INTEL® PARALLEL STUDIO XE ANALYZERS AND CLUSTER TOOLS OVERVIEW

For Distributed Performance

Agenda

Intel VTune Amplifier

- Overview, Predefined Analysis, VTune Platform Profiler, Demo

Intel Advisor

- Vectorization Advisor, Roofline, Demo

Intel Inspector

- Overview, Demo

Intel® MPI Library

- Basic Usage, Heterogeneous Jobs, Intel MPI Benchmarks, mpitune

Intel® Trace Analyzer and Collector

- Overview, Demo

Faster, Scalable Code, Faster

Intel® VTune™ Amplifier Performance Profiler

Accurate Data - Low Overhead

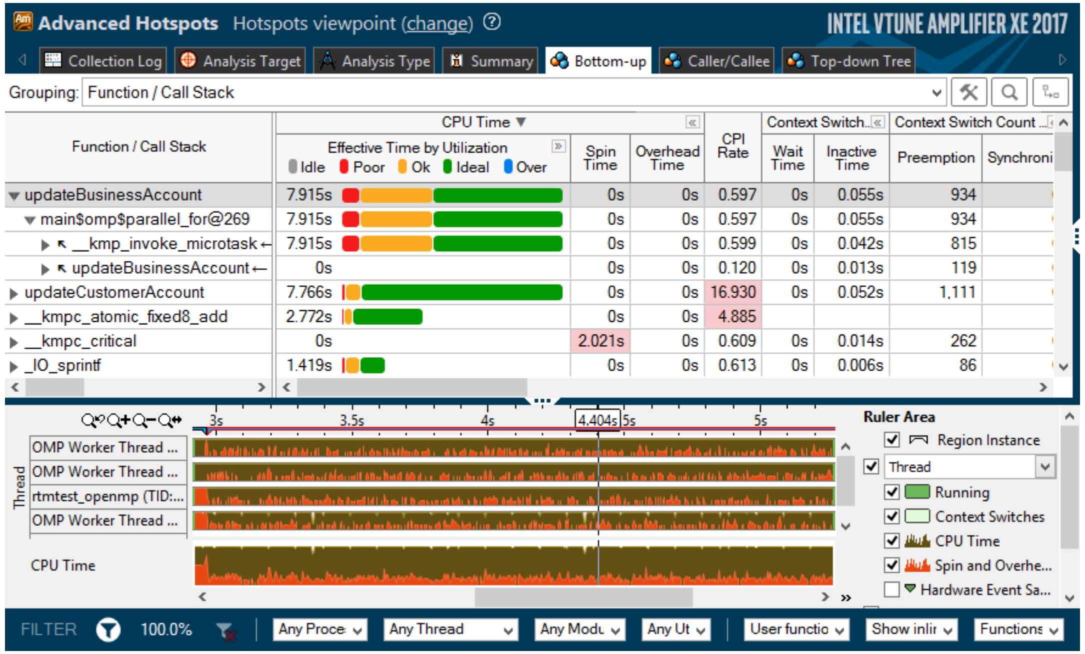
- CPU, GPU, FPU, threading, bandwidth…

Meaningful Analysis

- Threading, OpenMP region efficiency
- Memory access, storage device

Easy

- Data displayed on the source code
- Easy set-up, no special compiles



> "Last week, Intel® VTune™ Amplifier helped us find almost 3X performance improvement. This week it helped us improve the performance another 3X."

Claire Cates
Principal Developer
SAS Institute Inc.

http://intel.ly/vtune-amplifier-xe

Two Great Ways to Collect Data

Intel® VTune™ Amplifier

Software Collector	Hardware Collector
Uses OS interrupts	Uses the on chip Performance Monitoring Unit (PMU)
Collects from a single process tree	Collect system wide or from a single process tree.
~10ms default resolution	~1ms default resolution (finer granularity - finds small functions)
Either an Intel® or a compatible processor	Requires a genuine Intel® processor for collection
Call stacks show calling sequence	Optionally collect call stacks
Works in virtual environments	Works in a VM only when supported by the VM (e.g., vSphere*, KVM)
No driver required	Requires a driver - Easy to install on Windows - Linux requires root (or use default perf driver)

No special recompiles - C, C++, C#, Fortran, Java, Assembly

Find Answers Fast

Intel® VTune™ Amplifier

Adjust Data Grouping

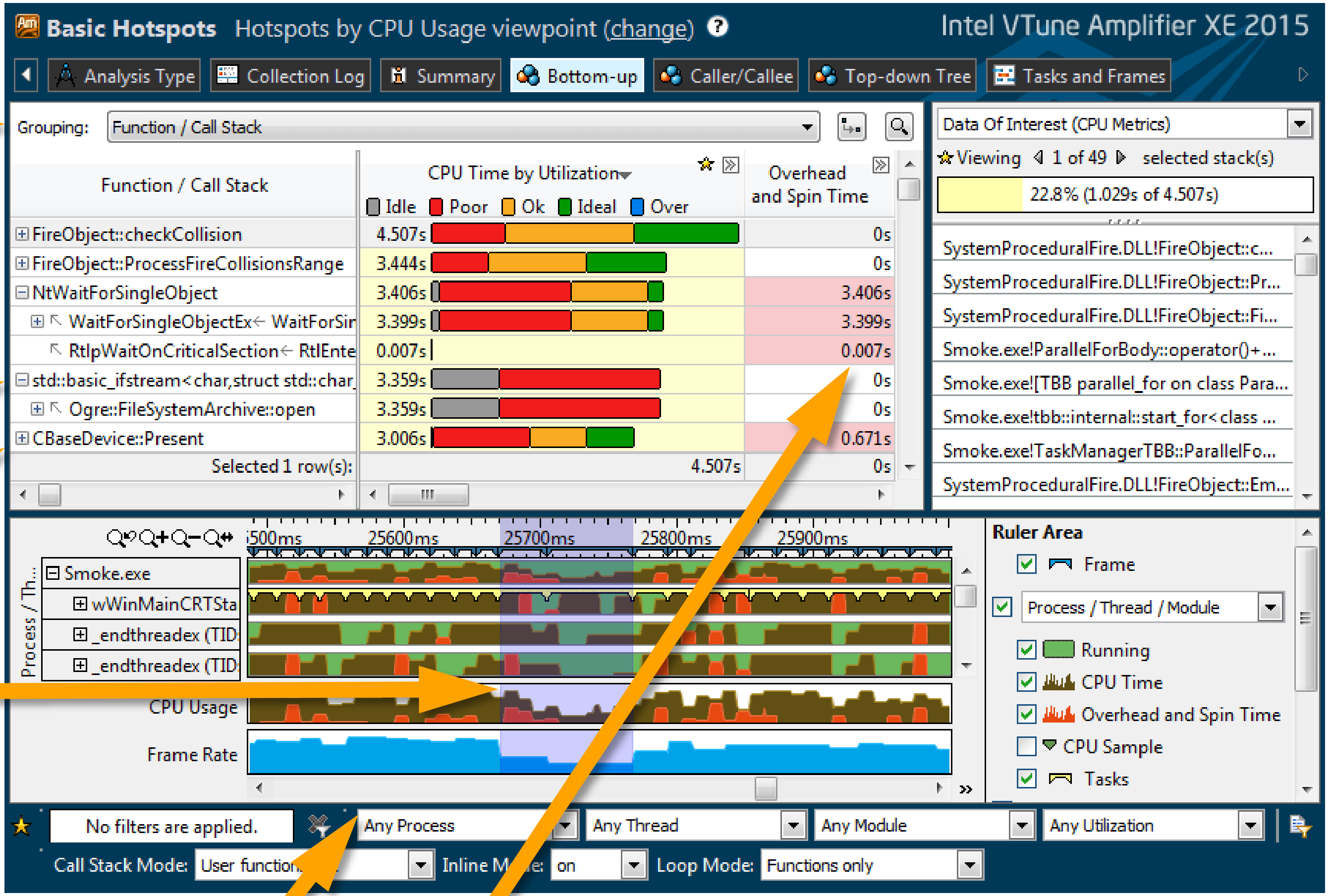
Function - Call Stack
Module - Function - Call Stack
Source File - Function - Call Stack
Thread - Function - Call Stack
... (Partial list shown)

Double Click Function to View Source

Click [+] for Call Stack

Filter by Timeline Selection (or by Grid Selection)

Zoom In And Filter On Selection
Filter In by Selection
Remove All Filters



Filter by Process & Other Controls

Tuning Opportunities Shown in Pink. Hover for Tips

See Profile Data On Source / Asm

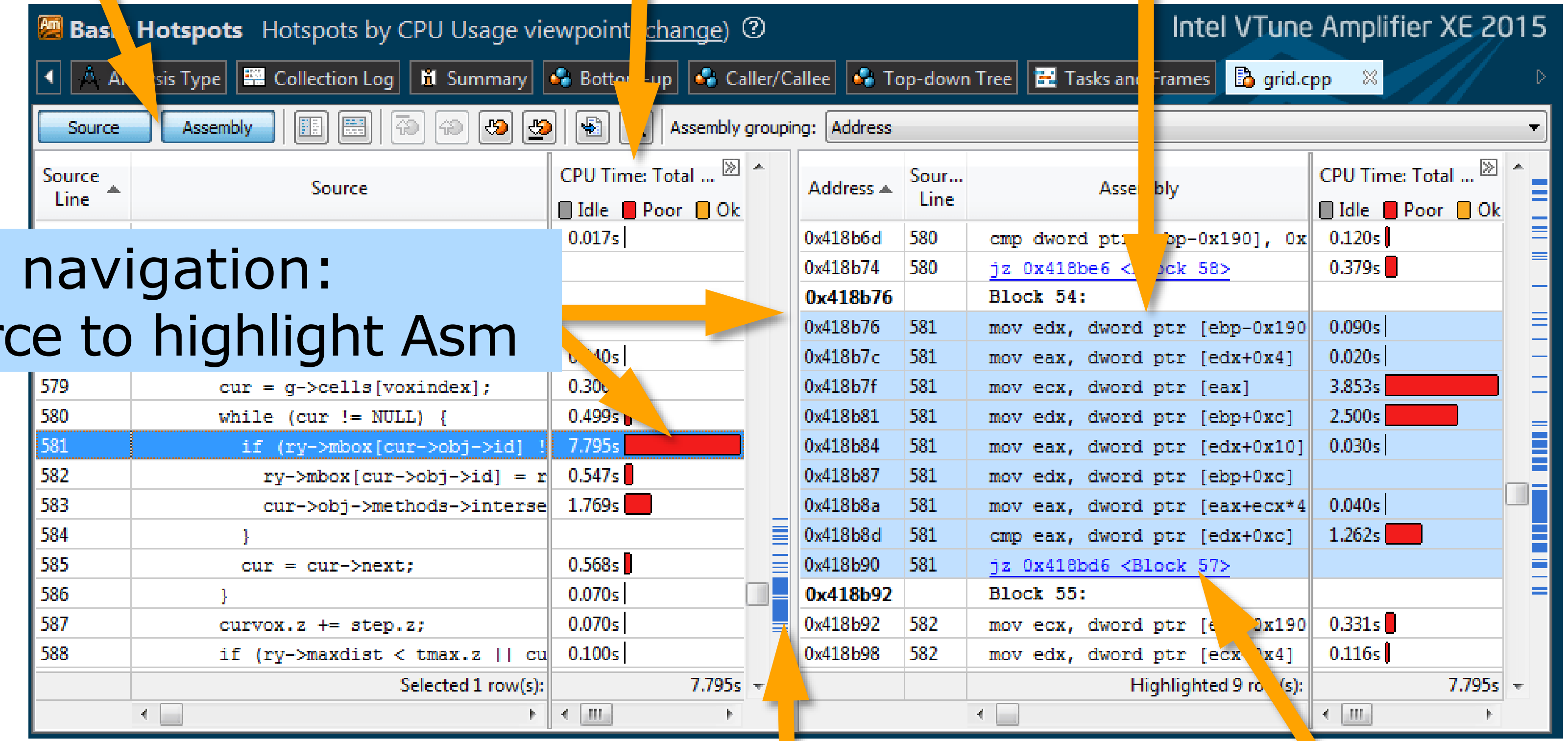
Double Click from Grid or Timeline

View Source / Asm or both

CPU Time

Right click for instruction reference manual

Quick Asm navigation:
Select source to highlight Asm

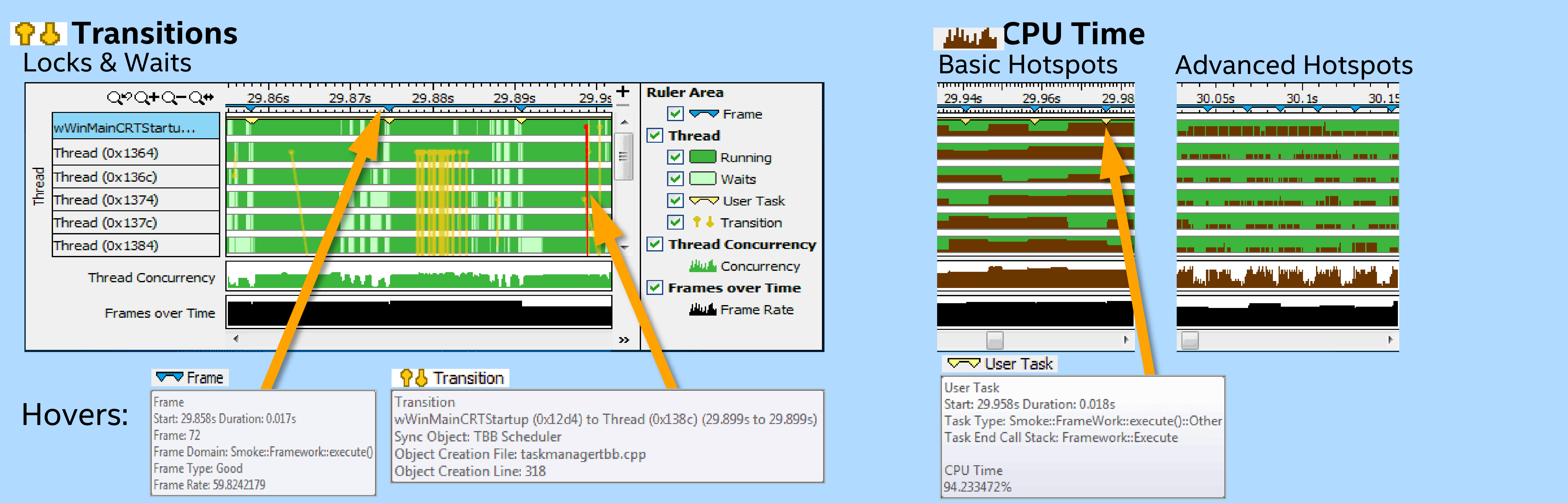


Scroll Bar “Heat Map” is an overview of hot spots

Click jump to scroll Asm

Timeline Visualizes Thread Behavior

Intel® VTune™ Amplifier



Optional: Use API to mark frames and user tasks Frame User Task

Optional: Add a mark during collection Mark Timeline

Command Line Interface

Automate analysis

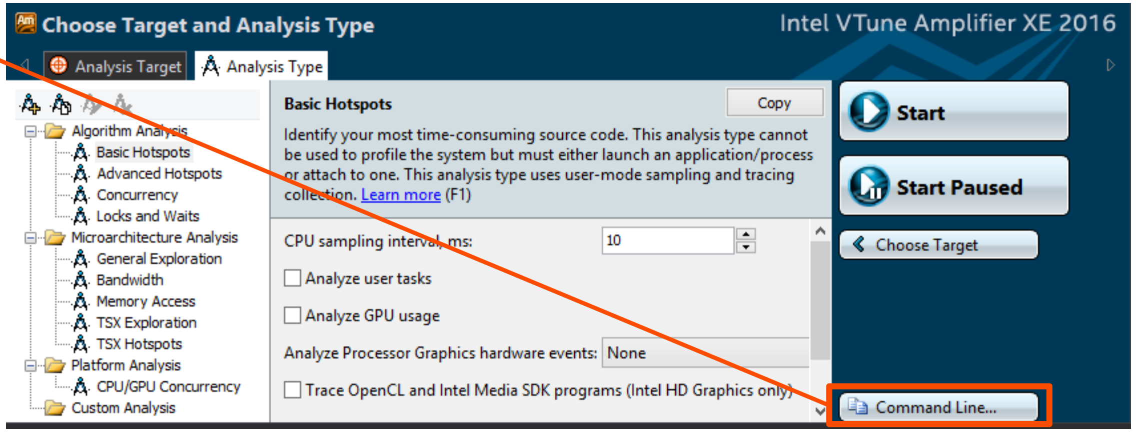
amplxe-cl is the command line:

