Intel Cluster Tools

Intel® MPI Library

Intel® Trace Analyzer and Collector

intel®

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. https://software.intel.com/en-us/articles/optimization-notice

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. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

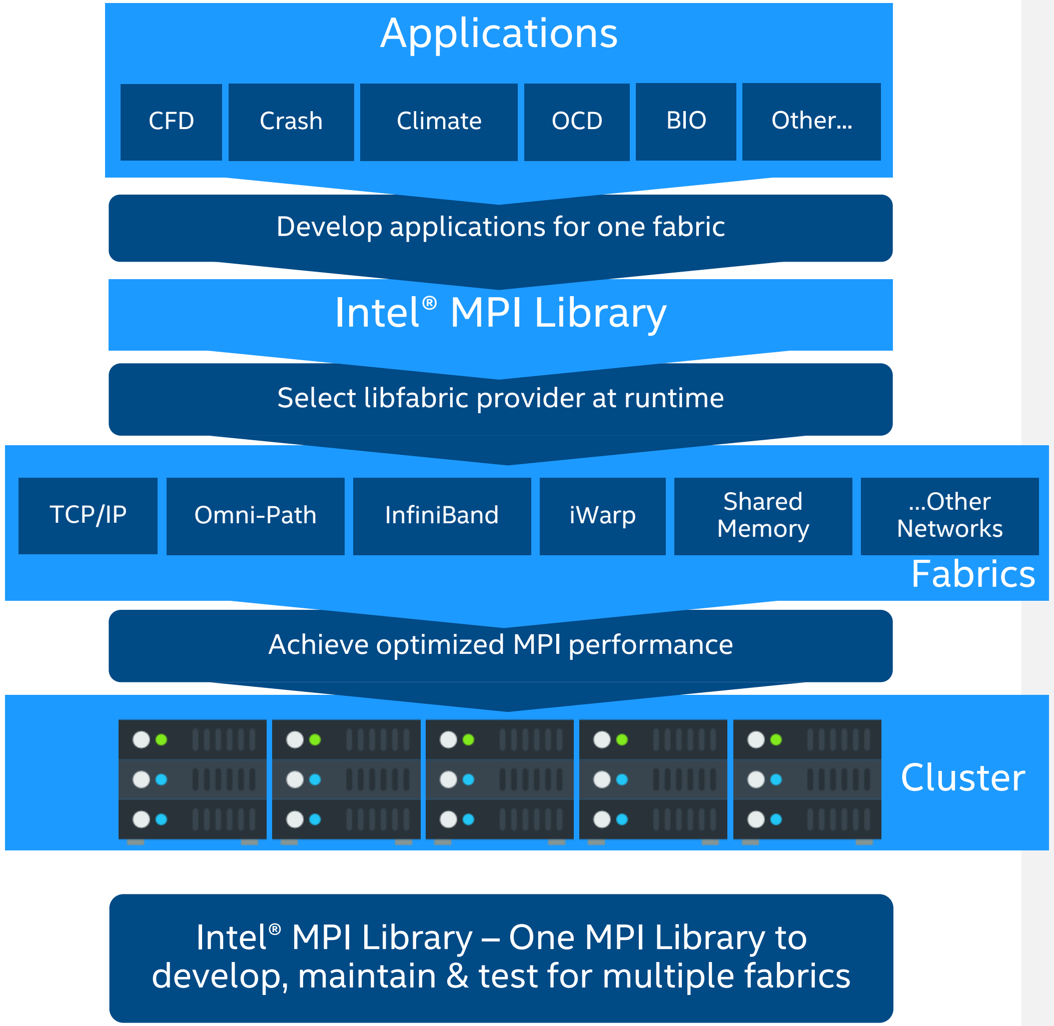
Agenda

- Distributed Performance with Intel® MPI Library
- Tuning MPI Application Performance with Intel® Trace Analyzer and Collector
- Summary and Resources

Intel® MPI Library

Intel® MPI Library Overview

- Optimized MPI application performance
  - Optimized collectives with topology and architecture awareness
- Lower-latency and multi-vendor interoperability
  - Industry leading latency
  - Performance optimized support for the fabric capabilities through OpenFabrics* (OFI) / libfabric
- Sustainable scalability up to 340K cores
  - Efficient path by relying on libfabric
  - New: Faster startup and finalization
- More robust MPI applications
  - Seamless interoperability with Intel® Trace Analyzer and Collector
- Extra Features
  - Conditional numerical reproducibility via I_MPI_CBWR
  - Automatic tuning via I_MPI_TUNING_MODE



Intel® MPI Library Overview

- Streamlined product setup
  - Part of Intel® oneAPI HPC Toolkit or standalone
  - Install as root or as standard user
  - Environment variable script setvars.sh or vars.sh sets paths
- Compilation scripts to handle details
  - One set to use Intel compilers, one set for user-specified compilers
- Environment variables for runtime control
  - I_MPI_* variables control many factors at runtime
    - Process pinning, collective algorithms, device protocols, and more

Compiling MPI Programs

- Compilation scripts automatically passes necessary libraries and options to underlying compiler
  - *mpiifort*, *mpiicpc*, and *mpiicc* use the Intel compiler by default
  - *mpif77*, *mpicxx*, *mpicc*, and others use GNU compiler by default
- Multiple ways to specify underlying compiler
  - I_MPI_F77, I_MPI_CXX, etc. environment variables
  - -f77, -cc, etc. command line options
  - Useful for makefiles portable between MPI implementations
- All compilers are found via PATH

