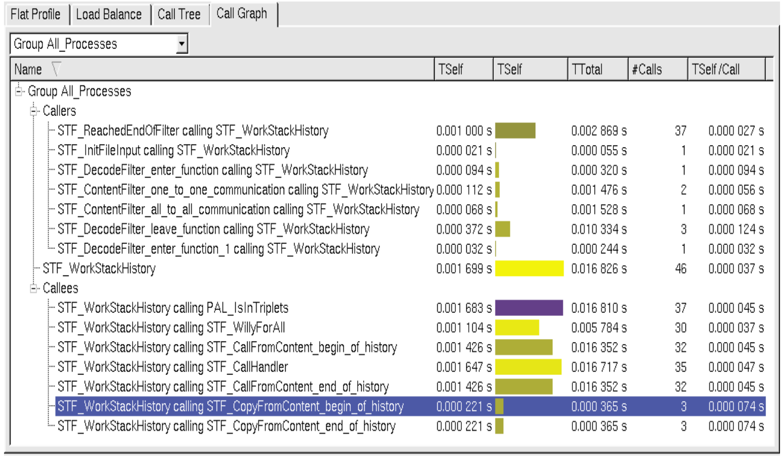
Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
MPI_Comm_dup					
MPI_Waitall					
Process 31 Thread 0	0.913 338 s		0.913 338 s	1 546	0.000 591 s
Process 24 Thread 0	0.801 994 s		0.801 994 s	1 546	0.000 519 s
Process 28 Thread 0	0.756 392 s		0.756 392 s	1 546	0.000 489 s
Process 23 Thread 0	0.721 329 s		0.721 329 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





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

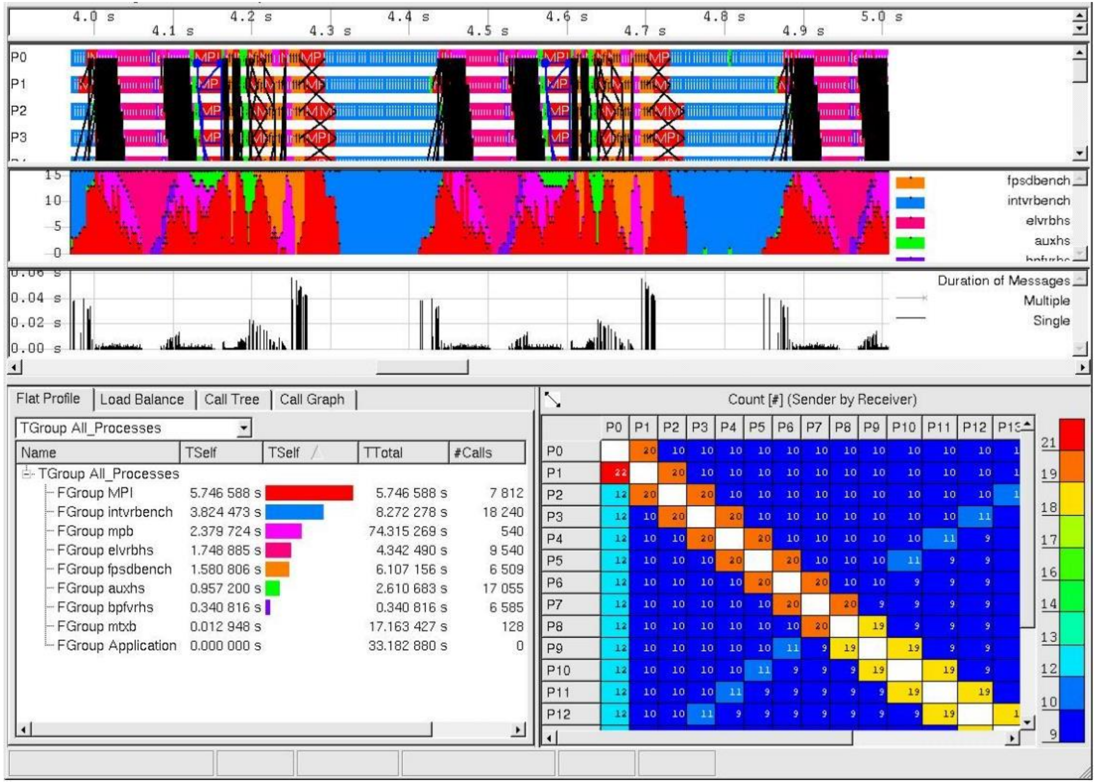
Total Time [s] (Collective Operation by Process)