- **Windows:** `C:\Program Files (x86)\Intel\VTune Amplifier XE \bin[32|64]\amplxe-cl.exe`
- **Linux:** `/opt/intel/vtune_amplifier_xe/bin[32|64]/amplxe-cl`

Help: `amplxe-cl -help`

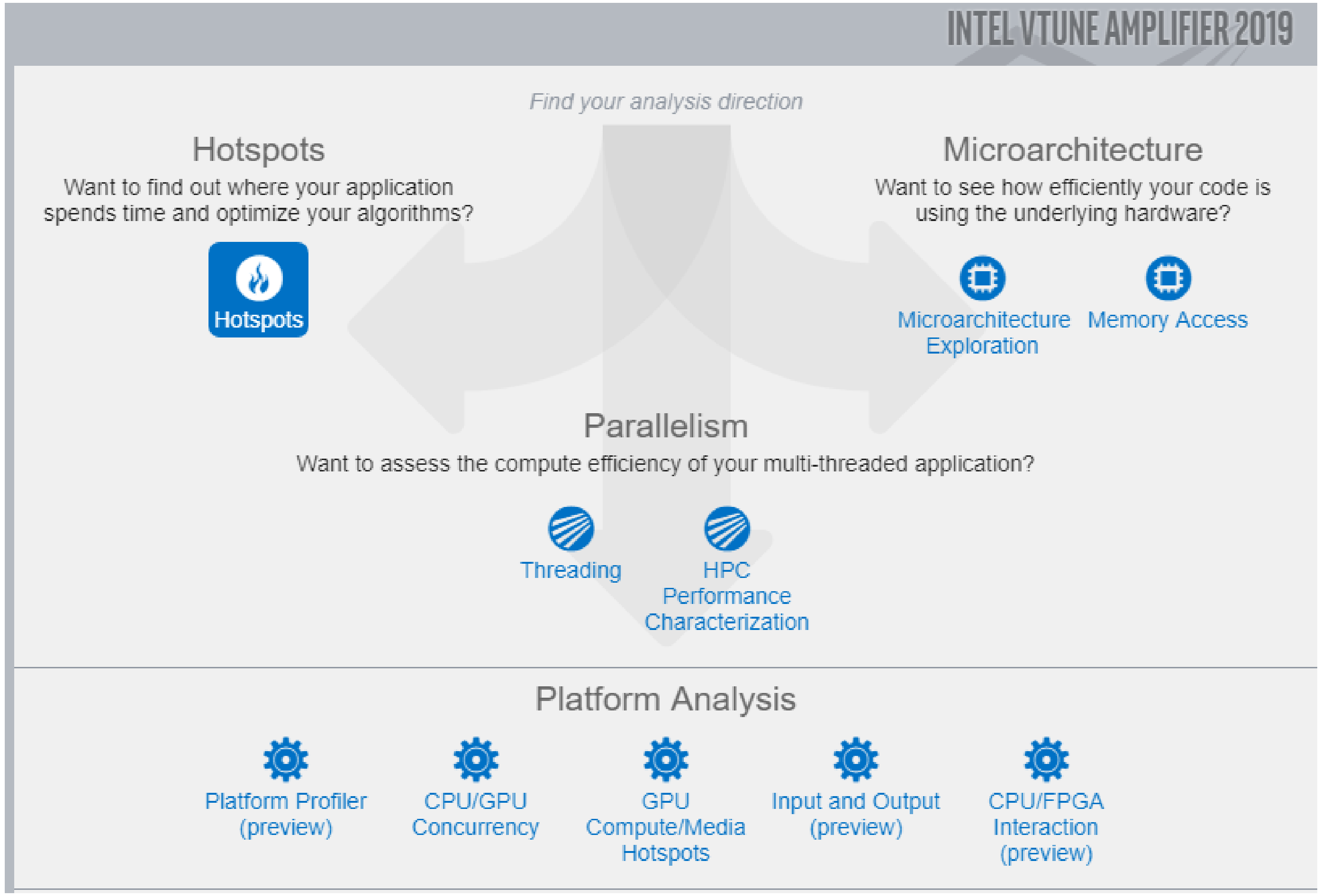
Use UI to setup

1) Configure analysis in UI
2) Press “Command Line...” button
3) Copy & paste command



Great for regression analysis – send results file to developer
Command line results can also be opened in the UI

VTune Predefined Analyses



VTune Command Line Analysis Configuration

How to Run VTune on MPI Applications

><mpi_launcher> – n N <vtune_command_line> ./app_to_run

- >srun –n 48 -N 16 amplxe-cl –collect memory-access **–trace-mpi** –r result_dir ./my_mpi_app
- >mpirun –n 48 -ppn 16 amplxe-cl –collect hotspots –r result_dir ./my_mpi_app

- Encapsulates ranks to per-node result directories suffixed with hostname
  - result_dir.hostname1 with 0-15, result_dir.hostname2 with 16-31, result_dir.hostname3 with 32-47

Add **–trace-mpi** option for VTune CL to enable per-node result directories for non-Intel MPIs

- Works for software and Intel driver-based collectors

VTune Command Line Analysis Configuration

Selective Rank Profiling

Superposition of application to launch and VTune command line for selective ranks to reduce trace size

Example: profile rank 1 from 0-15:

>mpirun -n 1 ./my_app : -n 1 <vtune_command_line> -- ./my_app : -n 14 ./my_app

- In the case of Intel MPI launcher –gtool option can be used:

Example: profile ranks 3, 7, 11-13 from 0-15:

>mpirun –gtool “amplxe-cl –collect advanced-hotspots –r result_dir:3,7,11-13” ./my_app

Finalization on KNL

Result finalization and viewing on KNL target might be slow

Use the recommended workflow:

1. Run collection on KNL deferring finalization to host:

   >amplxe-cl –collect memory-access –no-auto-finalize –r <my_result_dir> ./my_app

2. Finalize the result on the host

   - Provide search directories to the binaries of interest for resolving with –search-dir option

   >amplxe-cl –finalize –r <my_result_dir> –search-dir <my_binary_dir>

3. Generate reports, work with GUI

   >amplxe-cl –report hotspots –r <my_result_dir>

INTEL VTUNE™ AMPLIFIER – PLATFORM PROFILER

3-steps for optimizing complex workloads

Intel® VTune™ Amplifier

Characterize

- What is my workload doing
- Where are the bottlenecks

Right-size

- What configuration changes should I make

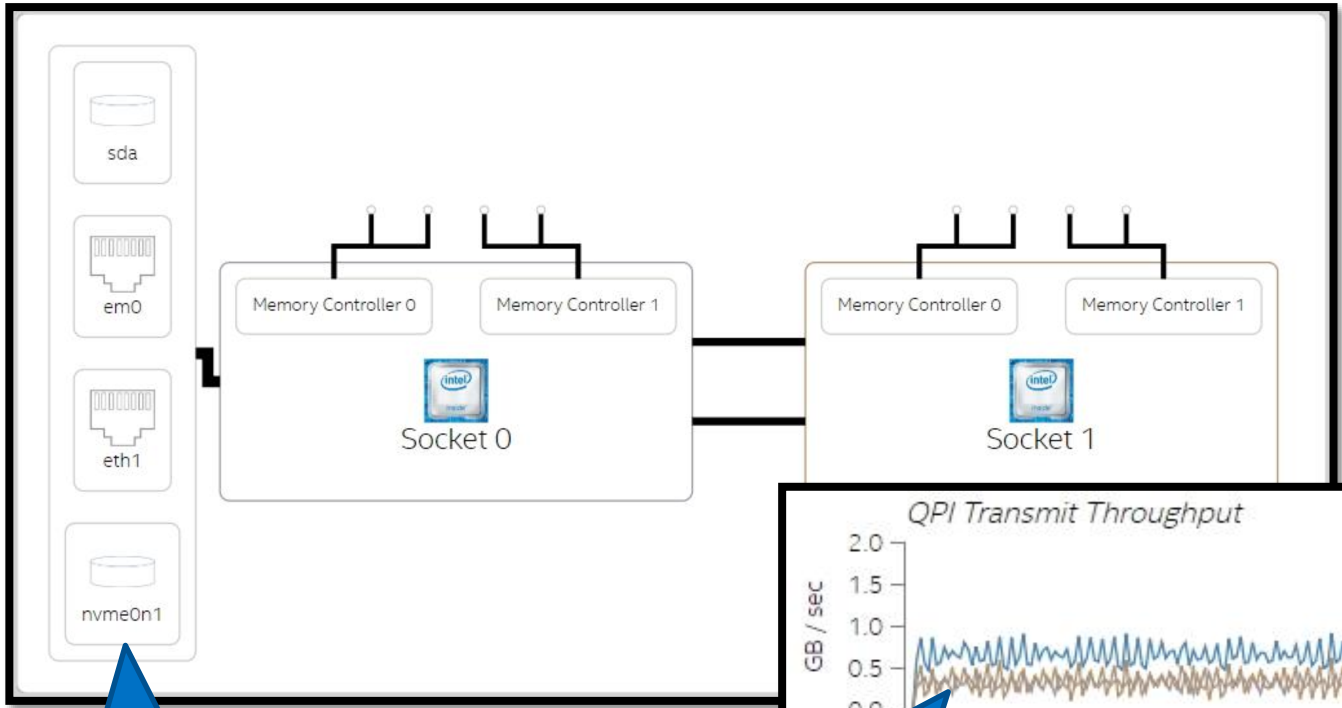
System Focus

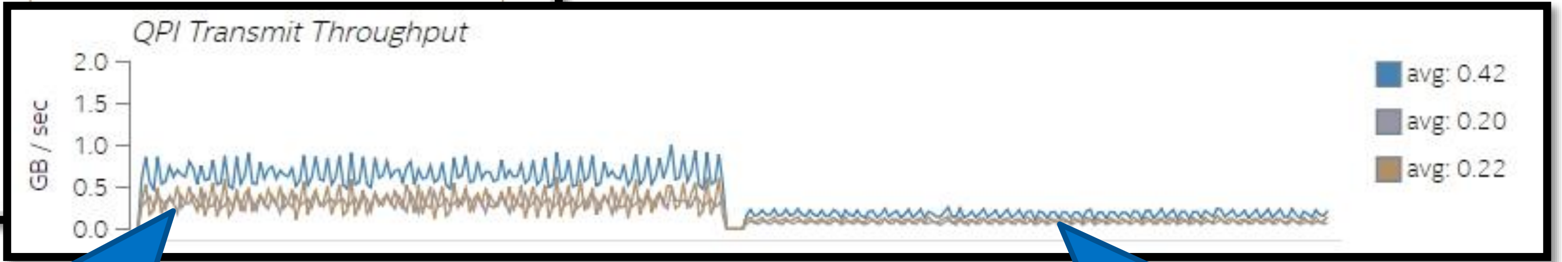
Optimize

- How do I get the most out of my platform

Application Focus

Configuration Matters!