MPI Launcher

- Robust launch command

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

- Options available for:
  - Rank distribution and pinning
  - Fabric selection and control
  - Environment propagation
  - And more

Process Placement

- Layout Across Nodes
  - Default placement puts one rank per core on each node
  - Use –ppn to control processes per node
  - Use a machinefile to define ranks on each node individually
  - Use arguments sets or configuration files for precise control for complex jobs
- Pinning on Node
  - Can pin to single or multiple cores
  - Multiple options for automatic distribution based on resources such as socket, shared cache level, NUMA arrangement
  - See documentation for details:
    - https://software.intel.com/en-us/mpi-developer-reference-linux-process-pinning
    - https://software.intel.com/en-us/mpi-developer-reference-linux-environment-variables-for-process-pinning
    - https://software.intel.com/en-us/mpi-developer-reference-linux-interoperability-with-openmp

GPU Pinning

- I_MPI_OFFLOAD=1 enables GPU features
- I_MPI_OFFLOAD_CELL defines unit of division for offload
  - tile – Single tile/subdevice
  - device – Single device (GPU)
- I_MPI_OFFLOAD_DOMAIN_SIZE sets the number of cells per rank
- I_MPI_OFFLOAD_DEVICES can limit which device numbers to use
- See https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/environment-variable-reference/gpu-support.html for more details

Fabric Control via libfabric

- I_MPI_OFI_PROVIDER chooses provider (select based on interconnect hardware):
  - Default is normally fine
  - tcp – Ethernet
  - psm2 – Intel® Omni-Path Architecture
  - mlx – InfiniBand* (requires at least Intel® MPI Library 2019 Update 5 and UCX 1.4)
  - efa – AWS* EFA (Elastic Fabric Adapter), see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa-start.html for setup process

Conditional Numerical Reproducibility

- I_MPI_CBWR
  - 0 (default) – no reproducibility controls, utilize all optimizations
  - 1 (weak) – disable topology aware optimizations, reproducible across different rank placements/topologies
  - 2 (strict) – disables topology aware optimizations and hardware optimizations, reproducible across hardware and topology
- MPI_Comm_dup_with_info
  - “I_MPI_CBWR”=“yes”, sets strict mode for communicator

Automatic Tuning via Autotuner

- Tuning happens behind the scenes during application run
- Tuning is per communicator
- To tune:
  - I_MPI_TUNING_MODE=auto
  - I_MPI_TUNING_BIN_DUMP=<tuning file> (optional)
- To use tuning results:
  - I_MPI_TUNING_BIN=<tuning file>
- Additional options for more control, see https://software.intel.com/en-us/mpi-developer-reference-linux-autotuning

Debugging MPI Applications

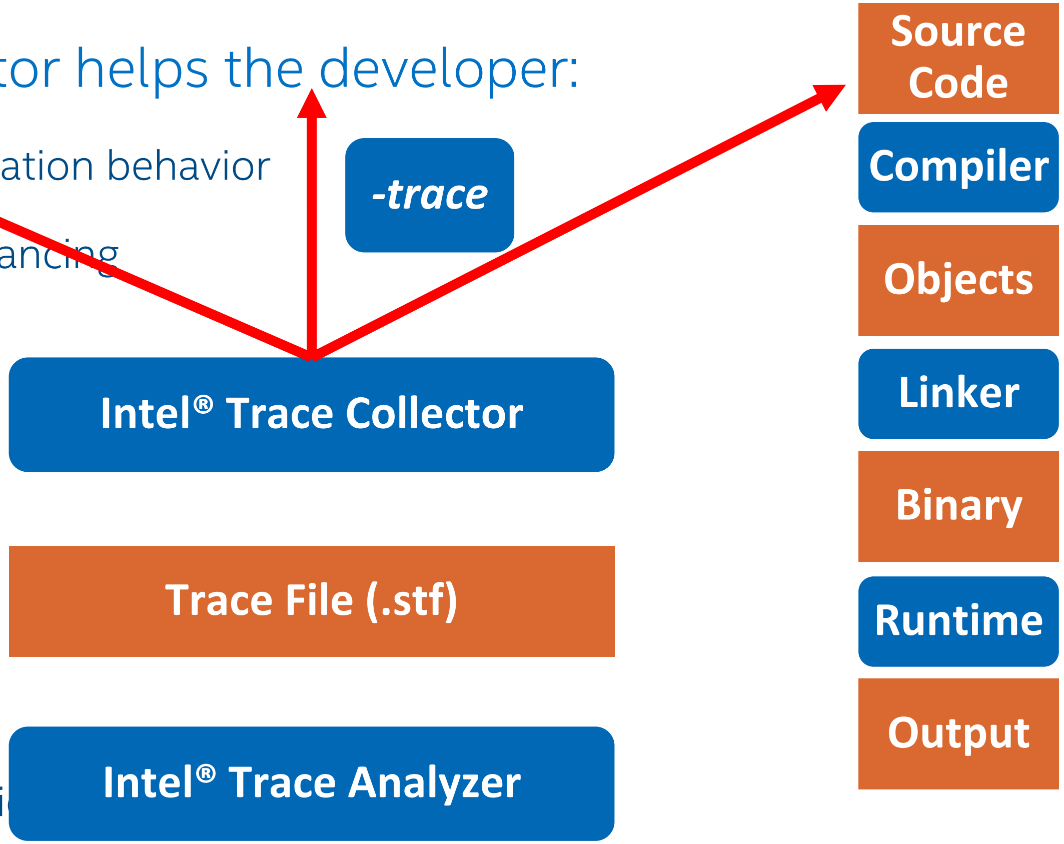
- GDB*
  - mpirun <mpi options> -gdb <application and options>
  - mpirun –n <nranks> -gdba <mpirun pid>
- gtool (https://software.intel.com/en-us/mpi-developer-reference-linux-gtool-options)
  - Set via –gtool option, -gtoolfile option, or I_MPI_GTOOL
  - “<prepend>:<rank set>[=launch mode][@arch]

