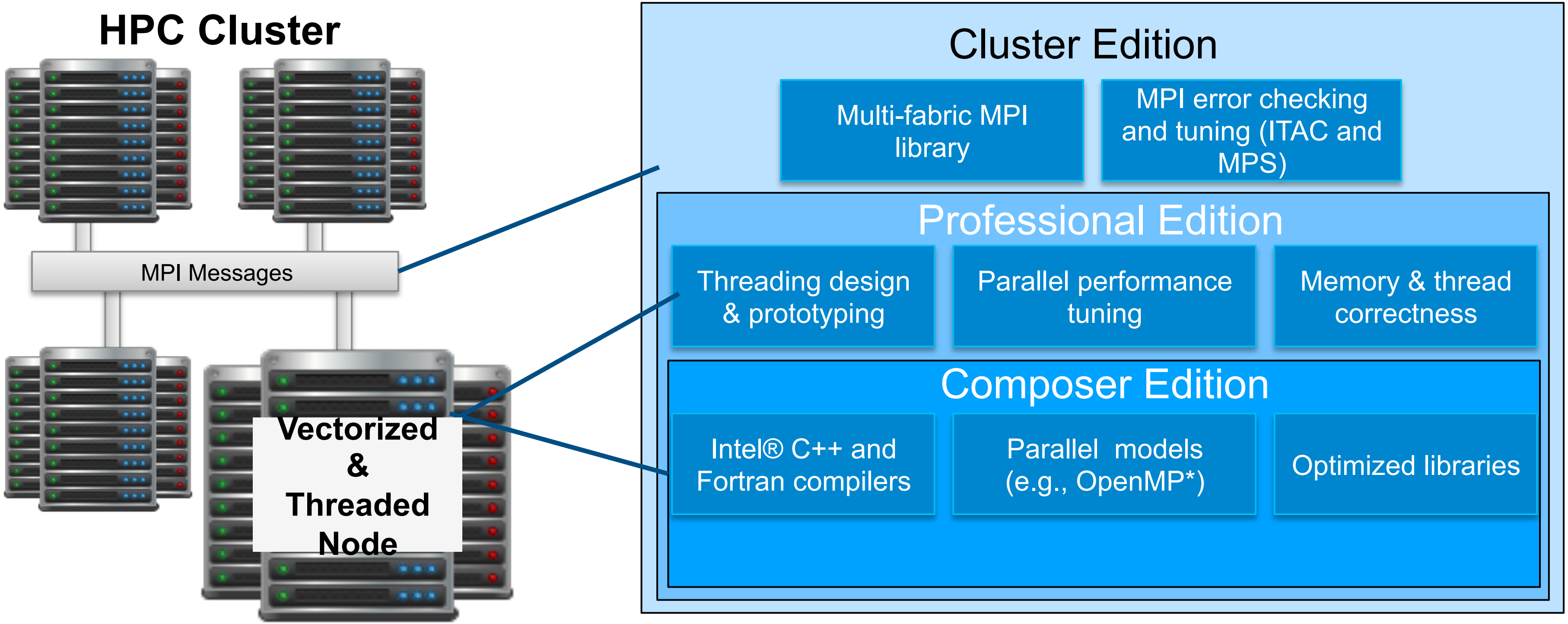
Intel® cluster tools on NERSC systems

For Distributed Performance

Mark Lubin, TCE, SSG/Intel

March 9th 2017

Intel® Parallel Studio XE 2017



Intel® MPI Library Overview

Streamlined product setup

- Install as root, or as standard user
- Environment variable script mpivars.(c)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 adds necessary links to MPI libraries and passes options to underlying compiler
- Use *mpiifort*, *mpiicpc*, or *mpiicc* to force usage of the associated Intel compiler
- Use *mpif77*, *mpicxx*, *mpicc*, or others to allow user to specify compiler (I_MPI_F77, … or –f77=, -cxx=, …)
  – 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
- https://software.intel.com/en-us/intel-mpi-library/documentation

Understanding MPI and Launcher Behavior

I_MPI_DEBUG=<level>

Debug Levels (cumulative):

- 0 – *Default*, no debug information
- 1 – Verbose error diagnostics
- 2 – Fabric selection process
- 3 – Rank, PID, node mapping
- 4 – Process pinning
- 5 – Display Intel® MPI Library environment variables
- 6 – Collective operation algorithm controls

I_MPI_HYDRA_DEBUG=1 turns on Hydra debug output

- Keep in mind that this gives a LOT of output. Only turn on if needed

Fabric Selection

I_MPI_FABRICS=<intranode fabric>:<internode fabric> *or* <fabric>

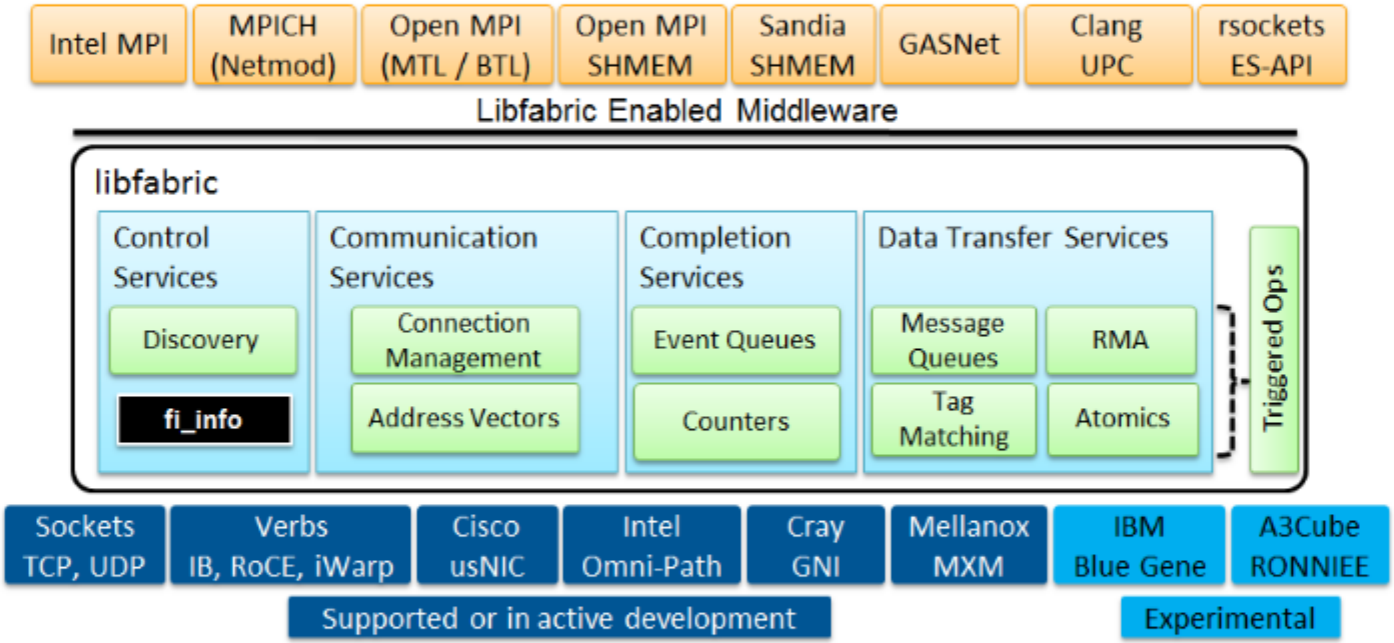
Fabric options

- shm – Shared Memory (only valid for intranode)
- *dapl – Direct Access Provider Library* -- default currently on Cori*
- ofa – Open Fabric Alliance (OFED* verbs)
- tmi – Tag Matching Interface
- tcp – Ethernet/Sockets
- ofi – OpenFabrics Interfaces* -- we advise to use OFI