	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

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

Zooming



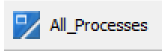
fpsdbench
intvrbench
elvrbhs
auxhs
Duration of Messages
Multiple
Single
Flat Profile
Load Balance
Call Tree
Call Graph
TGroup All_Processes
Name TSelf TSelf TTotal #Calls
TGroup All_Processes
FGroup MPI 5.746 588 s 5.746 588 s 7 812
FGroup intvrbench 3.824 473 s 8.272 278 s 18 240
FGroup mpb 2.379 724 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)

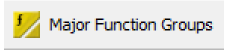
Grouping and Aggregation

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





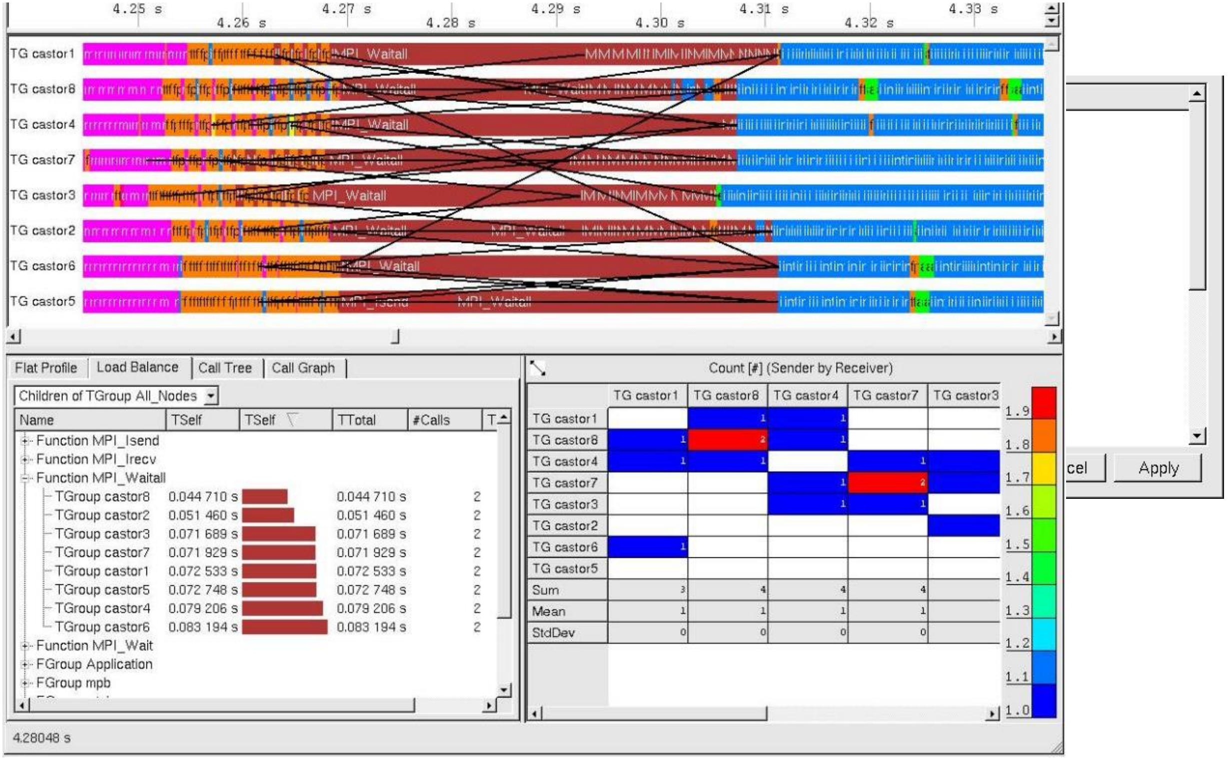
Arbitrary nesting is supported

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

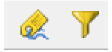
Aggregation is part of View-definition

- 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

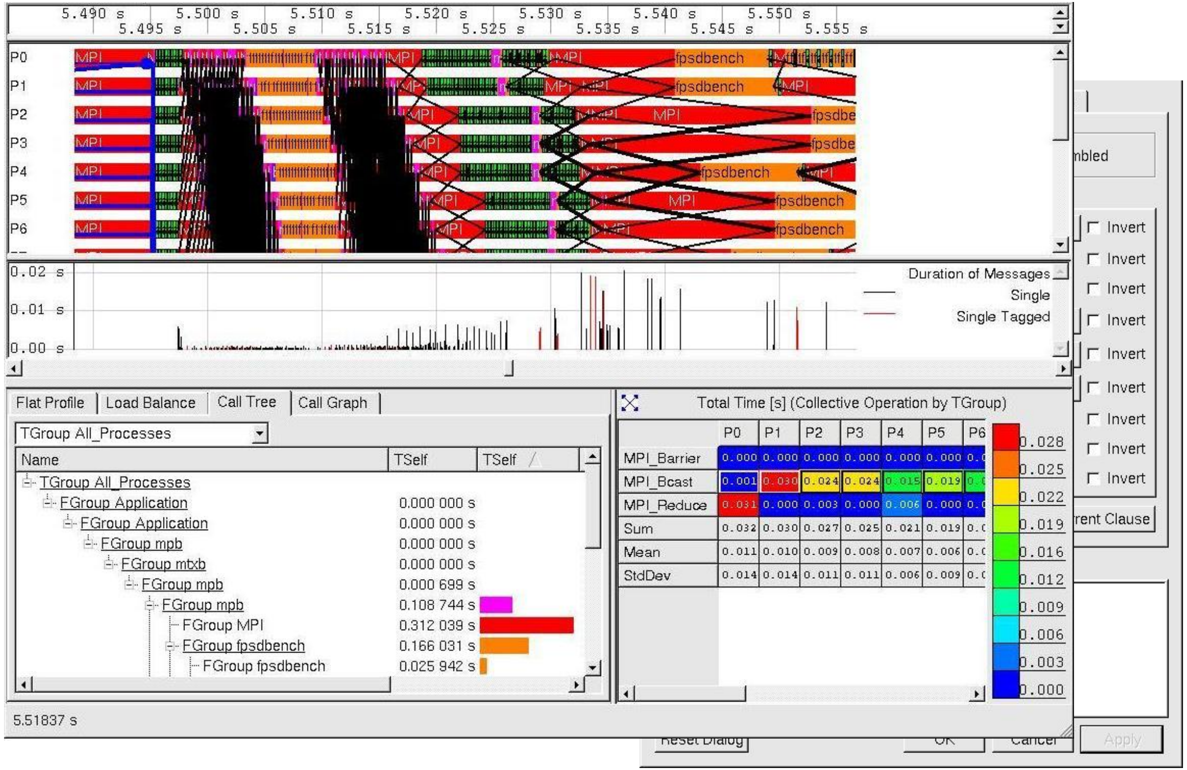
- E.g. all occurrences of function x
- E.g. all messages with tag y on communicator z