Intel® Trace Analyzer and Collector

Event-based Tracing for Distributed Applications

Intel® Trace Analyzer and Collector Overview

- Intel® Trace Analyzer and Collector helps the developer:
  - Visualize and understand parallel application behavior
  - Evaluate profilir … balancing
  - Identify communication hotspots
- Features
  - Event-based approach
  - Low overhead
  - Excellent scalability
  - Powerful aggregation and filtering functi
  - Performance Assistance and Imbalance Tuning



Strengths of Event-based Tracing

Predict — Detailed MPI program behavior

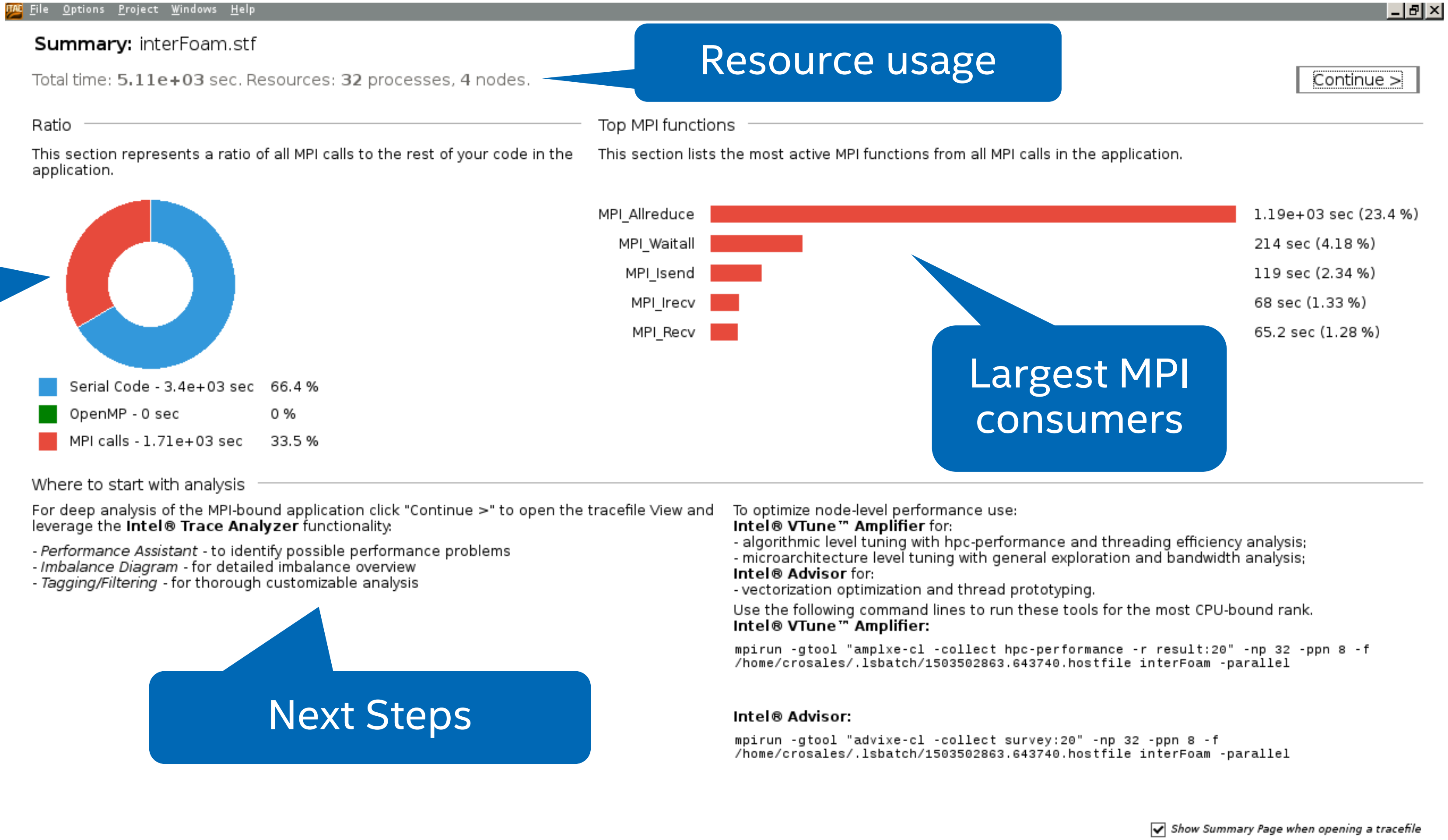
Record — Exact sequence of program states – keep timing consistent

Collect — Collect information about exchange of messages: at what times and in which order

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

Summary page shows computation vs. communication breakdown

Is your application MPI-bound?



