# INTEL® MPI ON THE INTEL® OMNI-PATH ARCHITECTURE

Profile, Analyze & Visualize MPI Applications

Dr.-Ing. Michael Klemm
Software and Services Group

Part of Intel® Parallel Studio XE Cluster Edition
and Available Individually

# 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.



**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

# Agenda

General MPI Tuning for Intel® Omni-Path Architecture

Application Performance Snapshot

Intel Trace Analyzer and Collector

# Agenda

General MPI Tuning for Intel® Omni-Path Architecture

Application Performance Snapshot

Intel Trace Analyzer and Collector

# WHAT'S INSIDE INTEL® PARALLEL STUDIO XE

Comprehensive Software Development Tool Suite

## COMPOSER EDITION

**BUILD**
Compilers & Libraries

C / C++ Compiler
Optimizing Compiler

Fortran Compiler
Optimizing Compiler

Intel® Threading Building Blocks
C++ Threading Library

Intel® Math Kernel Library

Intel® Integrated Performance Primitives
Image, Signal & Data Processing

Intel® Data Analytics Acceleration Library

Intel® Distribution for Python*
High Performance Scripting

## PROFESSIONAL EDITION

**ANALYZE**
Analysis Tools

Intel® VTune™ Amplifier
Performance Profiler

Intel® Inspector
Memory & Thread Debugger

Intel® Advisor
Vectorization Optimization & Thread Prototyping

## CLUSTER EDITION

**SCALE**
Cluster Tools

Intel® MPI Library
Message Passing Interface Library

Intel® Trace Analyzer & Collector
MPI Tuning & Analysis

Intel® Cluster Checker
Cluster Diagnostic Expert System

Intel® Architecture Platforms

Operating System: Windows*, Linux*, MacOS[1]*

More Power for Your Code - software.intel.com/en-us/parallel-studio-xe

# INTEL® MPI LIBRARY PRODUCTS

## Intel® MPI 2018 Update 1

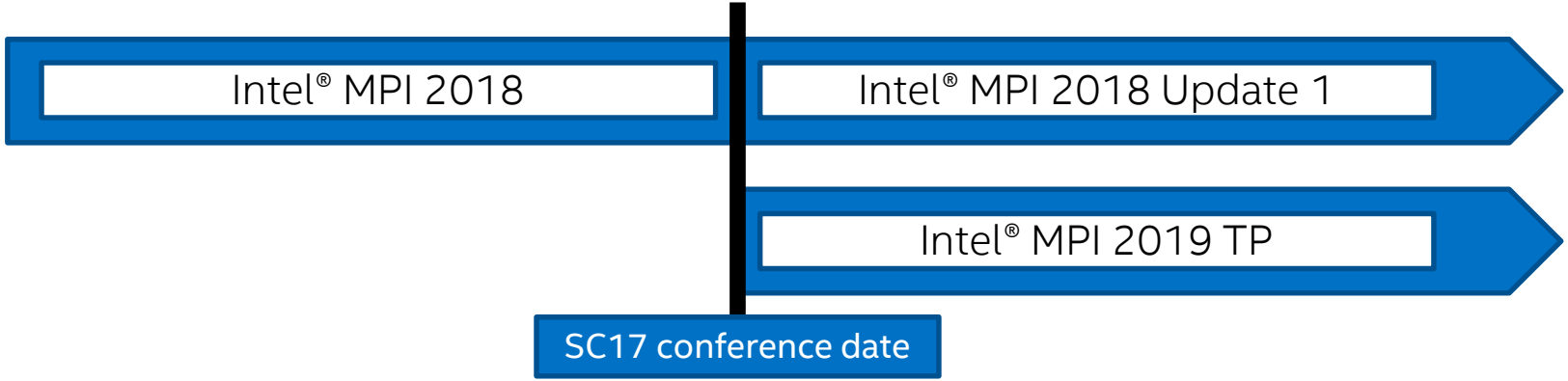
Main scope:

- bug fixes
- minor changes

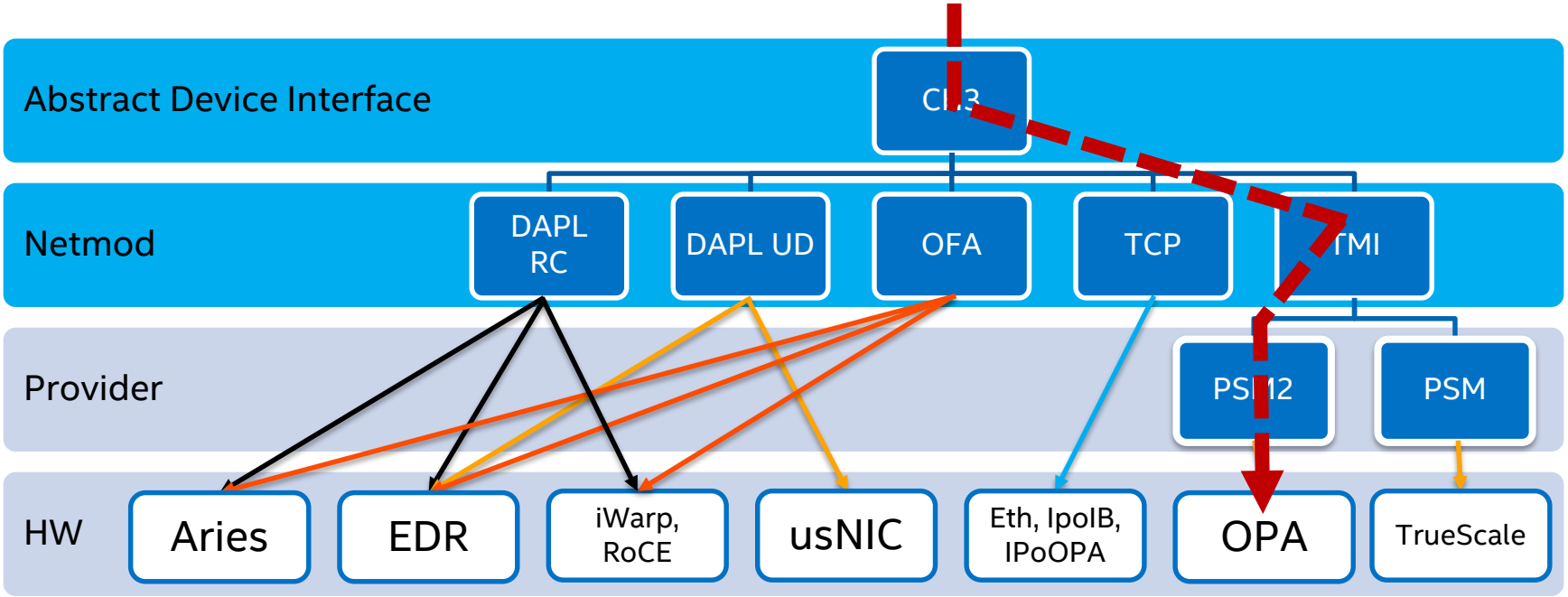
## Intel® MPI 2019 Technical Preview (TP)

Main scope:

- new features
- new infrastructure



# Intel MPI Software Stack – Choosing the Right Path



# Disable Fallback (for Benchmarking)

- Intel MPI Library falls back from the to ‘tcp’ and/or ‘shm:tcp’ if the fabric provider initialization fails
- Disable I_MPI_FALLBACK to ensure that the fast default fabric is working

