



Intel® Cluster Studio XE 2013

for Distributed Performance

Boost Performance. Code Reliably. Scale Forward

James Tullos

james.a.tullos@intel.com

Technical Computing, Analyzers, and Runtimes
Intel Software & Services Group



Software

Developers

ROCK YOUR CODE.

Legal Disclaimer

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.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

Copyright © 2012. Intel Corporation.

Optimization Notice

Optimization Notice

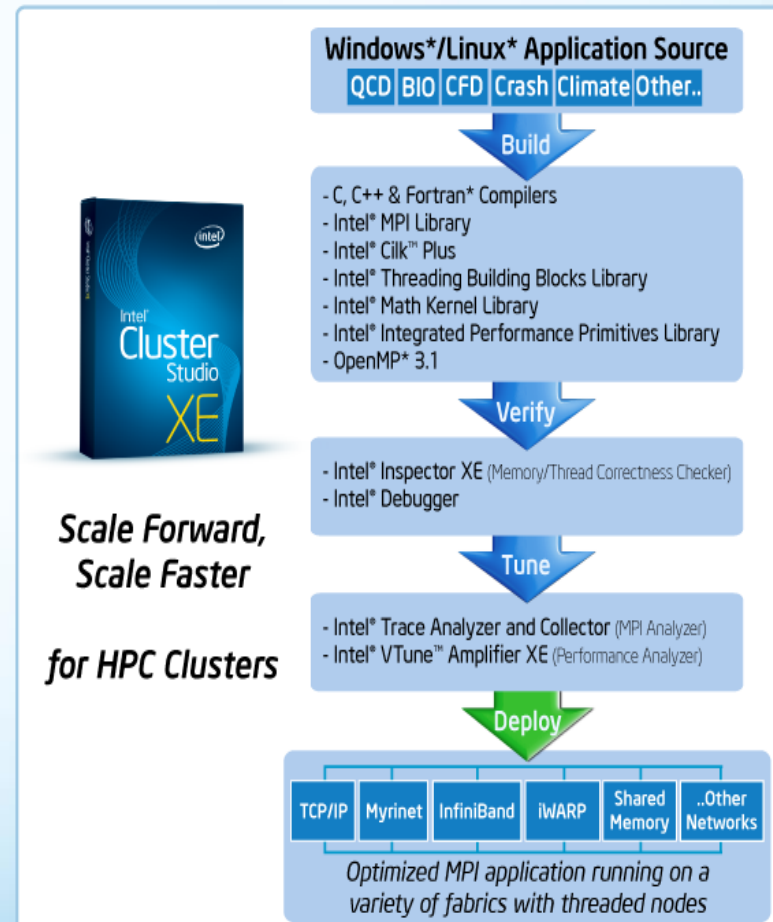
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, 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

Intel® Cluster Studio XE

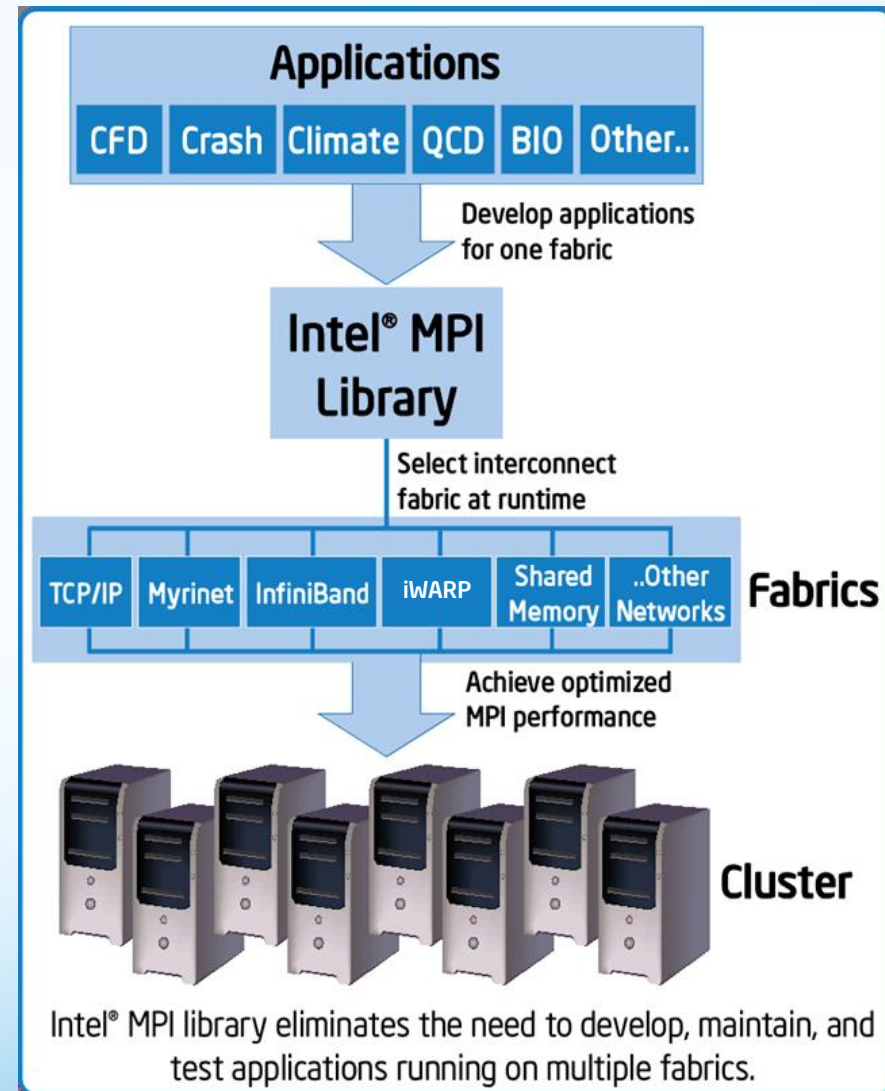
Scale Forward, Scale Faster – for HPC Clusters

- **Scale Performance – Perform on More Nodes**
 - MPI Latency - Intel® MPI Library - Up to 2.6X as fast as alternative MPI libraries
 - Compiler Performance – Industry leading Intel® C/C++ & Fortran compilers
- **Scale Forward – multicore now, many-core ready**
 - Intel® MPI Library scales beyond 120k processes
 - Parallel Programming Models – Commercially supported Intel® versions of open source Threading Building Blocks 4.0 and Intel® Cilk™ Plus 1.1, MPI, OpenMP 3.1, Coarray Fortran
 - Focused to preserve programming investments for multicore on future many-core machines
- **Scale Efficiently – Tune & Debug on More Nodes**
 - Thread & Memory Correctness Checking – Intel® Inspector XE now MPI enabled across many nodes
 - Rapid Node Level Performance Profiling – Intel VTune Amplifier XE can identify hotspots faster and on thousands of nodes



Intel® MPI Library Overview

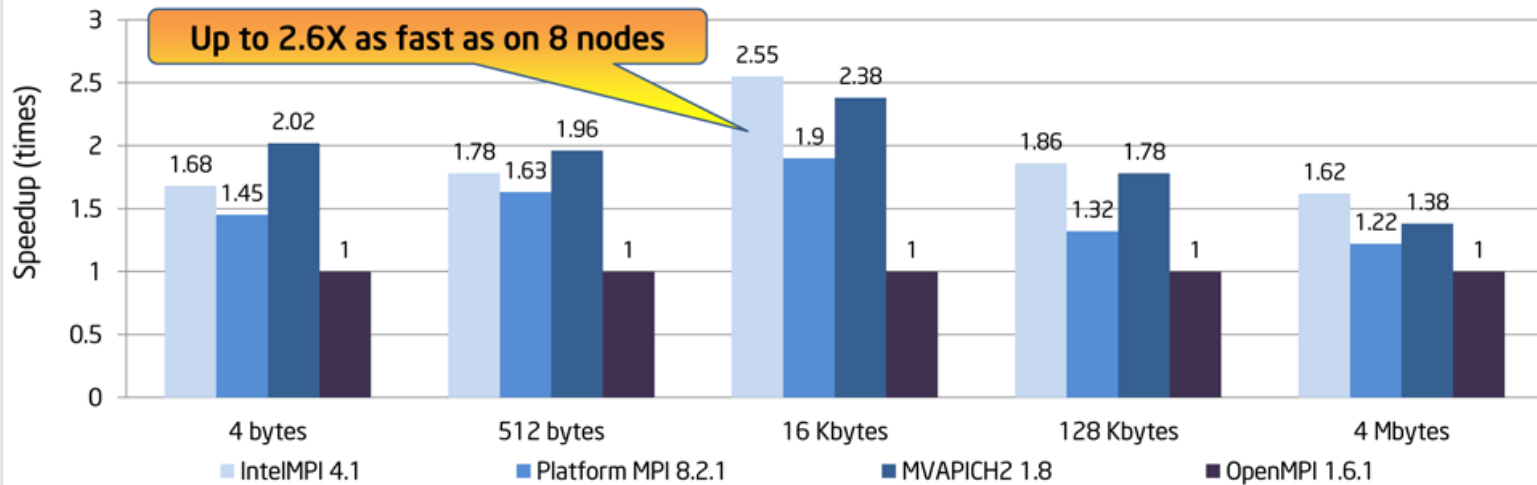
- Optimized MPI application performance
 - Application-specific tuning
 - Automatic tuning
- Lower latency and multi-vendor interoperability
 - Industry leading latency
 - Performance optimized support for the latest OFED capabilities through DAPL 2.0
- Faster MPI communication
 - Optimized collectives
- Simplify and accelerate clusters
 - “Intel® Cluster Ready”
- Sustainable scalability beyond 120K cores
 - Native InfiniBand* interface support allows for lower latencies, higher bandwidth, and reduced memory requirements
- More robust MPI applications
 - Seamless interoperability with Intel® Trace Analyzer and Collector



MPI Latency: 96 Processes / 8 Nodes on Intel processor running Linux* 64

Intel® MPI Library vs. alternative MPI libraries

Industry Leading Performance with Intel® MPI Library 4.1
 Relative (Geomean) MPI Latency Benchmarks on Linux* 64 (Higher is Better)
 96 Processes on 8 nodes (InfiniBand + shared memory)