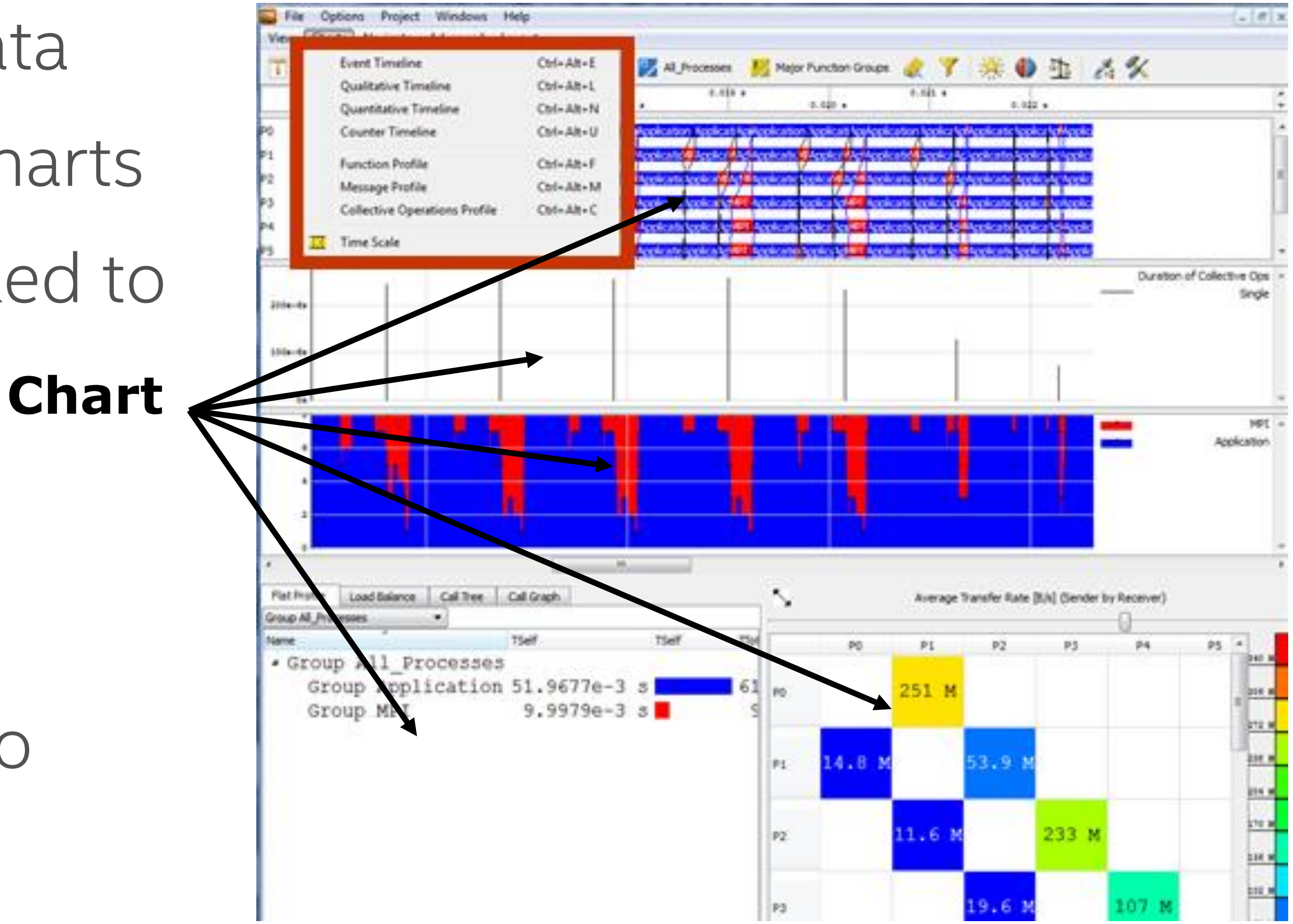
Resource usage

Largest MPI consumers

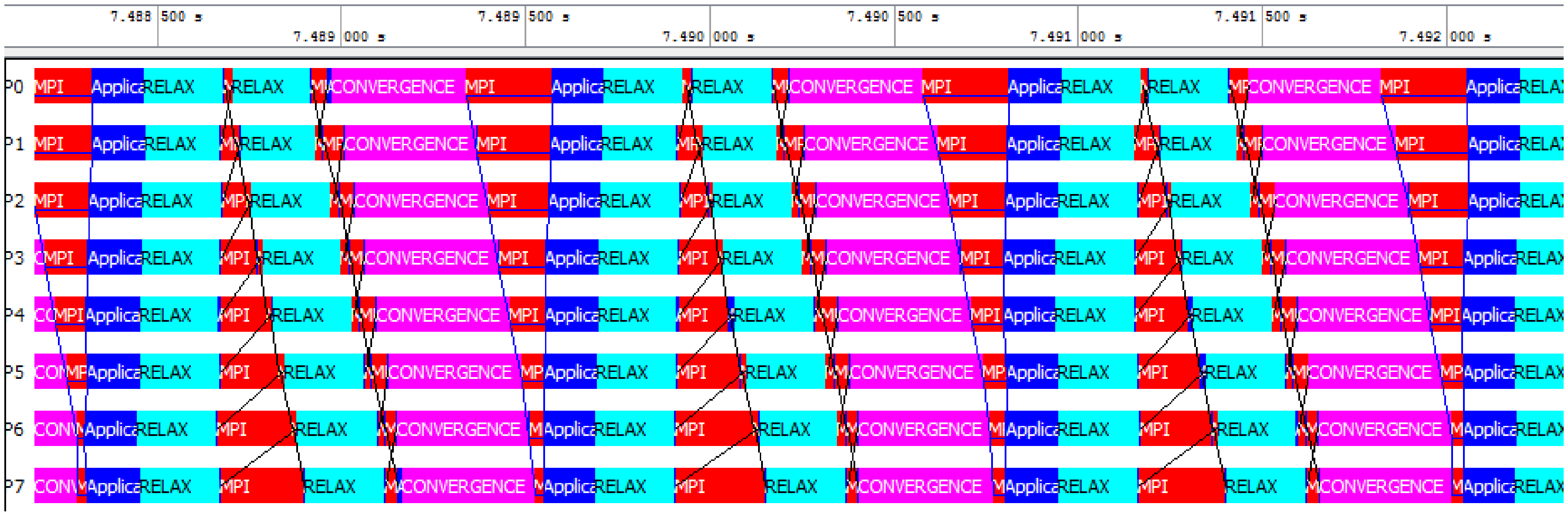
Next Steps

Views and Charts

- Helps navigate the trace data
- A View can show several Charts
- All Charts in a View are linked to a single:
  - time-span