Default behavior goes through a list to find first working fabric combination

If you specify a fabric, fallback is disabled, I_MPI_FALLBACK=1 to re-enable

Enhanced OFI support



More info: https://ofiwg.github.io/libfabric/

Resources

- A BKM for Working with libfabric* on a Cluster System when using Intel® MPI Library
  - https://software.intel.com/en-us/articles/a-bkm-for-working-with-libfabric-on-a-cluster-system-when-using-intel-mpi-library
- Howard Pritchard (LANL) repositories on OFI
  - https://github.com/hppritcha
  - https://github.com/ofi-cray/libfabric-cray/wiki
  - https://github.com/ofi-cray/libfabric-cray/wiki/Running-Intel-mpi

Using Intel MPI 2017 with OFI libfabric at NERSC

- Load the Intel 17 and Intel MPI 2017 modules, e.g.
  - module load intel
  - module load impi
- set environment variable to tell Intel MPI where to find the libfabric shared library, e.g.
  - % export I_MPI_OFI_LIBRARY=/global/homes/h/hpp/libfabric-v1.0.4rc1_install/lib/libfabric.so
- Specify the location of the SLURM PMI library. For Cori the setting is
  - % export I_MPI_PMI_LIBRARY=/usr/lib64/slurmpmi/libpmi.so
- tell Intel MPI to use OFI libfabric and GNI OFI provider
  - % export I_MPI_FABRICS=ofi
  - % export I_MPI_OFI_PROVIDER=gni

Tuning Methods (for MPI)

Library Tuning (algorithms, fabric parameters)

- Mpitune (next slide)

Application Tuning (load balance, MPI/threaded/serial performance)

- Intel® Trace Analyzer and Collector (ITAC)
- Intel® MPI Performance Snapshot (MPS)
- Intel® VTune™ Amplifier XE

Library 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)

Modes (see mpitune –h for options)

- Cluster-wide tuning

```
mpitune …
```

- Application-specific tuning

```
mpitune –application \"mpirun –n 32 ./exe\" …
```

Creates options settings which are used with the –tune flag

```
mpirun –tune …
```

Intel® Cluster Tools on NERSC clusters – all you need to know

- Intel® Advisor, Intel® Vtune and Intel® Inspector at NERSC:
  - https://www.nersc.gov/users/software/performance-and-debugging-tools/advisor/
  - https://www.nersc.gov/users/software/performance-and-debugging-tools/vtune/
  - https://www.nersc.gov/users/software/performance-and-debugging-tools/inspector/
- Intel® Tools ONLY WORK on Lustre file system at NERSC- please use $SCRATCH!
- Running Intel® Parallel Studio XE Analysis Tools on Clusters with Slurm*/srun
  - https://software.intel.com/en-us/articles/intel-parallel-studio-xe-analysis-tools-on-clusters-with-slurm-srun
  - Next slide in details

Attaching analysis tools, Intel® VTune Amplifier XE, Intel® Inspector XE or Intel® Advisor XE, to *srun*

- However, the following, in case of Vtune, will attach the tool to each MPI rank:
  - $ srun amplxe-cl –c hotspots –r my_result_1 -- ./my_application
- If the user is only interested in analyzing a subset of MPI ranks or shared memory nodes, they can leverage the multiple program configuration from srun.
- Create config file
  - cat > mpmd_vtune.cfg << EOF
  - 0-98   ./my_application
  - 99   amplxe-cl –c hotspots –r my_result_2 -- ./my_application
  - 100-255 ./my_application
  - EOF
- Run
  - srun --multi-prog ./mpmd_vtune.cfg

Intel® Trace Analyzer and Collector (ITAC) and MPI Performance Snapshot (MPS) on NERSC systems

- Intel® Cluster Tools in a Cray* environment
  - https://software.intel.com/en-us/articles/intel-mpi-itac-and-mps-in-a-cray-environment
  - $ module load itac
- ITAC - to collect the trace – just preload ITAC library
  - $export LD_PRELOAD=/path_to_ITAC_installation/intel64/slib/libVT.so
- MPS: preload library and collect statistics
  - $export LD_PRELOAD=/path_to_MPS_installation/intel64/slib/libmps_nopapi.so
  - $export I_MPI_STATS=20
  - $export I_MPI_STATS_COMPACT=1
- Run application
  - $srun –n 8 -c 8 ./app_name

Intel® Trace Analyzer and Collector (ITAC)

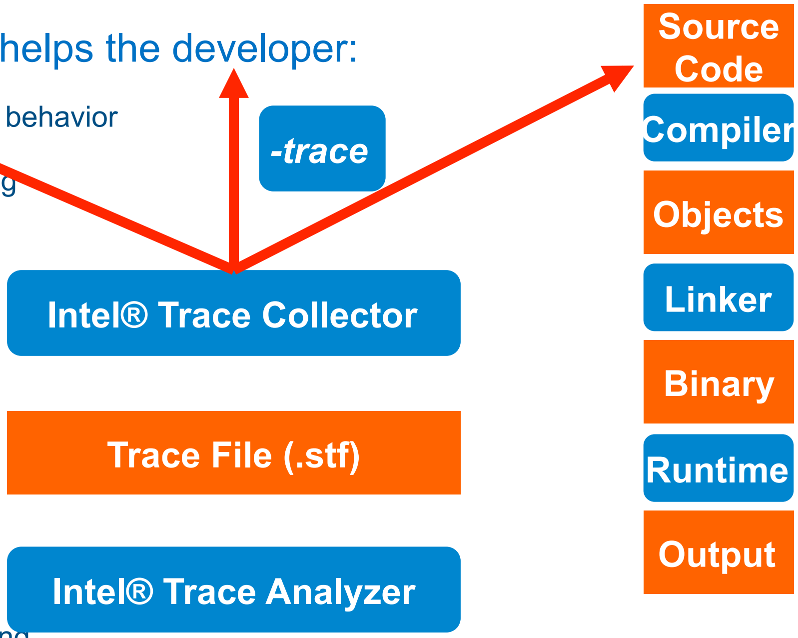
Intel® Trace Analyzer and Collector Overview

Intel® Trace Analyzer and Collector helps the developer:

- Visualize and understand parallel application behavior
- Evaluate profiling s … cing
- Identify communication hotspots