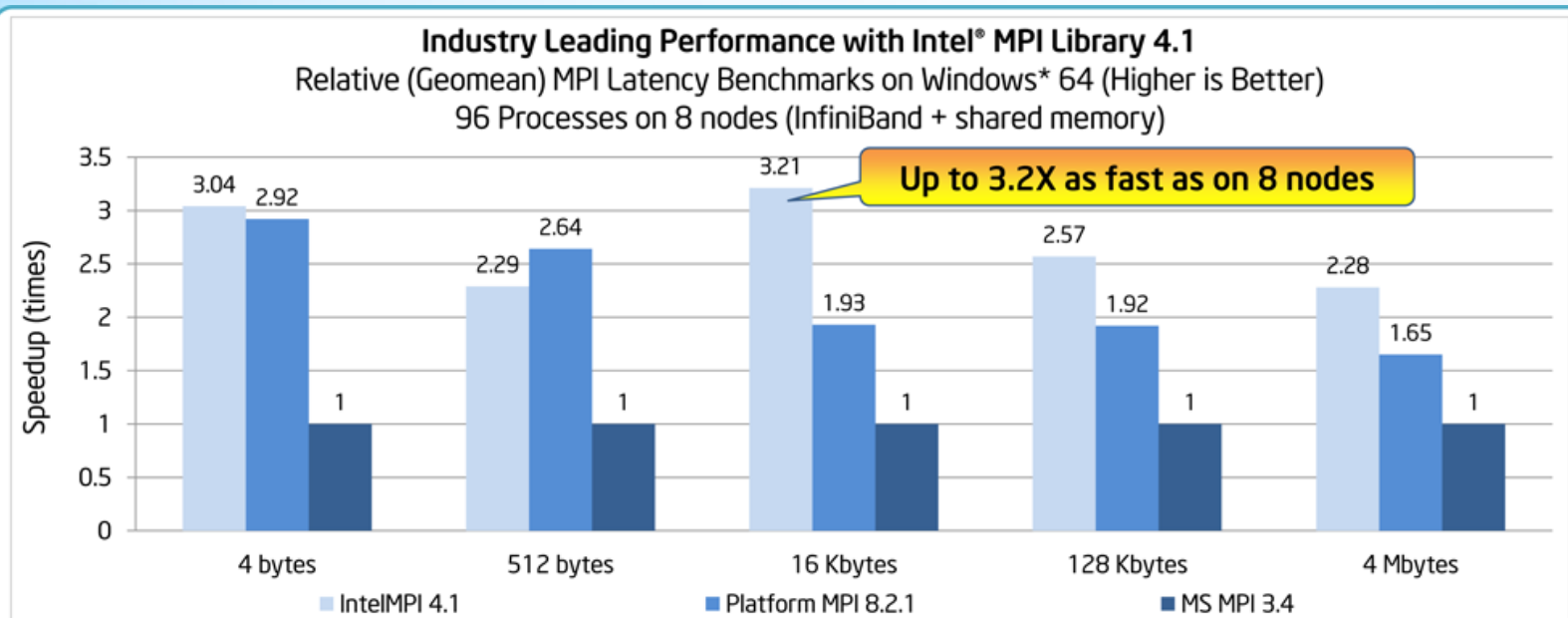


Configuration Info - SW Versions: Intel® C/C++ version 13.0, Intel® MPI Library 4.1, Platform MPI 8.2.1, MVAPICH2 1.8, Open MPI 1.6.1, Intel® MPI Benchmarks 3.2.4; Hardware: Intel® Xeon® CPU DP X5680 @ 3.33GHz, RAM 24GB; Interconnect: InfiniBand, ConnectX adapters; QDR; Operating System: SLES 11.1; Notes: 96 Processes on 8 nodes (InfiniBand + shared memory). All listed MPI libraries were built with the Intel® C++ Compiler 12.1 Update 10 for Linux*.

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation.

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-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

MPI Latency: 96 Processes / 8 Nodes on Intel processor running Windows* 64 Intel® MPI Library vs. alternative MPI libraries



Configuration Info -SW Versions: Intel® C/C++ version 13.0, Intel® MPI Library 4.1, Platform MPI 8.2.1, MS MPI 3.4, Intel® MPI Benchmarks 3.2.4; Hardware: Intel® Xeon® CPU DP X5680@3.33GHz, RAM 24GB; Interconnect: InfiniBand, ConnectX adapters; QDR; Operating System: Windows Server 2008 R2 x64 HPC Edition; Notes: 96 Processes on 8 nodes (InfiniBand + shared memory). All listed MPI libraries were built with the Intel® C++ Compiler 12.1 Update 10 for Windows*.

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation.

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

Intel® MPI Library Overview (cont'd)

- Streamlined product setup
 - Installation under root or ordinary user ID
 - mpivars.(c)sh scripts for easy path setting
- Simplified process management
 - mpiexec -perhost and -nolocal options
 - mpirun script that automates usage of the Hydra process manager
 - System-, user-, and session-specific configuration files
- Environment variables for runtime control over
 - Process pinning
 - Optimized collective operations
 - Device-specific protocol thresholds
 - Collective algorithm thresholds
 - Enhanced memory registration cache
 - Many others ...

Compile and Link Commands

- Using Intel compilers
 - `mpiicc, mpiicpc, mpiifort, ...`
- Using Gnu compilers (same underlying Intel MPI library)
 - `mpicc, mpicxx, mpif77, ...`
- Ease of use
 - Commands find the Intel® MPI Library include files automatically
 - Commands link the Intel® MPI libraries automatically
- Commands use compilers from PATH (or selected through options); compilers not hard-wired!
- Example:
 - Compile using the Intel Fortran compiler
 - \$ `mpiifort -o testf test.f`

Execution Commands

- All-inclusive

- `mpirun -f hostfile -n #processes executable`
- Most common usage scenario
 - Convenient
 - Uses new Hydra process manager by default
 - May be good for jobs in batch system
 - “In-session” mode: `mpirun` acquires the list of nodes from the batch system

- Example:

- Run the test program

```
$ mpirun -f hosts.file -n 2 ./testc
```

```
Hello world: rank 0 of 2 running on node1
```

```
Hello world: rank 1 of 2 running on node1
```

Process Placement

- Simple process placement (consecutive assignment of MPI ranks to round robin selection of nodes)

- `mpirun [-perhost #ppn] -n #procs executable`
- Place #ppn processes per node until the total number #procs of processes is reached

- Exact process placement using Argument Sets:

- `mpirun -n #p1 -hosts node1 exe1 : -n #p2 -hosts node2 exe2`
- Argument Set (separated by ":") is valid for the specified node:
- Place #p1 processes of exe1 on node1
- Place #p2 processes of exe2 on node2, ...
(usually: exe1 = exe2 = ...)
- "exe" may actually be "executable exeparams"

- Exact process placement with a config file

- One argument set per line in a file (without ":")
- Handy: comment unused lines with "#"
- Example config file:
`-n #p1 -hosts node1 exe1`
`-n #p2 -hosts node2 exe2`
`#-n #p3 -hosts dead node3 exe3`
`-n #p4 -hosts node4 exe4`
- `mpirun -configfile theconfigfile`
- No other *mpirun* flags on the command line!

Intel® MPI Library Fabric Selection

- Environment variable `I_MPI_FABRICS` selects the interconnect device at runtime
- `I_MPI_FABRICS` values:
 - `shm` (shared memory only)
 - `dapl` (DAPL fabrics)
 - `tcp` (sockets)
 - `tmi`
 - `ofa`
- `shm:dapl` fabrics is default
- Example
 - Check selected device

```
$ mpirun -f hosts.file -genv I_MPI_DEBUG 2 -n 2 ./testc
```

..... will use default fabric `shm:dapl` (RDMA-enabled device + shared memory)
 - Change selected device

```
$ mpirun -f hosts.file -genv I_MPI_DEBUG 2 \  
-genv I_MPI_FABRICS shm:ofa -n 2 ./testc
```

..... will use fabric `shm:ofa` (OFED verbs + shared memory)

Performance Tuning: mpitune

- Use the automatic tuning facility to tune the Intel® MPI Library for your cluster or application (done once, may take a long time)
- Example (see `mpitune -h` for options)
 - Cluster-wide tuning
`mpitune ...`
 - Application-specific tuning
`mpitune --application \"mpiexec -n 32 ./exe\" ...`

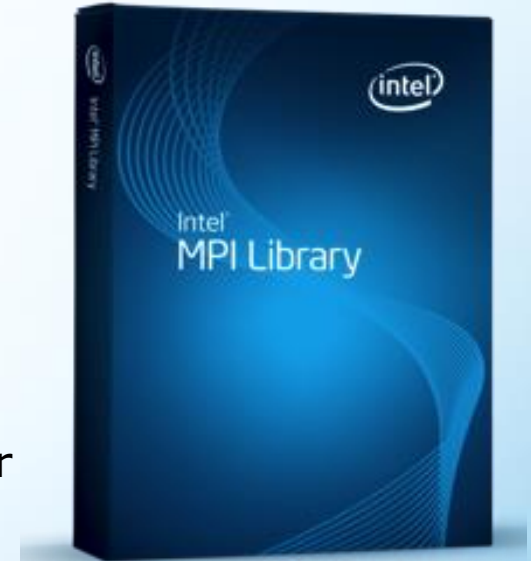
Creates options settings which are used with the `-tune` flag

```
mpirun -tune ...
```

Intel® MPI Library 4.1

What's New

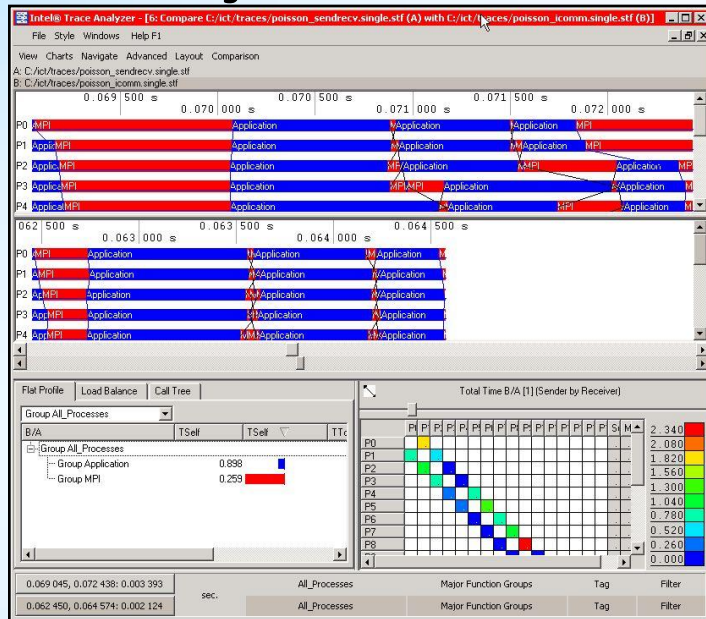
- Increased MPI application performance and scalability
 - Support for the latest Intel® Composer XE 2013 and new Intel® Architectures including:
 - Intel® Xeon Phi™ coprocessor in all programming models: offload, native, and symmetric
 - New Intel® MPI Library architecture provides industry-leading performance and sustainable scalability beyond 120K cores
 - Added conformance to the MPI-2.2 standard
 - Backwards compatibility with Intel® MPI Library 4.0.x based applications
 - Tighter integration with the PBS Pro* job manager
 - Improved heterogeneous support when running over different Intel® Architecture processors
 - Support for Berkeley Lab Checkpoint/Restart (BLCR)
 - Brand new HTML documentation format



Intel® MPI Library 4.1 contains leading edge technology to further improve performance, scalability and usability

Scale Performance Tune Hybrid Cluster MPI and Thread Performance

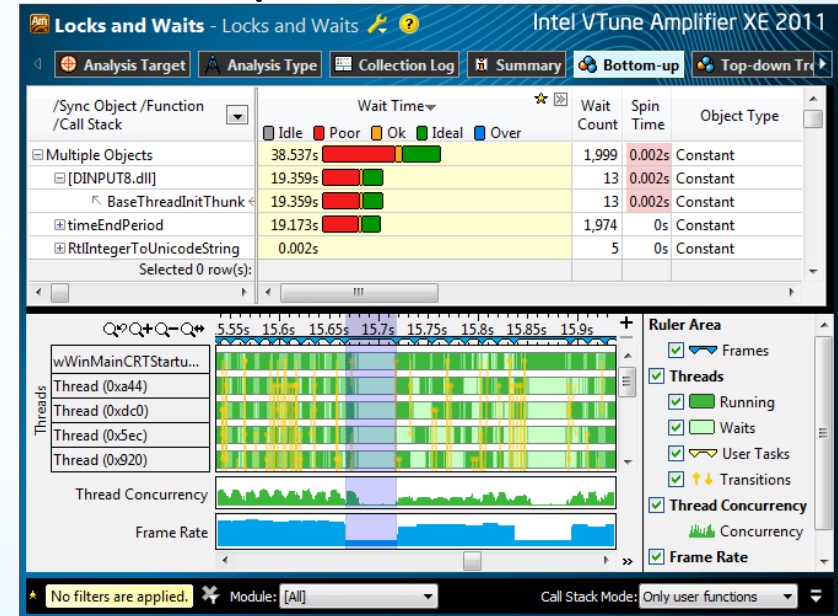
Intel Trace Analyzer and Collector



Tune cross-node MPI

- Visualize MPI behavior
- Evaluate MPI load balancing
- Find communication hotspots