5% performance drop when running from Socket 1 which is "far" from the storage device

Storage device connected to Socket 0

Interconnect traffic from an application running on Socket 1 doing IO

Interconnect traffic from an application running on Socket 0 doing IO

The Long & Short of Performance Analysis

Get the big picture first with a Snapshot or Platform Profiler

	Snapshot Quickly size potential performance gain. Run a test "during a coffee break".	**In-Depth** Advanced collection & analysis. Insight for effective optimization.
Application Focus • HPC App developer focus • 1 app running during test	VTune Amplifier's **Application Performance Snapshot** L🕑	**VTune Amplifier** • Many profiles S-M🕑 **Intel Advisor** • Vectorization S🕑 **ITAC** • MPI Optimization S-L🕑
System Focus • Deployed system focus • Full system load test	VTune Amplifier's **Storage Performance Snapshot** L🕑	**VTune Amplifier** - System-wide sampling S-M🕑 - Platform Profiler L🕑

Maximum collection times: L🕑=long (hours) M🕑=medium (minutes) S🕑=short (seconds-few minutes)

Platform Profiler

Identify system performance & configuration issues and headroom

Target User

- Infrastructure Architects
- Software Architects & QA

Analyze performance

- Display current configuration
- ~150 hardware and ~20 OS metrics

Identify system configuration issues

- Inefficient memory module placements
- Need for faster storage, larger/faster memory

Identify potential software issues

- Inefficient CPU/storage/memory utilization
- Near vs. far memory accesses (NUMA)

Platform Profiler

Common use cases

Visualize workload behavior

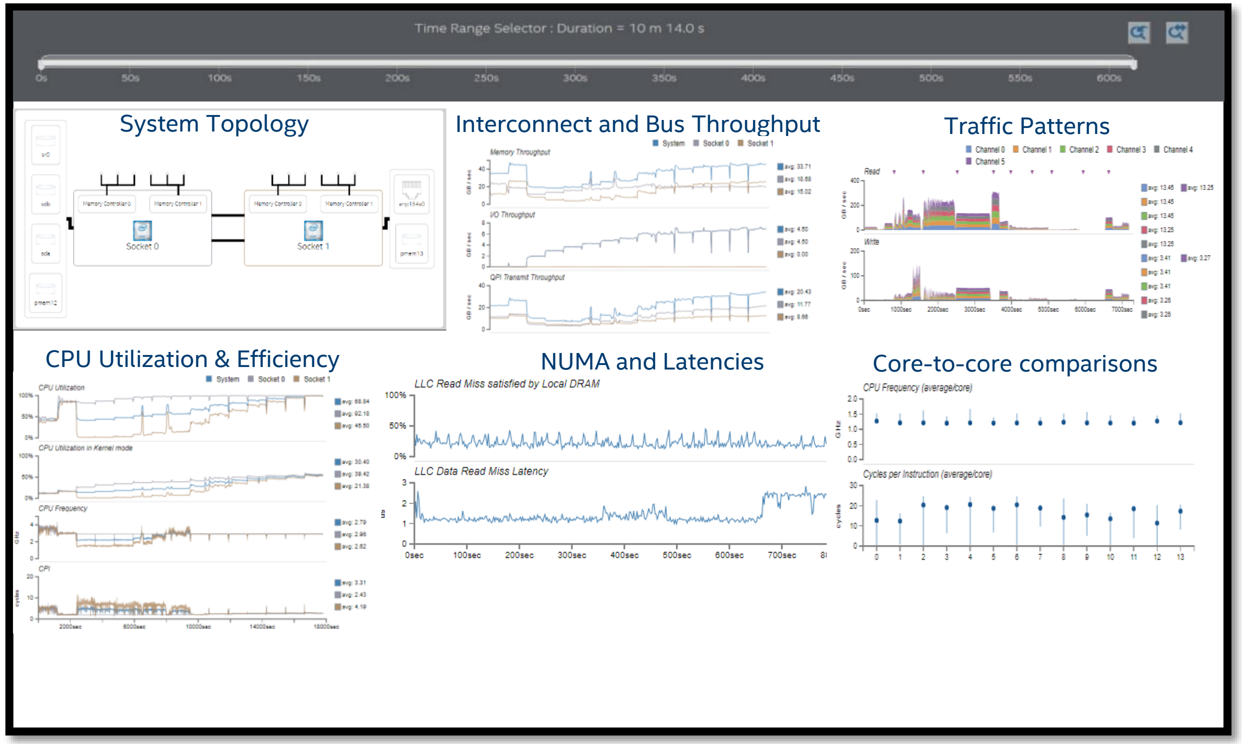
- Very low overhead ~2%
- Long running workloads (hours)

Plan tuning strategy

- Which app needs it most?
- Which phase of the app?
- Optimize cache? I/O? Threading?

Optimize hardware configuration

- Add more I/O? Add memory?



Platform Profiler

Progressive disclosure methodology

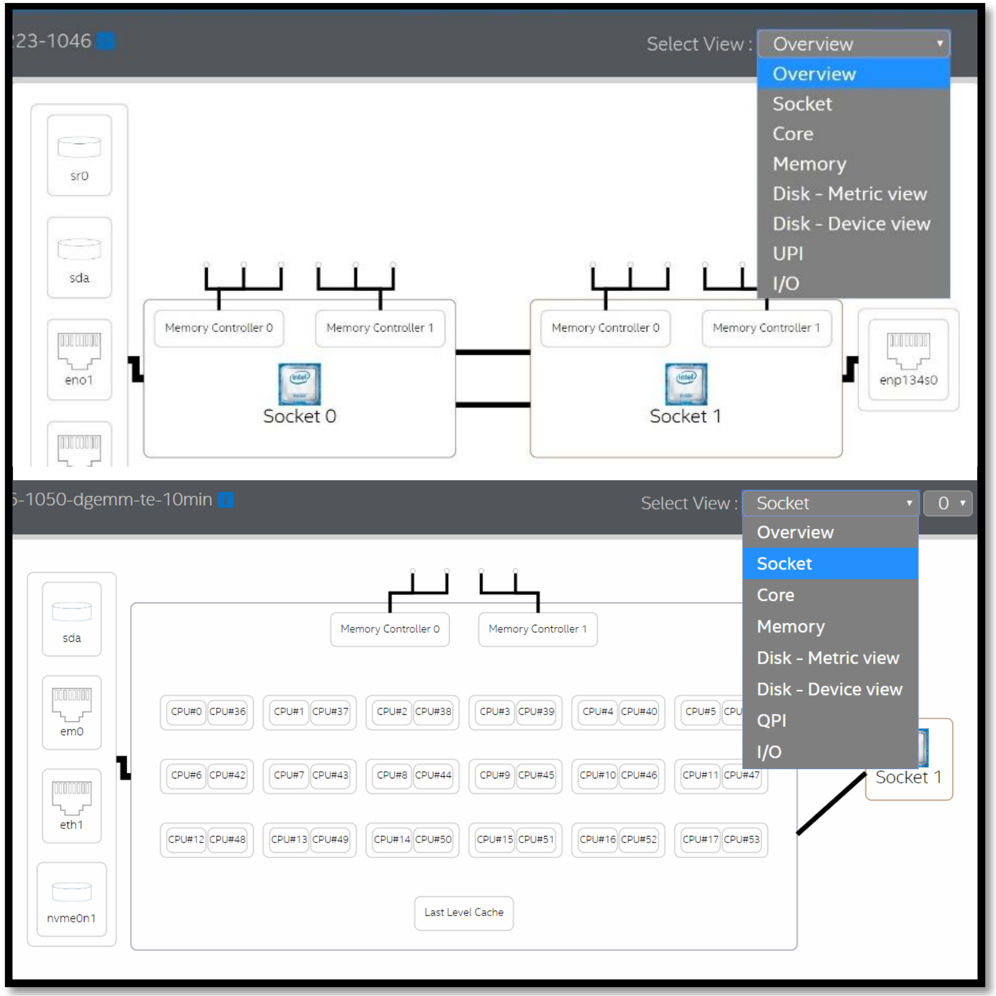
Start with Overview and Topology

View different aspects of system

- Socket, Core, Memory, Disk, UPI, I/O

Drill down

- Socket → Core → Internal Caches
- Socket → Memory Link → Memory Module



Platform Profiler

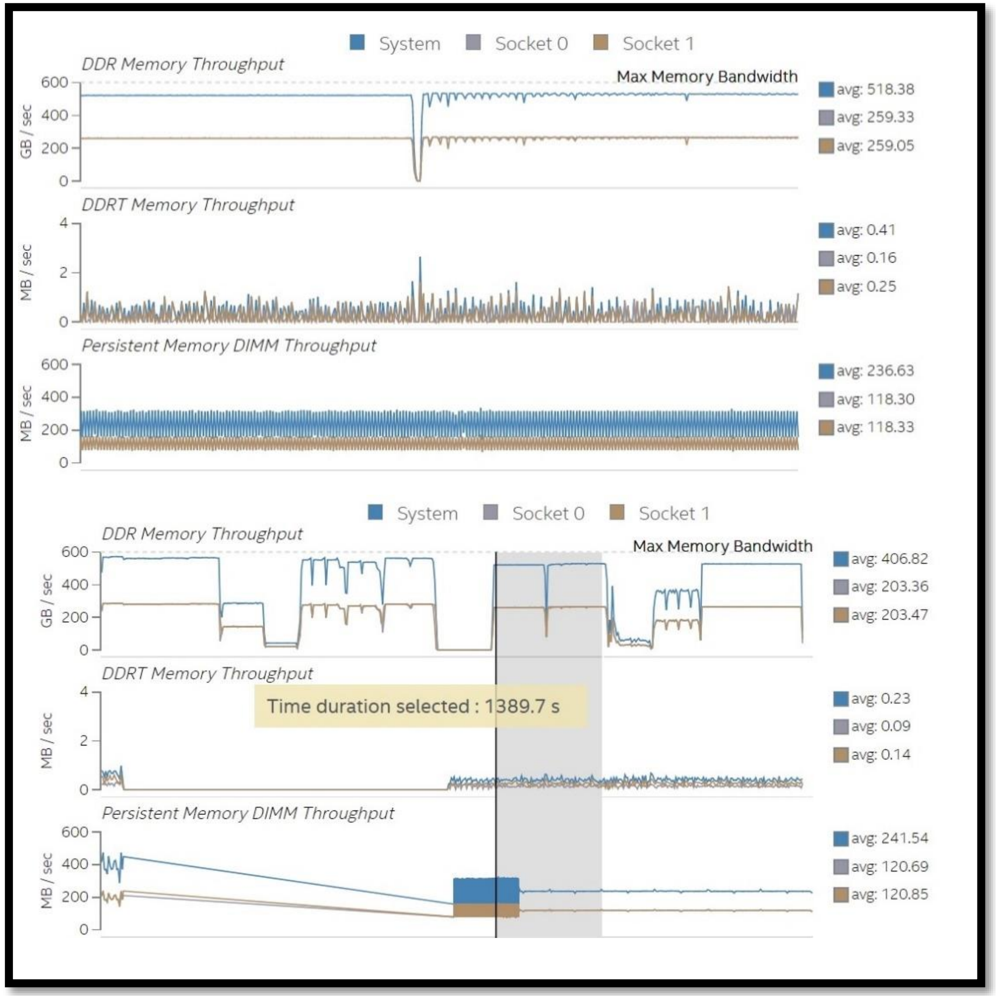
Telemetry data collected (and displayed) at a granularity of 100ms

Analyze hours of execution

- No process-level info
- Average won't hide erratic behavior

Select and zoom to desired execution region

- All graphs and averages adjust to selection)



Platform Profiler

Data collection

Data Collector

Platform Profiler Analysis

Data Collection “Targets”

- Windows*
- Linux*

Analysis and visualization

- Web-based UI
- Works best with Chrome

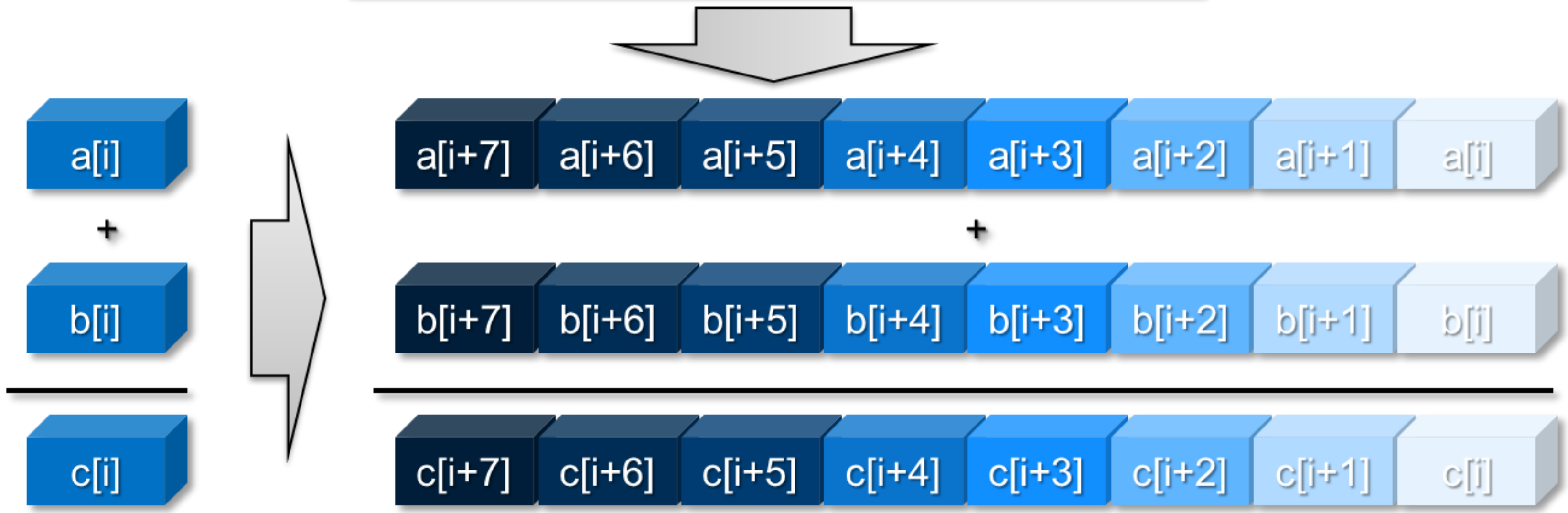
INTEL® ADVISOR

Vectorization of code

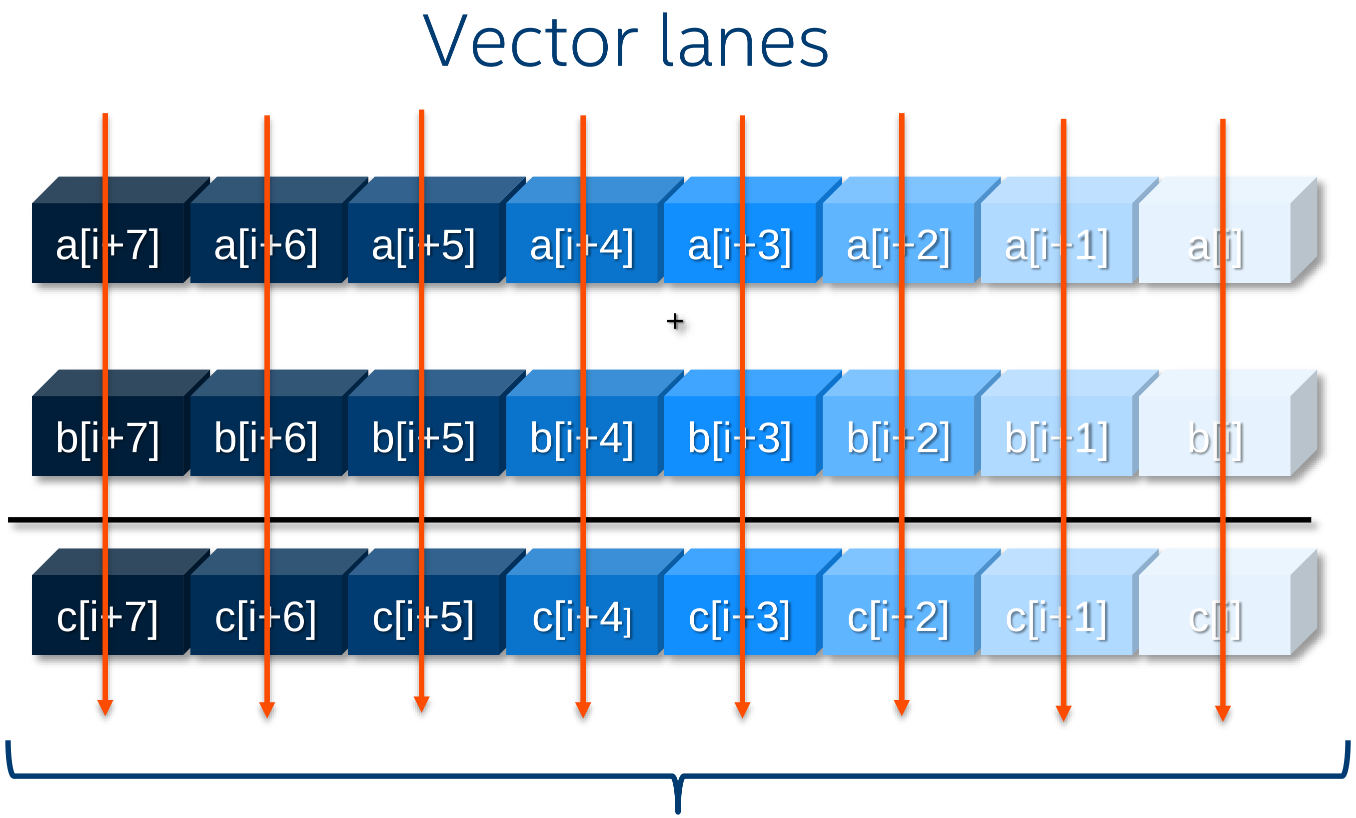
Transform sequential code to exploit vector processing capabilities (SIMD) of Intel processors