  Fallback is disabled by default if I_MPI_FABRICS is set (use one of the following):

```
I_MPI_FALLBACK=0
I_MPI_FALLBACK=disable
I_MPI_FALLBACK=off
```

# Use Lightweight Statistics

- Set I_MPI_STATS to a non-zero integer to gather MPI communication statistics (max. 10)
- Change scope with I_MPI_STATS_SCOPE
- Example here:

  Gromacs rank 0 with

  **I_MPI_STATS=3**

  **I_MPI_STATS_SCOPE=coll**

```
Communication Activity by actual args
Collectives
Operation        Context  Algo  Comm size    Message size      Calls  Cost(%)
-------------------------------------------------------------------------------
Allreduce
1                     58     1          4              24          1     0.00
2                     58     1          4               4          8     0.00
3                     58     1          4               8         12     0.03
4                     58     1          4            1376        181     0.04
5                     58     1          4            1344         19     0.01
6                     58     1          4            1216          1     0.00
7                     58     1          4             224          1     0.00
8                      0     5        192               8          2     0.00
9                      0     5        192             968          1     0.00
10                     0     5        192             288          2     0.01
11                     0     5        192             768          2     0.00
Barrier
1                     62     5        160               0          1     0.00
2                      0     5        192               0          1     0.00
Bcast
...
Gather
1                     52     3          5              32         25     0.01
2                     54     3          4              36         25     0.00
3                     56     3          8              28         25     0.01
Reduce
1                     60     1         40              24          1     0.00
2                     60     1         40               4          8     0.00
3                     60     1         40               8         12     0.01
4                     60     1         40            1376        181     0.21
5                     60     1         40            1344         19     0.03
6                     60     1         40            1216          1     0.00
7                     60     1         40             224          1     0.00
Scatter
1                     62     1        160               8          1     0.00
Scatterv
1                     62     1        160          315840          2     0.03
2                     62     1        160           52640          1     0.08
===============================================================================
```

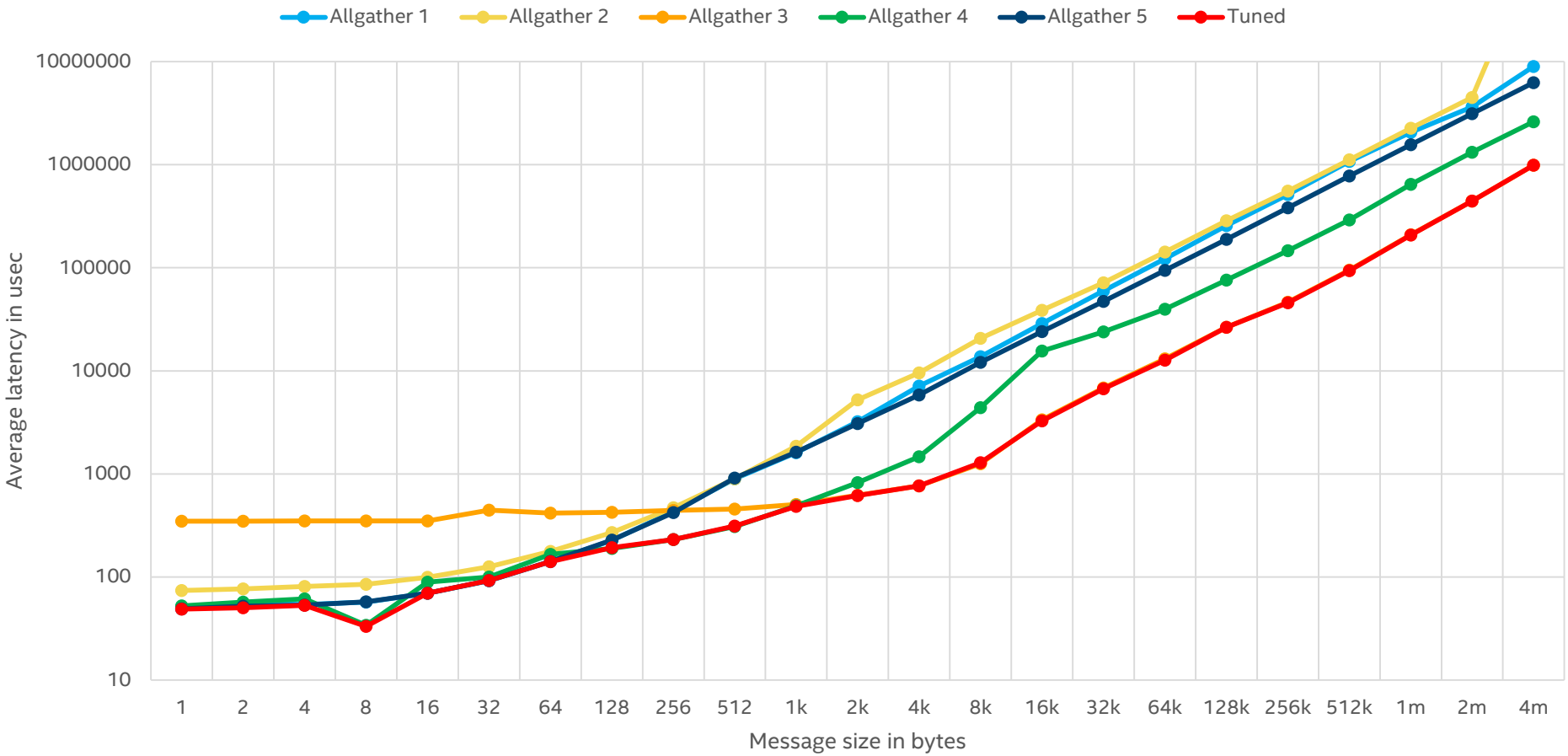
# Choose the Best Collective Algorithm

Use one of the I_MPI_ADJUST_<opname> knobs to change the algorithm

- Focus on the most critical collective operations (see statistics)
- Use the Intel® MPI Benchmarks by selecting various algorithms to find out the right algorithms for collective operations
- Or use mpitune for automatic (/fast) tuning!

```
I_MPI_ADJUST_<collective>=<algorithm#>
```

# Allgather on 8 HSW-EP nodes N256/PPN32



# Tuning for Numerical Stability

## Which algorithms of the collective reduction operations are topologically-aware?

| Collective MPI Operation using Reductions | Intel MPI Library Collective Operation Control Environment | Non-Topologically Aware Algorithms |
|---|---|---|
| **MPI_Allreduce** | I_MPI_ADJUST_ALLREDUCE | (1) , 2 , 3 , 5 , 7 , 8, 9 |
| **MPI_Exscan** | I_MPI_ADJUST_EXSCAN | 1 |
| **MPI_Reduce_scatter** | I_MPI_ADJUST_REDUCE_SCATTER | 1 , 2 , 3 , 4 |
| **MPI_Reduce** | I_MPI_ADJUST_REDUCE | 1 , 2 , 5, [7] |
| **MPI_Scan** | I_MPI_ADJUST_SCAN | 1 |

