

# USING INTEL® CLUSTER TOOLS TO Optimize MPI Programs



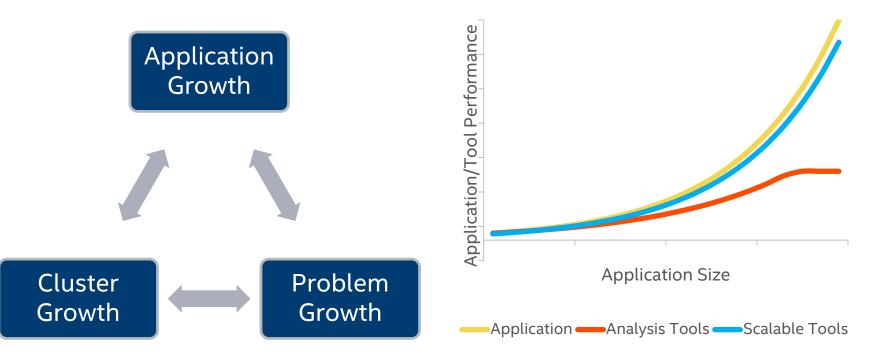
Scaling with MPI Performance Snapshot

Tuning MPI Performance with Intel® Trace Analyzer and Collector



# **MPI SCALING ANALYSIS CHALLENGES**

## MPI Scaling Analysis Challenges To Exascale... and Beyond



#### Optimization Notice

# 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<sup>®</sup> Trace Analyzer and Collector or Intel<sup>®</sup> VTune<sup>™</sup> 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

**Optimization Notice** 



### 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

#### **Optimization Notice**

Copyright © 2016, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others. Intel Confidential

### 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

#### **Optimization Notice**



# **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

**Optimization Notice** 

Copyright © 2016, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others. Intel Confidential



### **MPS Output**

Summary

Files and folders:

- stats.txt
  - MPI statistics
- app\_stat.txt
  - MPS collector statistics
- \_mps/results.<node>/
  - VTune results

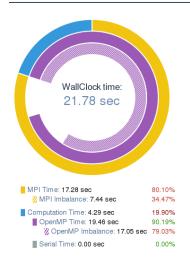
	======= GENERAL STATISTICS ====================================
Total time:	448.391 sec (All ranks)
MPI:	40.73%
NON_MPI:	59.27%
WallClock :	
MIN :	89.594 sec (rank 1)
MAX :	89.975 sec (rank 4)
	====== MEMORY USAGE STATISTICS ====================================
All ranks:	226.969 MB
MIN:	24.172 MB (rank 2)
MAX:	96.465 MB (rank 0)
	===== MPI IMBALANCE STATISTICS ====================================
MPI Imbalance	e: 31.798 sec 7.092% (All ranks)
MII	N: 2.219 sec 2.467% (rank 4)
MA	X: 9.157 sec 10.219% (rank 0)

#### Optimization Notice

### **HTML** Reporting

#### MPI Performance Snapshot Summary

WallClock time:



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 wa of the MPI time above. High values are usually bad.	iting for data. This time is part
This value is HIGH. The application workload is NOT well balanced between MPI ranks. More	e 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 usually good.	ne Serial time. High values are
This value is LOW.	
Open MP Time: 19.46 sec     Mean time per process spent in the OpenMP parallel regions. High values are usually good a     is well-threaded.     This value is HIGH.	90.19% nd indicate that the application
■ OpenMP Time: 19.46 sec Mean time per process spent in the OpenMP parallel regions. High values are usually good a is well-threaded. This value is HIGH.	
OpenMP Time: 19.46 sec Mean time per process spent in the OpenMP parallel regions. High values are usually good a is well-threaded.	nd indicate that the application 79.03%
OpenMP Time: 19.46 sec Mean time per process spent in the OpenMP parallel regions. High values are usually good a is well-threaded. This value is HIGH.     // OpenMP Imbalance: 17.05 sec Mean unproductive wait time per process spent in OpenMP parallel regions (normally at	nd indicate that the application 79.03% synchronization barriers). High
OpenMP Time: 19.46 sec Mean time per process spent in the OpenMP parallel regions. High values are usually good a is well-threaded. This value is HIGH. <u>OpenMP Imbalance: 17.05 sec     Mean unproductive wait time per process spent in OpenMP parallel regions (normally at values are usually bad. </u>	nd indicate that the application 79.03% synchronization barriers). High
CopenMP Time: 19.46 sec Mean time per process spent in the OpenMP parallel regions. High values are usually good at is well-threaded. This value is HIGH. CopenMP Imbalance: 17.05 sec Mean unproductive wait time per process spent in OpenMP parallel regions (normally at values are usually bad. This value is HIGH. The application's OpenMP work sharing is NOT well load-balanced.	nd indicate that the application 79.03% synchronization barriers). High More details 0.00%