Intel VTune™ Amplifier XE



Tune single node threading

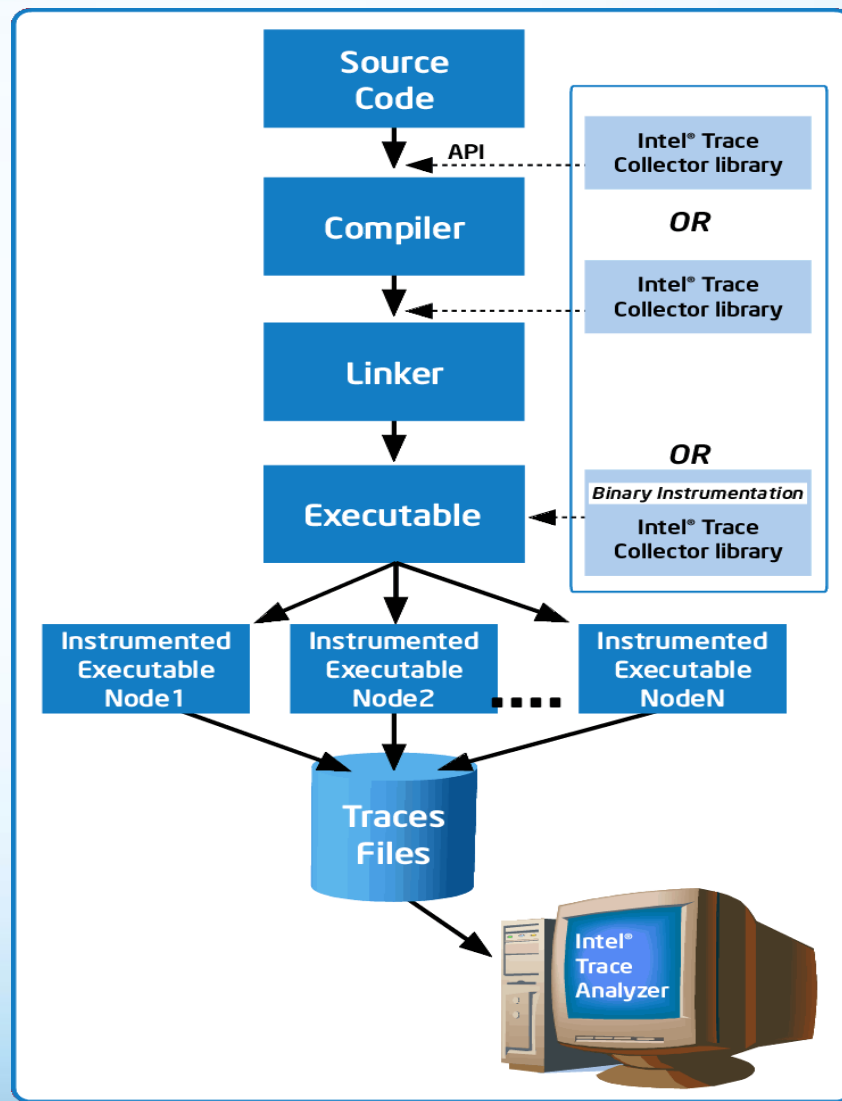
- Visualize thread behavior
- Evaluate thread load balancing
- Find thread sync. bottlenecks

Introduction – What is Tracing?

- Record program execution
 - Program events such as function enter/exit, communication
- 1:1 protocol of the actual program execution
 - Sampling gathers statistical information
- Accurate data
- Easily get loads of data

Intel® Trace Analyzer and Collector Overview

- Intel® Trace Analyzer and Collector helps the developer:
 - Visualize and understand parallel application behavior
 - Evaluate profiling statistics and load balancing
 - Identify communication hotspots



Event based approach

- Event = time stamp + thread ID + description
- Function entry/exit
- Messages
- Collective operations
- Counter samples

Strengths of Event-based Tracing

- Predict detailed program behavior
- Record **exact sequence** of program states – keep timing consistent
- Collect information about exchange of messages: **at what times** and **in which order**

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

Key Features

- Low Overhead
- Catch all MPI events
- Powerful configuration mechanism
 - Filters, settings, features
- Automatic source-code references
- Instrumentation
 - Rich API
 - Binary instrumentation (itcpin)
 - Compiler based (-tcollect)
- Fail-safe version
- Comparison of multiple profiles
- Idealizer
- MPI Correctness Checking

How to use Intel® Trace Analyzer and Collector

- **Step 1:** Run your binary and create a tracefile
run the binary for a representative amount of time (to reduce initialization influences) on representative data (no corner cases)

```
$ mpirun -trace -n 2 ./test
```

- Alternative 1: Generate an instrumented binary via re-linking

```
$ mpiicc -trace test.c -o test.inst
```

```
$ mpirun -n 2 ./test.inst
```

- Alternative 2: Instrument binary

```
itcpin --run -- ./test
```

- **Step 2:** To view the generated trace file, start the GUI:

```
traceanalyzer &
```

Support for the Intel® Xeon Phi™ coprocessor

- The tracing libraries have been fully ported to Xeon Phi™
- Make sure the necessary libraries are accessible to the card via:
 - NFS-sharing the /opt/intel & \$HOME directories (*preferred*)
 - Manually copying the files:

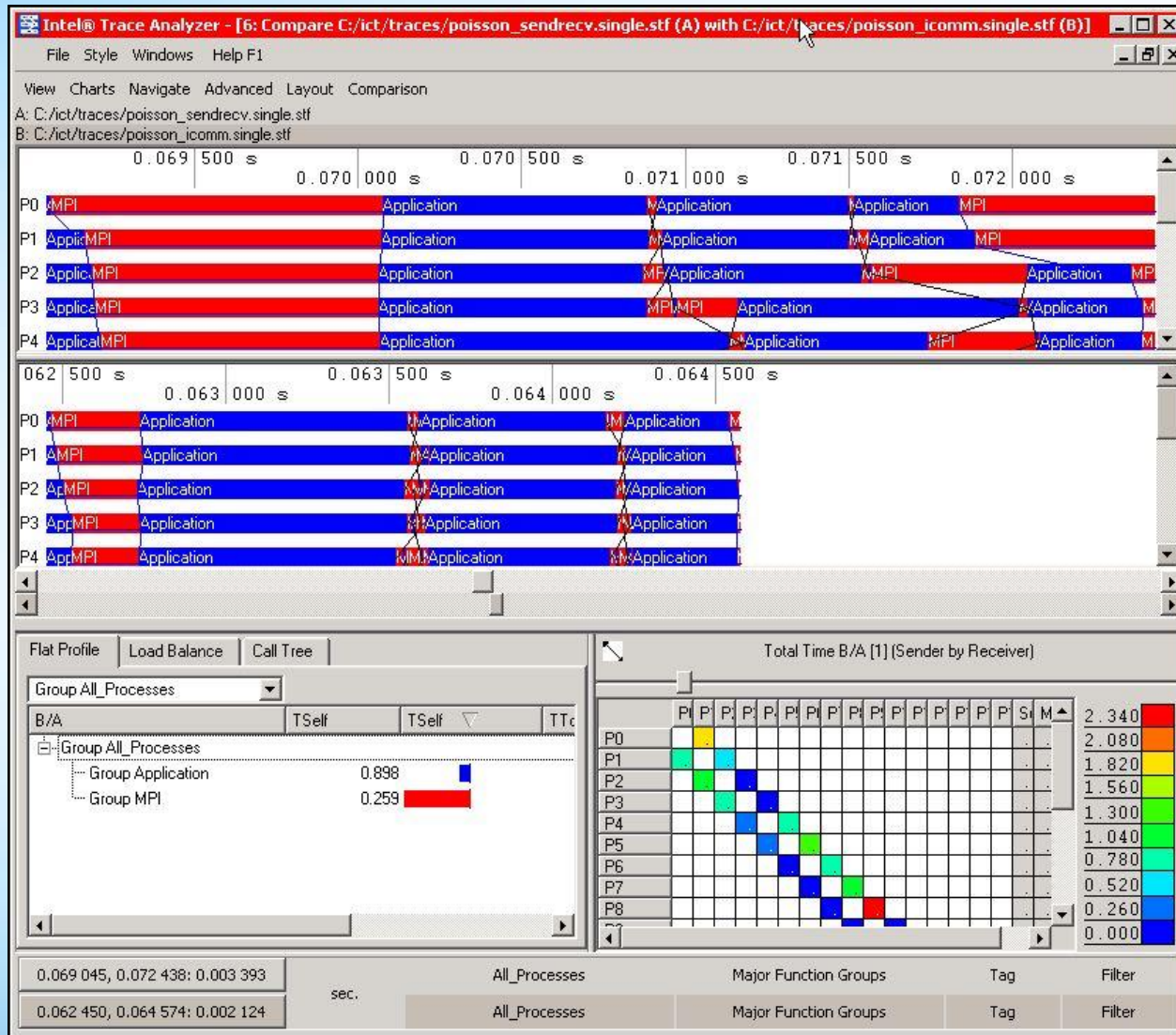
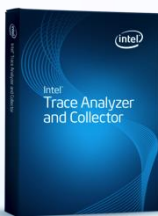
```
$ scp /opt/intel/itac/<version>/mic/slib/libVT.so mic0:/lib64
```
- Now run as shown previously:

```
$ mpirun -trace -n 2 ./test
```
- Make sure you have all trace files in the same directory:
 - If your card is NFS-shared, all trace files will be accessible in the directory where the executable is located (*preferred*)
 - If your card is not NFS-shared, you might have partial trace files created on the Xeon Phi™ coprocessor that you need to copy:

```
$ scp mic0:/home/<user>/test.stf.* node0:~
```
- Finally view using the GUI:

```
$ traceanalyzer test.single.stf &
```