Further information can be found in the PUM #21

Notes:
(1) – The first algorithm of MPI_Allreduce is not topologically aware, it does however not guarantee to provide conditionally reproducible results
[7] – The Knomial algorithm provides reproducible results, only if the I_MPI_ADJUST_<COLLECTIVE-OP-NAME>_KN_RADIX environment is kept stable – or unmodified

# Tuning for numerical stability

```
program rep
  use mpi
  implicit none
  integer :: n_ranks,rank,errc
  real*8 :: global_sum,local_value

  call MPI_Init(errc)
  call MPI_Comm_size(MPI_COMM_WORLD, n_ranks, errc)
  call MPI_Comm_rank(MPI_COMM_WORLD, rank, errc)

  local_value = 2.0 ** -60

  if(rank.eq.15) local_value= +1.0
  if(rank.eq.16) local_value= -1.0

  call MPI_Reduce(local_value,global_sum,1,MPI_DOUBLE_PRECISION, &
       MPI_SUM,0,MPI_COMM_WORLD, errc)

  if(rank.eq.0) write(*,'(f22.20)') global_sum

  call MPI_Finalize(errc)
end program rep
```

```
$ cat ${machinefile_A}
ehk248:16
ehs146:16
ehs231:16
ehs145:16
$ cat ${machinefile_B}
ehk248:32
ehs146:32
ehs231:0
ehs145:0
$ mpiifort -fp-model strict –o ./rep.x ./rep.f90

$ export I_MPI_ADJUST_REDUCE=3
$ mpirun -n 64 -machinefile ${machinefile_A} ./rep.x
0.00000000000000000000
$ mpirun -n 64 -machinefile ${machinefile_B} ./rep.x
0.00000000000000004163

$ export I_MPI_ADJUST_REDUCE=1
$ mpirun -n 64 -machinefile ${machinefile_A} ./rep.x
0.00000000000000004163
$ mpirun -n 64 -machinefile ${machinefile_B} ./rep.x
0.00000000000000004163
```

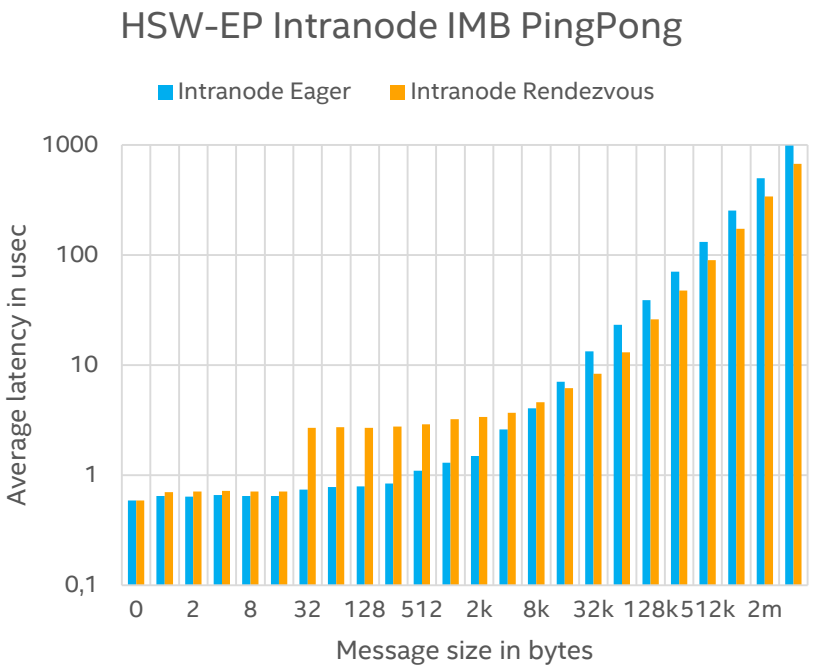
# Adjust the Eager/Rendezvous Protocol Threshold

Two communication protocols:

1. "Eager" sends data immediately regardless of receive request availability - used for short messages
2. "Rendezvous" notices receiving site on data pending and transfers when receive request is set (RTS/CTS)

**I_MPI_EAGER_THRESHOLD**

Default is 256k bytes



# Async Progress for Non-blocking Operations

- Overlapping communication and computation is possible by spawning a helper thread
- Can cause oversubscription - therefore deactivated by default

Enabling asynchronous progress

```
MPICH_ASYNC_PROGRESS=1
```

# Leverage the KNL High Bandwidth Memory (HBM)

IMPI offers several different ways to leverage MCDRAM on KNL, including options to bind the whole target application onto the fast memory.

```
I_MPI_HBW_POLICY=
    <user memory>[,<mpi memory>][,<win_allocate>]
```

| policy | Description |
|---|---|
| **hbw_preferred** | Allocate the local HBW memory for each process. If the HBW memory is not available, allocate the local dynamic random access memory. |
| **hbw_bind** | Allocate only the local HBW memory for each process. |
| **hbw_interleave** | Allocate the HBW memory and dynamic random access memory on the local node in the round-robin manner. |

# KNL-F – Integrated Omni Path Fabric Usage

Use either the OpenFabrics Interface (OFI) or the Tag Matching Interface (TMI) for the KNL-F integrated Omni Path fabric.

**`I_MPI_FABRICS=ofi`** or

**`I_MPI_FABRICS=tmi`** and **`I_MPI_TMI_PROVIDER=psm2`**

While the fabric can be utilized in combination with Shared Memory (SHM),

**`I_MPI_FABRICS=shm:ofi`**

It might be faster to leverage the pure fabric (as shown above) without SHM – in order to activate the direct (short) receive path within IMPI (implicitly).

**`I_MPI_OFI_DRECV=1`**

# Special OPA Tuning for Multiple Nodes

## OPA configuration parameters (KNL, recommended by OPA team)

```
export PSM2_MQ_RNDV_HFI_WINDOW=4194304
export PSM2_MQ_EAGER_SDMA_SZ=65536
export PSM2_MQ_RNDV_HFI_THRESH=200000
export PSM2_IDENTIFY=1

export HFI_NO_CPUAFFINITY=1

export I_MPI_FABRICS=shm:tmi
export I_MPI_TMI_PROVIDER=psm2
export I_MPI_FALLBACK=0
```

# Agenda

General MPI Tuning for Intel® Omni-Path Architecture

Application Performance Snapshot

Intel Trace Analyzer and Collector

# Intel® Performance Snapshots

## Three Fast Ways to Discover Untapped Performance

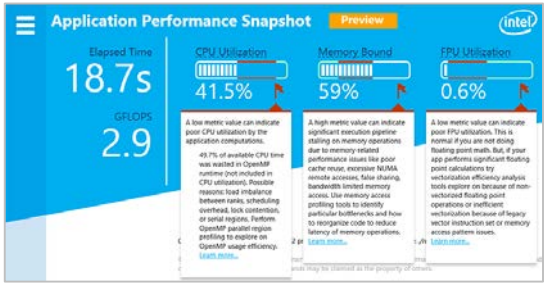
### Is your application making good use of modern computer hardware?

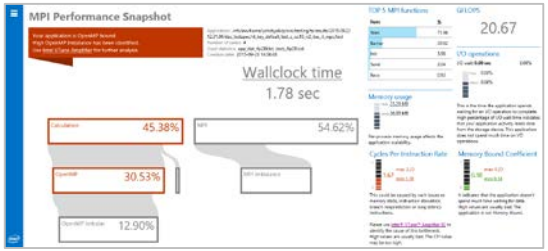
- Run a test case during your coffee break.
- High level summary shows which apps can benefit most from code modernization and faster storage.