Total application lifetime. The time is elapsed time for the slowest process. This metric includes the MPI Time and the Computation time

21.78 sec

#### 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

#### **Optimization Notice**



### MPS HTML Report Breakdown – MPI Time

MPI Time - Time spent in MPI calls

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

Lower is better



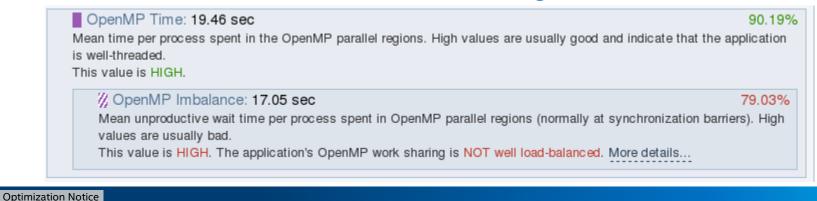
### MPS HTML Report Breakdown – OpenMP Time

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<sup>®</sup> Advisor to find opportunities to add more threading



# TUNING MPI APPLICATION PERFORMANCE WITH INTEL® TRACE ANALYZER AND COLLECTOR

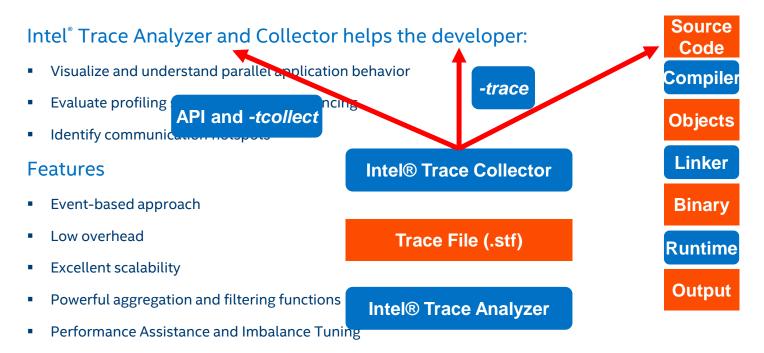
### Intel<sup>®</sup> Trace Analyzer and Collector

### **Value Proposition**

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



# Intel<sup>®</sup> Trace Analyzer and Collector Overview

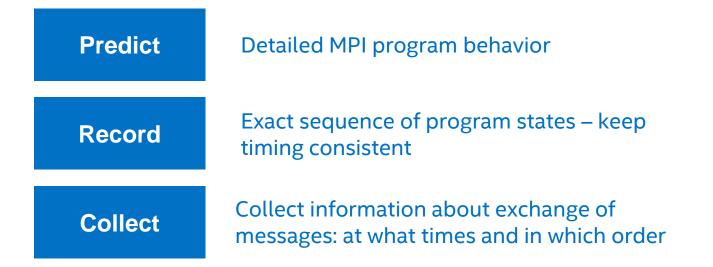


NEW in 9.1: MPI Performance Snapshot

#### **Optimization Notice**



### Strengths of Event-based Tracing



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.

\*Other names and brands may be claimed as the property of others.

### **Views and Charts**

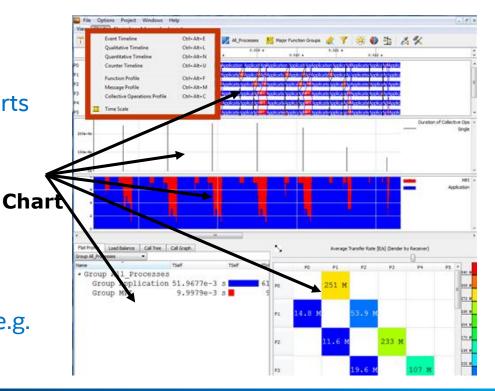
Helps navigating through the trace data and keep orientation