Intel® Trace Analyzer and Collector

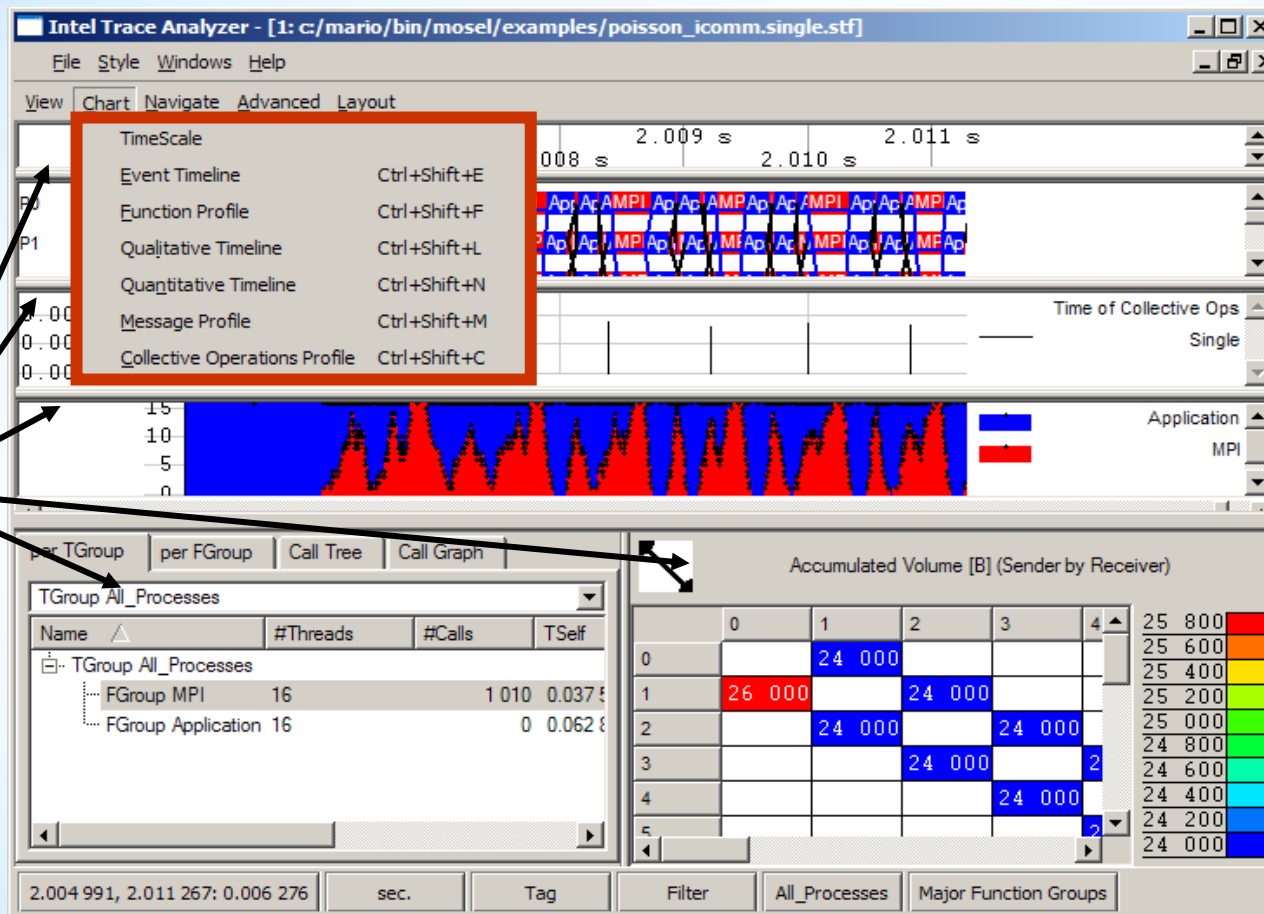


Compare the event timelines of two communication profiles

Blue = computation
Red = communication

Chart showing how the MPI processes interact

Chart

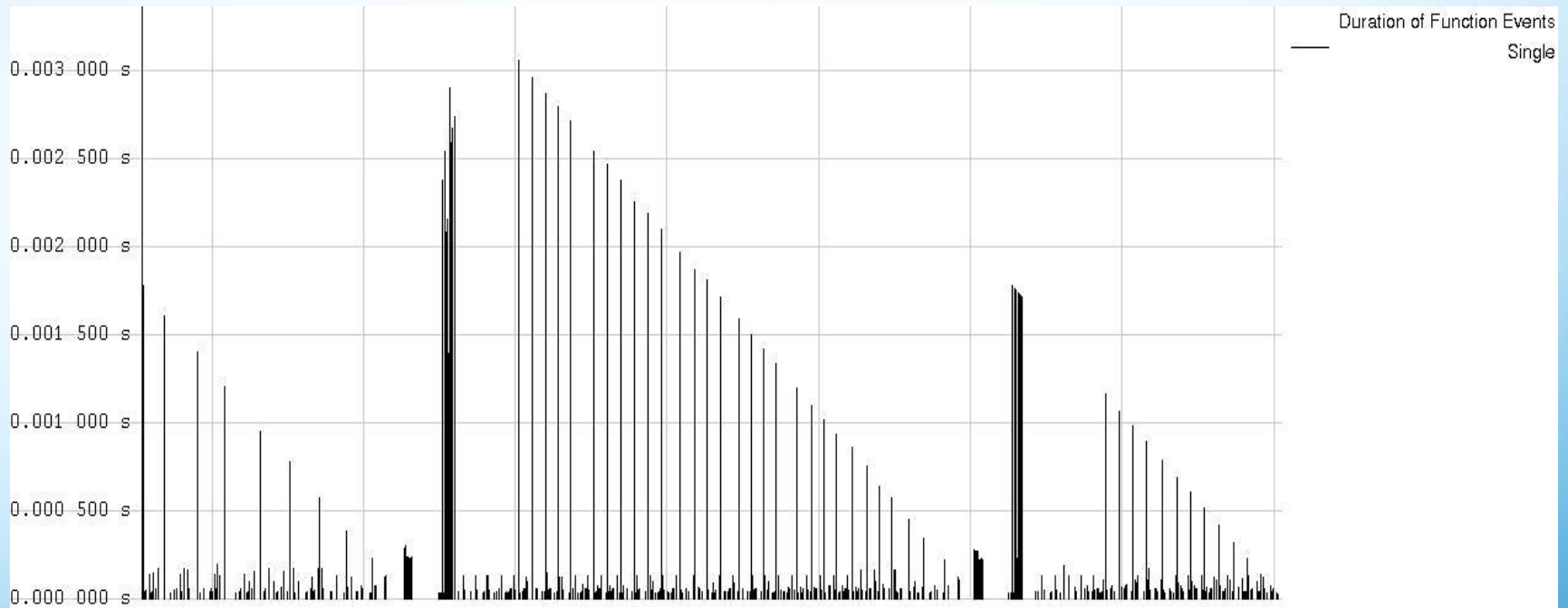


Chart

A Chart is a numerical or graphical diagram

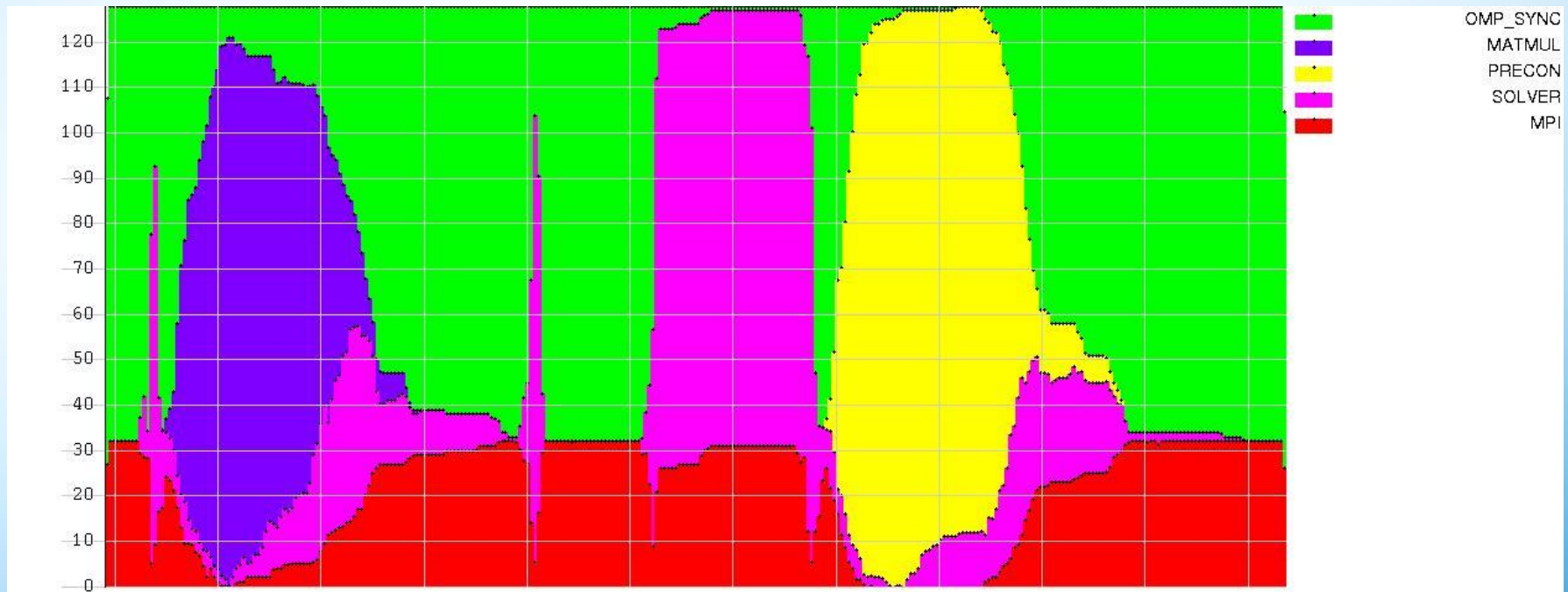
Timelines: Qualitative Timeline

- Find patterns and irregularities
- Display attributes of functions, messages or collective operations as they occur for any process/thread
- Retrieval of detailed event information



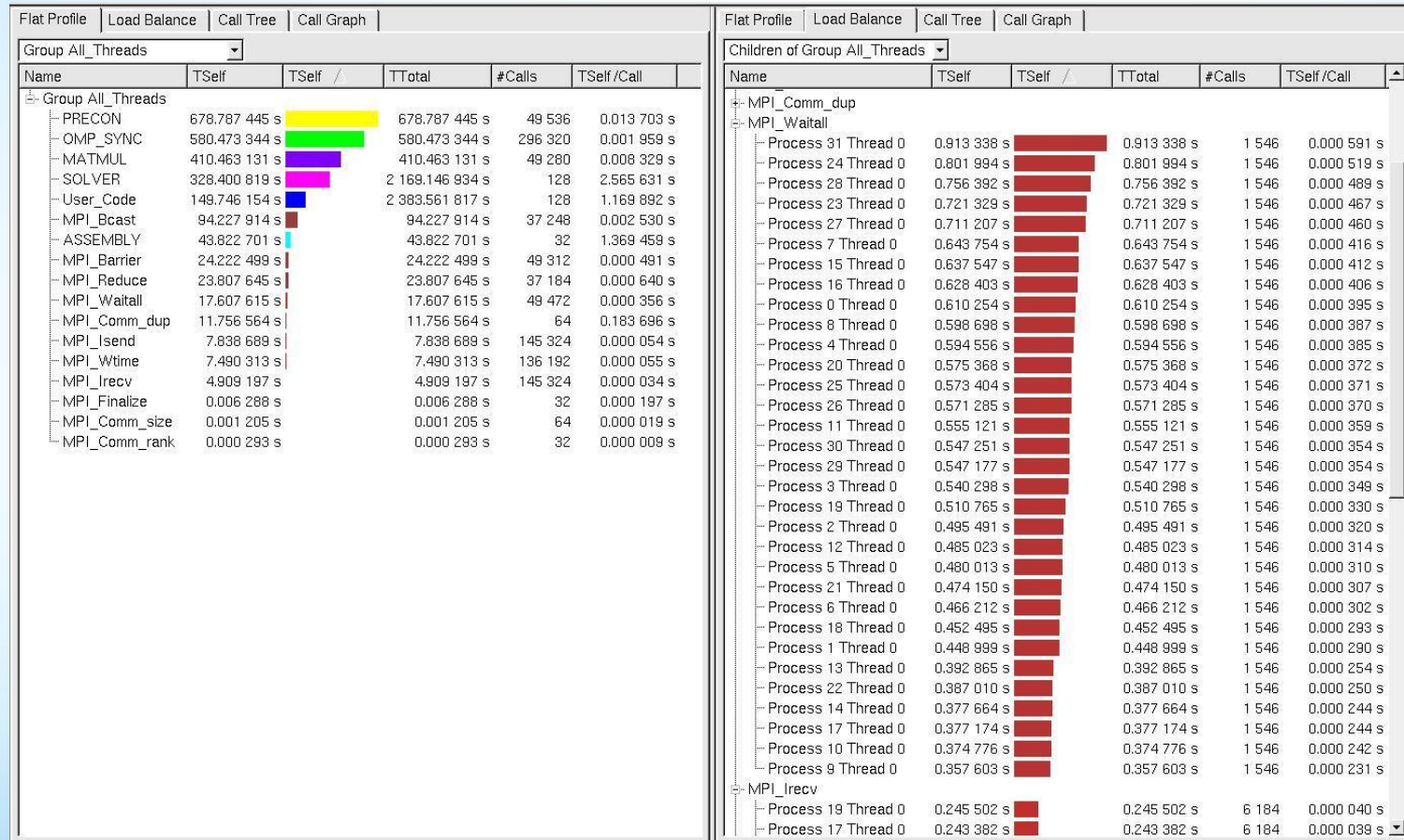
Timelines: Quantitative Timeline

- Get impression on parallelism and load balance
- Show for every function how many threads/processes are currently executing it