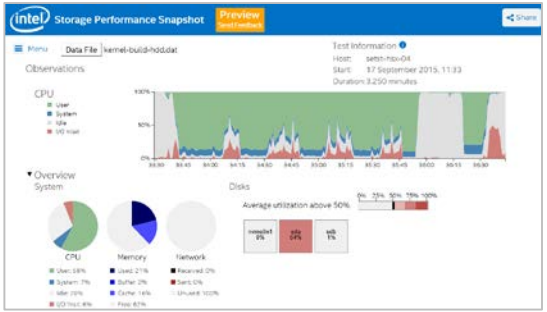
### Pick a Performance Snapshot:

- **Application** – for non-MPI apps
- **MPI** – for MPI apps
- **Storage** – for systems. Servers and workstations with directly attached storage.

**Free download:** http://www.intel.com/performance-snapshot
Also included with Intel® Parallel Studio and Intel® VTune™ Amplifier products.



# Application Performance Snapshot

Preview

Elapsed Time

18.7s

GFLOPS

2.9

CPU Utilization

41.5%

A low metric value can indicate poor CPU utilization by the application computations.

49.7% of available CPU time was wasted in OpenMP runtime (not included in CPU utilization). Possible reasons: load imbalance between ranks, scheduling overhead, lock contention, or serial regions. Perform OpenMP parallel region profiling to explore on OpenMP usage efficiency.
Learn more...

Memory Bound

59%

A high metric value can indicate significant execution pipeline stalling on memory operations due to memory-related performance issues like poor cache reuse, excessive NUMA remote accesses, false sharing, bandwidth limited memory access. Use memory access profiling tools to identify particular bottlenecks and how to reorganize code to reduce latency of memory operations.
Learn more...

FPU Utilization

0.6%

A low metric value can indicate poor FPU utilization. This is normal if you are not doing floating point math. But, if your app performs significant floating point calculations try vectorization efficiency analysis tools explore on because of non-vectorized floating point operations or inefficient vectorization because of legacy vector instruction set or memory access pattern issues.
Learn more...

# MPI Performance Snapshot

Your application is OpenMP bound.
High OpenMP imbalance has been identified.
Use Intel VTune Amplifier for further analysis.

Application: /nfs/inn/home/yshchyok/p/svn/testing/ts/results/2015.09.23 12.31.09/itac_testspec/vt_key_default_test_c_icc15_n2_itac_it_mps/test
Number of ranks: 4
Used statistics: app_stat_4p28t.txt, stats_4p28t.txt
Creation date: 2015-09-28 14:58:48

## Wallclock time

1.78 sec

Calculation 45.38%

MPI 54.62%

OpenMP 30.53%

MPI Imbalance

OpenMP Imbalan 12.90%

## TOP 5 MPI functions

| Func | % |
|---|---|
| Wait | 71.98 |
| Barrier | 20.92 |
| Init | 3.98 |
| Send | 2.04 |
| Recv | 0.93 |

## GFLOPS

20.67

## I/O operations

I/O wait: **0.00 sec** 0.00%

PEAK 0.00%
MEAN 0.00%

This is the time the application spends waiting for an I/O operation to complete. High percentage of I/O wait time indicates that your application actively reads data from the storage device. This application does not spend much time on I/O operations.

## Memory usage

PEAK 25.29 MB
MEAN 20.95 MB

Per-process memory usage affects the application scalability.

## Cycles Per Instruction Rate

4
1.67
max: 2.23
min: 1.10
0

This could be caused by such issues as memory stalls, instruction starvation, branch misprediction or long latency instructions.

Please use Intel® VTune™ Amplifier XE to identify the cause of this bottleneck. High values are usually bad. The CPI value may be *too high*.

## Memory Bound Coefficient

1
0.18
max: 0.23
min: 0.14
0

It indicates that the application doesn't spend much time waiting for data. High values are usually bad. The application is *not Memory Bound*.

**Free download:** http://www.intel.com/performance-snapshot. Also included with Intel® Parallel Studio Cluster Edition.

# Storage Performance Snapshot

Preview
Send Feedback

Share

Menu

Data File | kernel-build-hdd.dat

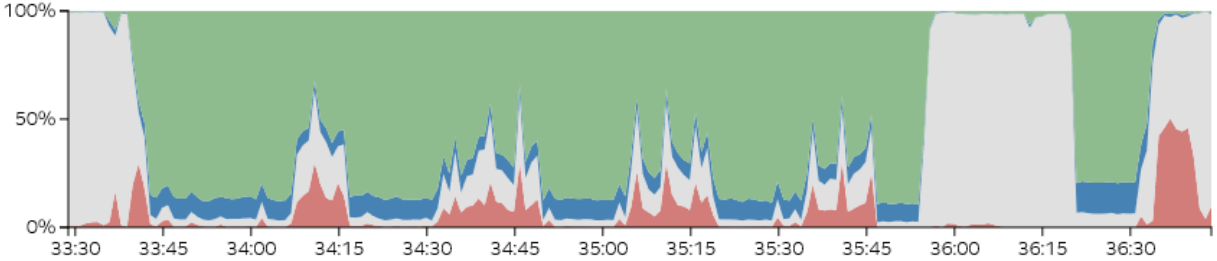
## Observations

### Test Information

Host: setst-hsx-04
Start: 17 September 2015, 11:33
Duration: 3.250 minutes

### CPU

- User
- System
- Idle
- I/O Wait

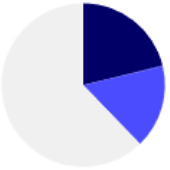



## Overview

### System

CPU

- User: 58%
- System: 7%
- Idle: 29%
- I/O Wait: 6%



Memory

- Used: 21%
- Buffer: 0%
- Cache: 16%
- Free: 62%

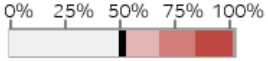


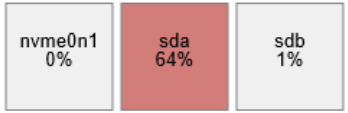
Network

- Received: 0%
- Sent: 0%
- Unused: 100%

### Disks

Average utilization above 50%



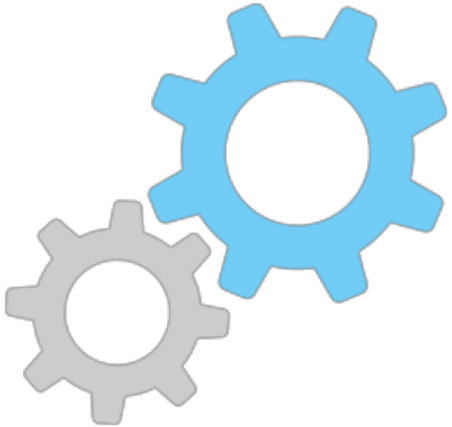
# Agenda

General MPI Tuning for Intel® Omni-Path Architecture

Application Performance Snapshot

Intel Trace Analyzer and Collector

# Intel® Trace Analyzer & Collector 2018



Learn More: software.intel.com

## Powerful Profiler, Analysis & Visualization Tool for MPI Applications

- Low overhead for accurate profiling, analysis & correctness checking
- Easily visualize process interactions, hotspots & load balancing for tuning & optimization
- Workflow flexibility: Compile, Link or Run

## What's New in 2018 version

- Support of OpenSHMEM* applications
- Supports the latest Intel® Xeon® processor (codenamed Skylake)



Certain technical specifications and select processors/skus apply. See product site for more details.