Every View can contain several Charts

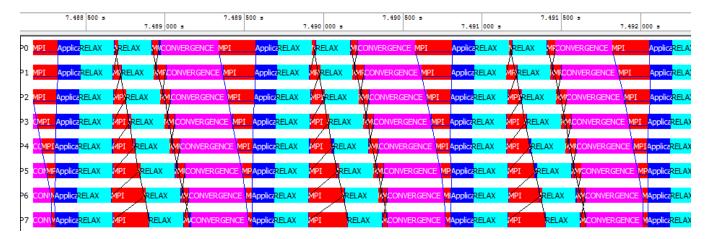
All Charts in a View are linked to a single:

- 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

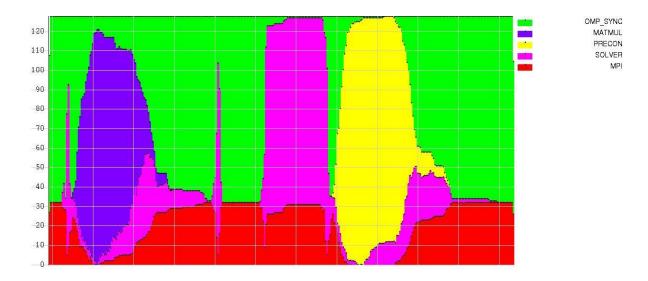
Retrieval of detailed event information



### **Quantitative Timeline**

Get impression on parallelism and load balance

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



Optimization Notice Copyright © 2016, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others.

23

### **Flat Function Profile**

### Statistics about functions

roup All_Threads	-					Children of Group All_Thread	s •					
ame	TSelf	TSelf /	TTotal	#Calls	TSelf /Call	Name	TSelf	TSelf /	TTotal	#Calls	TSelf /Call	1
Group All_Threads						+-MPI Comm dup						
- PRECON	678.787 445		678.787 445 s	49 536		Here MPI Waitall						
- OMP_SYNC	580.473 344	5	580.473 344 s	296 320	0.001 959 s	Process 31 Thread 0	0.913 338	s s	0.913 338 9	1 546	0.000 591	s
MATMUL	410.463 131	5	410.463 131 s	49 280	0.008 329 s	Process 24 Thread 0	0.801 994	s	0.801 994 9	1 5 4 6	0.000 519	IS
SOLVER	328.400 819	5	2 169.146 934 s	128	2.565 631 s	Process 28 Thread 0	0.756 392	s <b>see s</b>	0.756 392 s	1 5 4 6	0.000 489	s
-User_Code	149.746 154	S	2 383.561 817 s	128	1.169 892 s	Process 23 Thread 0	0.721 329	5	0.721 329 s	1 5 4 6	0.000 467	s
-MPI_Bcast	94.227 914	s 📕	94.227 914 s	37 248	0.002 530 s	Process 27 Thread 0	0.711 207		0.711 207 s			
- ASSEMBLY	43.822 701	s	43.822 701 s	32	1.369 459 s	Process 7 Thread 0	0.643 754		0.643 754 9			
- MPI_Barrier	24.222 499	s	24.222 499 s	49 312	0.000 491 s	- Process 15 Thread 0	0.637 547		0.637 547 9			
- MPI_Reduce	23.807 645	5	23.807 645 s	37 184	0.000 640 s	- Process 16 Thread 0	0.628 403		0.628 403 9			
-MPI_Waitall	17.607 615	s	17.607 615 s	49 472	0.000 356 s	Process 0 Thread 0	0.610 254		0.610 254 9			
- MPI_Comm_dup	11.756 564	s	11.756 564 s	64	0.183 696 s	Process 8 Thread 0	0.598 698		0.598 698 5			
-MPI Isend	7.838 689	s	7.838 689 s	145 324	0.000 054 s	Process 4 Thread 0	0.594 556		0.594 556 s			
- MPI_Wtime	7.490 313	s	7.490 313 s	136 192	0.000 055 s	Process 20 Thread 0	0.575 368		0.575 368 9			
-MPI_Irecv	4.909 197	s	4.909 197 s	145 324	0.000 034 s	Process 25 Thread 0	0.573 404		0.573 404 9			
-MPI_Finalize	0.006 288	5	0.006 288 s	32	0.000 197 s	Process 26 Thread 0	0.571 285		0.571 285 9			
- MPI_Comm_size	0.001 205	5	0.001 205 s	64	0.000 019 s	Process 11 Thread 0	0.555 121		0.555 121 s			
-MPI_Comm_rank	0.000 293	s	0.000 293 s	32	0.000 009 s	Process 30 Thread 0	0.547 251		0.547 251 s			
						Process 29 Thread 0	0.547 177		0.547 177 5			
						Process 3 Thread 0	0.540 298		0.540 298 9			
						- Process 19 Thread 0	0.510 765		0.510 765 9			
						- Process 2 Thread 0	0.495 491		0.495 491 9			
						- Process 12 Thread 0	0.485 023		0.485 023 s			
						- Process 5 Thread 0	0.480 013		0.480 013 s			
						Process 21 Thread 0	0.474 150		0.474 150 s			
						Process 6 Thread 0	0.466 212		0.466 212 5			
						Process 18 Thread 0	0.452 495		0.452 495 9			
						- Process 1 Thread 0	0.448 999		0.448 999 9			
						- Process 13 Thread 0	0.392 865		0.392 865 9			
						Process 22 Thread 0	0.387 010		0.387 010 s			
						Process 14 Thread 0	0.377 664		0.377 664 s			
						Process 17 Thread 0	0.377 174		0.377 174 s			
						Process 10 Thread 0	0.374 776		0.374 776 s			
						Process 9 Thread 0	0.357 603		0.357 603 s			
						- MPI Irecv	0.007 000		0.007 000 5	1 340	0.000 231	9
						Process 19 Thread 0	0.245 502		0.245 502 \$	6 184	0.000 040	1 0
						Process 17 Thread 0	0.243 382		0.243 382 5			