Combine several filters: Intersection, Union, Complement

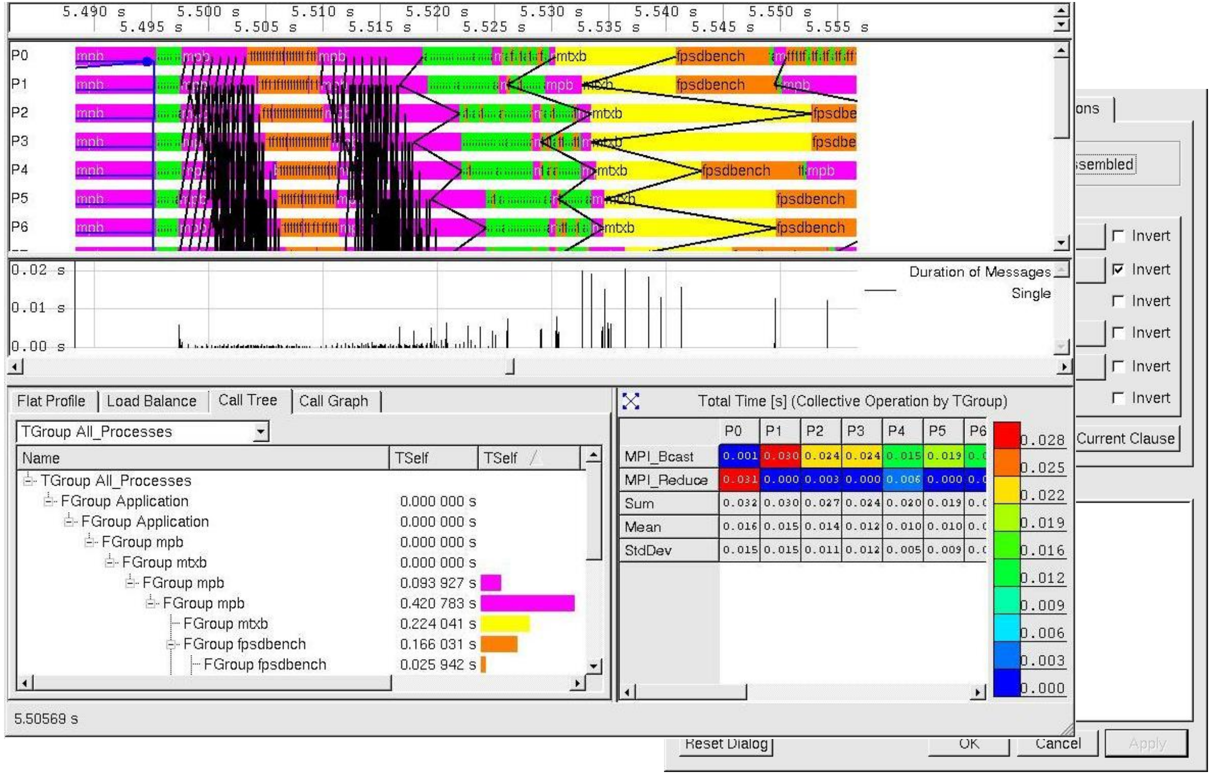
Apply it

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

Tagging Example



Filtering Example

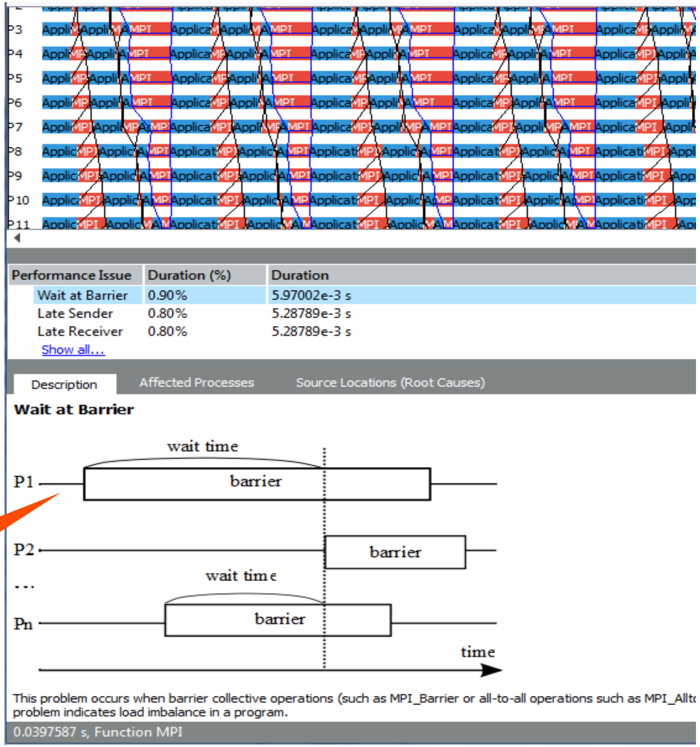


MPI Performance Assistance

Automatic Performance Assistant

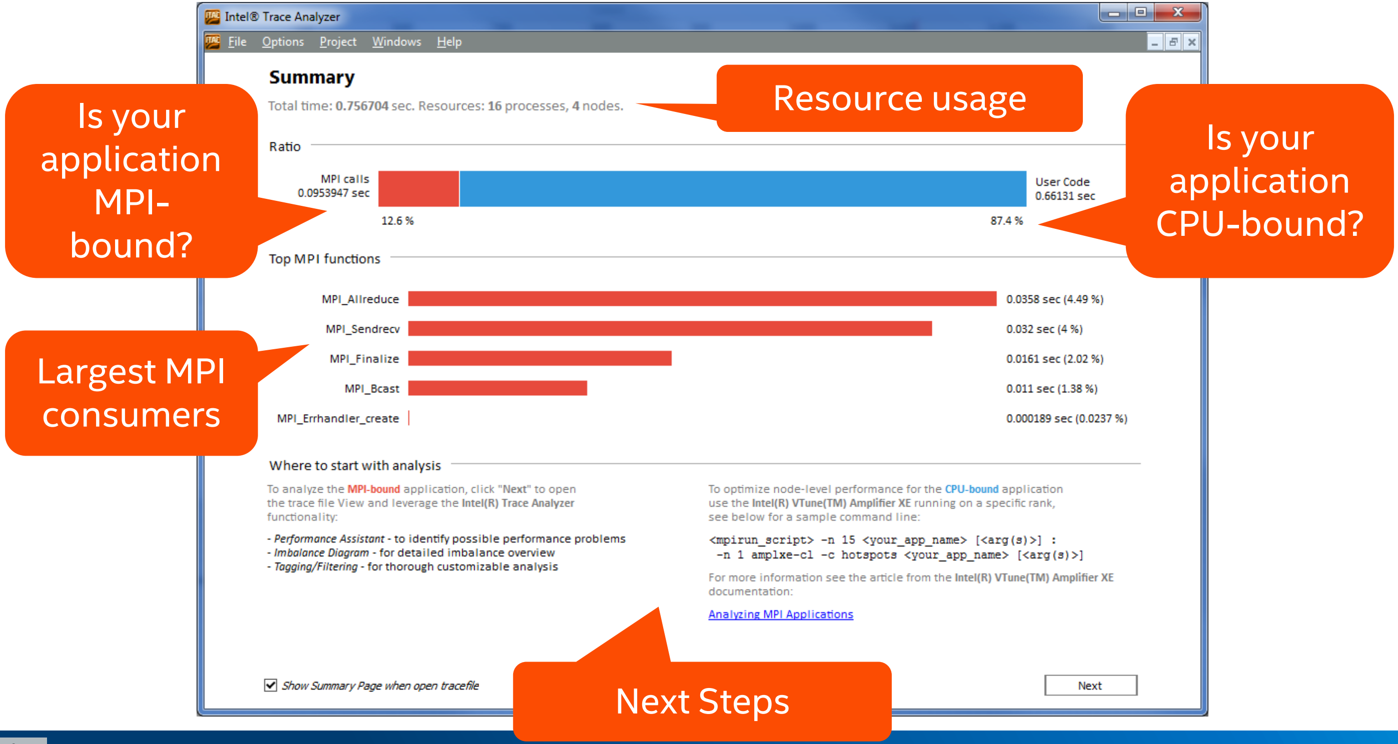
Detect common MPI performance issues

Automated tips on potential solutions



Automatically detect performance issues and their impact on runtime

Summary page shows computation vs. communication breakdown



MPI-3.0 Support

Support for major MPI-3.0 features

- Non-blocking collectives
- Fast RMA
- Large counts

Non-blocking Allreduce (MPI_Iallreduce)

Legal Disclaimer & Optimization Notice

experience
what's inside™