- Manually by explicit syntax
- Automatically by tools like a compiler

```
for(i = 0; i <= MAX;i++)
    c[i] = a[i] + b[i];
```



Vectorization terms



Vector length(VL): Elements of the vector

All elements of vector are of the same data types

Many Ways To Vectorize

Compiler:
Auto-vectorization (no change of code)

Compiler:
Auto-vectorization hints (`#pragma vector`, …)

Compiler:
OpenMP* 4.0

SIMD intrinsic class
(e.g.: `F32vec`, `F64vec`, …)

Vector intrinsic
(e.g.: `_mm_fmadd_pd(…)`, `_mm_add_ps(…)`, …)

Assembler code
(e.g.: `[v]addps`, `[v]addss`, …)

Ease of use

Programmer control

Tuning for Skylake - Compiler options

Both Skylake and Knights Landing processors have support for Intel® AVX-512 instructions. There are three ISA options in the Intel® Compiler:

`-xCORE-AVX512` : Targets Skylake, contains instructions not supported by Knights Landing

`-xCOMMON-AVX512` : Targets both Skylake and Knights Landing

`-xMIC-AVX512` : Targets Knights Landing, includes instructions not supported by Skylake

Intel® Compiler is conservative in its use of ZMM (512bit) registers so to enable their use with Skylake the additional flag **`-qopt-zmm-usage=high`** must be set.

Validating Vectorization Success I: Compiler report

- -qopt-report[=n]: tells the compiler to generate an optimization report
  - n: (Optional) Indicates the level of detail in the report. You can specify values 0 through 5. If you specify zero, no report is generated. For levels n=1 through n=5, each level includes all the information of the previous level, as well as potentially some additional information. Level 5 produces the greatest level of detail. If you do not specify n, the default is level 2, which produces a medium level of detail.
- -qopt-report-phase[=list]: specifies one or more optimizer phases for which optimization reports are generated.
  - loop: the phase for loop nest optimization
  - vec: the phase for vectorization
  - par: the phase for auto-parallelization
  - all: all optimizer phases
- -qopt-report-filter=string: specified the indicated parts of your application, and generate optimization reports for those parts of your application.

Validating Vectorization Success II

- -S: assembler code inspection
  - Most reliable way and gives all details of course
  - Check for scalar/packed or (E)VEX encoded instructions:
    Assembler listing contains source line numbers for easier navigation
  - Compiling with -qopt-report-embed (Linux*, macOS*) helps interpret assembly code
- Performance validation
  - Compile and benchmark with -no-vec –qno-openmp-simd or on a loop by loop basis via #pragma novector or !DIR$ NOVECTOR
  - Compile and benchmark with selected SIMD feature
  - Compare runtime differences

Intel® Advisor

Boosting your application by threading and vectorization

Compiler will not always vectorize

- Check for Loop Carried Dependencies
  using **Intel® Advisor**
- All clear? Force vectorization.
  C++ use: pragma simd, Fortran use: SIMD directive

Arrays of structures are great for intuitively organizing data, but are much less efficient than structures of arrays. Use the Intel® SIMD Data Layout Templates (Intel® SDLT) to map data into a more efficient layout for vectorization.

Not all vectorization is efficient vectorization

- Stride of 1 is more cache efficient than stride of 2 and greater. Analyze with **Intel® Advisor**.
- Consider data layout changes
  Intel® SIMD Data Layout Templates can help

Get Faster Code Faster! Intel® Advisor

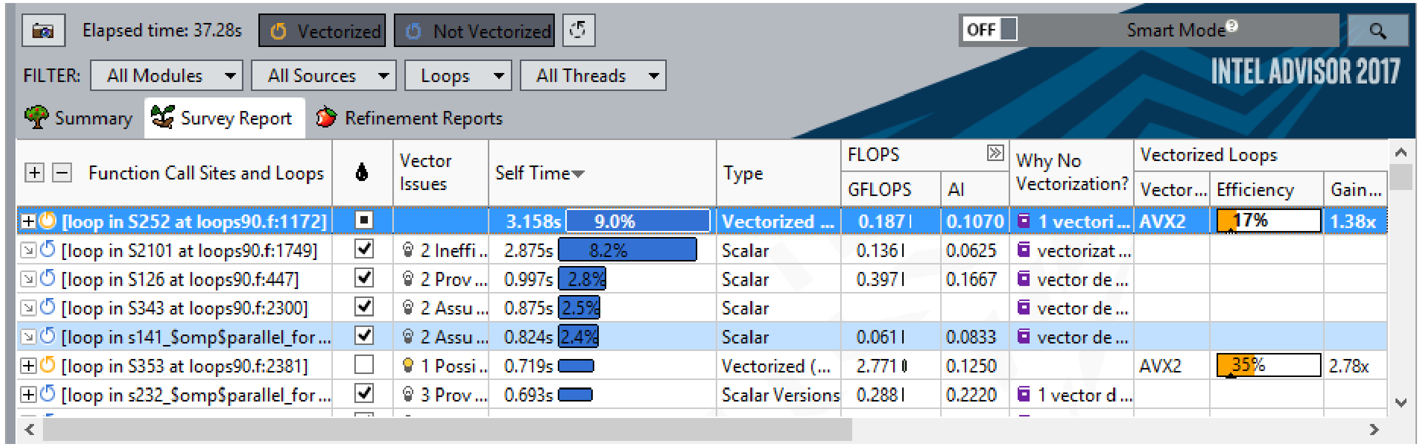
Vectorization Optimization

Have you:

- Recompiled for AVX2 with little gain
- Wondered where to vectorize?
- Recoded intrinsics for new arch.?
- Struggled with compiler reports?

Data Driven Vectorization:

- What vectorization will pay off most?
- What's blocking vectorization? Why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?



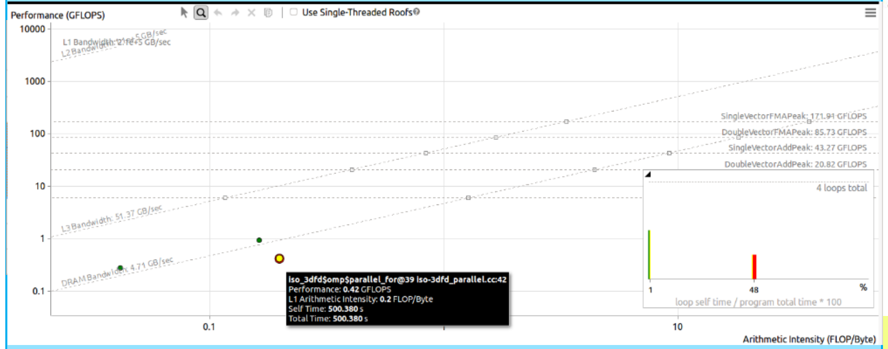
"Intel® Advisor's Vectorization Advisor permitted me to focus my work where it really mattered. When you have only a limited amount of time to spend on optimization, it is invaluable."

Gilles Civario
Senior Software Architect
Irish Centre for High-End Computing

4 Steps to Efficient Vectorization

Intel® Advisor – Vectorization Advisor

1. Compiler diagnostics + Performance Data + Roofline Model + SIMD efficiency information



2. Guidance: detect problem and recommend how to fix it

2

8

Issue: Peeled/Remainder loop(s) present

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at Vector Essentials, Utilizing Full Vectors...

Recommendation: Align memory access

Projected maximum performance gain: High

Projection confidence: Medium

The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned. This example aligns memory using a 32-byte boundary:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

// Somewhere else
__assume_aligned(array, 32);
// Use array in loop
```

3. Loop-Carried Dependency Analysis

Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P1		Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P6		Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7		Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	New

4. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cxx:1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns | Correctness Report

ID		Stride	Type	Source	Modules	Alignment
P22		0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	

```
635   j2 = ( j2 & 64-1 ) ;
636   p[ip][0] += y[i2+32];
637   p[ip][1] += z[j2+32];
638   i2 += e[i2+32];
639   j2 += f[j2+32];
```

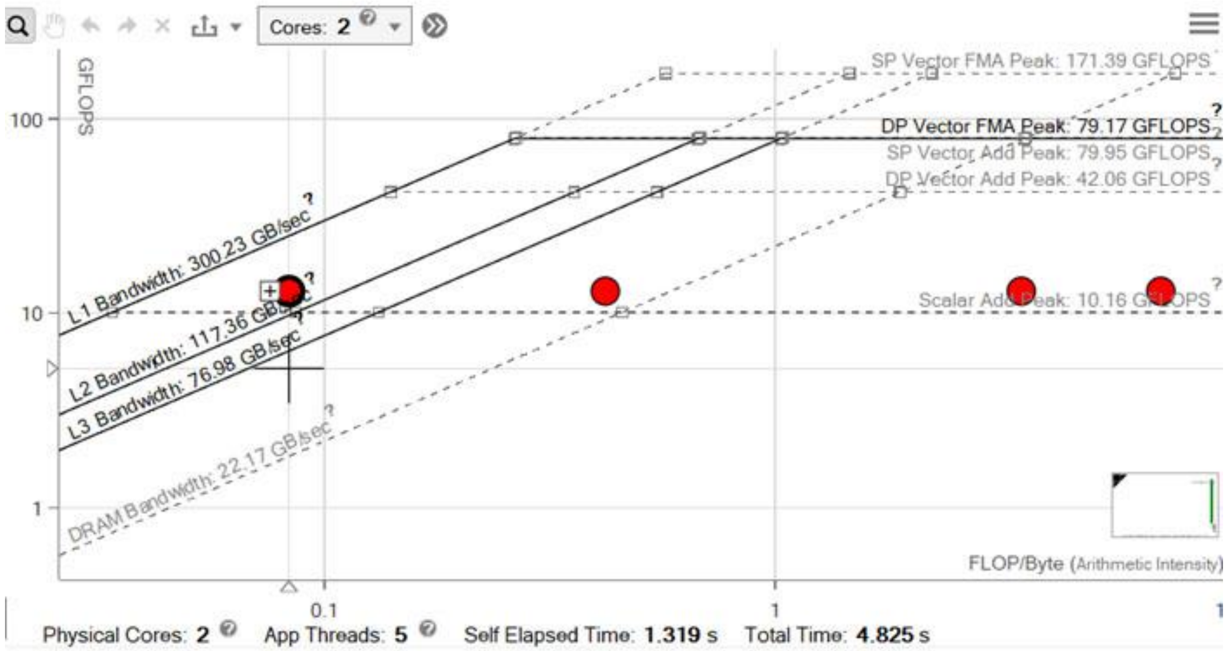
ID		Stride	Type	Source	Modules	Alignment
P23		0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
P30		-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628	lcals.exe	