#### **Optimization Notice**

### Call Tree and Call Graph

### Function statistics including calling hierarchy

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

Vame	TSelf	TSelf	TTotal	#Calls	TSelf/Call	TSelf/Call /	
- Process 5	Total	Tooli	TTOCA	#Guild	Toolinotei	Toenroun /	Ľ
Process 4							
Process 3							
≜-User Cade	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	1	
- ExchangeStart	4.567 565 s		7.396 478 s	4 458	0.001 025 s		
- MPI Isend	1.435 251 s	í	1.435 251 s	8 9 1 6	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		1
L_MPI_Comm_rank	0.000 194 s		0.000 194 s	2	0.000 097 s		
- ExchangeEnd	0.000 567 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	1	
-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		
L 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				
- MPI_Camm_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				
- MPI_Barrier	0.040 269 s		0.040 269 s	2	0.020 134 s	1	
iteration	14.859 618 s		162.377 108 s				
- MPI_Allreduce	88.085 492 s		88.085 492 s	4 458	0.019 759 s	1	

Flat Profile   Load Balance   Call Tree   Call Graph					
Group All_Processes					
Name 🗸	TSelf	TSelf	∏otal	#Calls	TSelf /Call
B-Group All_Processes					
🕂 Callers					
<ul> <li>STF_ReachedEndOfFilter calling STF_WorkStackHistory</li> </ul>	0.001 000 s		0.002 869 s	37	0.000 027 s
<ul> <li>STF_InitFileInput calling STF_WorkStackHistory</li> </ul>	0.000 021 s		0.000 055 s	1	0.000 021 s
<ul> <li>STF_DecodeFilter_enter_function calling STF_WorkStackHistory</li> </ul>	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					
<ul> <li>STF_WorkStackHistory calling PAL_IsInTriplets</li> </ul>	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
<ul> <li>STF_WorkStackHistory calling STF_CallHandler</li> </ul>	0.001 647 s		0.016 717 s	35	0.000 047 s
<ul> <li>STF_WorkStackHistory calling STF_CallFromContent_end_of_history</li> </ul>	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

#### **Optimization Notice**

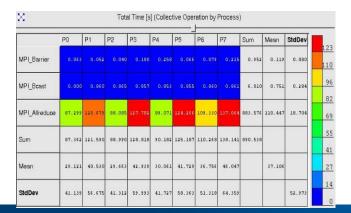
### **Communication Profiles**

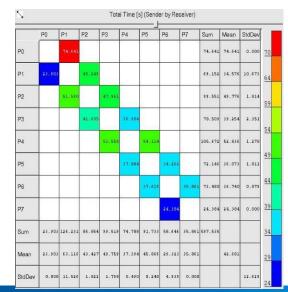
Statistics about point-to-point or collective communication Matrix supports grouping by attributes in each dimension