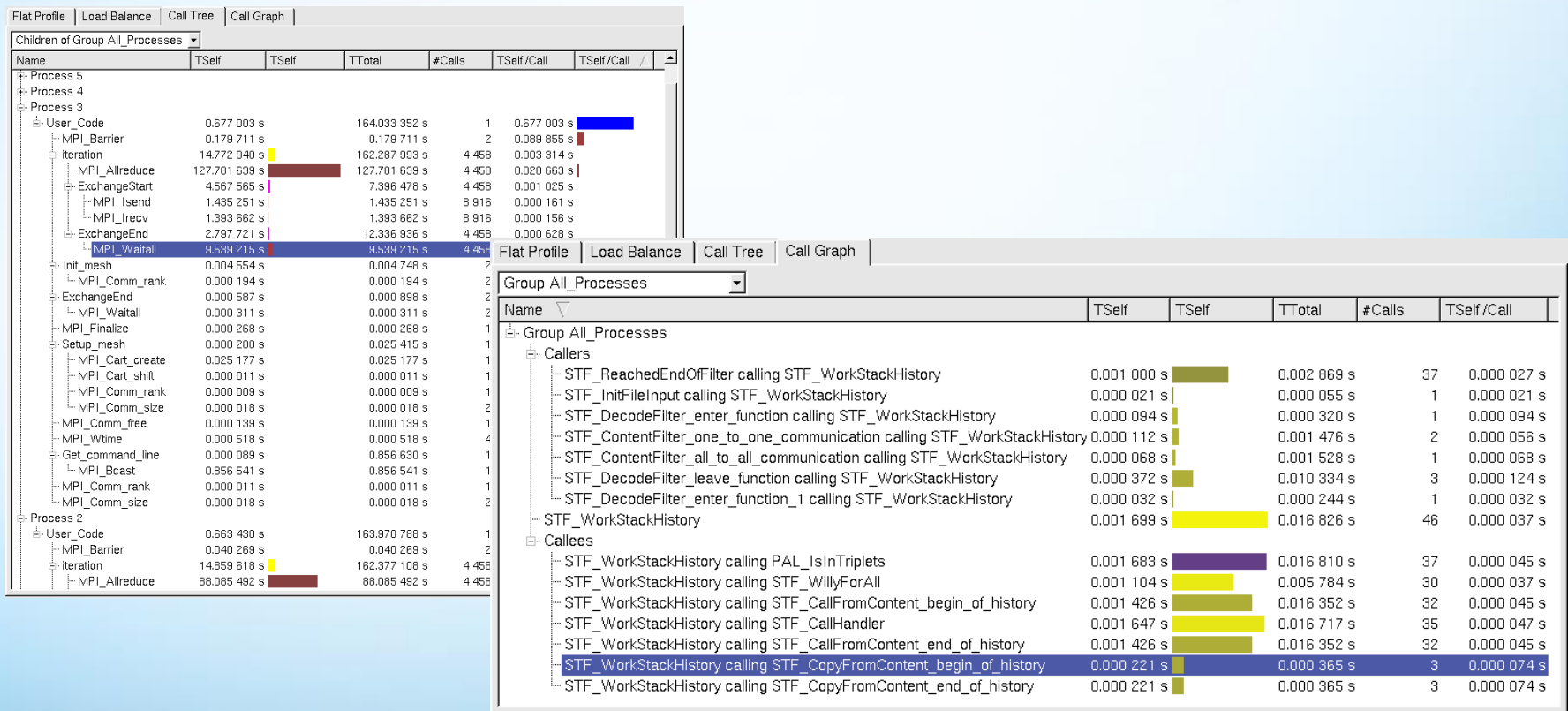
Profiles: Flat Function Profile

- Statistics about functions



Profiles: Call-Tree and Call-Graph

- Function statistics including calling hierarchy
 - Tree: call-stack
 - Graph: calling dependencies



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 msg, 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		45.243						69.152	34.576	10.673
P2		51.590		47.951					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

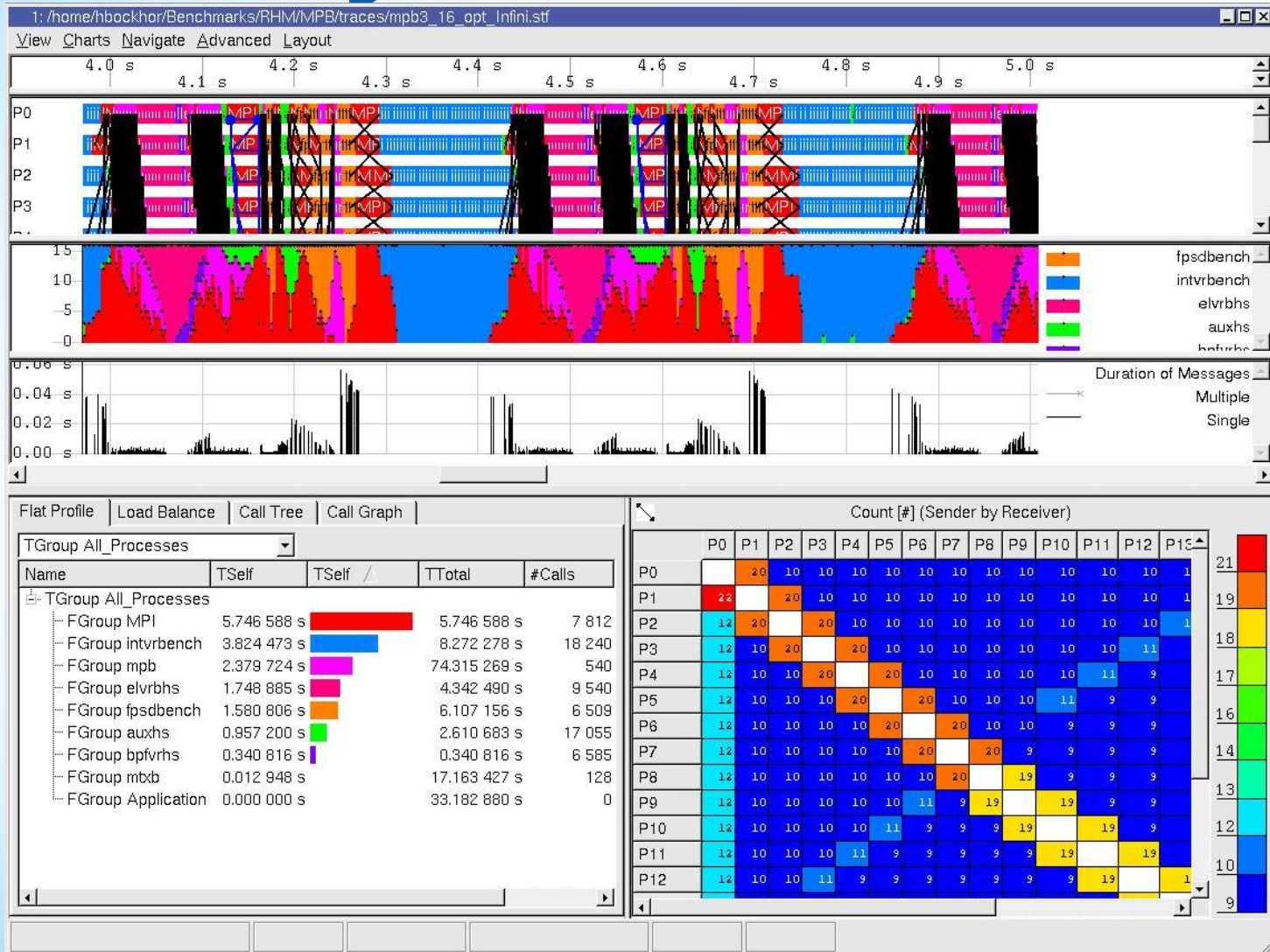
Total Time [s] (Collective Operation by Process)

	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

View

- Helps navigating through the trace data and keep orientation
- Every **View** can contain several **Charts**
- A View on a file is defined by a triplet of
 - time-span
 - set of threads
 - set of functions
- All Charts follow changes to View (e.g. zooming)
- Timelines are correctly aligned along time

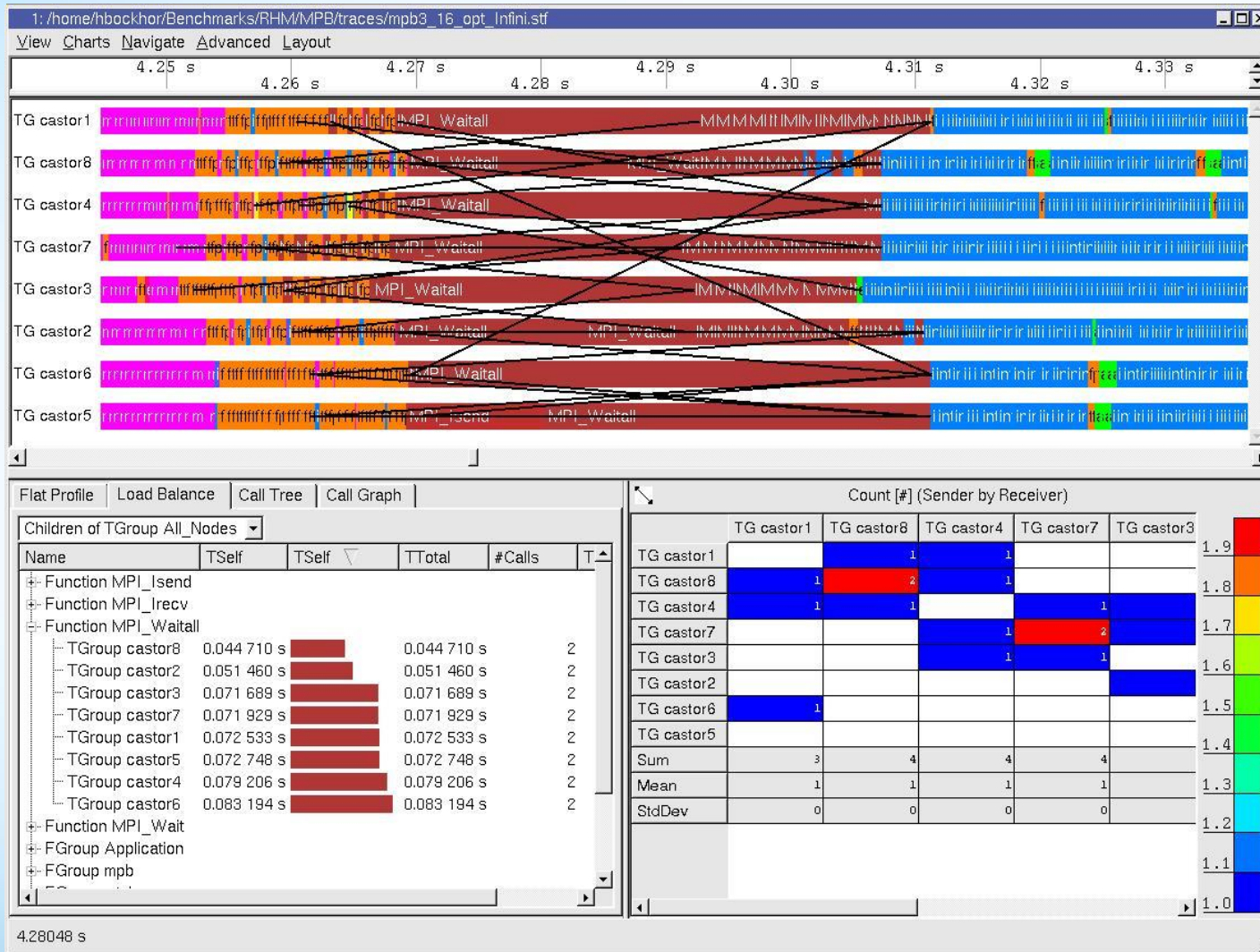
View - zooming



Grouping and Aggregation

- Allow analysis on different levels of detail by aggregating data upon group-definitions
- Functions and threads can be grouped hierarchically
 - Function Groups and Thread 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