Features

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



Strengths of Event-based Tracing

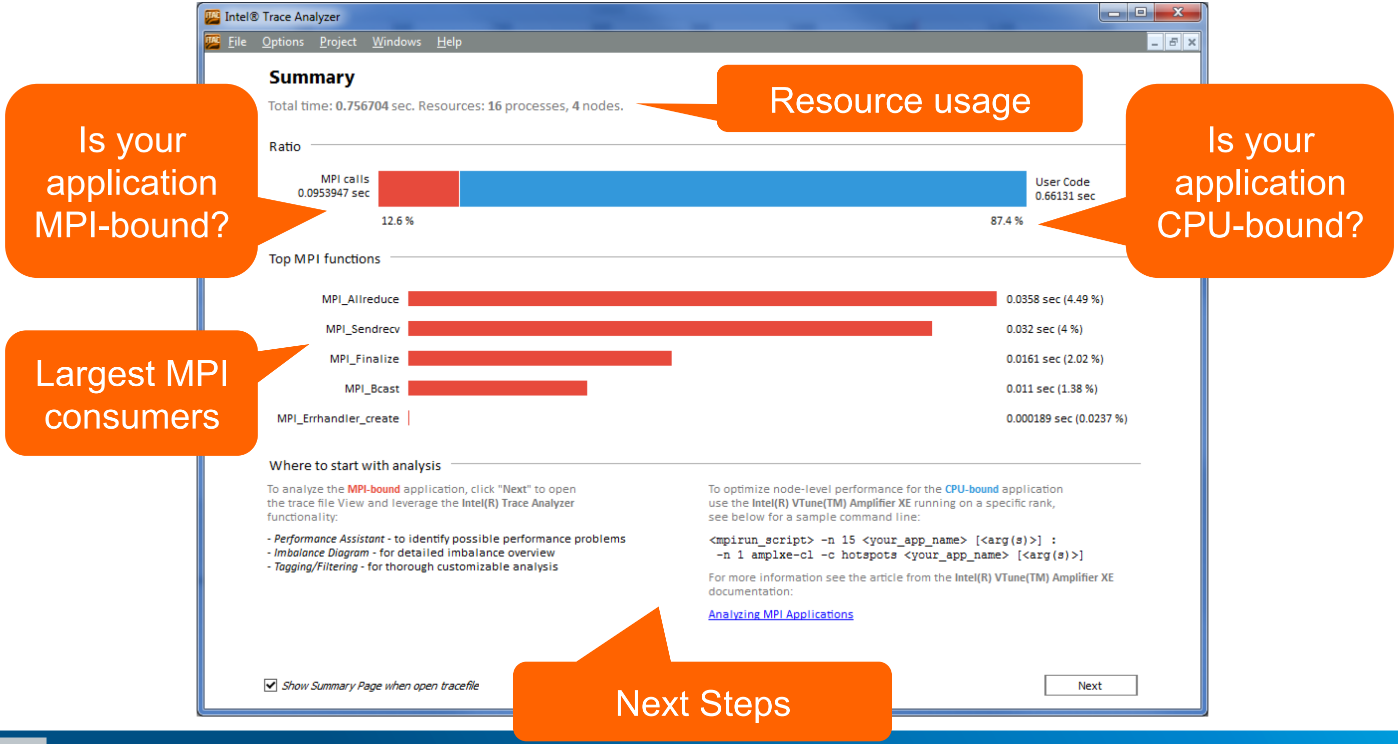
Predict	Detailed MPI program behavior
Record	Exact sequence of program states – keep timing consistent
Collect	Collect information about exchange of messages: at what times and in which order

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

Multiple Methods for Data Collection

Collection Mechanism	Advantages	Disadvantages
Run with –trace or preload trace collector library.	**Automatically collects all MPI calls, requires no modification to source, compile, or link.**	**No user code collection.**
Link with –trace.	Automatically collects all MPI calls.	No user code collection. Must be done at link time.
Compile with –tcollect.	Automatically instruments all function entries/exits.	Requires recompile of code.
Add API calls to source code.	Can selectively instrument desired code sections.	Requires code modification.

Summary page shows computation vs. communication breakdown



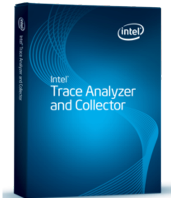
MPI-3.0 Support

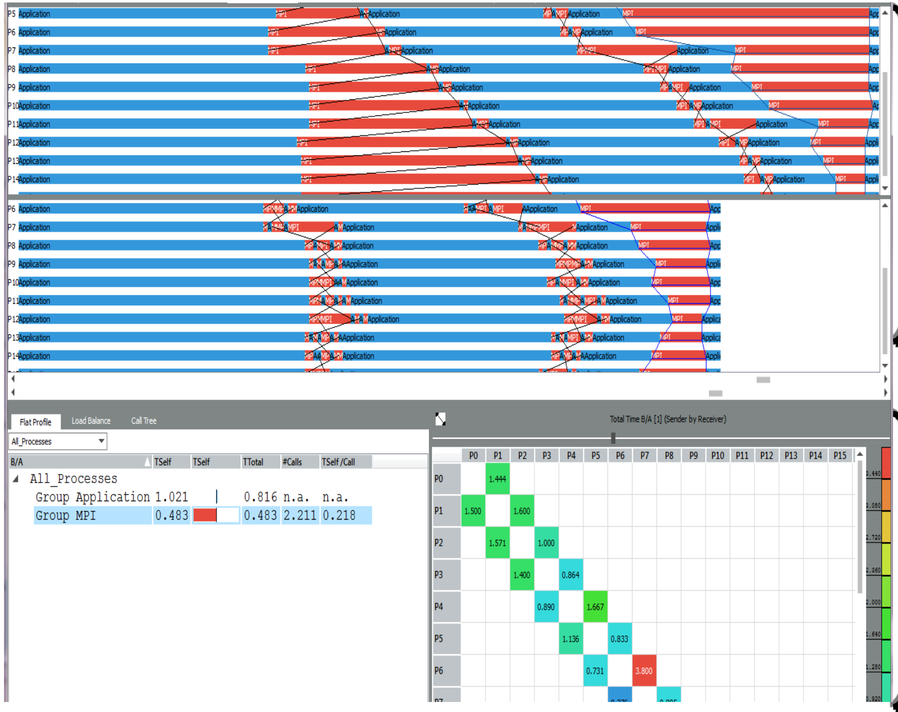
Support for major MPI-3.0 features

- Non-blocking collectives
- Fast RMA
- Large counts

Non-blocking Allreduce (MPI_Iallreduce)

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

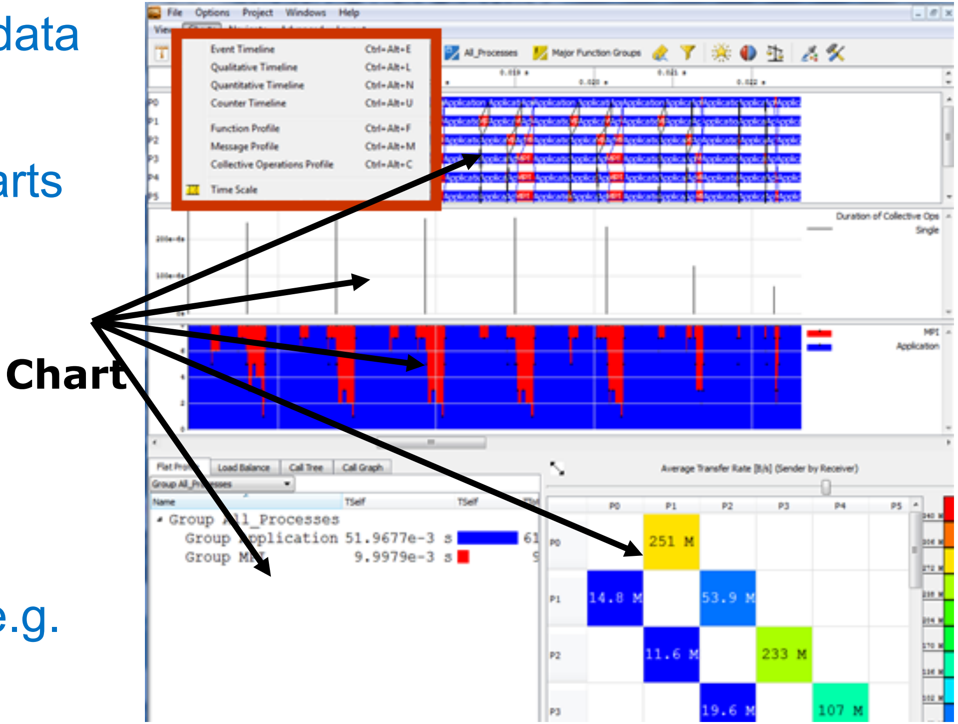
Helps navigating through the trace data and keep orientation