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

Available attributes

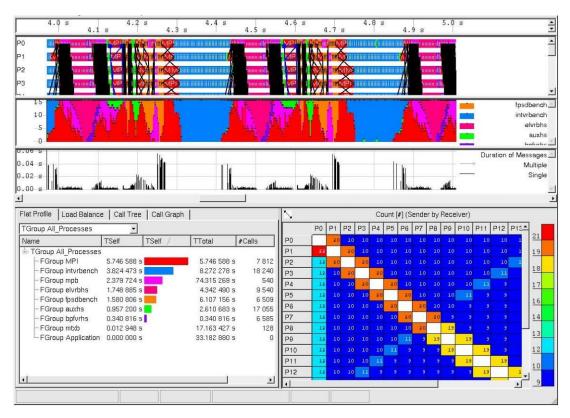
Count, Bytes transferred, Time, Transfer rate





**Optimization Notice** 

### Zooming





# **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



Major Function Groups

### Arbitrary nesting is supported

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

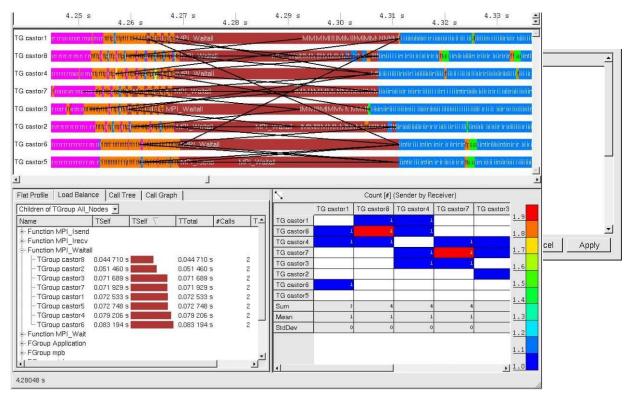
### Aggregation is part of View-definition

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

#### **Optimization Notice**



## **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

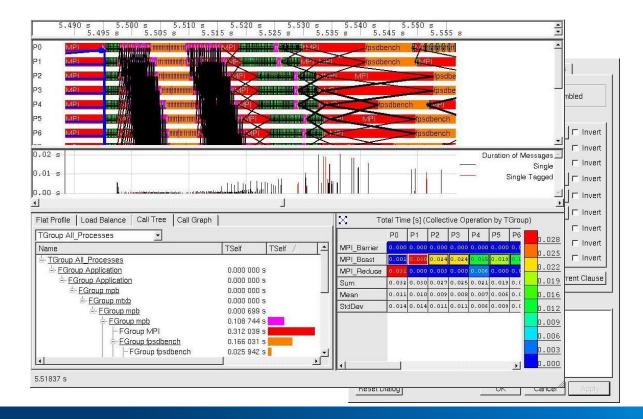
### Apply it

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

#### **Optimization Notice**



## **Tagging Example**

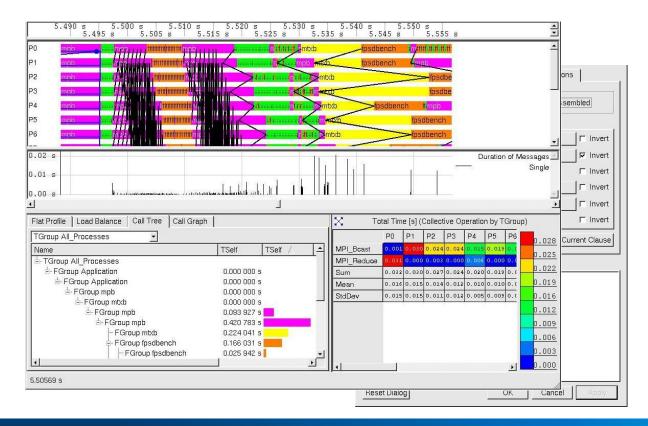


**Optimization Notice** 

Copyright © 2016, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others. 31



## Filtering Example



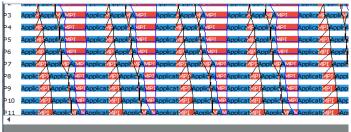
#### **Optimization Notice**

### **MPI Performance Assistance**

Automatic Performance Assistant

Detect common MPI performance issues

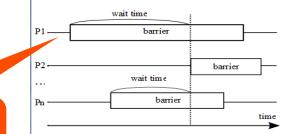
Automated tips on potential solutions



Performance Issue	Duration (%)	Duration
Wait at Barrier	0.90%	5.97002e-3 s
Late Sender	0.80%	5.28789e-3 s
Late Receiver Show all	0.80%	5.28789e-3 s