mpb3_16_opt_Infini.s

cel Apply

Tagging & 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

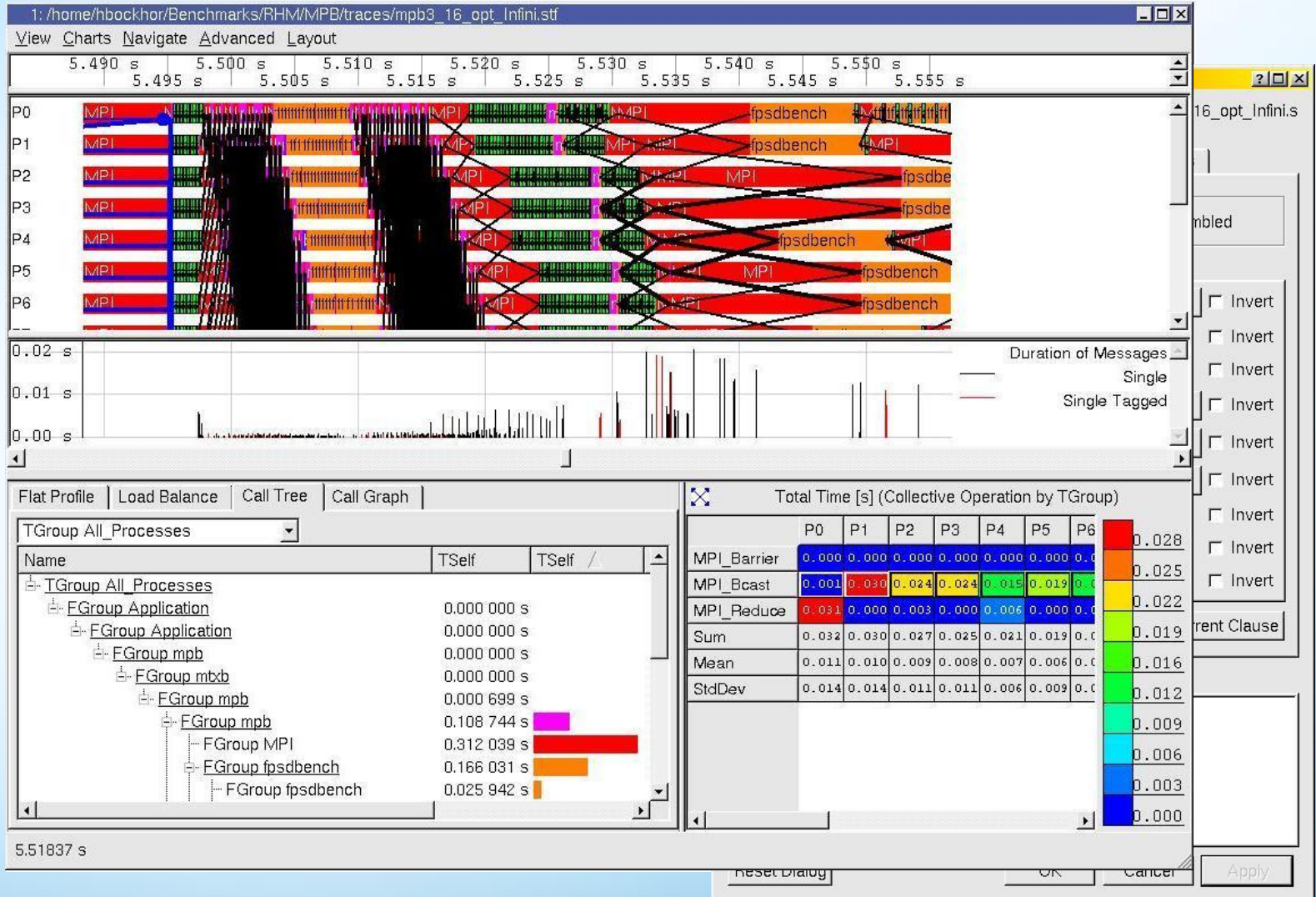
Combine several filters:

Intersection, Union, Complement

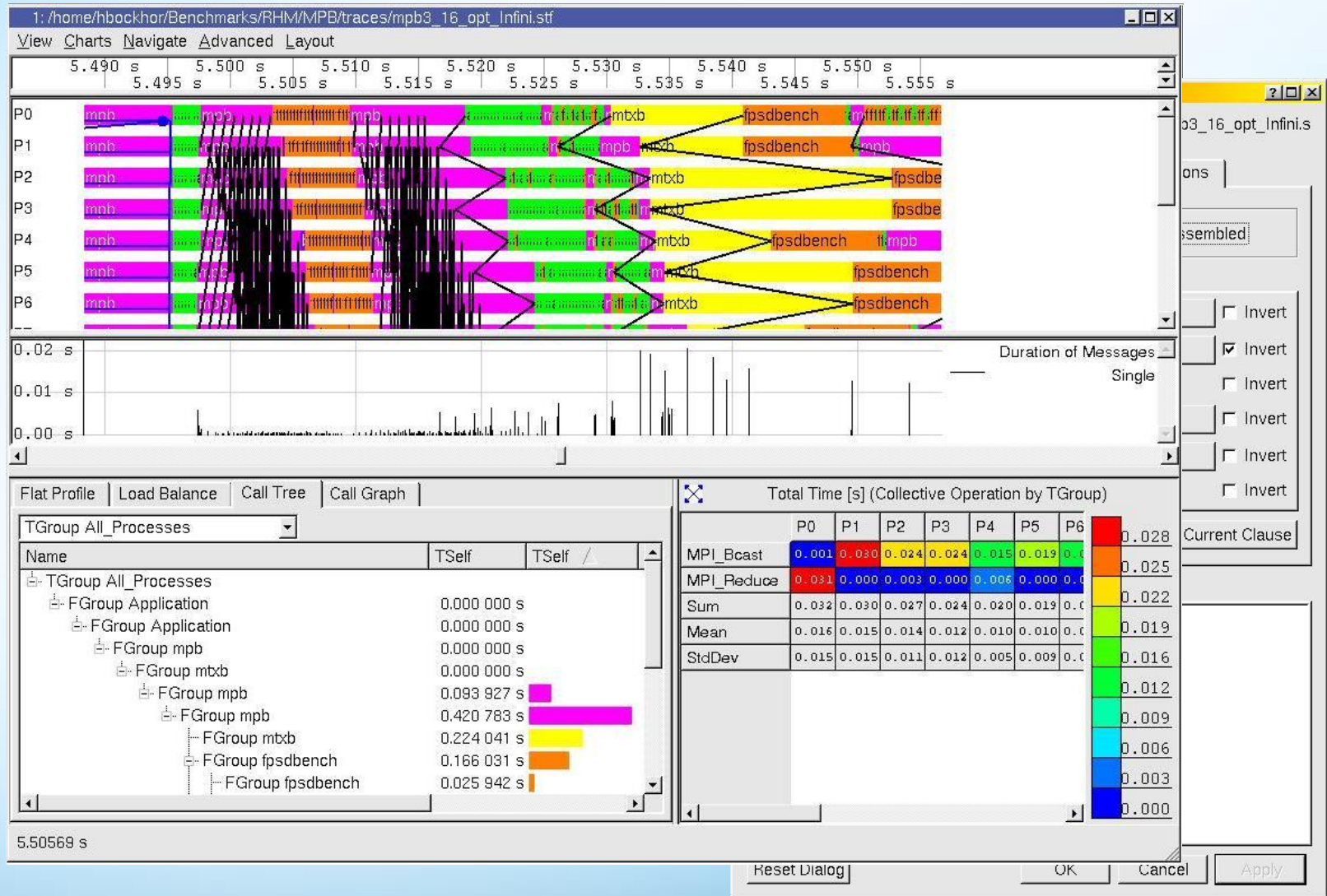
Apply it

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

Tagging Example



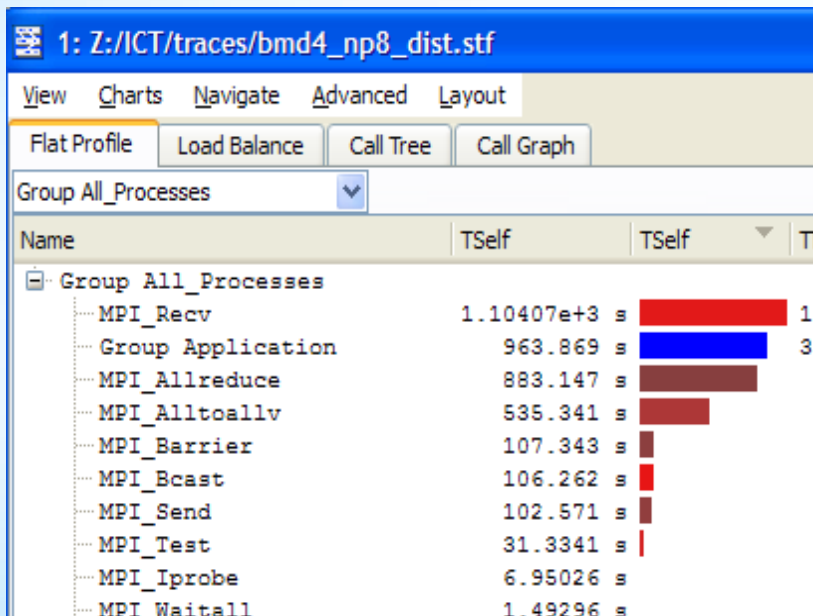
Filtering Example



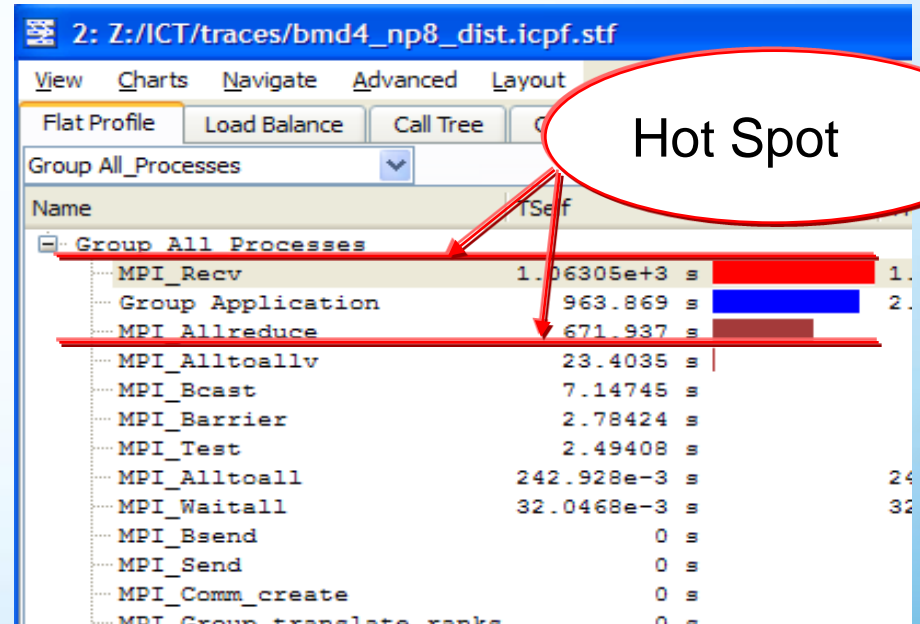
Ideal Interconnect Simulator (Idealizer)