Every View can contain several Charts

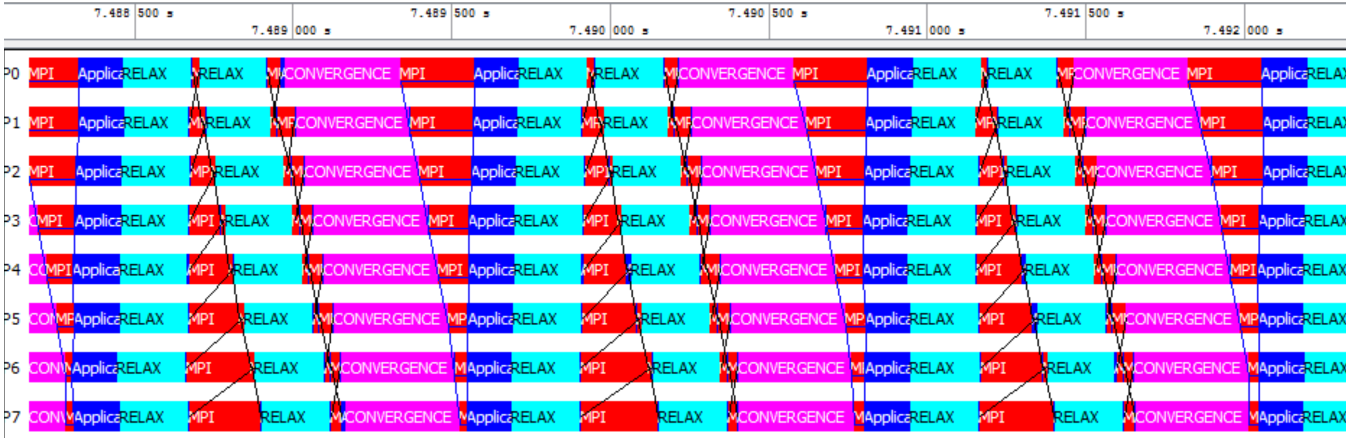
All Charts in a View are linked to a single:

- time-span
- set of threads
- set of functions

All Charts follow changes to View (e.g. zooming)



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

Communication Profiles

Statistics about point-to-point or collective communication

Matrix supports grouping by attributes in each dimension

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

Available attributes

- Count, Bytes transferred, Time, Transfer rate

Total Time [s] (Collective Operation by Process)

	P0	P1	P2	P3	P4	P5	P6	P7	Sum	Mean	StdDev
MPI_Barrier	0.063	0.052	0.040	0.180	0.258	0.066	0.079	0.215	0.952	0.119	0.080
MPI_Bcast	0.000	0.860	0.865	0.857	0.853	0.855	0.860	0.861	6.010	0.751	0.284
MPI_Allreduce	87.299	120.679	88.085	127.782	89.071	124.266	109.330	137.064	883.576	110.447	18.704
Sum	87.362	121.590	88.990	128.818	90.182	125.187	110.268	138.141	890.538		
Mean	29.121	40.530	29.663	42.939	30.061	41.729	36.756	46.047		37.106	
StdDev	41.139	56.675	41.312	59.993	41.727	58.363	51.318	64.359			52.973

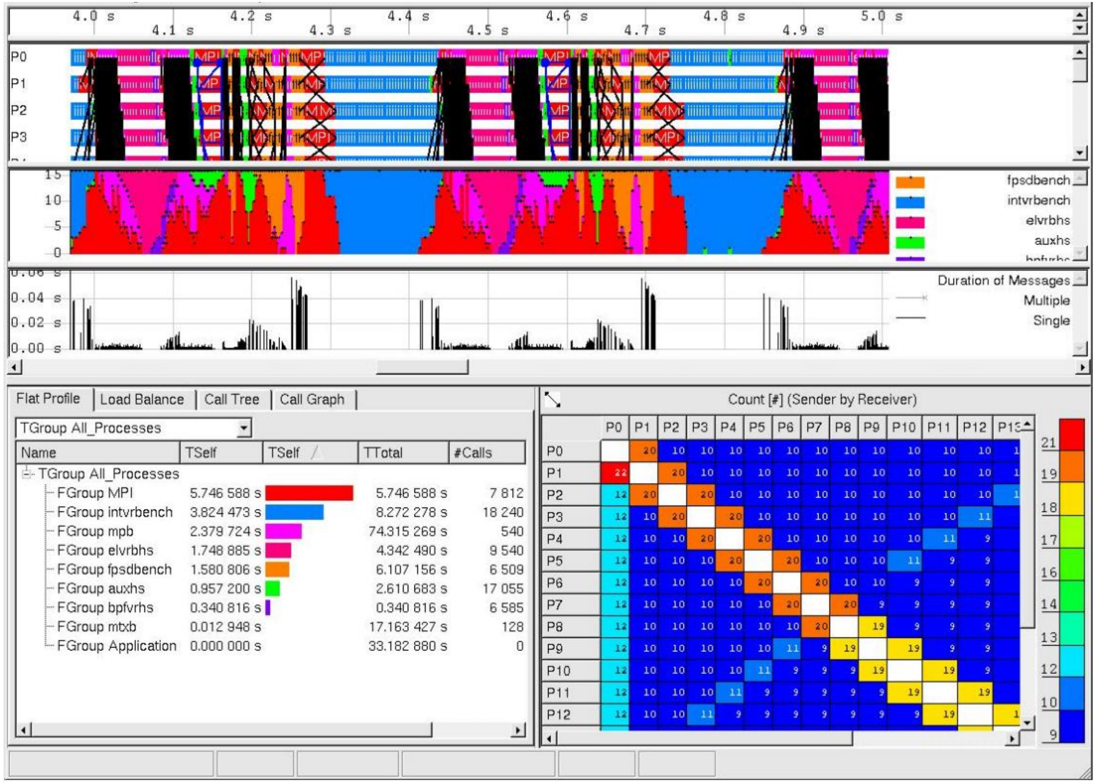
Color scale: 123, 110, 96, 82, 69, 55, 41, 27, 14, 0