```
626   i1 &= 64-1;
627   j1 &= 64-1;
628   p[ip][2] += b[j1][i1];
```

Integrated Roofline model

In the Intel® Advisor Integrated Roofline chart the Arithmetic Intensity and memory traffic for each level of the memory hierarchy is represented separately.

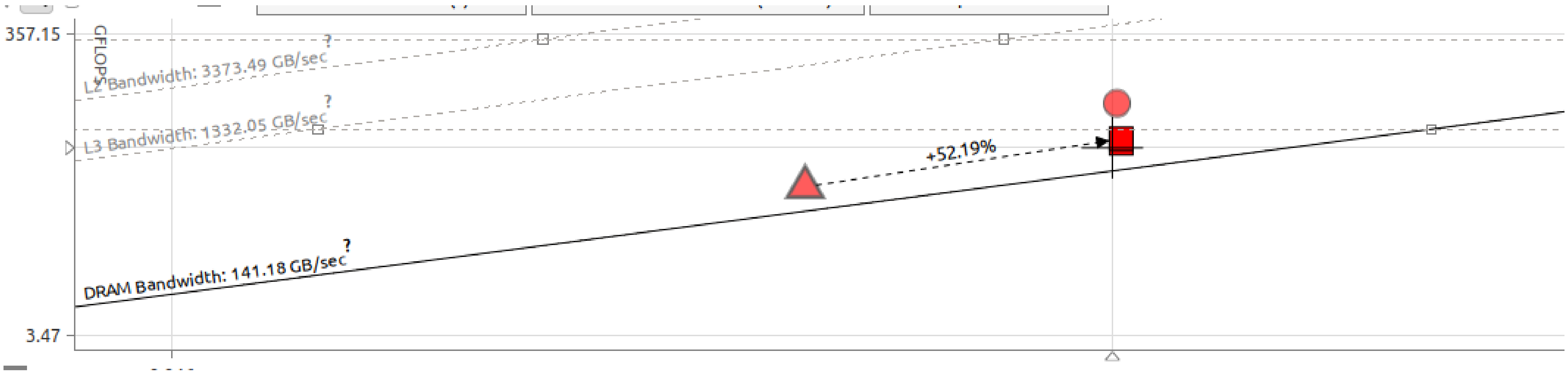
You can visualize the levels that need further optimization.



Roofline compare

Visualize multiple roofline charts on the same graph.

Test optimization strategies and see how much progress your are making.



New and Improved Summary

Program metrics

Elapsed Time	7.71s
Vector Instruction Set	AVX2, AVX
Number of CPU Threads	36

GFLOPS	61.89
GFLOP Count	477.428
FP Arithmetic Intensity	0.30422
GINTOPS	1.29

Overall metrics

Performance characteristics

Metrics	*Total*	
Total CPU time	167.50s	100%
Time in **2** vectorized loops	109.30s	65.3%
Time in scalar code	58.20s	34.7%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	6.87x	86%
Program Approximate Gain	4.83x	

Ratio of vectorized/unvectorized code

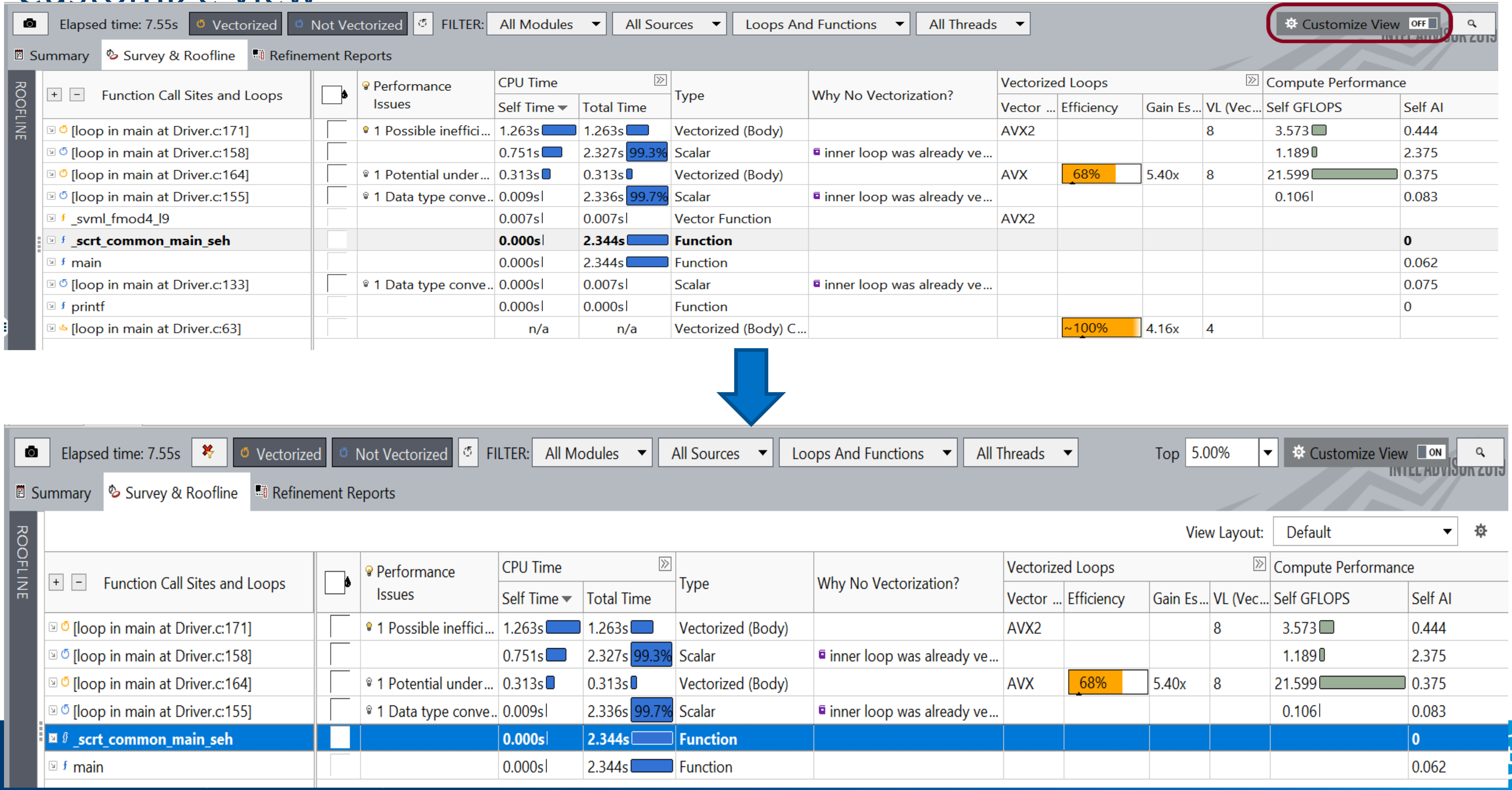
OP/S and Bandwidth

Effective OP/S And Bandwidth		*Utilization*		*Hardware Peak*
GFLOPS	**61.89**	4.7%	out of	1318 (DP) FLOPS
		2.3%	out of	2646 (SP) FLOPS
GINTOPS	**1.292**	0.19%	out of	662.8 (Int64) INTOPS
		0.097%	out of	1326 (Int32) INTOPS
CPU <-> Memory [L1+NTS GB/s]	**203.4**	1.6%	out of	12370 GB/s [bytes]
L2 Bandwidth [GB/s]	**105.9**	3.1%	out of	3374 GB/s [cacheline bytes]
L3 Bandwidth [GB/s]	**69.07**	5.2%	out of	1332 GB/s [cacheline bytes]
DRAM Bandwidth [GB/s]	**14.88**	11%	out of	141.2 GB/s [cacheline bytes]

Informations on Operations and Memory transfers