  - set of threads
  - set of functions
- All Charts follow changes to View (e.g. zooming)



Event Timeline



Get detailed impression of program structure

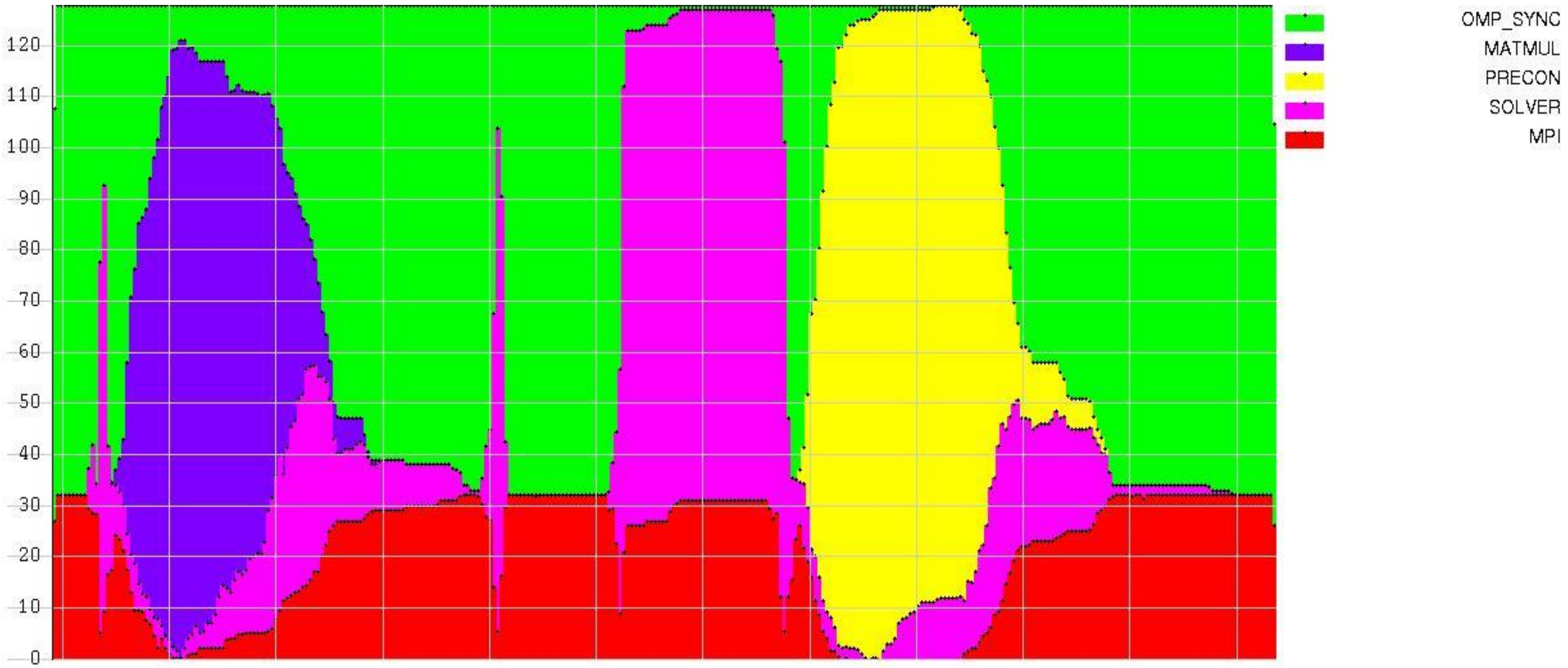
Display functions, messages, and collective operations for each rank/thread along time-axis