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

Color scale: 70, 64, 59, 54, 49, 44, 39, 34, 29, 24

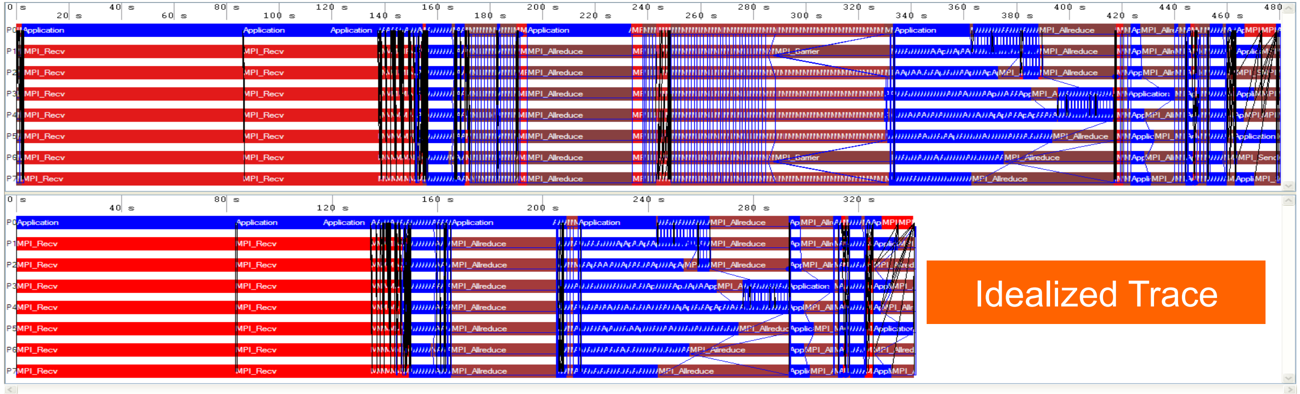
Zooming


4.0 s
4.1 s
4.2 s
4.3 s
4.4 s
4.5 s
4.6 s
4.7 s
4.8 s
4.9 s
5.0 s
P0
P1
P2
P3
fpsdbench
intvrbench
elvrbhs
auxhs
0.04 s
0.02 s
0.00 s
Duration of Messages
Multiple
Single
Flat Profile
Load Balance
Call Tree
Call Graph
TGroup All_Processes
Name
TSelf
TSelf
TTotal
#Calls
TGroup All_Processes
FGroup MPI 5.746 588 s 5.746 588 s 7 812
FGroup intvrbench 3.824 473 s 8.272 278 s 18 240
FGroup mpb 2.379 724 s 74.315 269 s 540
FGroup elvrbhs 1.748 885 s 4.342 490 s 9 540
FGroup fpsdbench 1.580 806 s 6.107 156 s 6 509
FGroup auxhs 0.957 200 s 2.610 683 s 17 055
FGroup bpfvrhs 0.340 816 s 0.340 816 s 6 585
FGroup mtxb 0.012 948 s 17.163 427 s 128
FGroup Application 0.000 000 s 33.182 880 s 0
Count [#] (Sender by Receiver)

Ideal Interconnect Simulator (Idealizer)

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

Actual trace

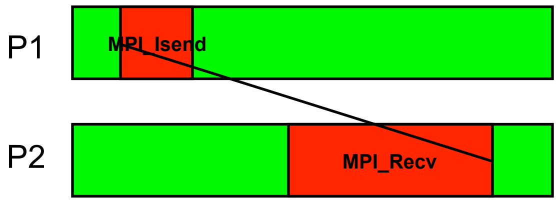


Idealized Trace

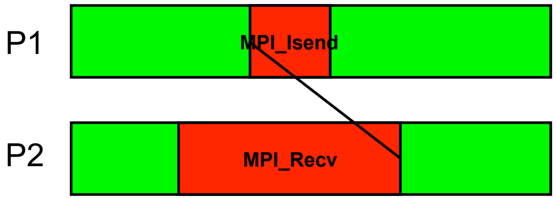
Easy way to identify application bottlenecks