Column Configurator

Customize view

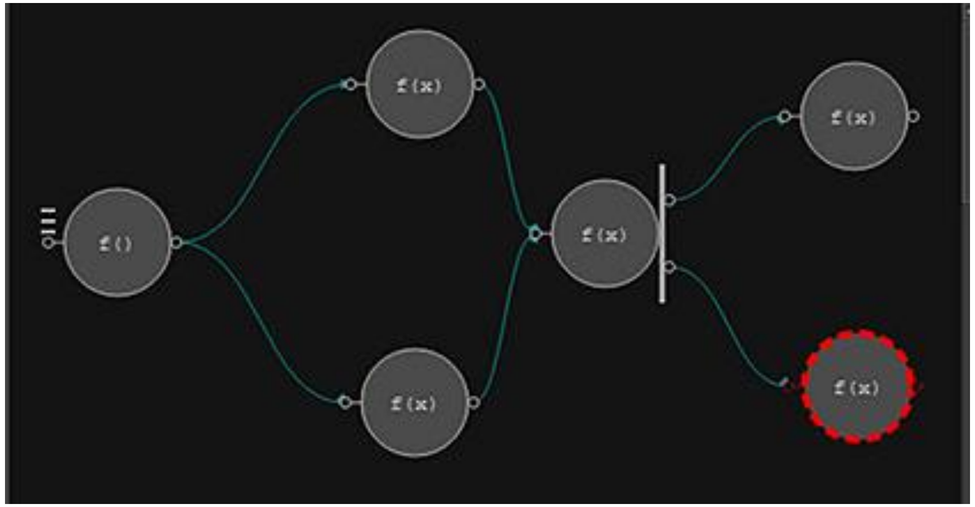


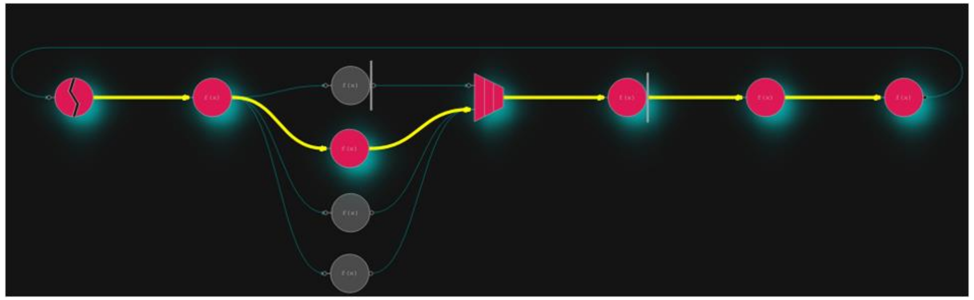
Visualize Parallelism—Interactively Build, Validate & Analyze Algorithms

Intel® Advisor—Flow Graph Analyzer (FGA)

Design mode

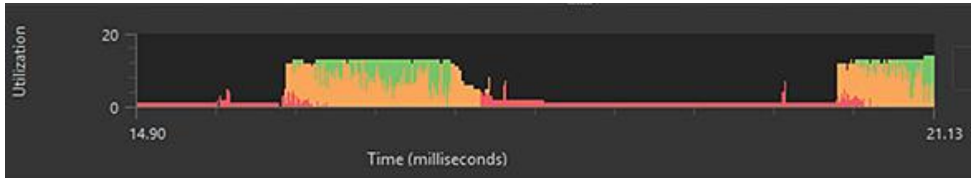
- Allows you to create a graph topology interactively
- Validate the graph and explore what-if scenarios
- Add C/C++ code to the node body
- Export C++ code using Threading Building Blocks (TBB) flow graph API





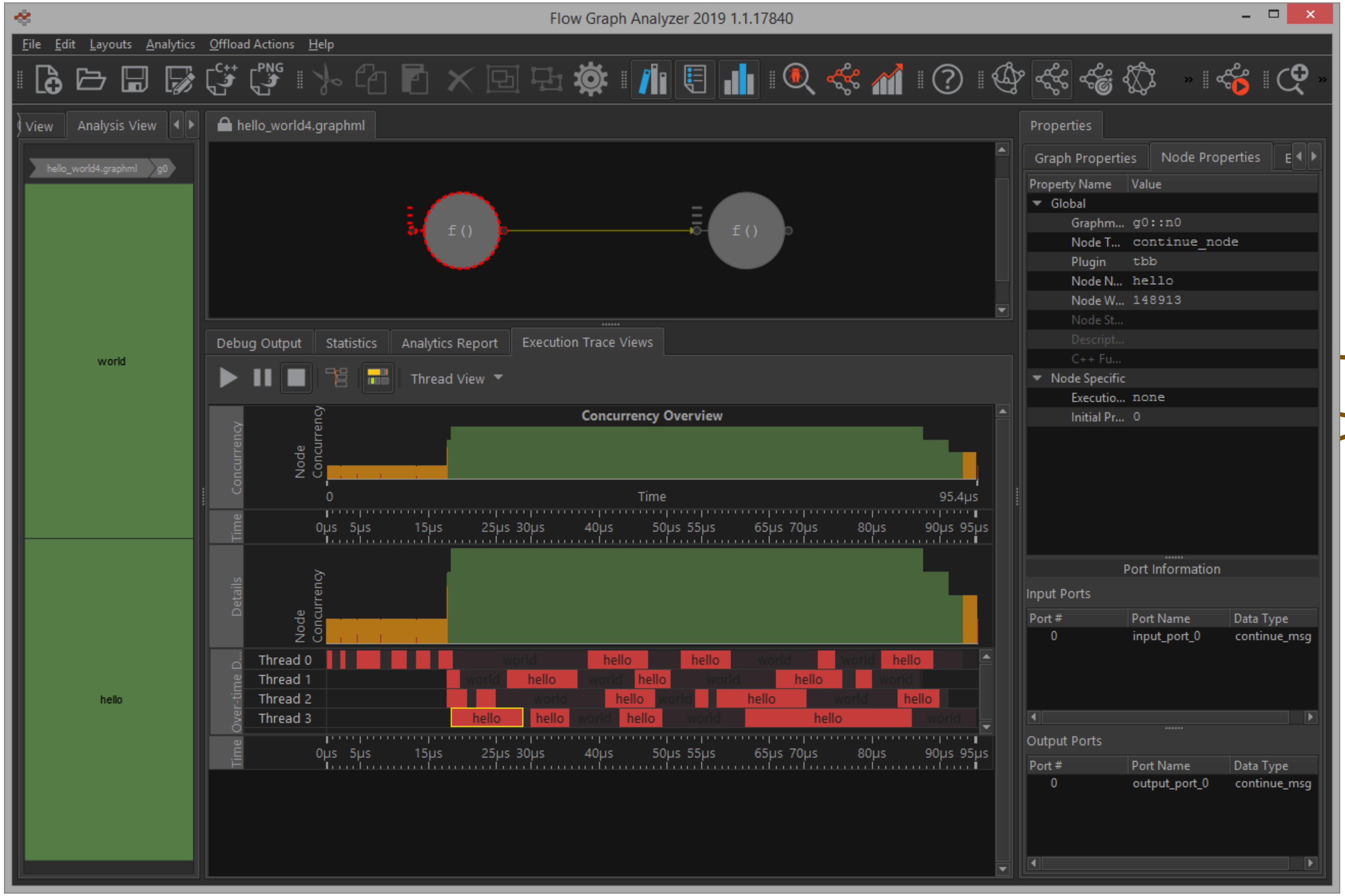
Analysis mode

- Compile your application (with tracing enabled)
- Capture execution traces during the application run
- Visualize/analyze in Flow Graph Analyzer



Use Intel® TBB or OpenMP* 5 (draft) OMPT APIs

Intel® Advisor – Flow Graph Analyzer (Analysis mode)



Trace program execution

Show correlation

Trace Playback

How to set it up (command line) ?

Before running an analysis, run:

- $ export ADVIXE_EXPERIMENTAL=int_roofline
- Use –integrated option

Run the survey

- advixe-cl -collect survey -integrated ……. -- ./my_application param1 param2 …

Run the trip count and flop

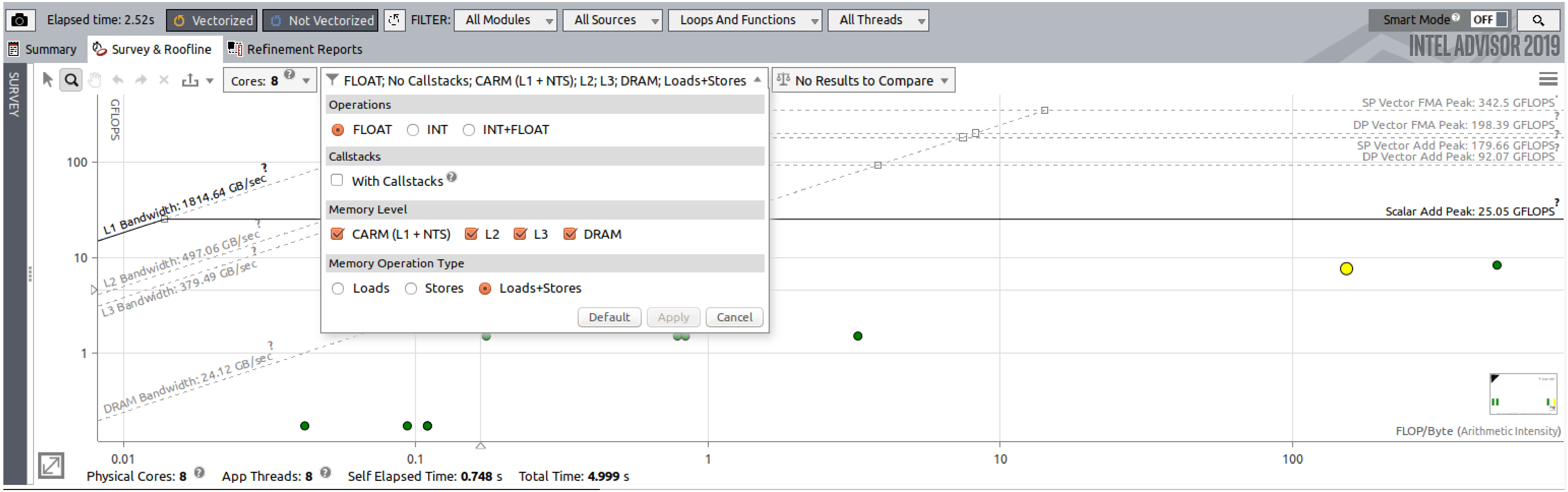
- advixe-cl –collect tripcounts -flop –integrated -enable-cache-simulation ……. -- ./my_application param1 param2 …

- ## Or Run Roofline analysis

- advixe-cl –collect roofline -integrated… -- ./my_application param1 param2

How to display the Integrated Roofline chart

You can select which memory level you want to display. Each memory level selected display an additional dot for every kernel. Each dot of the same kernel has the same performance but different Arithmetic Intensities. Here we selected CARM, L2, L3 and DRAM