Retrieval of detailed event information

Quantitative Timeline

Get impression on parallelism and load balance

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



Flat Function Profile

Statistics about functions

Flat Profile | Load Balance | Call Tree | Call Graph

Group All_Threads

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

Flat Profile | Load Balance | Call Tree | Call Graph

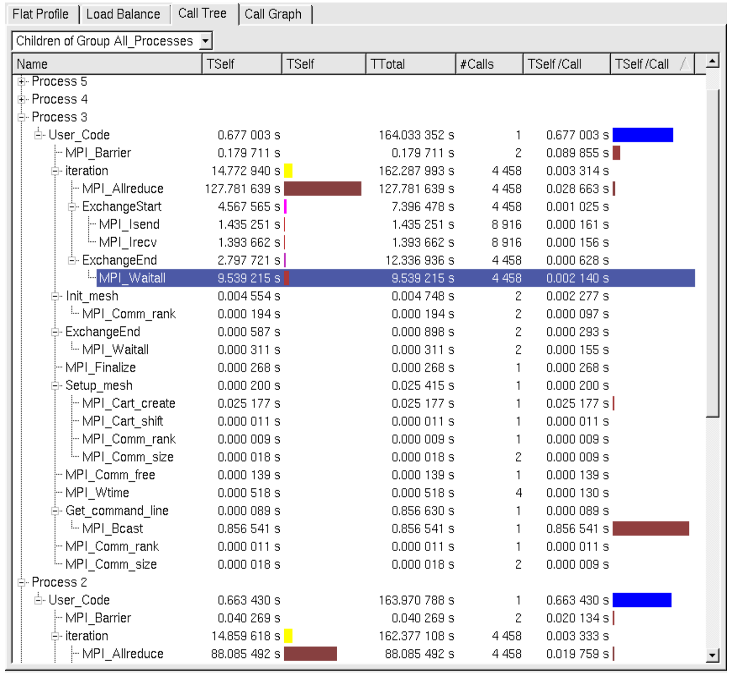
Children of Group All_Threads

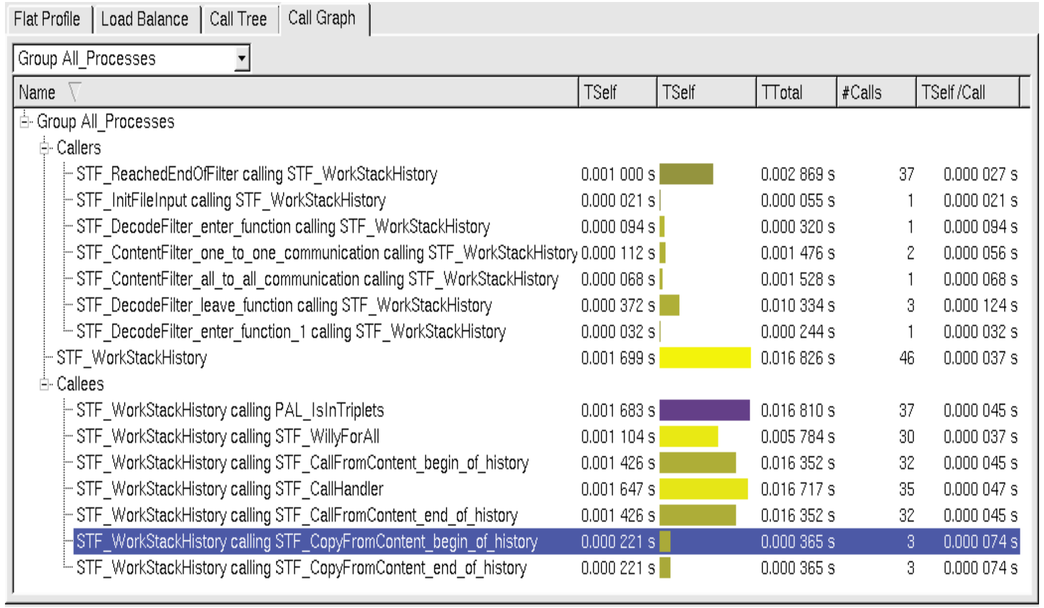
Name	TSelf	TSelf ∕	TTotal	#Calls	TSelf /Call
MPI_Comm_dup					
MPI_Waitall					
Process 31 Thread 0	0.913 338 s		0.913 338 s	1 546	0.000 591 s
Process 24 Thread 0	0.801 994 s		0.801 994 s	1 546	0.000 519 s
Process 28 Thread 0	0.756 392 s		0.756 392 s	1 546	0.000 489 s
Process 23 Thread 0	0.721 329 s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

Call Tree and Call Graph

Function statistics including calling hierarchy

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





Communication Profiles

Statistics about point-to-point or collective communication

Matrix supports grouping by attributes in each dimension

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

Available attributes

- Count, Bytes transferred, Time, Transfer rate

Total Time [s] (Sender by Receiver)