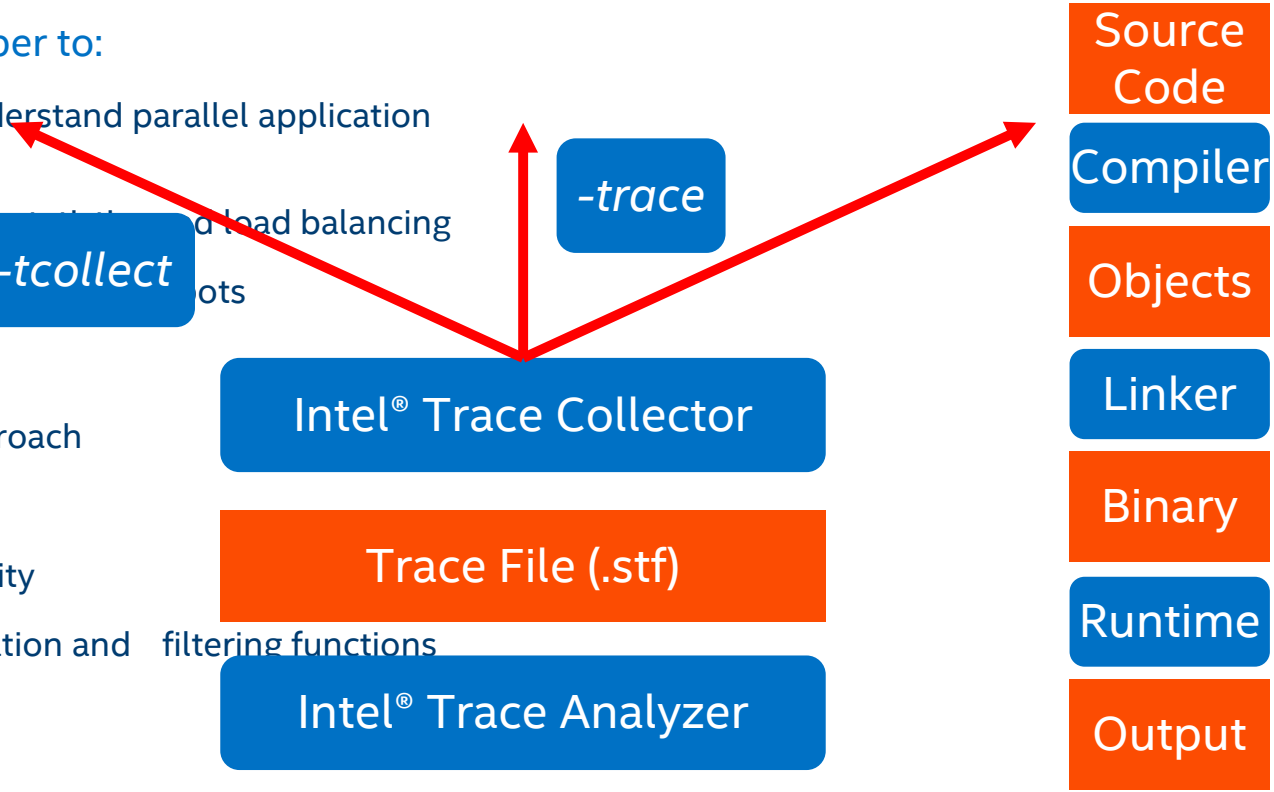
# Intel® Trace Analyzer and Collector Overview

## Helps the developer to:

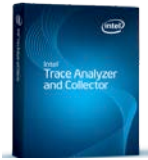
- Visualize and understand parallel application behavior
- Eval d load balancing
- Iden ots

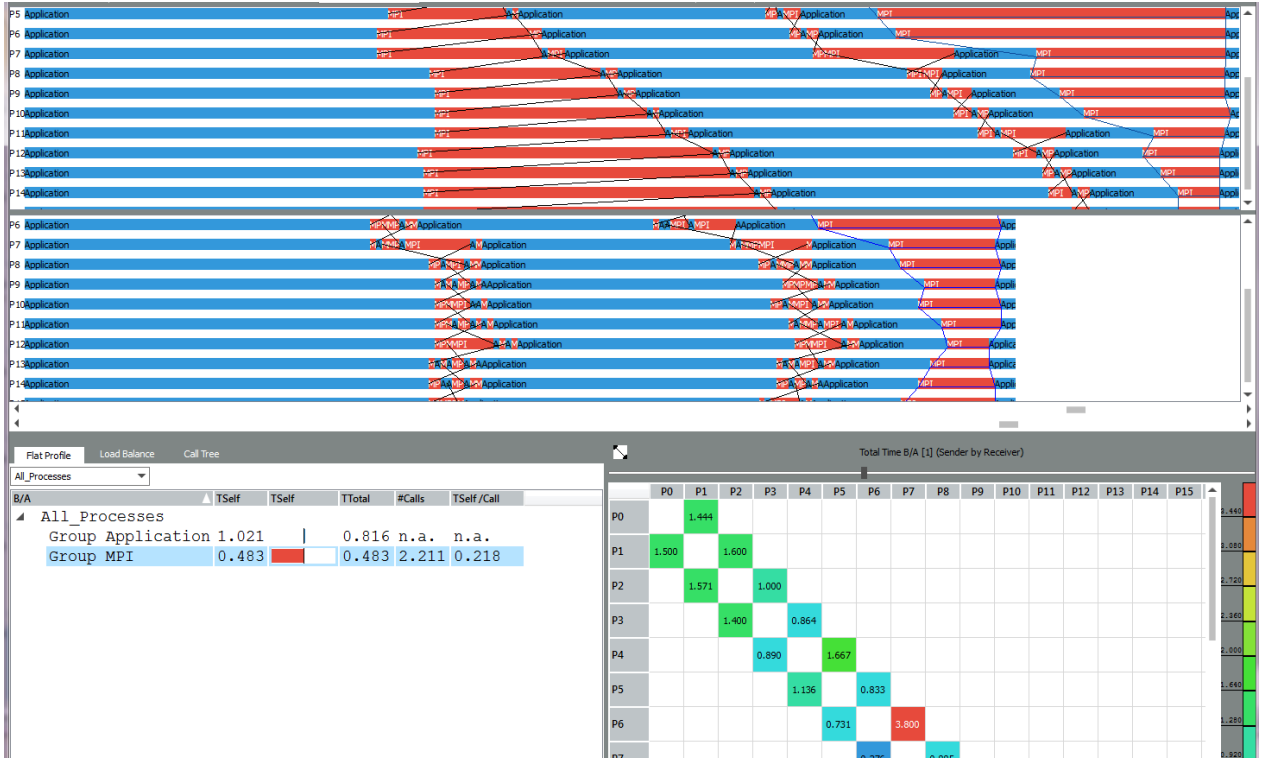
## Features

- Event-based approach
- Low overhead
- Excellent scalability
- Powerful aggregation and filtering functions
- Idealizer