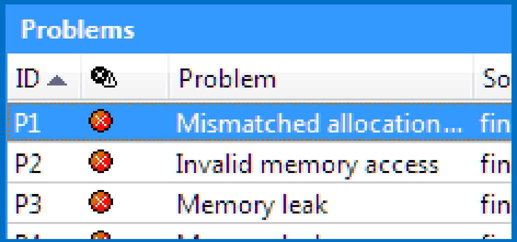


INTEL® INSPECTOR

Memory and thread debugger

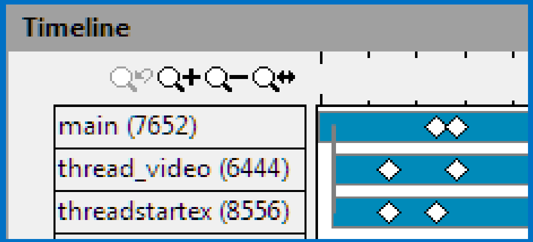
Motivation for Intel® Inspector

Memory Errors



- Invalid Accesses
- Memory Leaks
- Uninitialized Memory Accesses

Threading Errors



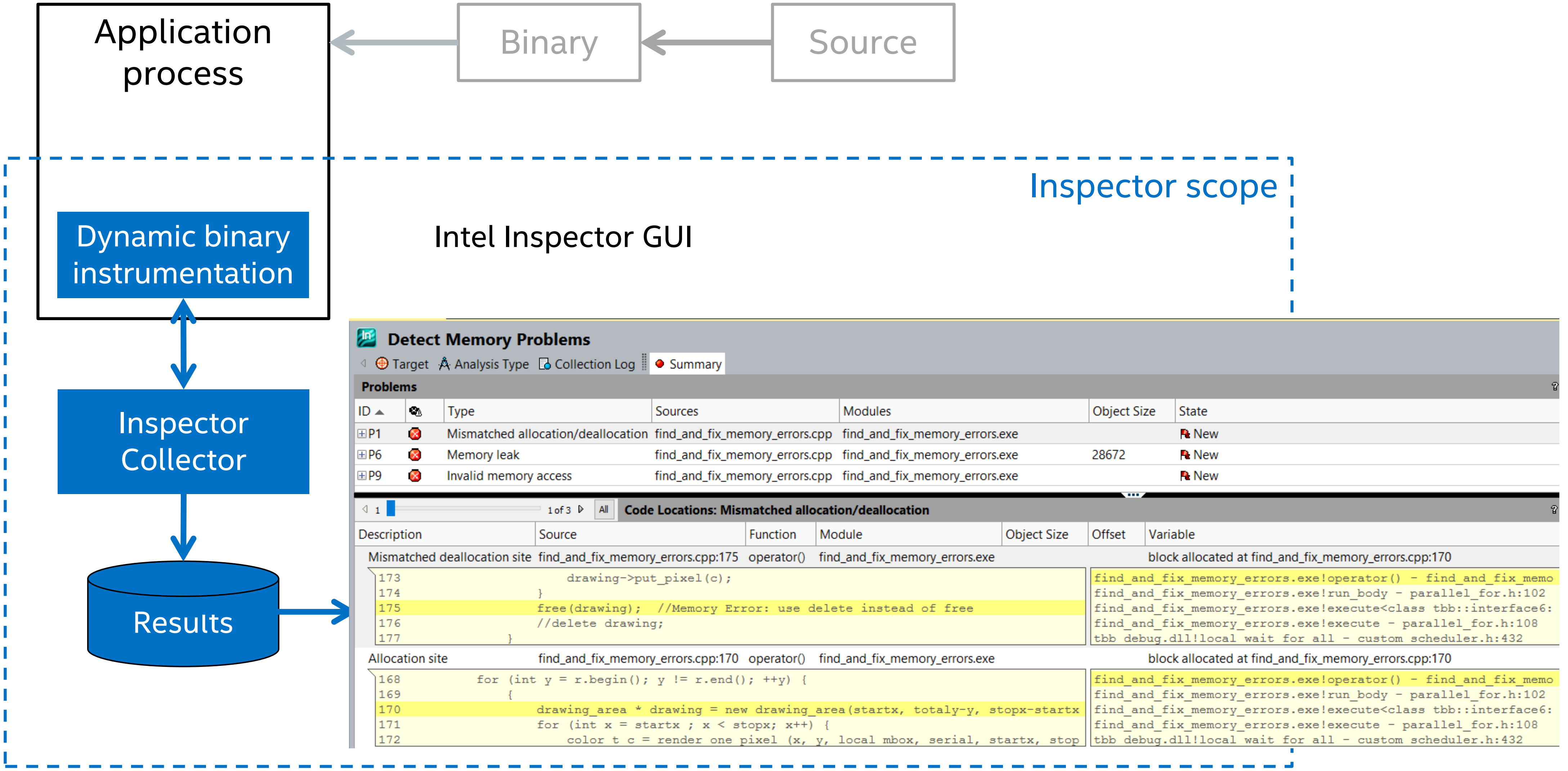
- Data Races
- Deadlocks
- Cross Stack References

Multi-threading problems

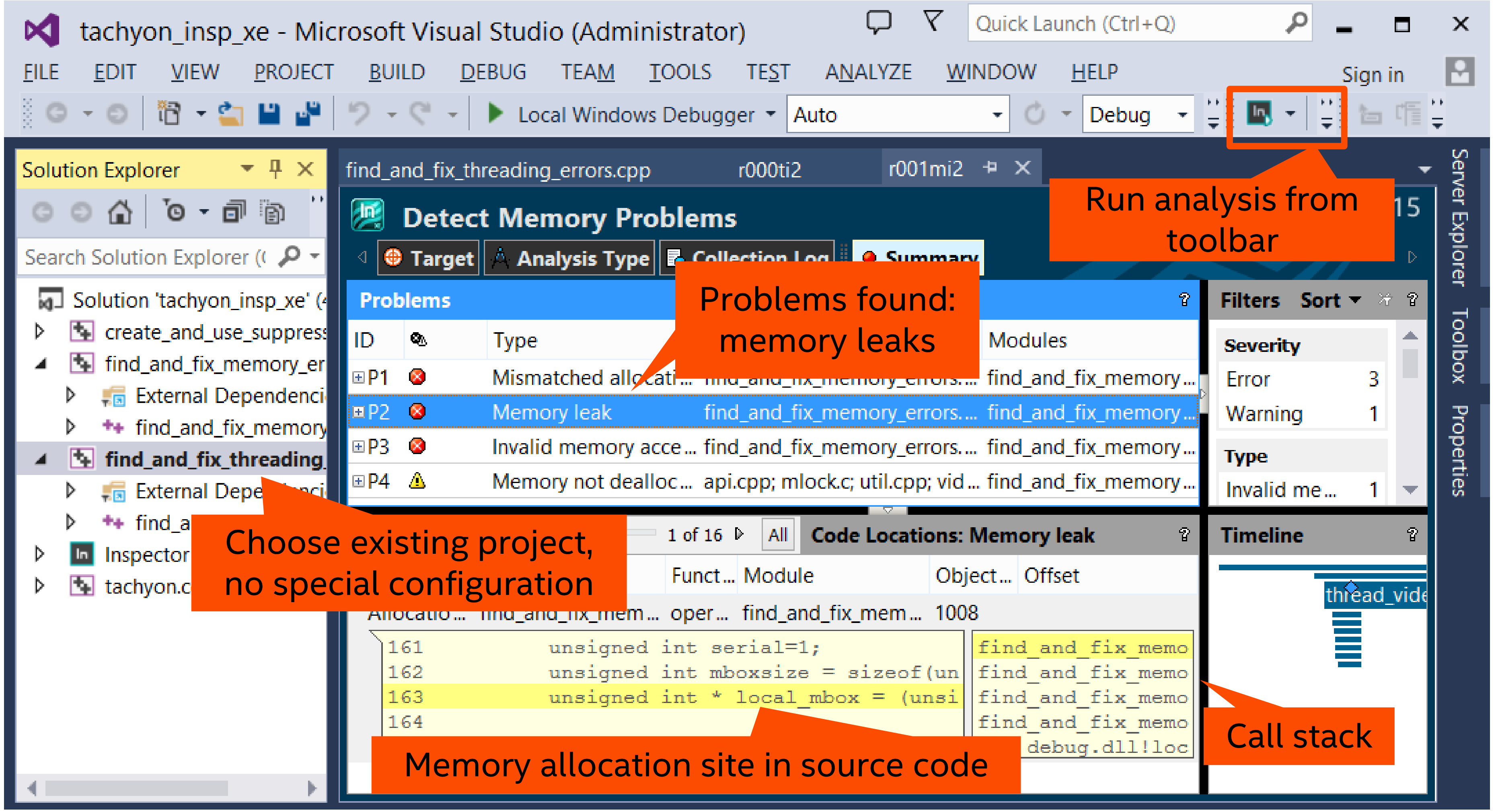
- Hard to reproduce,
- Difficult to debug
- Expensive to fix

Let the tool do it for you

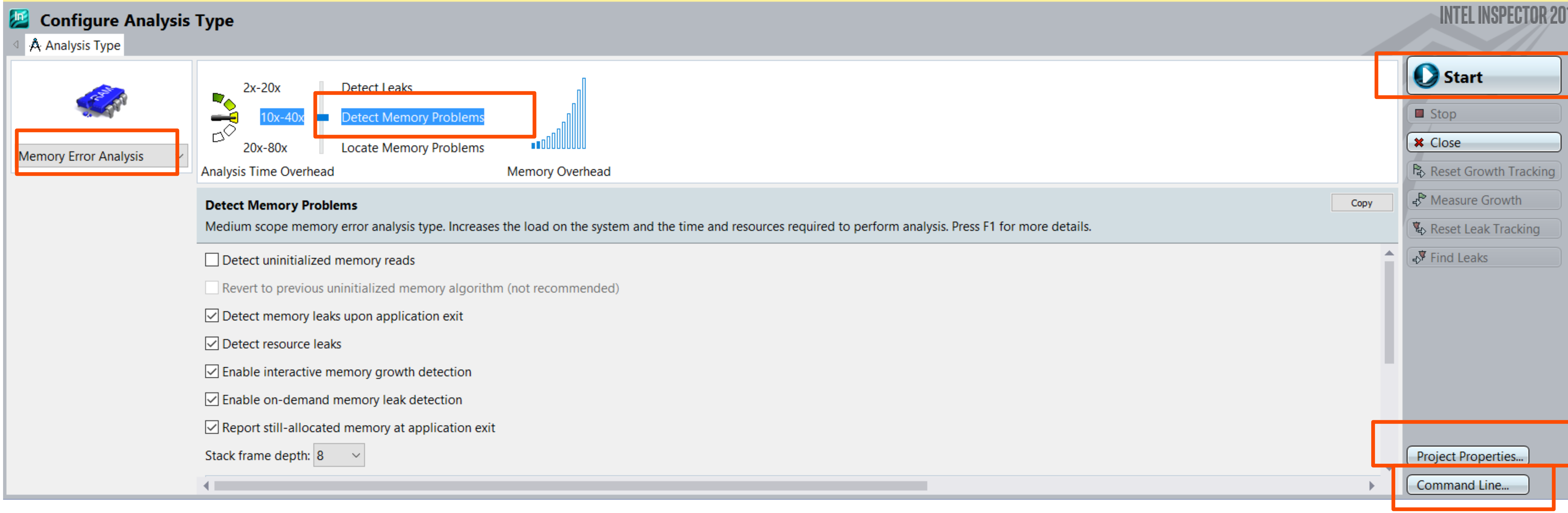
Intel® Inspector: Dynamic Analysis



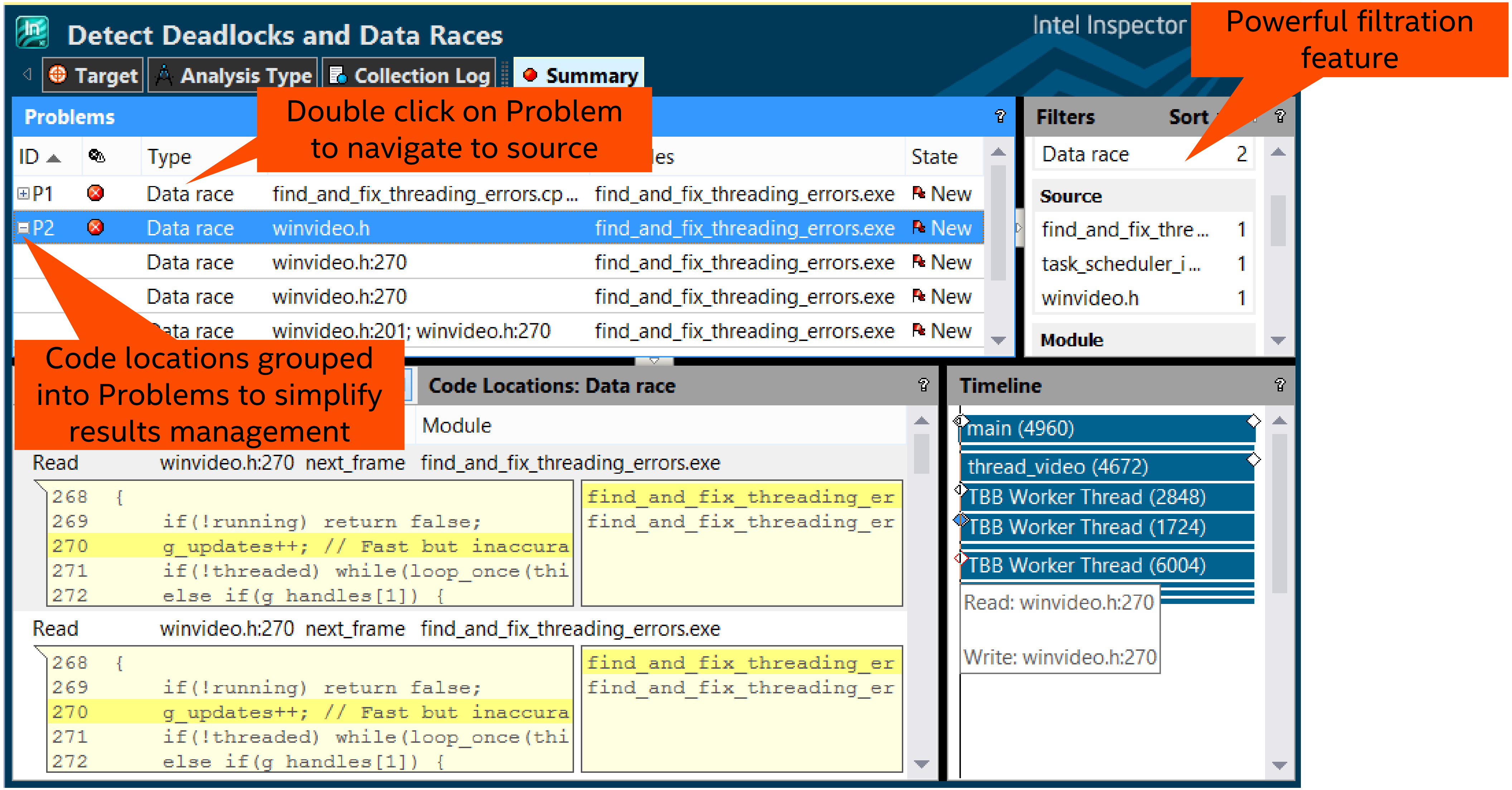
How it Looks: Visual Studio* Integration



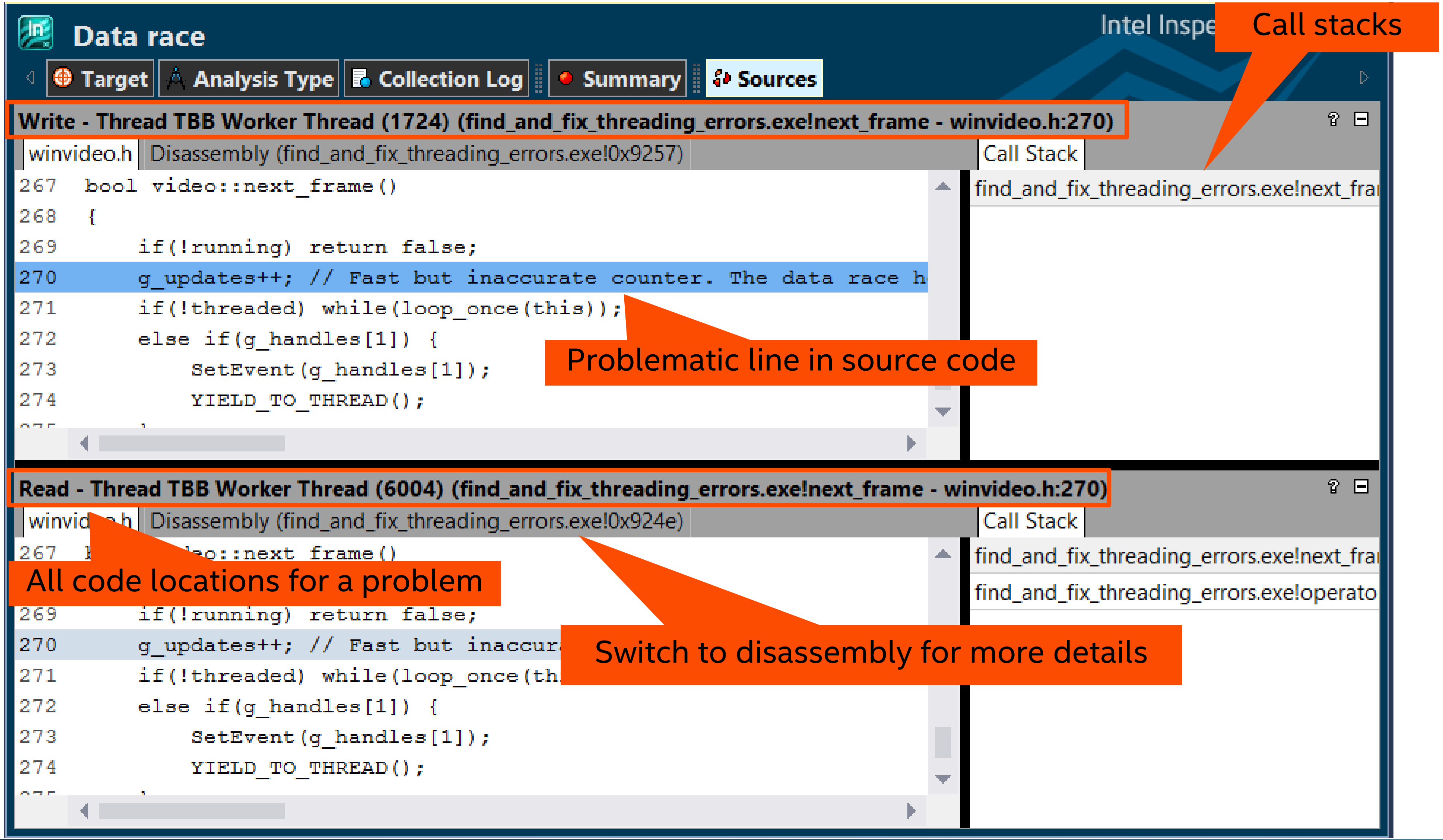
Standalone GUI: Windows* and Linux*



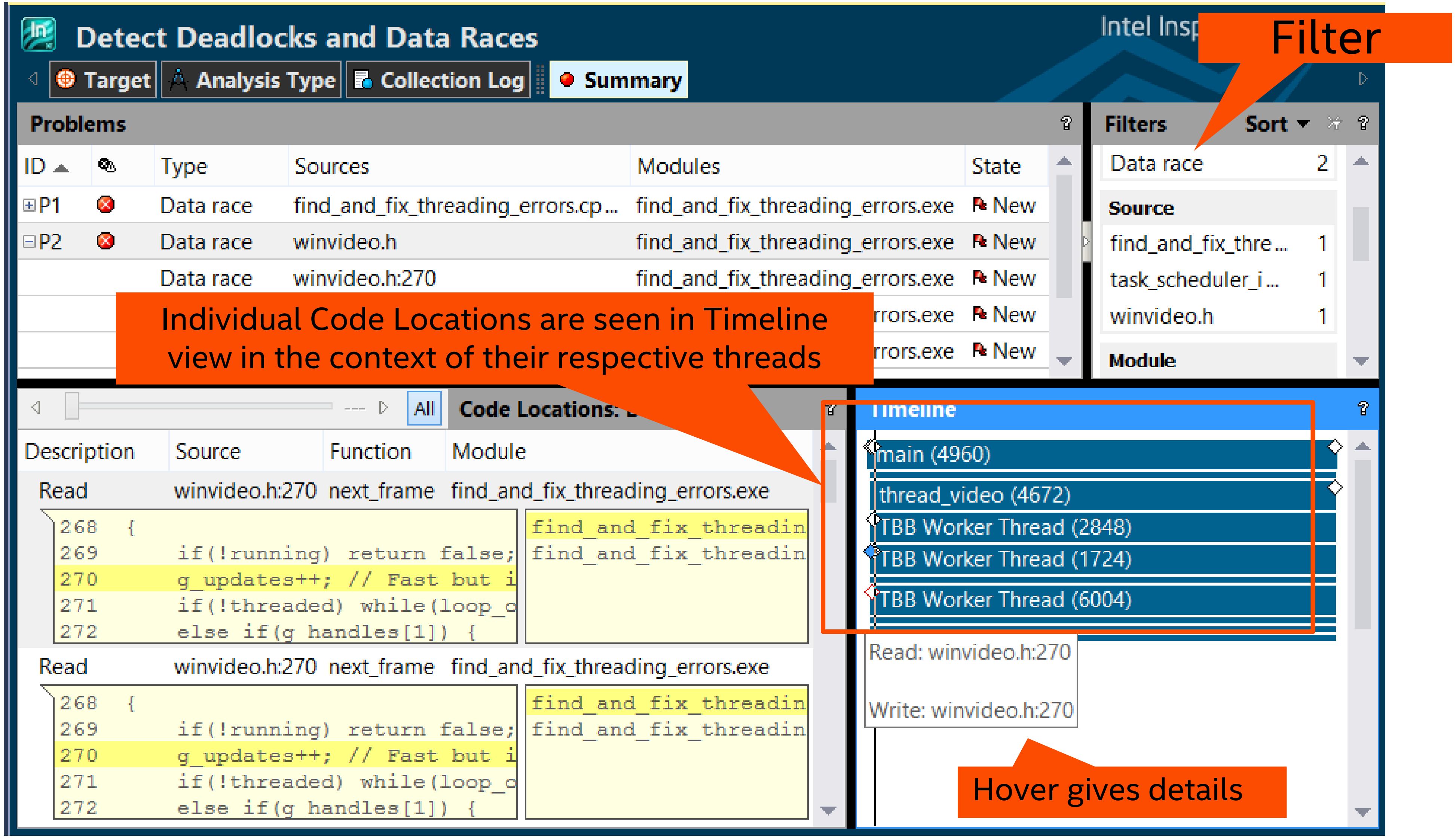
Workflow: manage results



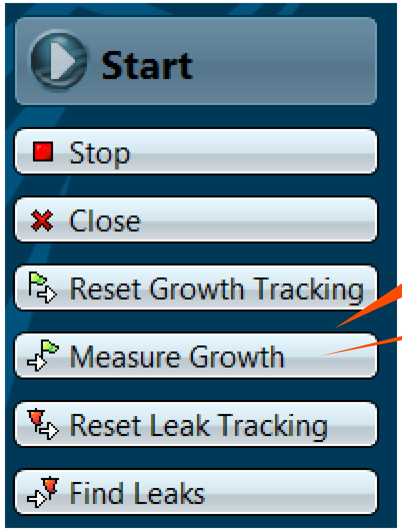
Workflow: navigate to sources



Workflow: timeline view



Analyze Memory Growth



During Analysis:

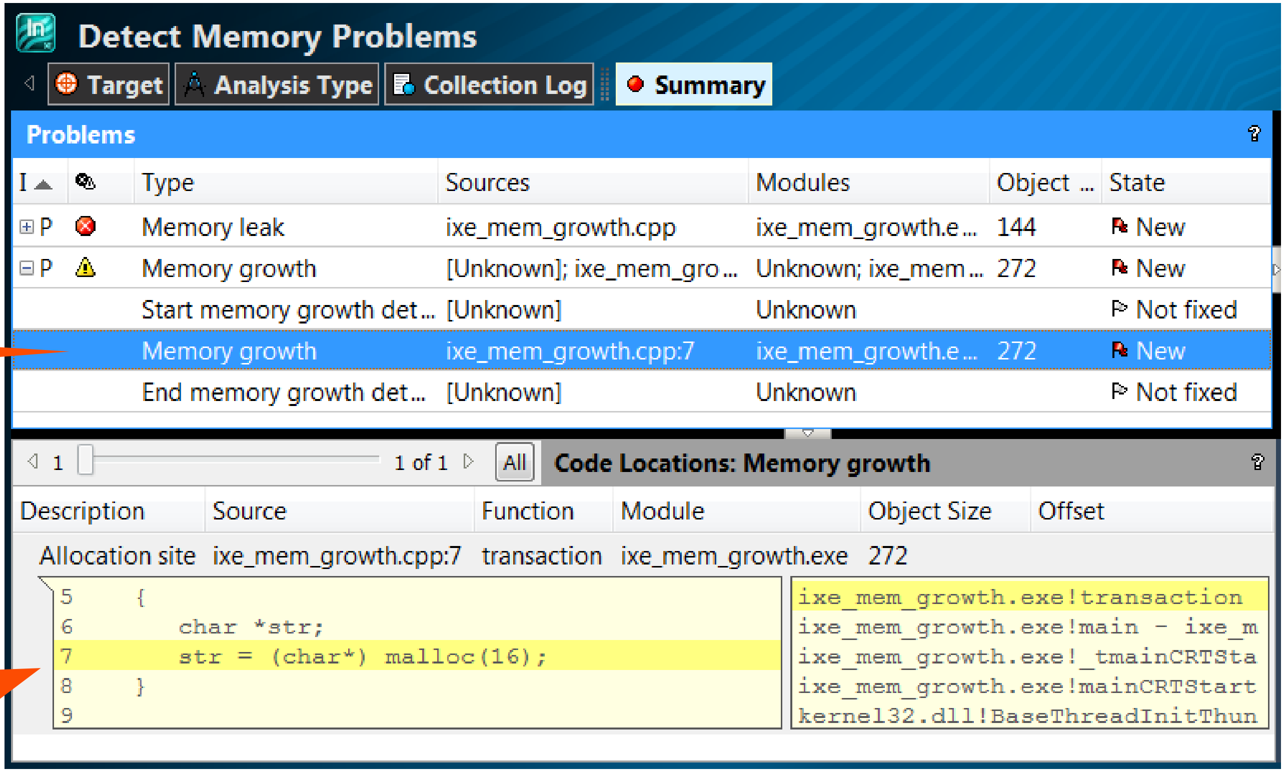
Set Start Point

Set End Point

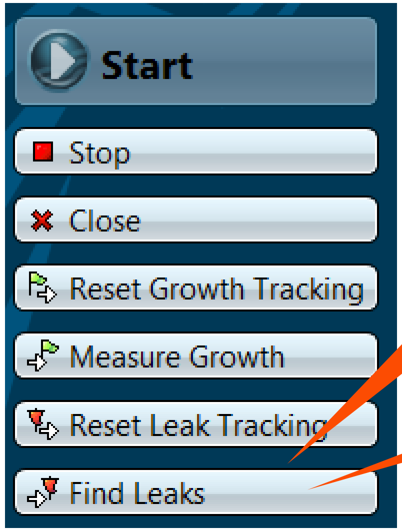
Analysis Results:

Memory Growth Problem Set

Code location for each block of memory that was allocated but not de-allocated during the time period



On-demand leak detection



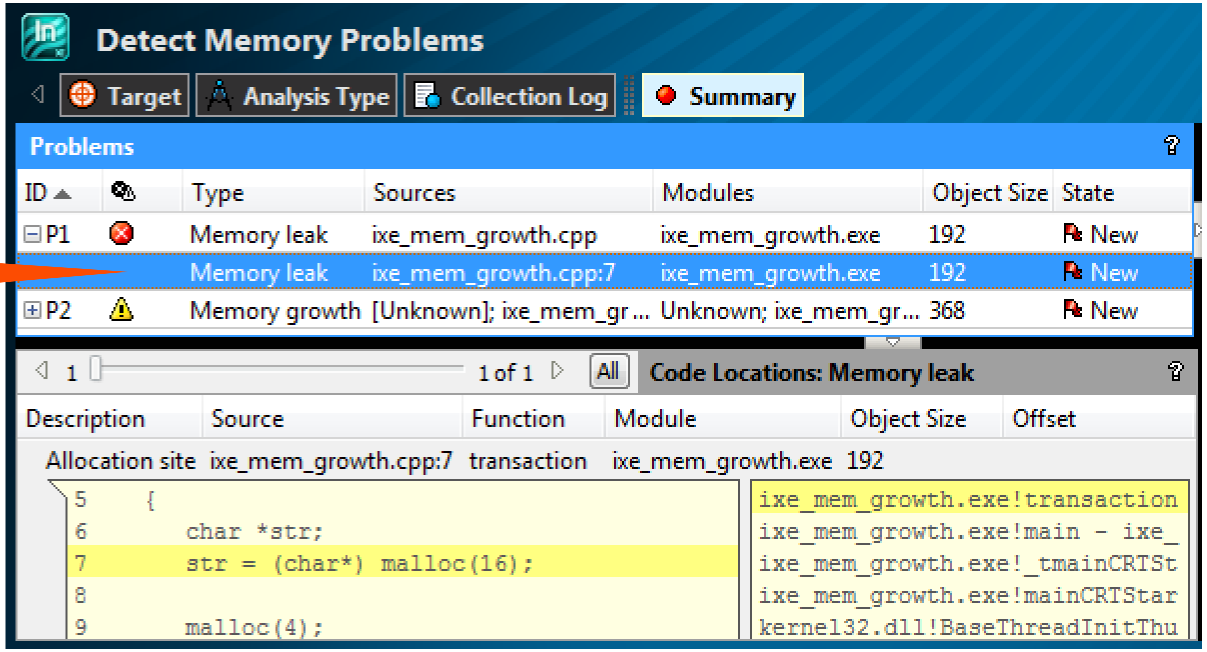
Set Start Point

Set End Point

- Check code regions between points 'A' and 'B' for leaks
- Check daemon processes for leaks
- Check crashing processes for leaks

Analysis Results:

Memory Leak shown during run time



Define analysis scope in source code