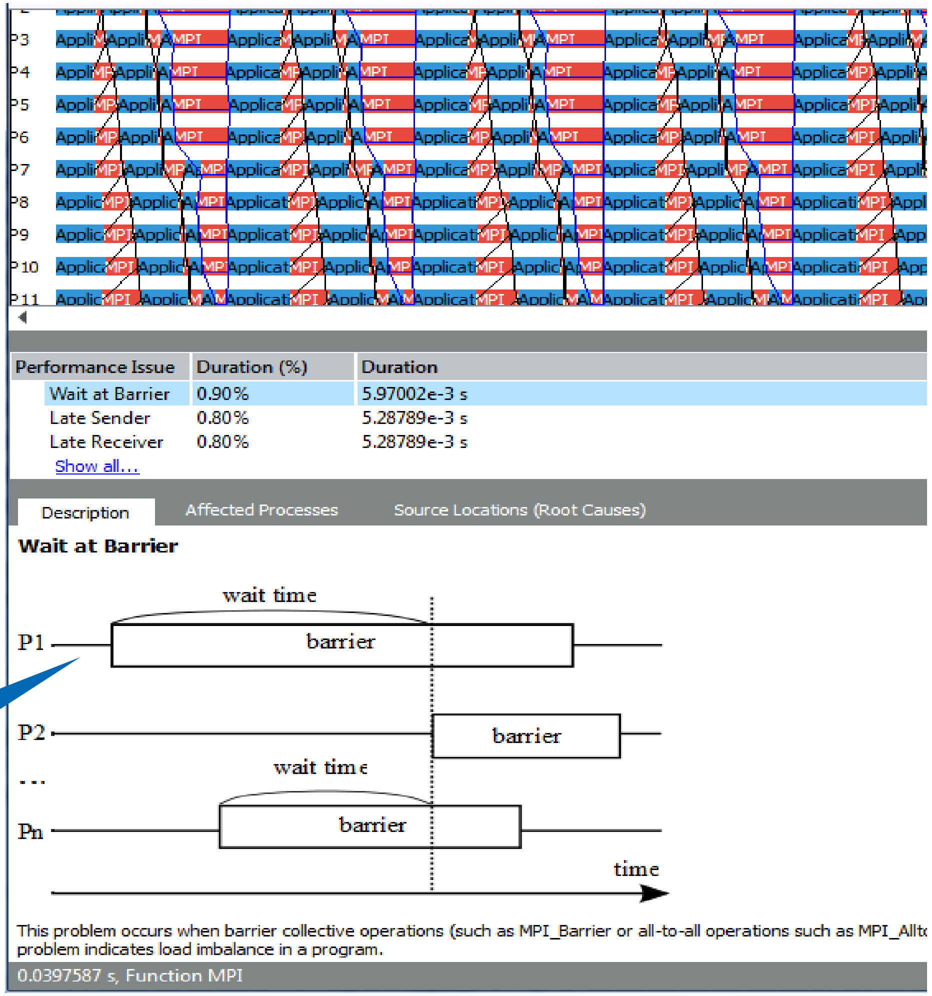
	P0	P1	P2	P3	P4	P5	P6	P7	Sum	Mean	StdDev
P0		74.641							74.641	74.641	0.000
P1	23.903		45.249						69.152	34.576	10.673
P2		51.590		47.961					99.551	49.776	1.814
P3			41.605		36.904				78.509	39.254	2.351
P4				51.558		54.114			105.672	52.836	1.278
P5					37.884		34.262		72.146	36.073	1.811
P6						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

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

MPI Performance Assistant

- Automatic Performance Assistant
- Detect common MPI performance issues
- Automated tips on potential solutions



Automatically detect performance issues and their impact on runtime

Checking MPI Application Correctness

Runtime Correctness Checks

Integration with Debuggers

MPI Correctness Checking

Solves two problems:

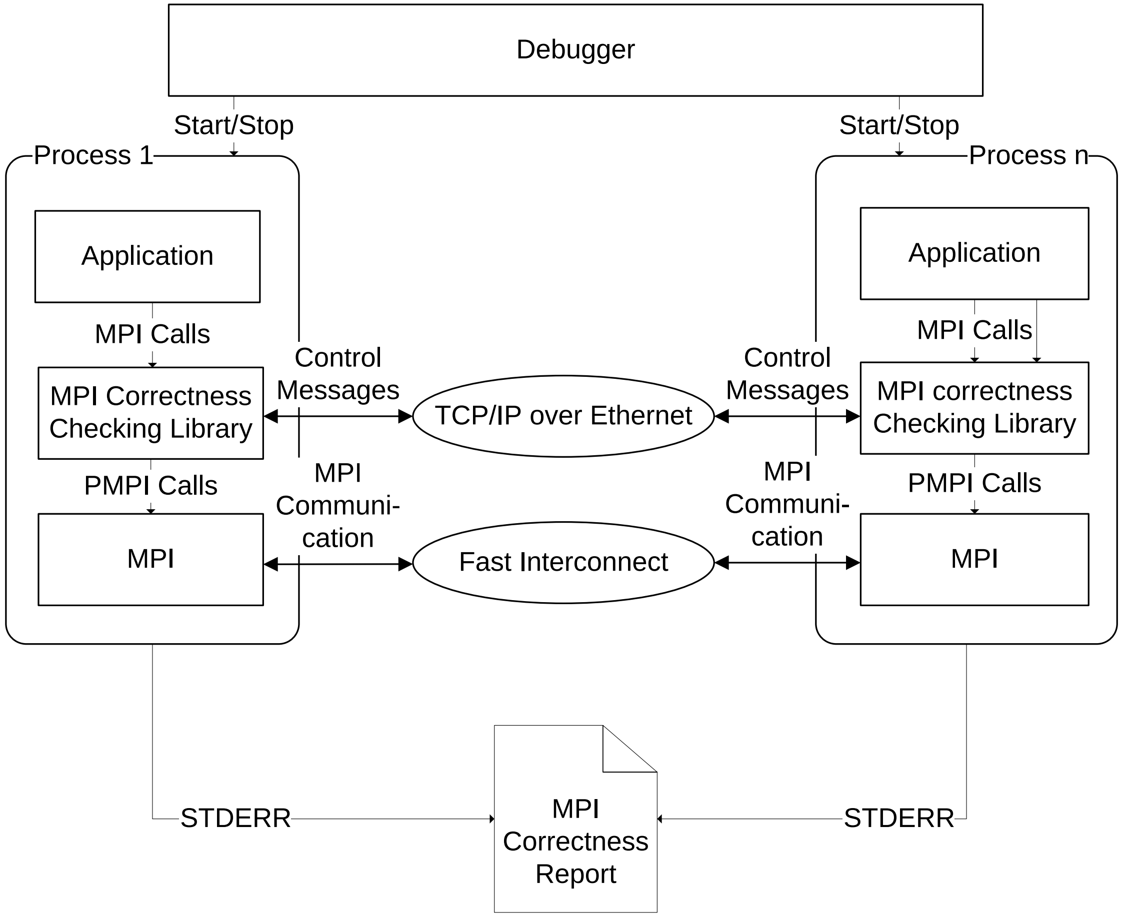
- Finding programming mistakes which need to be fixed by the application developer
- Detecting errors in the execution environment