Description Affected Processes

Wait at Barrier



Automatically detect performance issues and their impact on runtime

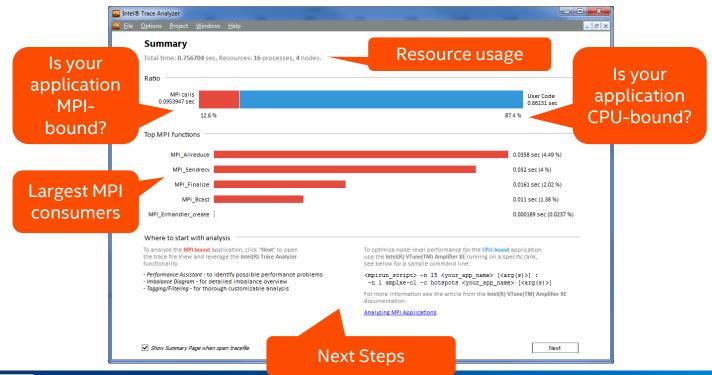
This problem occurs when barrier collective operations (such as MPI\_Barrier or all-to-all operations such as MPI\_Alltr problem indicates load imbalance in a program.

0.0397587 s, Function Mi

#### **Optimization Notice**



# Summary page shows computation vs. communication breakdown



#### Optimization Notice

### MPI-3.0 Support

### Support for major MPI-3.0 features

- Non-blocking collectives
- Fast RMA
- Large counts

Non-blocking Allreduce (MPI\_Iallreduce)

🖬 🏋 🛄	0.073 582 - 0.073 813	: 0.000 231 Sec	onds 🔻	All_Processes	MPI expanded in		ups) 😥 🍸 🔆 🏠 👔	Σ
P0 Application V	PI Iall Application	MPI Wait	-				4 MIA IN A MPEA Application	
P1 Application	MPI IallApplication	MPI Wait					MMM/AMM/PApApplication	
P1 Application							MMM ACAMPA Application	
P2 con	MPI_IallrApplication	MPI_Wait						
<u>- I_Waital</u>	AppliApplication	MPI_IallrAp	dication MR	PI_Wait			MMM WAY AND Waite ArApplica	tion
ApplicationMPM	MPMMMMPI_Waitall	ArApplicat	ion MPI	Iallr Application	MPI_Wait		MMMPM/NIPI MP	ArApplication
P5 Application	MPM	PMP MMMMPT	ApApplication	MPI_Ialr/	pplication M2	I_Wait	MINING MARKAN	ApApplicat
P6 Application A	MAMMANNMPI_Waital	ArAppli	ation MP	I Ialre Application	MPI_Wait		Man MARSIN MP Waitel WA	polication
P7 MPI_Waital	AApplication	MPI_TalkAA	oplication X	PI Wait			MMMINA A PLAAppication	/
P8 Application	MPI Jalle AApplicatio						Market Babant And	
P9 Application	PI Ial Application	MPI Wait						
PS Mppication		HP1_Walt						
•								
Flat Profile	Load Balance Call Tree	e Call Graph			Performance Issue	Duration (%)	Duration	
All Processes	-				Wait at Barrier	0.00%	4.83046e-3 s	
					Late Receiver	0.00%	4.22804e-3 s	
Name	TSelf	TSelf	TTotal	#Calls TS	Late Sender	0.00%	186.548e-6 s	
All_Process	ses plication 28.7039e-3		84.1394e-3	1511 18				i i i i i i i i i i i i i i i i i i i
MPI Ially			4.082e-3		Description	Affected Processes	Source Locations (Root Causes)	
MPI_Isend			3.41077e-3		Wait at Barrier			
	/ 1.78376e-3		1.78376e-3	1698 1.				

#### **Optimization Notice**

### Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

#### **Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