# Intel® Trace Analyzer and Collector




Compare the event timelines of two communication profiles

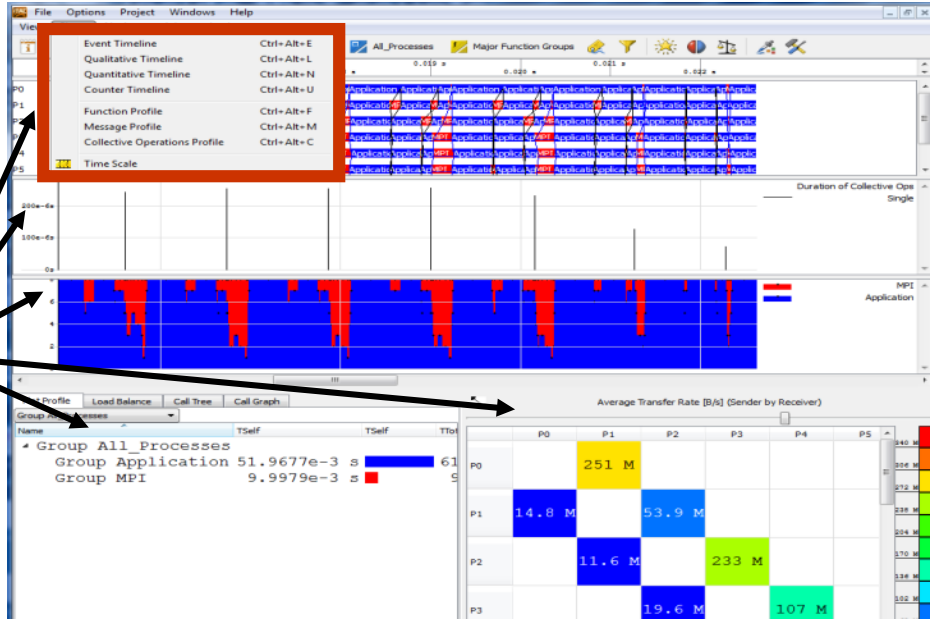
Blue = Computation
Red = Communication

Chart showing how the MPI processes interact

# 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)

Chart

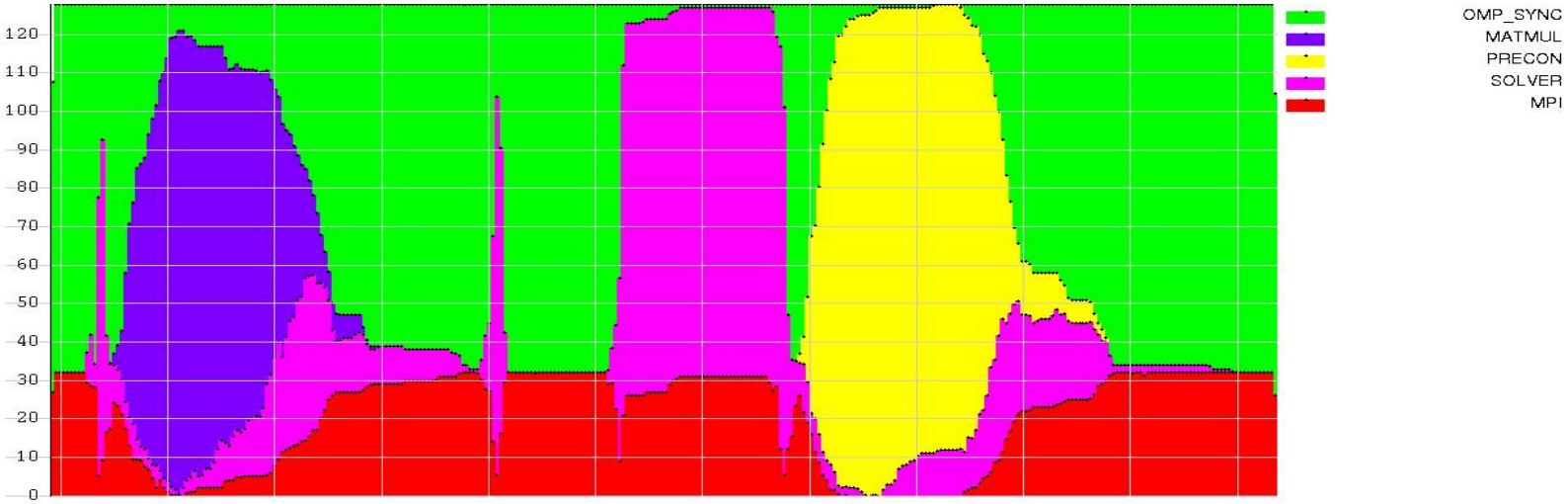


# Event Timeline


- Get detailed impression of program structure.
- Display functions, messages, and collective operations for each process/thread along time axis.
- Retrieve detailed event information.

# Quantitative Timeline

- Get impression on parallelism and load balance.
- Show for every function how many threads/processes 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

- Tree: Call-stack
- Graph: 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 | 8.539 215 s | | 8.539 215 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
- Generic matrix supports grouping by several attributes in each dimension: Sender, receiver, data volume per message, tag, communicator, type
- Available attributes: Count, bytes transferred, time, transfer rate

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

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 | 0.204 |
| 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 |