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

Real trace



Ideal trace

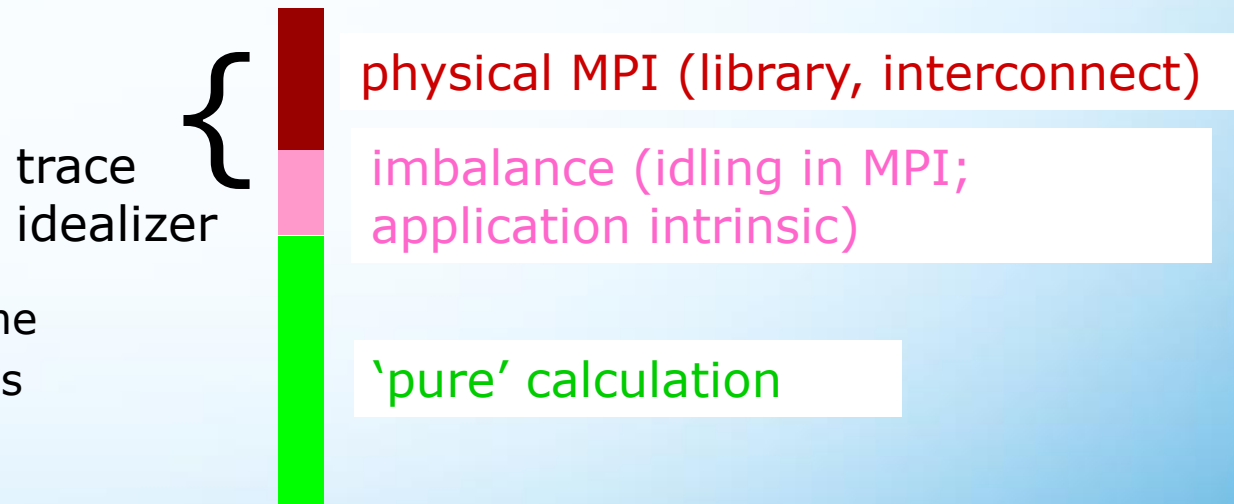


Easy way to identify application bottlenecks

Application Imbalance diagram

- Intuitive diagram for simplified application analysis
- Combined information in one location:
 - Load Imbalance
 - MPI overall time
 - MPI Interconnect time
 - Different Breakdowns etc

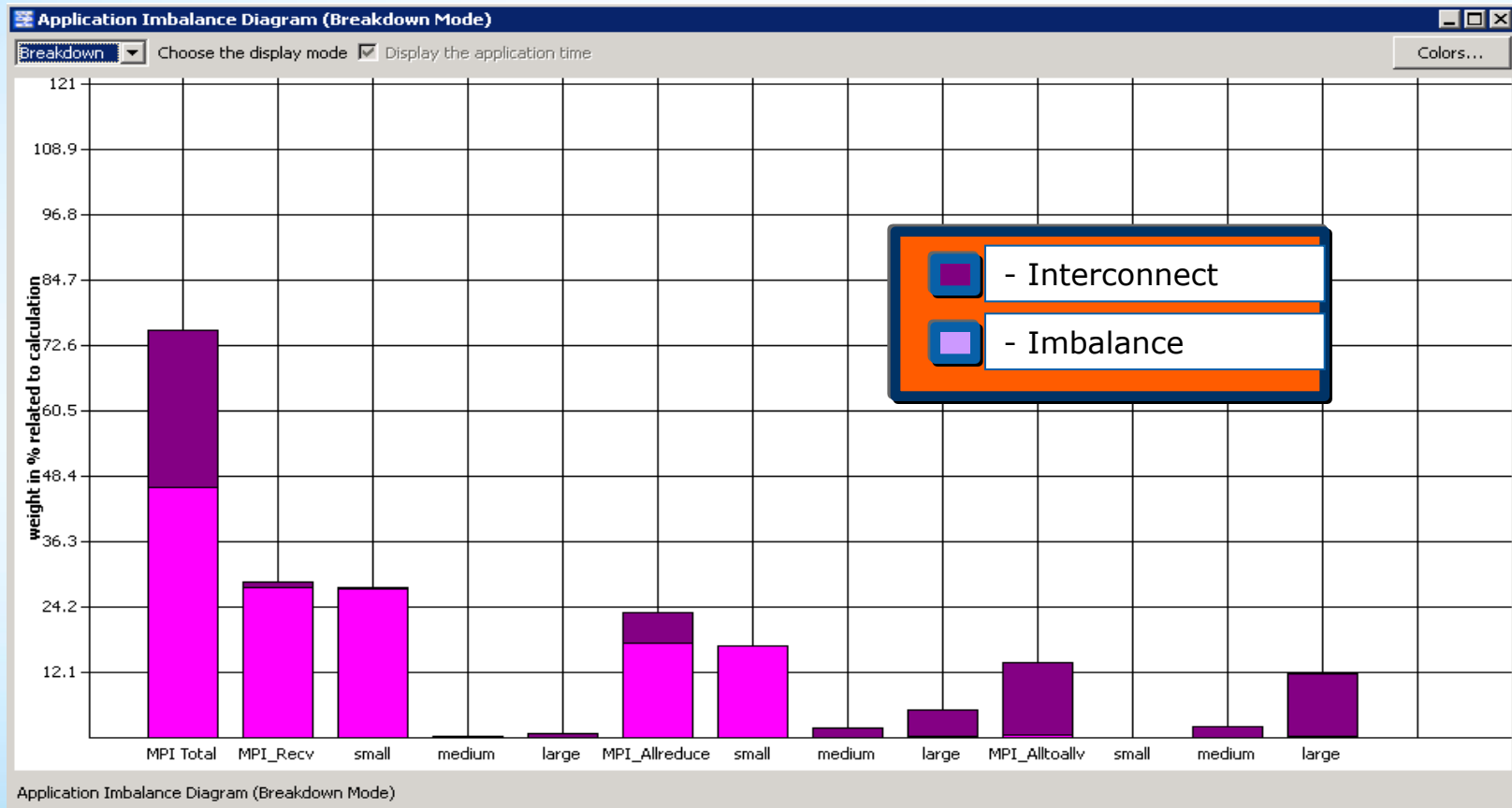
Basic building block: breakdown of a single run time into 3 colors



Simplified application analysis helps to identify performance issues

Application Imbalance diagram (cont.)

Breakdown mode



MPI Correctness Checking: automatically checks MPI correctness

- Solves two problems:

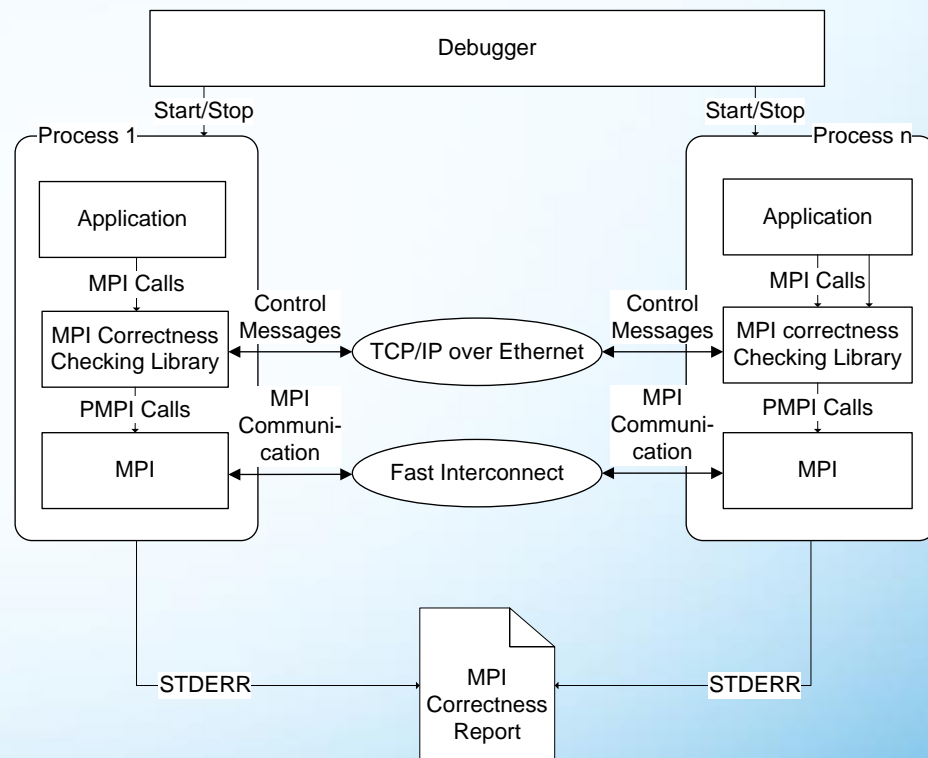
1. Finding programming mistakes in MPI application which need to be fixed by the application developer.
2. Detecting errors in the execution environment.

- Two aspects:

1. *error detection* – done ***automatically*** by the tool
2. *error analysis* – manually by the user based on
 - information provided about an error
 - knowledge of source code, system, ...

MPI Correctness Checking: How it works

- All checks are done at runtime in MPI wrappers.
- Detected problems are reported on stderr immediately in textual format.
- A debugger can be used to investigate the problem at the moment when it is found.



MPI Correctness Checking: Categories of checks

- Local checks: isolated to single process

- Unexpected process termination
- Buffer handling
- Request and data type management
- Parameter errors found by MPI

- Global checks: all processes

- Global checks for collectives and p2p ops
 - Data type mismatches
 - Corrupted data transmission
 - Pending messages
 - Deadlocks (hard & potential)
- Global checks for collectives – one report per operation
 - Operation, size, reduction operation, root mismatch
 - Parameter error
 - Mismatched MPI_Comm_free()

MPI Correctness Checking: Severity of Checks

- Levels of severity:
 - **Warnings:** application can continue
 - **Error:** application can continue but almost certainly not as intended
 - **Fatal error:** application must be aborted
- Some checks may find both warnings and errors
 - Example: CALL_FAILED check due to invalid parameter
 - Invalid parameter in MPI_Send() => msg cannot be sent => **error**
 - Invalid parameter in MPI_Request_free() => resource leak => **warning**

MPI Correctness Checking: Usage (Part I)

- Command line option via `-check_mpi` flag for Intel MPI Library:

```
$ mpirun -check_mpi -n 2 overlap
```

```
[...]
```

```
[0] WARNING: LOCAL:MEMORY:OVERLAP: warning
```

```
[0] WARNING:      New send buffer overlaps with currently active send buffer  
at address 0x7fbffffec10.
```

```
[0] WARNING:      Control over active buffer was transferred to MPI at:
```

```
[0] WARNING:      MPI_Isend(*buf=0x7fbffffec10, count=4, datatype=MPI_INT,  
dest=0, tag=103, comm=COMM_SELF [0], *request=0x508980)
```

```
[0] WARNING:      overlap.c:104
```

```
[0] WARNING:      Control over new buffer is about to be transferred to MPI  
at:
```

```
[0] WARNING:      MPI_Isend(*buf=0x7fbffffec10, count=4, datatype=MPI_INT,  
dest=0, tag=104, comm=COMM_SELF [0], *request=0x508984)
```

```
[0] WARNING:      overlap.c:105
```


MPI Correctness Checking: Usage (Part II)

Function		Issue			
Process	Show Source	Time [s]	Type	Level	Description
+ P4		11.909 909	LOCAL:MPI:CALL_FAILED	warning	Null MPI_Request

Warnings indicate potential problems that could cause unexpected behavior (e.g., incomplete message requests, overwriting a send/receive buffer, potential deadlock, etc.).