Two aspects:

- Error Detection – done automatically by the tool
- Error Analysis – manually by the user based on:
  - Information provided about an error
  - Knowledge of source code, system, …

How Correctness Checking 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



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()

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*

Correctness Checking on Command Line

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 0x7fbfffec10.
[0] WARNING:    Control over active buffer was transferred to MPI at:
[0] WARNING:       MPI_Isend(*buf=0x7fbfffec10, 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=0x7fbfffec10, count=4, datatype=MPI_INT, dest=0, tag=104,
comm=COMM_SELF [0], *request=0x508984)
[0] WARNING:       overlap.c:105
```

Correctness Checking in GUI

Enable correctness checking info to be added to the trace file:

- Enable VT_CHECK_TRACING environment variable:

```
$ mpirun –check_mpi –genv VT_CHECK_TRACING on –n 4 ./a.out
```

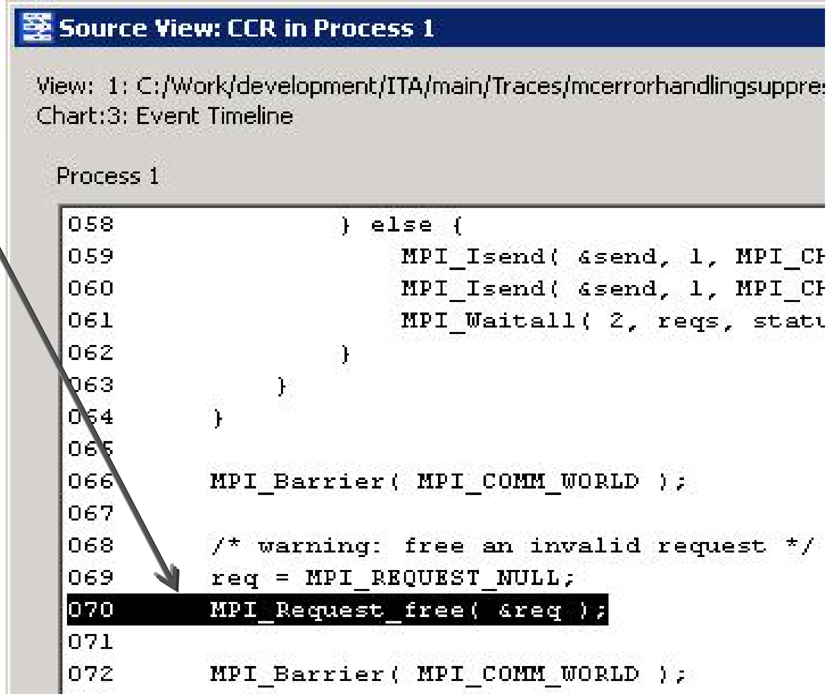
Errors

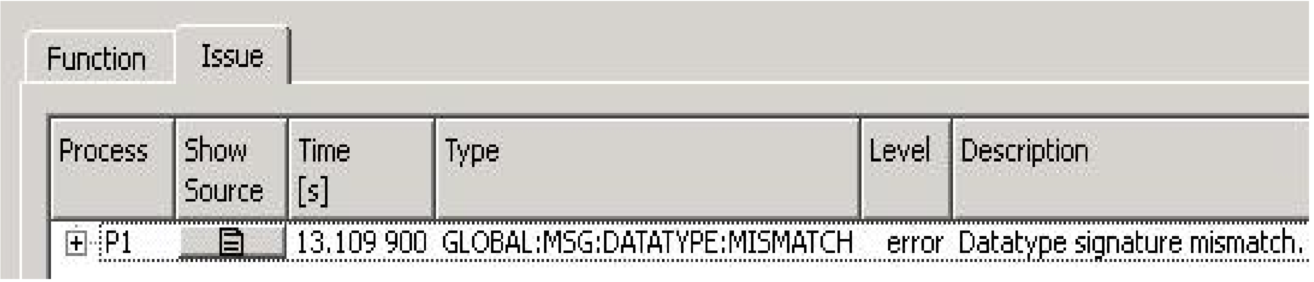
Warnings

Viewing Source Code

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.).





Debugger Integration

Debugger must be in control of application before error is found

A breakpoint must be set in MessageCheckingBreakpoint()

Trace of a Simple MPI Program

Demo

Online Resources

Intel® MPI Library product page

- www.intel.com/go/mpi

Intel® Trace Analyzer and Collector product page

- www.intel.com/go/traceanalyzer

Intel® oneAPI HPC Toolkit Forum

- https://community.intel.com/t5/Intel-oneAPI-HPC-Toolkit/bd-p/oneapi-hpc-toolkit

Intel® MPI Library Tuning Files

- https://software.intel.com/en-us/articles/replacing-tuning-configuration-files-in-intel-mpi-library