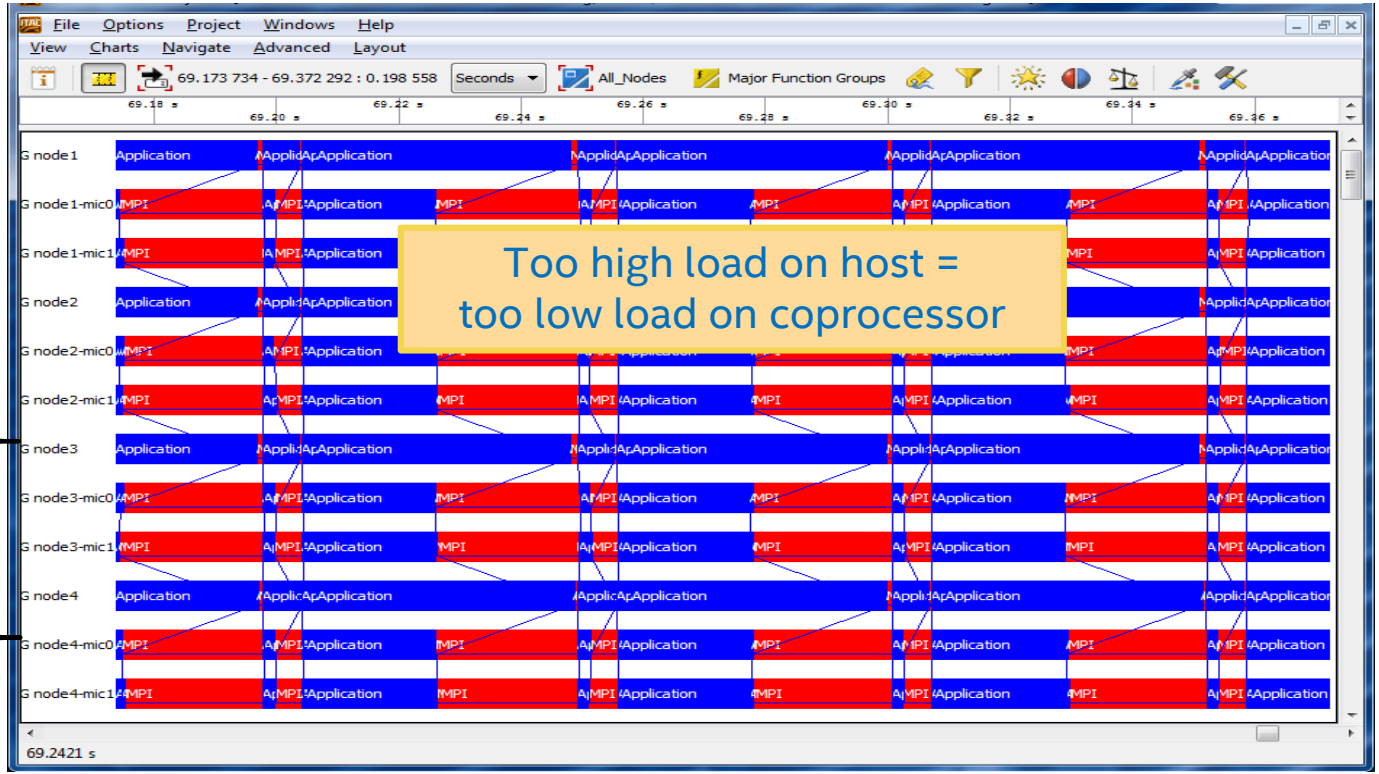
123 110 96 82 69 55 41 27 14 0

# Improving Load Balance: Real-World Case

Collapsed data per node and coprocessor card

**Host**
16 MPI procs x
1 OpenMP* thread

**Coprocessor**
8 MPI procs x
28 OpenMP threads



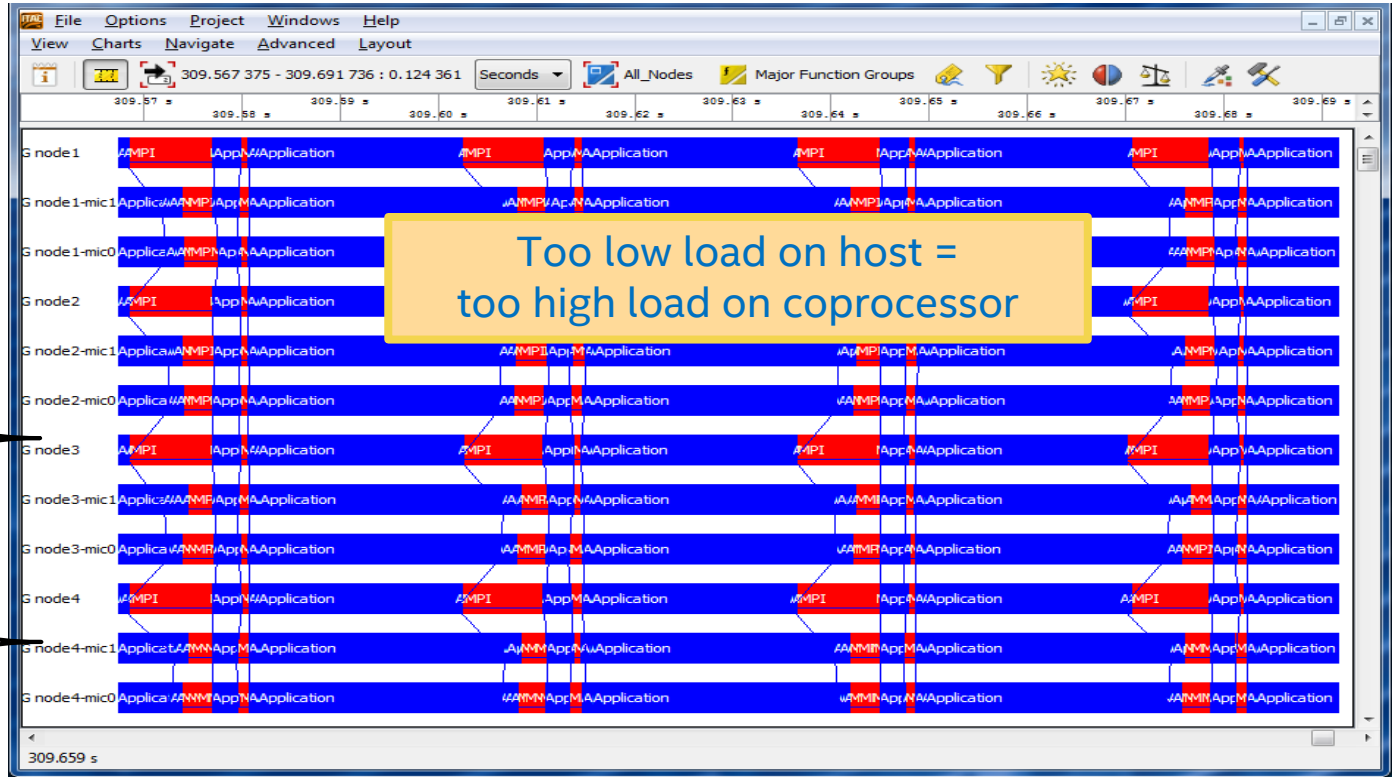

Too high load on host =
too low load on coprocessor

# Improving Load Balance: Real-World Case

Collapsed data per node and coprocessor card

**Host**
16 MPI procs x
1 OpenMP* thread

**Coprocessor**
24 MPI procs x
8 OpenMP threads

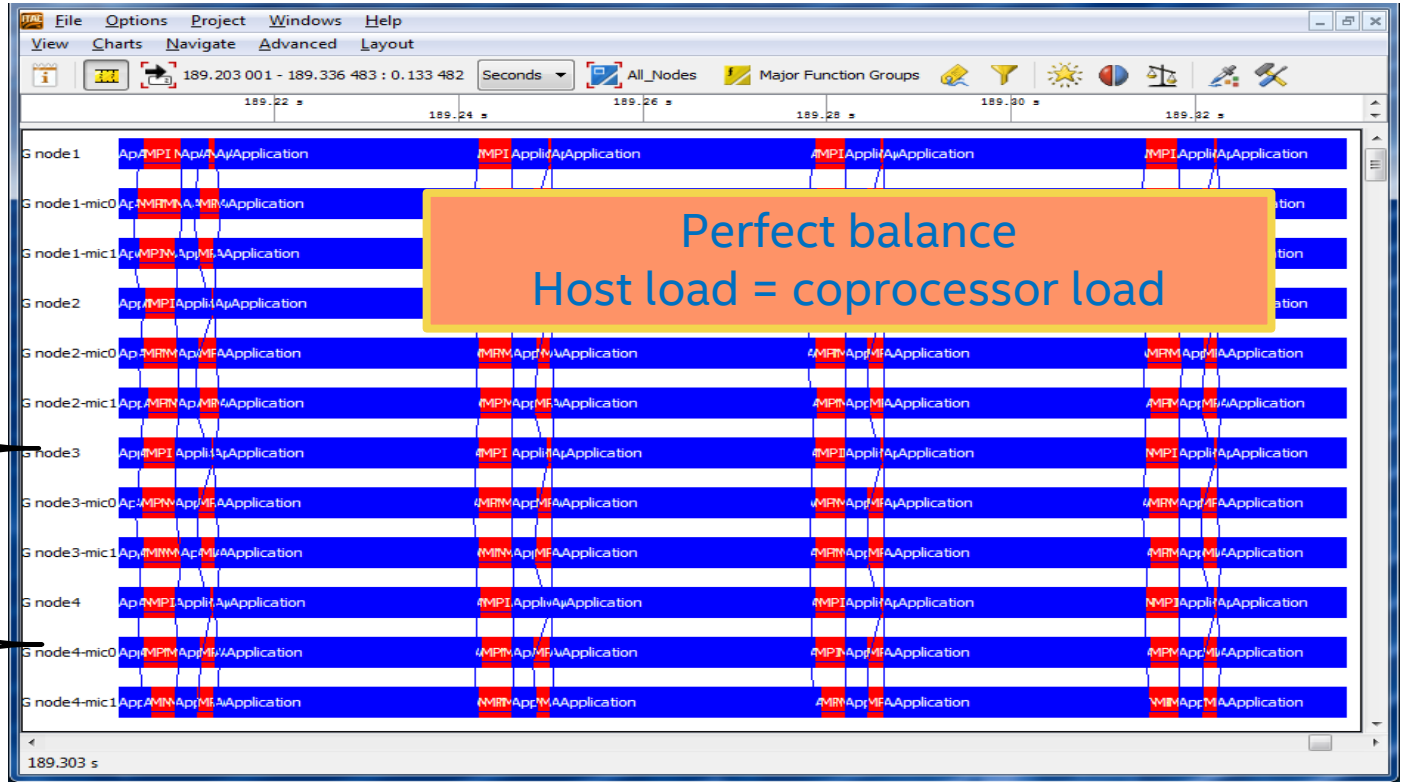


# Improving Load Balance: Real-World Case

Collapsed data per node and coprocessor card

**Host**
16 MPI procs x