Errors indicate problems that violate the MPI standard or definitely cause behavior not intended by the programmer (e.g., incomplete collectives, API errors, corrupting a send/receive buffer, deadlock, etc.).

```
Source View: CCR in Process 1
View: 1: C:/Work/development/ITA/main/Traces/mcerrorhandlingsuppre
Chart:3: Event Timeline

Process 1
058         } else {
059             MPI_Isend( &send, 1, MPI_CH
060             MPI_Isend( &send, 1, MPI_CH
061             MPI_Waitall( 2, reqs, statu
062         }
063     }
064 }
065
066 MPI_Barrier( MPI_COMM_WORLD );
067
068 /* warning: free an invalid request */
069 req = MPI_REQUEST_NULL;
070 MPI_Request_free( &req );
071
072 MPI_Barrier( MPI_COMM_WORLD );
```

Function		Issue			
Process	Show Source	Time [s]	Type	Level	Description
+ P1		13.109 900	GLOBAL:MSG:DATATYPE:MISMATCH	error	Datatype signature mismatch.

MPI Correctness Checking: Debugger Integration

- Debugger must be in control of application before error is found.
- A breakpoint must be set in `MessageCheckingBreakpoint()`

Can be done automatically by configuring the debugger, instructions for TotalView, gdb and idb contained in documentation.

MPI Correctness Checking: Usage of Debugger

python<overlap>.0

File Edit View Group Process Thread Action Point Tools Window Help

Group (Control) Go Halt Kill Restart Next Step Out Run To

Rank 0: python<overlap>.0 (At Breakpoint 1)

Thread 1 (At Breakpoint 1)

Stack Trace	Stack Frame
MessageCheckingBreakpoint, FP=7fbffffdfc0	Function "main":
VT_CheckErrorImpl, FP=7fbffffe1d0	argc: 1
VT_CheckErrorArray, FP=7fbffffe1f0	argv: 0x7fbffffedb8 -> 0x7fbffff0dd ->
VT_CheckOverlapActive, FP=7fbffffe840	Block "\$b1":
VT_CheckOverlapAddReq, FP=7fbffffe8d0	statuses: (MPI_Status[4])
VT_CheckOverlapAdd, FP=7fbffffe940	Local variables:
MPI_..., FP=7fbffffea20	datatype: 0
MPI_..., FP=7fbffffea90	blocklens: (int[10])
main, FP=7fbffffe90	indices: (MPI_..., [10])
main, FP=7fbffffed90	oldtypes: (MPI_..., [10])
main, FP=7fbffffeda0	ints: (int[10])
main, FP=7fbffffeda0	rank: (int)

Action Points | Processes | Threads

STOP 1 MessageCheckingBreakpoint MessageCheckingBreakpoint

error detected, process stopped at breakpoint

access to MPI parameter interface wrapper

full access to application source code and data

Tune MPI Apps Single Node Threading

Intel® VTune™ Amplifier XE Performance Profiler

- Launch Intel® VTune™ Amplifier XE
 - Use mpirun or mpiexec
 - List your app as a parameter
- Results organized by MPI rank
- Review results
 - Graphical user interface
 - Command line report

Tune for Scalable Multicore Performance

Using the Intel® VTune™ Amplifier XE with MPI

- Use the command-line tool under the MPI run scripts to gather report data

```
$ mpirun -n 4 amplxe-cl --result-dir ampl_results -collect hotspots --  
./example.exe
```

- A results directory is created for each MPI rank
 - Can use arg sets to filter on a subset of ranks
- Launch the GUI and view the results for each particular rank

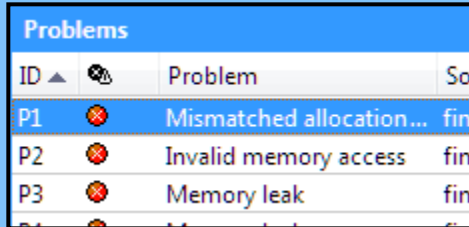
```
$ amplxe-gui ampl_results.<rank#>
```

Scale Efficiently

Intel® Cluster Studio XE correctness tools find errors early in the design cycle

Where are the application's...

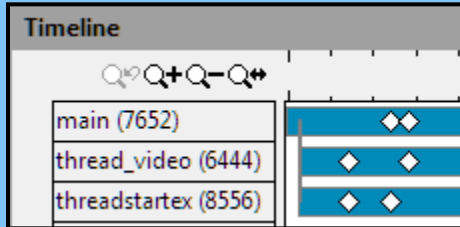
Memory Errors



ID	Problem	So
P1	Mismatched allocation...	fin
P2	Invalid memory access	fin
P3	Memory leak	fin

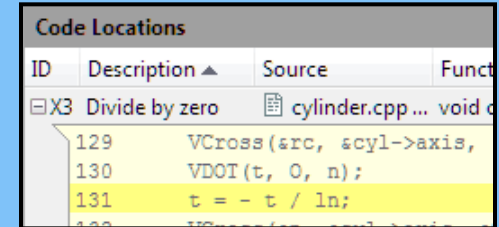
- Invalid Accesses
- Memory Leaks
- Uninitialized Memory Accesses

Threading Errors



- Races
- Deadlocks
- Cross Stack References

Security Errors



```
Code Locations
ID Description Source Funct
X3 Divide by zero cylinder.cpp ... void c
129 VCross(src, scyl->axis,
130 VDOT(t, 0, n);
131 t = - t / ln;
```

- Buffer overflows and underflows
- Incorrect pointer usage
- Over 250 error types...

- MPI aware, cluster friendly
- Both dynamic and static analysis
- Multiple tools - common GUI
- Windows* & Linux*

"Having such a tool this early in the development stage frees the validation from trivial bug reports and gives our engineers the opportunity to code more efficiently from the very beginning of the product cycle."

Jean Kypreos
Advanced Video Processing Team Manager
Envivio

Developer friendly tools help you find errors earlier

Intel® Cluster Studio XE Correctness Tools

Analyze MPI Apps For Memory, Threading and Security Errors

Dynamic Analysis

- Launch Intel® Inspector XE
 - Use mpirun or mpiexec
 - List your app as a parameter
- Results organized by MPI rank
- Review results
 - Graphical user interface
 - Command line report

Static Analysis

- Source analyzed for errors (similar to a build)
- Review results
 - Graphical user interface
 - Command line report

Find errors earlier when they are less expensive to fix.

Using the Intel® Inspector XE with MPI

- Use the command-line tool under the MPI run scripts to gather report data

```
$ mpirun -n 4 inspxe-cl --result-dir insp_results -collect mil --  
./insp_example.exe
```

- A results directory is created for each MPI rank
 - Can use arg sets to filter on a subset of ranks
- Launch the GUI and view the results for each particular rank

```
$ inspxe-gui insp_results.<rank#>
```

Intel® MPI Benchmarks 3.2.4

Overview and What's New

Standard benchmarks with **OSI-compatible CPL license**

- Enables testing of interconnects, systems, and MPI implementations
- Comprehensive set of MPI kernels that provide performance measurements for:
 - Point-to-point message-passing
 - Global data movement and computation routines
 - One-sided communications
 - File I/O

Enhancements:

- Support for the Intel® Xeon Phi™ Coprocessor

The Intel® MPI Benchmarks provide a simple and easy way to measure MPI performance on your cluster

What customers say...

"Intel Trace Analyzer and Collector for Linux helped to drastically improve the performance of RIKEN's molecular dynamics cluster software. We were able to shorten MPI communication time by half by finding and removing bottlenecks with non-blocking communication patterns. Since Intel Trace Analyzer and Collector can embed instrumentation into the program, we can tell the execution time of each function and its load balance, which enabled us to very easily understand where to optimize. Intel's MPI library and Cluster tools provide us the best cluster development environment."

Dr. Takahiro Koishi, Computational Astrophysics Laboratory, RIDEN, Japan.

"Using Intel VTune Amplifier XE makes my work easier and speeds up the development process...it has helped us achieve performance gains from 20% to 360%"

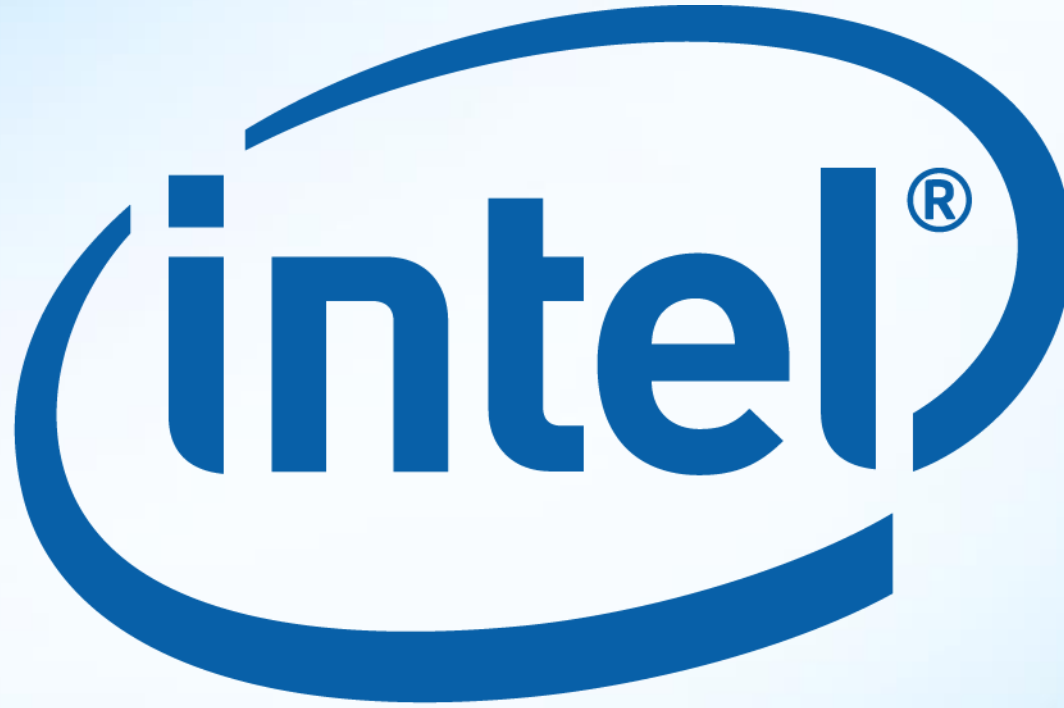
Sergey Zaritchny - Open Cascade SAS

"Intel Inspector XE 2011 is a must-use to craft reliable code in C++. It helped me to quickly localize threading and memory problems in my code, making it easier to fix even the most difficult ones"

Jorge Martinis - Research & Development Engineer, BR&E Inc.

"We're delighted by the efforts of the Intel cluster tools team in helping us scale our applications to 10s of thousands of cores with Intel MPI Library 4.0. and raising the performance bar in providing us with the most scalable commercial MPI library for Intel architecture based processors."

Dr. Daniel Gruner, Chief Technical Officer - Software, University of Toronto



Software