Building Blocks: Elementary Messages

Early Send /
Late Receive

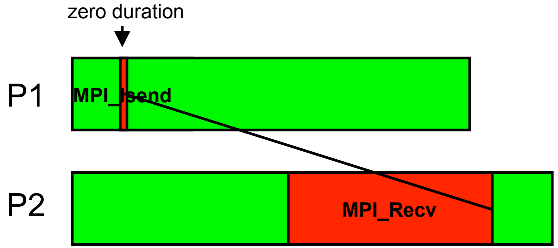


Late Send /
Early Receive

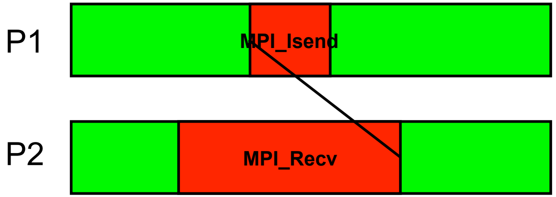


Building Blocks: Elementary Messages

Early Send /
Late Receive

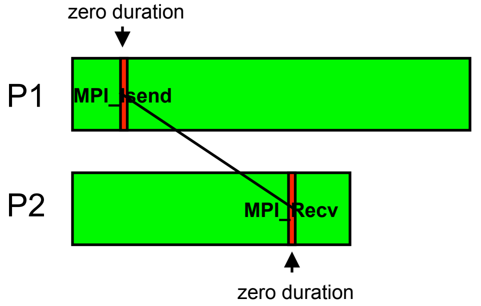


Late Send /
Early Receive

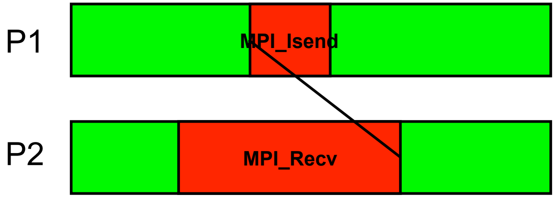


Building Blocks: Elementary Messages

Early Send /
Late Receive

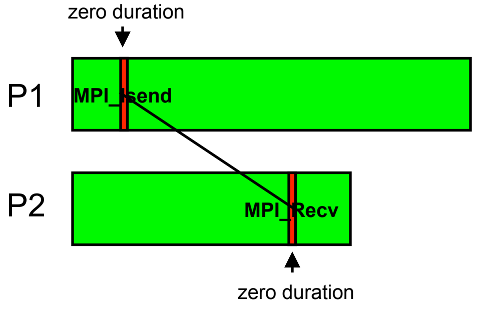


Late Send /
Early Receive

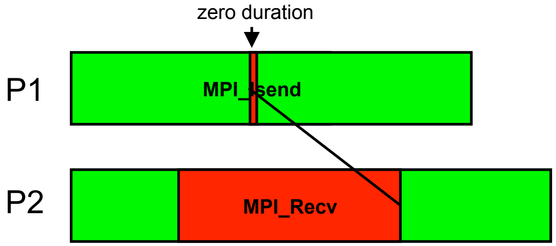


Building Blocks: Elementary Messages

Early Send /
Late Receive

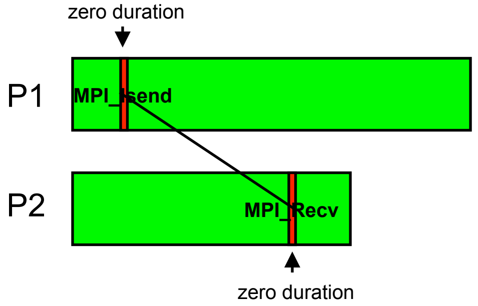


Late Send /
Early Receive

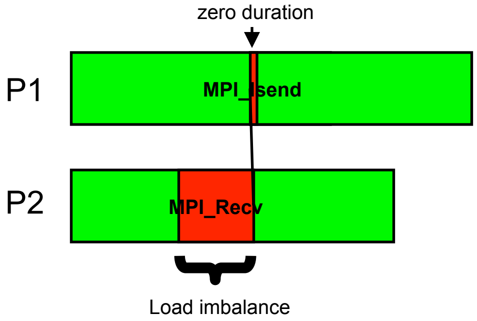


Building Blocks: Elementary Messages

Early Send /
Late Receive



Late Send /
Early Receive

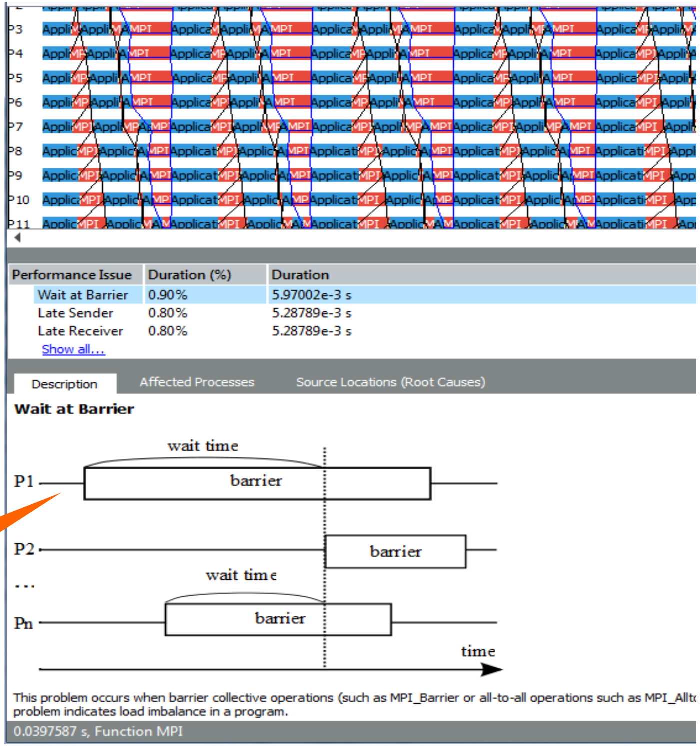


MPI Performance Assistance

Automatic Performance Assistant

Detect common MPI performance issues

Automated tips on potential solutions



Automatically detect performance issues and their impact on runtime

MPI Performance Snapshot (MPS)

MPI Performance Snapshot (MPS)

- New tool available with Intel® Trace Analyzer and Collector
- Enables developer to quickly collect performance summary of large jobs
  - Tested to 37,000 ranks
- Lightweight, scalable collection
- Report on application imbalances between MPI, OpenMP*, and serial time
- Report hardware counters via Intel® VTune™ Amplifier XE or PAPI
- Separates statistical analysis from event analysis

Why MPI Performance Snapshot (MPS)?

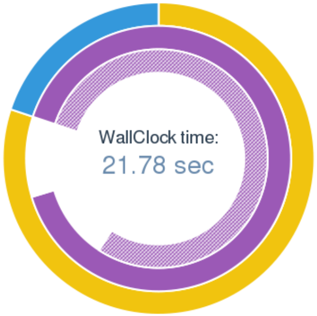
- Advantages
  - Get an initial profile of the application very quickly
  - Performance variation at scale can be detected and triaged quickly
  - Provides development recommendations to developers based on analysis
    - Intel® Trace Analyzer and Collector or Intel® VTune™ Amplifier XE for deeper analysis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

Complementary MPI Analysis Tools

	Intel® Trace Analyzer and Collector		MPI Performance Snapshot
Scalability	0 – 4K ranks		**1K – 100K ranks (37K tested)**
Collection Details	**High (events, source hooks)**		Low (aggregation)
Collection Size	Huge (~17 GB for 1K ranks)		**Much less (~0.8 GB for 1K ranks; ~4.5 GB for 37K)**
Event-based Analysis	**Yes**	+	EBS or PAPI events
Statistics Analysis	**Yes**		**Yes**
Quick Processing	No		**Yes**
Small & flexible	No		**Yes**
Collector	Intel® Trace Collector		**Intel® MPI Library built-in statistics and MPI/OpenMP* imbalance**

HTML Reporting

MPI Performance Snapshot Summary



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

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

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

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

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

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

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

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

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

High Capacity MPI Profiler

Combination of lightweight collector (EBS/PAPI events + OMP itt_notify metrics + MPI wait time metrics) and internal MPI statistics

Metrics collected:

- MPI time vs. application time
- Sum of time spent in each MPI function
- MPI message size and transfer data (total and per rank)
- HW counters from EBS or PAPI (e.g. FP, vectorized DP, memory access instructions)
- Memory usage stats (total and per rank)
- MPI/OpenMP/Serial imbalance

Shows OpenMP/MPI imbalance and HW counters all in one result

Easy to use ‘entry point’ for starting analysis

Currently only available on Linux* as command line tool

MPS Usage

4 quick steps to getting started

Install Intel® MPI Library and Intel® Trace Analyzer and Collector

Setup your environment

```
$ source /opt/intel/itac/9.1/bin/mpsvars.sh --vtune
```

Run with the MPI Performance Snapshot enabled

```
$ mpirun -mps -n 1024 ./exe
```

Analyze your results

```
$ mps ./stats_<timestamp>.txt ./app_stat_<timestamp>.txt
```

MPS Output

Summary

Files and folders:

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

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

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

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

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

MPS HTML Report Breakdown – MPI Time

MPI Time – Time spent in MPI calls

MPI Imbalance – MPI time spent waiting

Lower is better

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

MPI Time: 17.28 sec 80.10%

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

MPI Imbalance: 7.44 sec 34.47%

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

MPS HTML Report Breakdown – OpenMP Time

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

OpenMP Imbalance – OpenMP Time spent waiting – lower is better

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

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

OpenMP Time: 19.46 sec 90.19%

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

This value is HIGH.

OpenMP Imbalance: 17.05 sec 79.03%

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

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

Useful MPS Reports – MPI Function Summary

```
[~/projects/Cardiac_demo-master]$ mps app_stat_20160310-035458.txt stats_20160310-035458.txt -f
| Reading: app_stat_20160310-035458.txt
| Reading: stats_20160310-035458.txt
| Done.
|
| Function summary for all ranks
|---------------------------------------------------------------------------------------------------
|          Function      Time(sec)       Time(%)     Volume(MB)     Volume(%)           Calls
|---------------------------------------------------------------------------------------------------
              Wait         259.42         88.30           0.00          0.00         2560144
              Test          15.59          5.31           0.00          0.00        21843828
           Barrier          10.31          3.51           0.00          0.00            1040
              Send           2.96          1.01        4568.14         49.97         4000252
              Init           2.80          0.95           0.00          0.00              17
         Allreduce           1.14          0.39           0.01          0.00            1584
| [skipped 9 lines]
|===================================================================================================
| TOTAL                    293.80        100.00        9142.59        100.00        32408964
|
```

Useful MPS Reports – MPI Time per Rank

```
[~/projects/Cardiac_demo-master]$ mps app_stat_20160310-035458.txt stats_20160310-035458.txt -t
| Reading: app_stat_20160310-035458.txt
| Reading: stats_20160310-035458.txt
| Done.
|
| MPI Time per Rank
|---------------------------------------------------------
| Rank     LifeTime(sec)    MPI Time(sec)     MPI Time(%)
|---------------------------------------------------------
  0015             21.50            19.52           90.81
  0013             21.50            19.31           89.83
  0007             21.50            19.29           89.71
  0009             21.50            19.28           89.69
  0004             21.50            19.23           89.43
  0012             21.50            19.20           89.33
  0011             21.50            19.17           89.14
  0014             21.50            19.12           88.96
  0010             21.50            19.12           88.94
  0005             21.50            19.08           88.75
  0006             21.50            19.06           88.67
  0008             21.50            18.74           87.15
  0000             21.78            18.25           83.80
  0016             21.78            15.93           73.13
  0003             21.76            13.55           62.30
  0002             21.76            13.00           59.77
  0001             21.76             2.94           13.52
|=========================================================
| TOTAL           366.81           293.80           80.10
| AVG              21.58            17.28           80.10
|
```

Intel® VTune™ Amplifier XE

Using Intel® VTune™ Amplifier XE on MPI programs

Run VTune underneath MPI

VTune can run multiple instances per node

- Results are grouped into one result per node
  - <result folder>.<node name>
- Within result, ranks indicate rank number

```
$ mpirun <mpi args> amplxe-cl <vtune args> -- <application and args>
```

New!