1 OpenMP* thread

**Coprocessor**
16 MPI procs x
12 OpenMP thrds

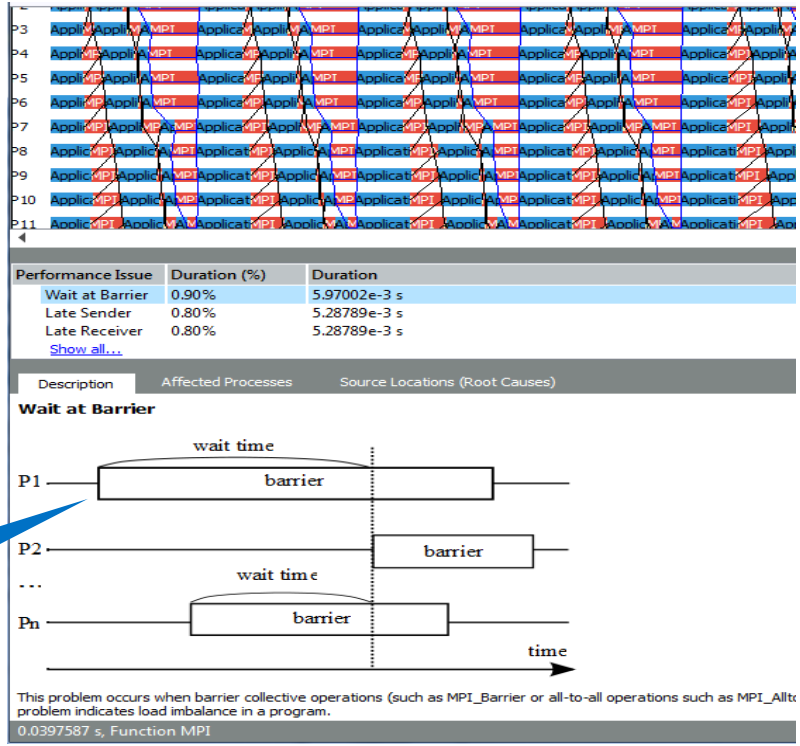


# MPI Performance Analysis

## Initial MPI-3.x Support

### Automatic Performance Assistant

- Detects common MPI performance issues
- Automated tips on potential solutions

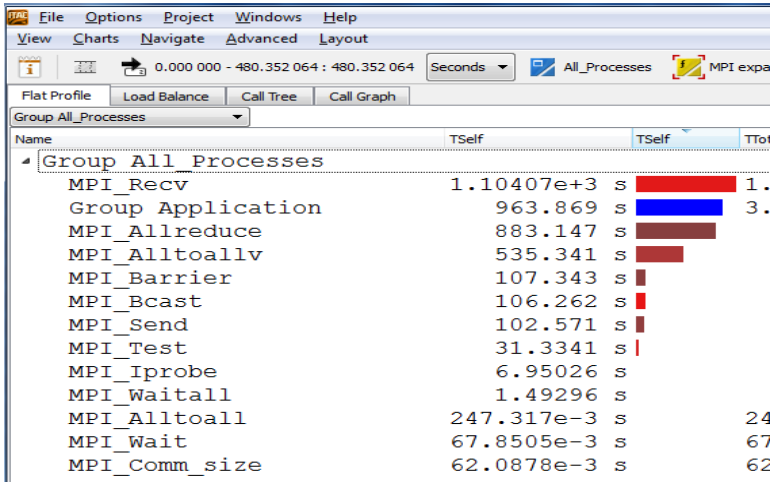



Automatically detect performance issues and their impact on runtime

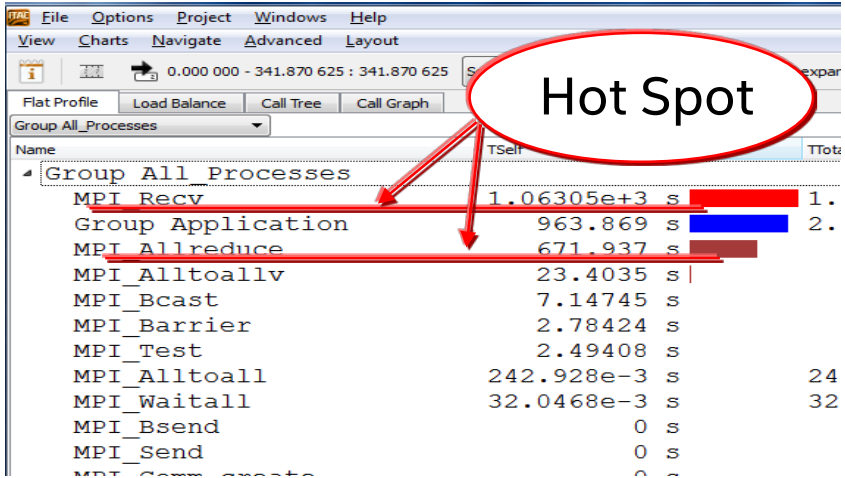
# Ideal Interconnect Simulator (Idealizer)

## Helps to Figure Out Application's Imbalance, Simulating its Behavior in the 'Ideal' Communication Environment

**Real trace**




**Ideal trace**




**Easy Way to Identify Application Bottlenecks**

intel®