Easier Multi-Rank Analysis of MPI + OpenMP

Tune hybrid parallelism using ITAC + VTune Amplifier

Tune OpenMP performance of high impact ranks in VTune Amplifier

Ranks sorted by MPI Communication Spins – ranks on the critical path are on the top

Process names link to OpenMP metrics

Per-rank OpenMP Potential Gain and Serial Time metrics

Top OpenMP Processes by MPI Communication Spin Time

This section lists processes sorted by MPI Communication Spin time. The lower MPI Communication Spin time
a critical path of MPI application execution. Explore OpenMP efficiency metrics by MPI processes laying on the

Process	PID	MPI Communication Spinning	(%)	OpenMP Potential Gain	(%)		
heart_demo (rank 7)	32394	5.122s	8.1%	19.929s	31.3%	2.875s	4.5%
heart_demo (rank 10)	32397	5.463s	8.6%	19.482s	30.6%	2.867s	4.5%
heart_demo (rank 11)	32398	5.593s	8.8%	20.183s	31.7%	2.873s	4.5%
heart_demo (rank 6)	32393	6.264s	9.8%	19.429s	30.5%	2.868s	4.5%
heart_demo (rank 9)	32396	6.595s	10.4%	19.379s	30.5%	2.864s	4.5%

Detailed OpenMP metrics per MPI ranks

Advanced Hotspots Hotspots viewpoint (change) Inte

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

Process / OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack	Elapsed Time	OpenMP Potential Gain: Imbalance	Loc.. Co..	Cre..	Sche...	Red..	Ato.	Other	MPI Com.. Spin...	Numb... of Open... threads	Ins... Cou.	Ope. Loo.. Chu.	Open... Loop Sche... Type	Avg Open.. Loop Itera...
heart_demo (rank 7)	33.938s	16.780s	0s	0s	0.086s	0s	0s	0.095s	5.122s		1			
[Serial - outside any region]	2.875s							0s	2.536s					
solveompparallel:18@unknown:527:561	31.062s	16.780s	0s	0s	0.086s	0s	0s	0.095s	2.586s	18	1			
make_rk_stepomploop_barrier_segment@unknown:352	8.887s	1.124s	0s	0s	0.012s	0s	0s	0.013s	0s	18		98	Static	1,762
update_coupling_v2omploop_barrier_segment@unknov	16.461s	11.565s	0s	0s	0.025s	0s	0s	0.043s	0.281s	18		98	Static	1,762
make_rk_stepompbarrier_segment@unknown:247	5.715s	4.090s	0s	0s	0.049s	0s	0s	0.039s	2.305s	18				
heart_demo (rank 10)	63.619s	19.108s	0s	0s	0.188s	0s	0s	0.187s	5.463s		1			
heart_demo (rank 11)	63.615s	19.824s	0s	0s	0.176s	0s	0s	0.183s	5.593s		1			
heart_demo (rank 6)	63.617s	19.091s	0s	0s	0.148s	0s	0s	0.190s	6.264s		1			
heart_demo (rank 9)	63.616s	19.002s	0s	0s	0.173s	0s	0s	0.203s	6.595s		1			

Checking MPI Application Correctness

MPI Correctness Checking

Solves two problems:

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

Two aspects:

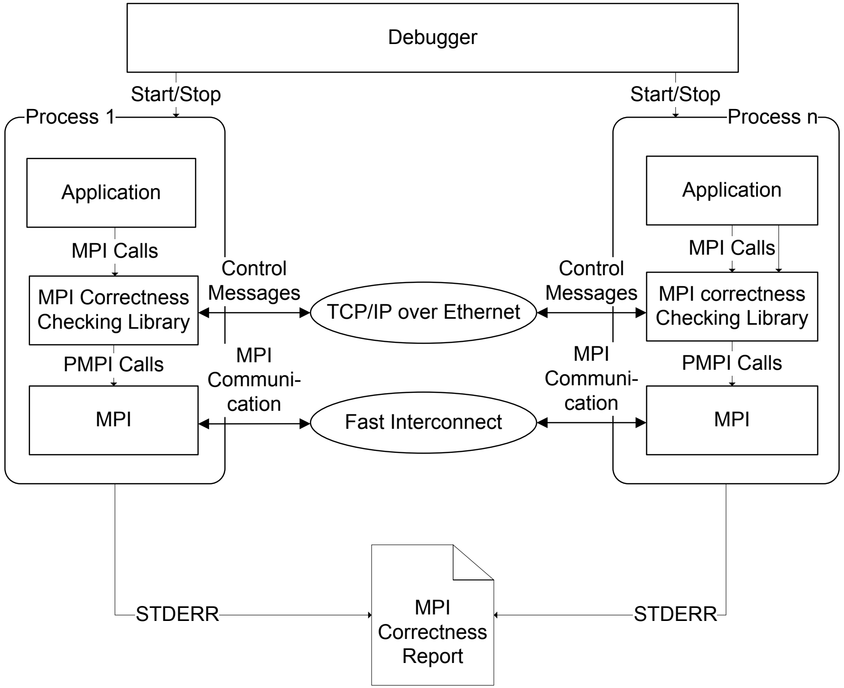
- Error Detection – done automatically by the tool
- Error Analysis – manually by the user based on:
  - Information provided about an error
  - Knowledge of source code, system, …

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:

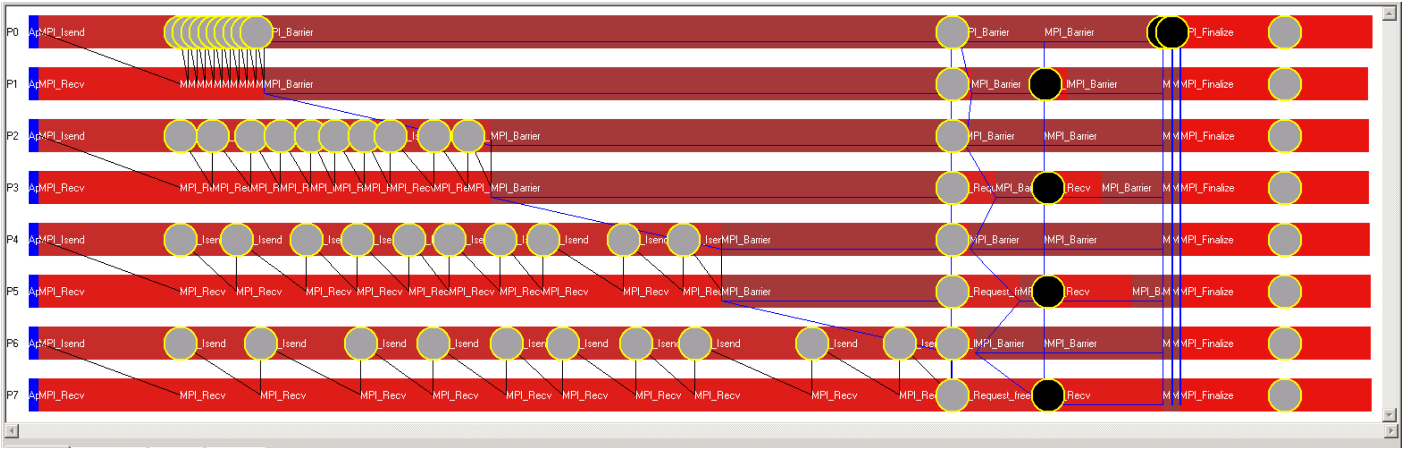
```
$ mpirun –check_mpi -n 2 overlap
[...]
[0] WARNING: LOCAL:MEMORY:OVERLAP: warning
[0] WARNING:    New send buffer overlaps with currently active send buffer at address 0x7fbfffec10.
[0] WARNING:    Control over active buffer was transferred to MPI at:
[0] WARNING:      MPI_Isend(*buf=0x7fbfffec10, 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:

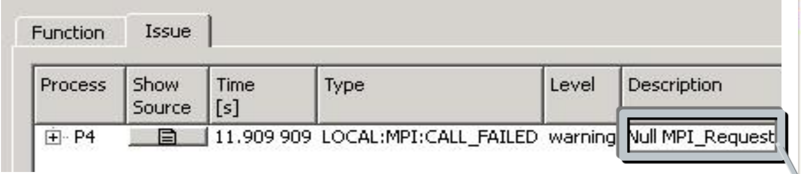
- Enable VT_CHECK_TRACING environment variable:

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



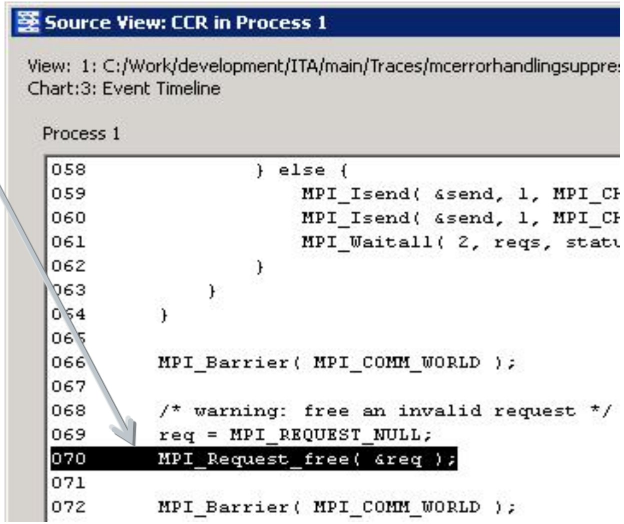
Errors **Warnings**

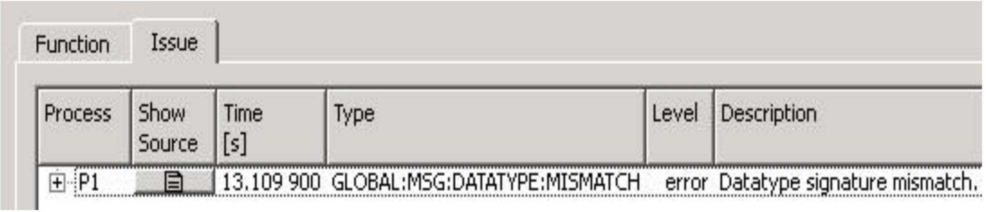
Viewing Source Code



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.).





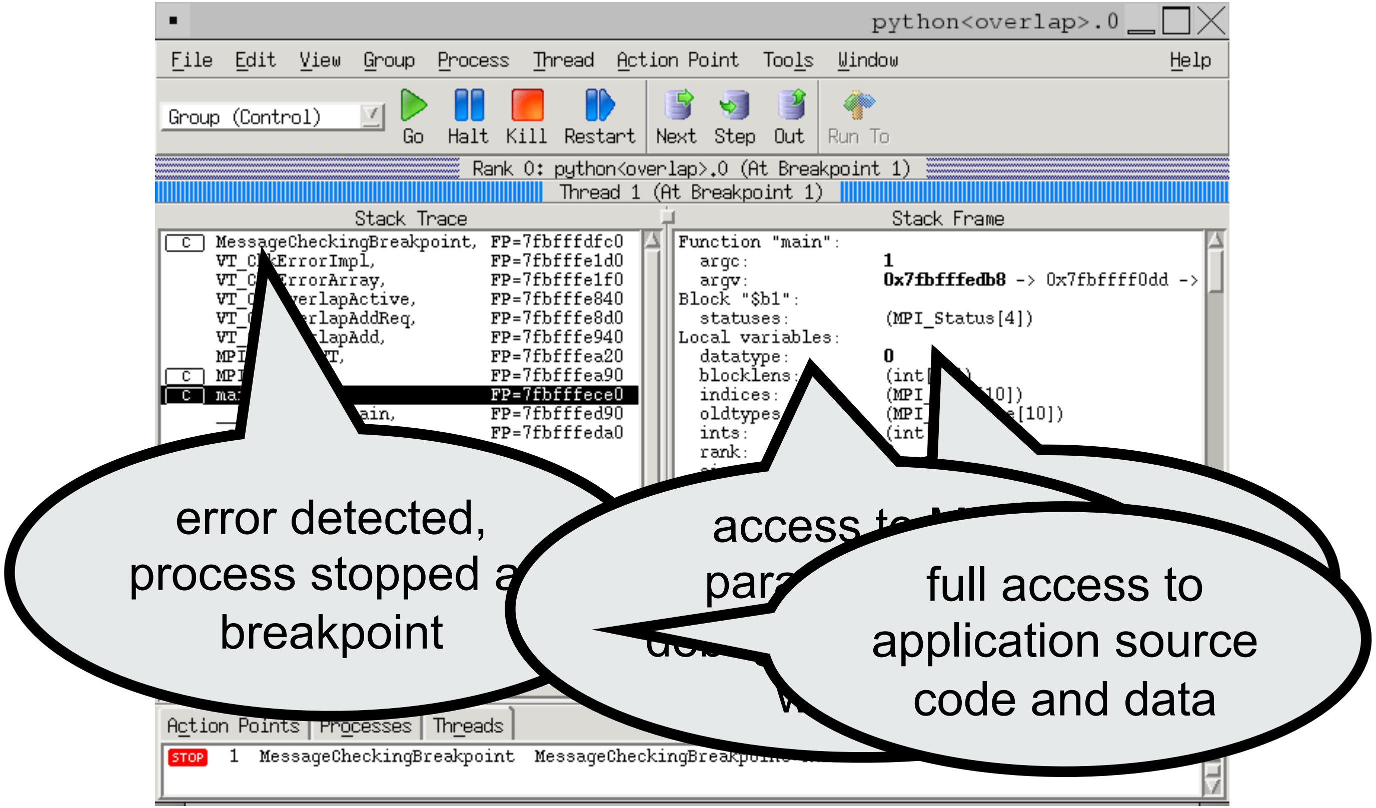
Debugger Integration

Debugger must be in control of application before error is found

A breakpoint must be set in MessageCheckingBreakpoint()

Documentation contains instructions for automating this process for TotalView*, gdb, and idb.

Usage of Debugger



Intel® Inspector XE

Dynamic Analysis

Launch Intel® Inspector XE

- Use mpirun
- 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

Find errors earlier when they are less expensive to fix

Using Intel® Inspector XE with MPI

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

```
$ mpirun -n 4 inspxe-cl -r my_result -collect mi1 -- ./test
```

Argument Sets can be used for more control

- Only collect data on certain ranks
- Different collections or options on different ranks

A unique results directory is created for each analyzed MPI rank

Launch the GUI and view the results for each rank

Benchmarking MPI and Cluster Performance

Intel® MPI Benchmarks 4.1

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
  - Supports MPI-1.x, MPI-2.x, and MPI-3.x standards
-

What's New:

Introduction of new benchmarks

- Measure cumulative bandwidth and message rate values

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

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® Clusters and HPC Technology forums

- http://software.intel.com/en-us/forums/intel-clusters-and-hpc-technology

Intel® Xeon Phi™ Coprocessor Developer Community

- http://software.intel.com/en-us/mic-developer experience what's inside™

Legal Disclaimer & 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.

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