```
#include <ittnotify.h>

void ProcessPipeline()
{
     __itt_heap_reset_detection(__itt_heap_leaks);  // Start measuring memory leaks
     pipeline_stage1();                             // Run pipeline stage 1
     __itt_heap_record(__itt_heap_leaks);           // Report leaks in stage 1

     DoSomeOtherWork();

     __itt_heap_reset_detection(__itt_heap_growth); // Start measuring memory growth
     pipeline_stage2();                                                      // Run
     pipeline stage 2
     __itt_heap_record(__itt_heap_growth);              // Report memory growth in
     stage 2
}
```

Correctness analyses overhead

Inspector XE tracks

- Thread and Sync APIs
- Memory accesses

Inspector performs binary instrumentation using PIN

- Dynamic instrumentation system provided by Intel (http://www.pintool.org)
- Injected code used for observing the behavior of the running process
- Source modification/recompilation is not needed

Increases execution time and memory consumed (potentially significantly)

The Inspector XE dilates both time and memory consumed significantly!

Workload guidelines

Use small data set

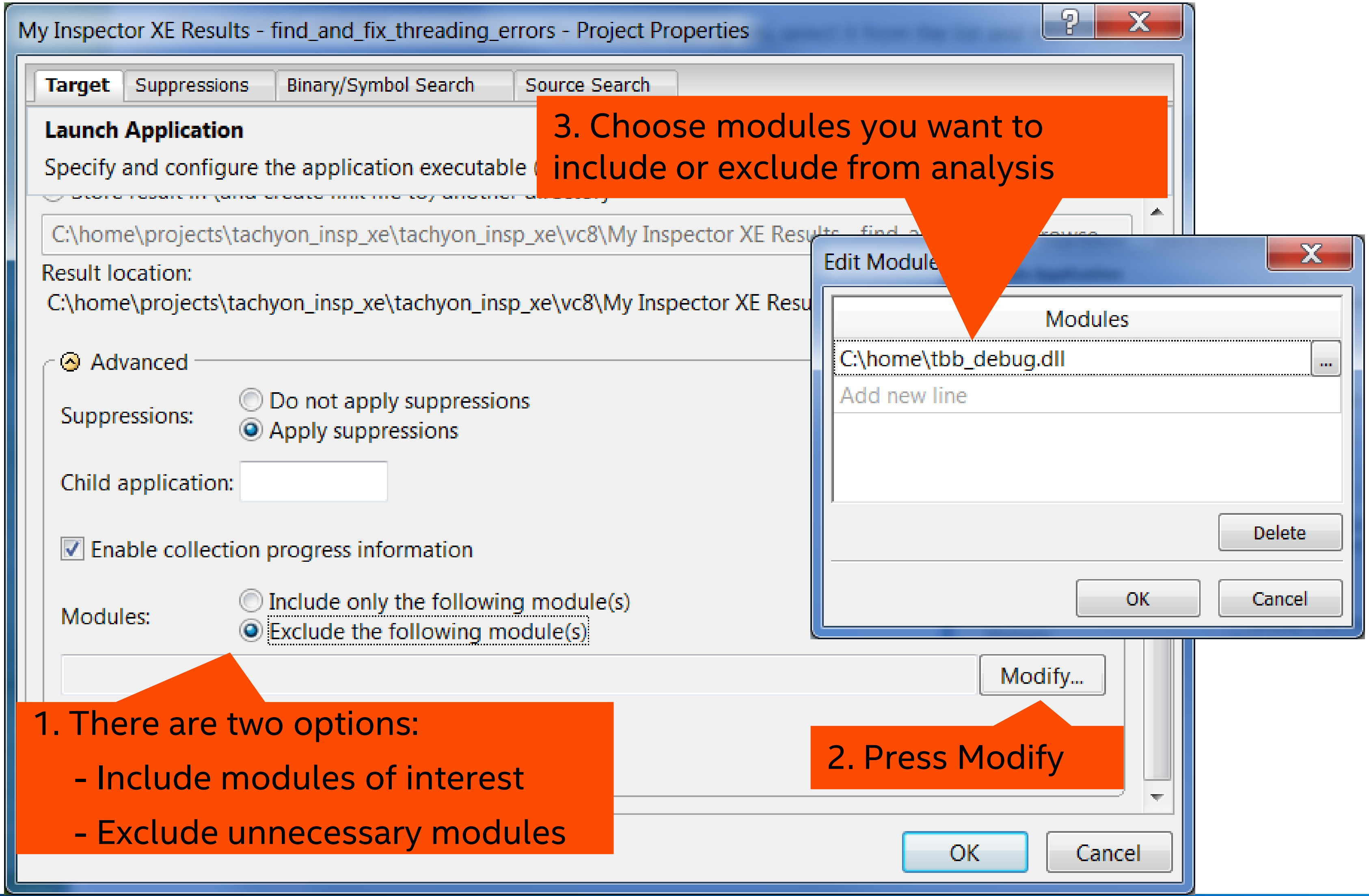
- Smaller number of threads
- Minimize data set size (e.g. smaller image sizes)
- Minimize loop iterations or time steps
- Minimize update rates (e.g. lower frames per second)

Use small but representative data set

- Only **actually executed** code paths are analyzed

Scale down workload to speed up analysis!

Include and Exclude modules



Debugger integration

Break into debugger

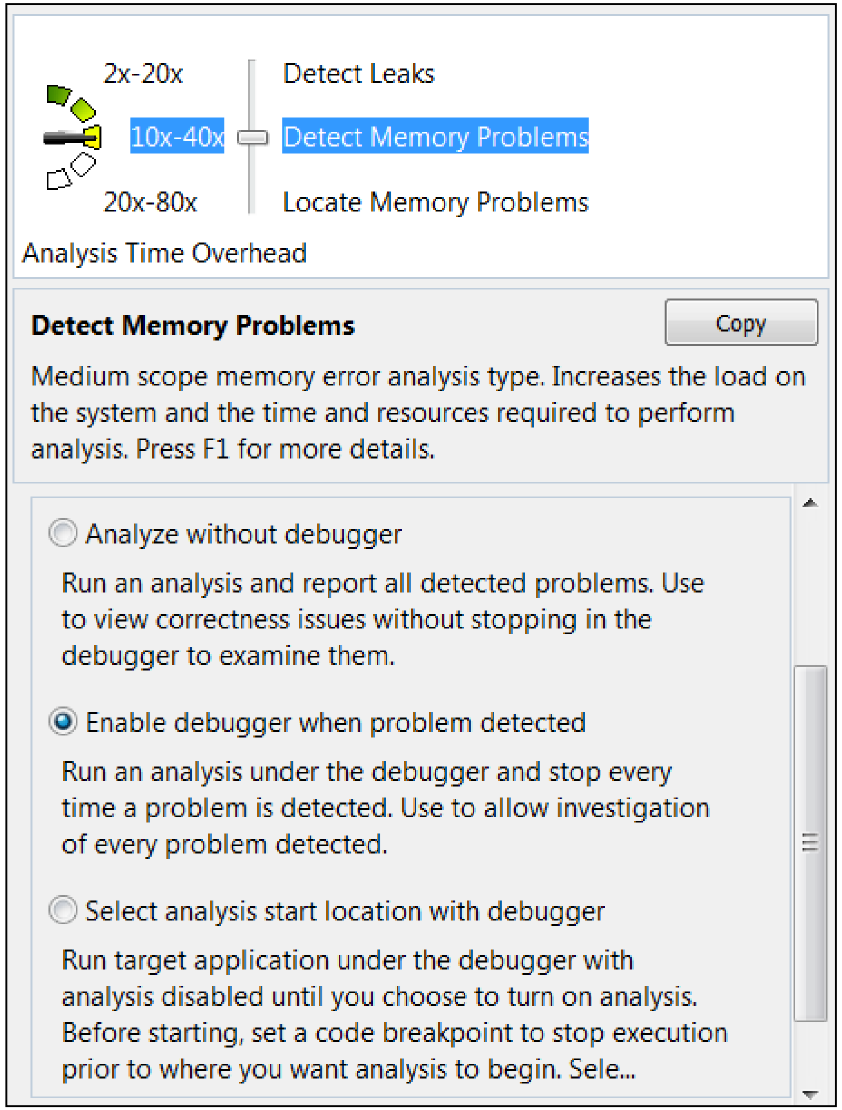
- Analysis can stop when it detects a problem
- User is put into a standard debugging session

Windows*

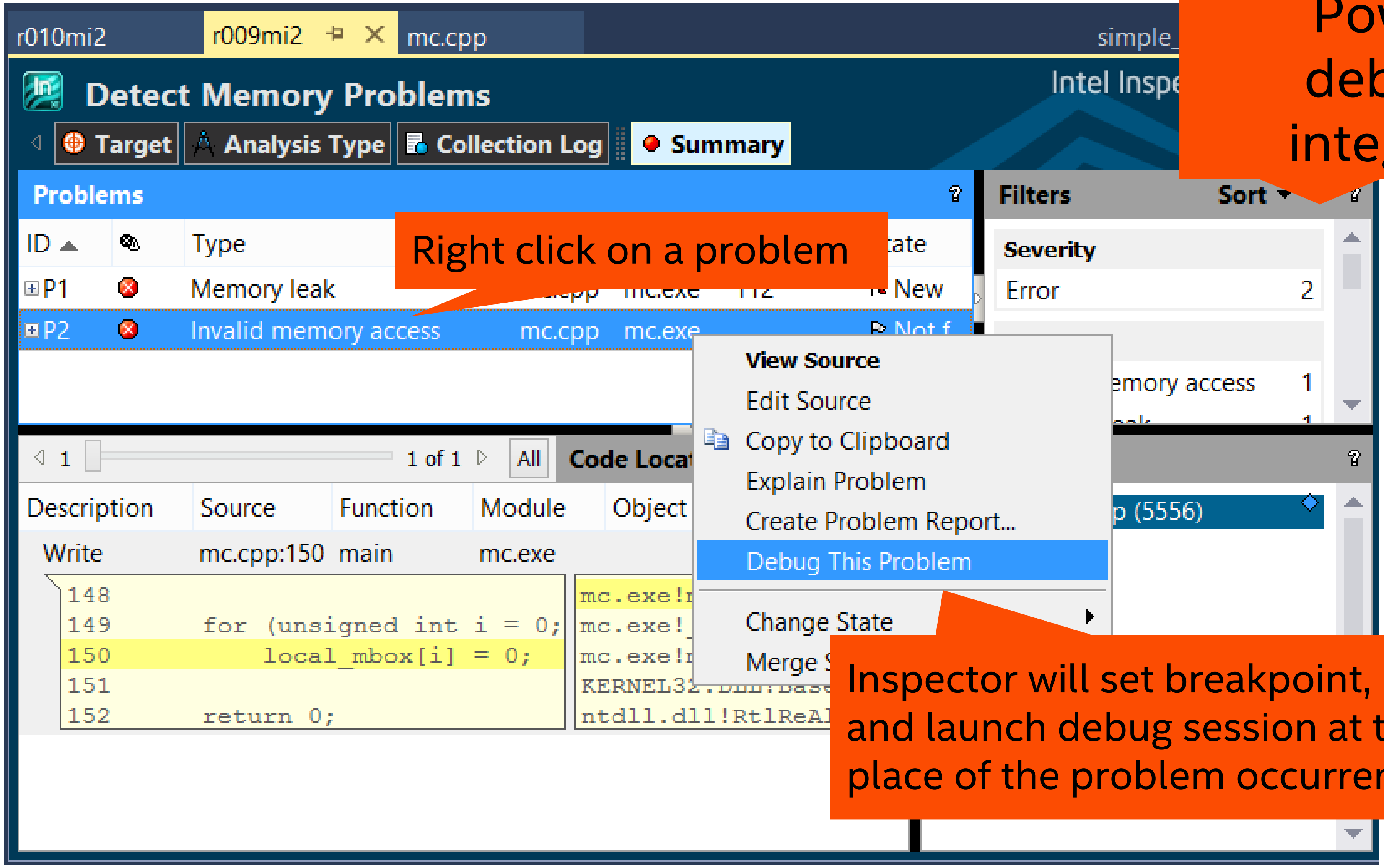
- Microsoft* Visual Studio Debugger

Linux*

- gdb



Debug this problem

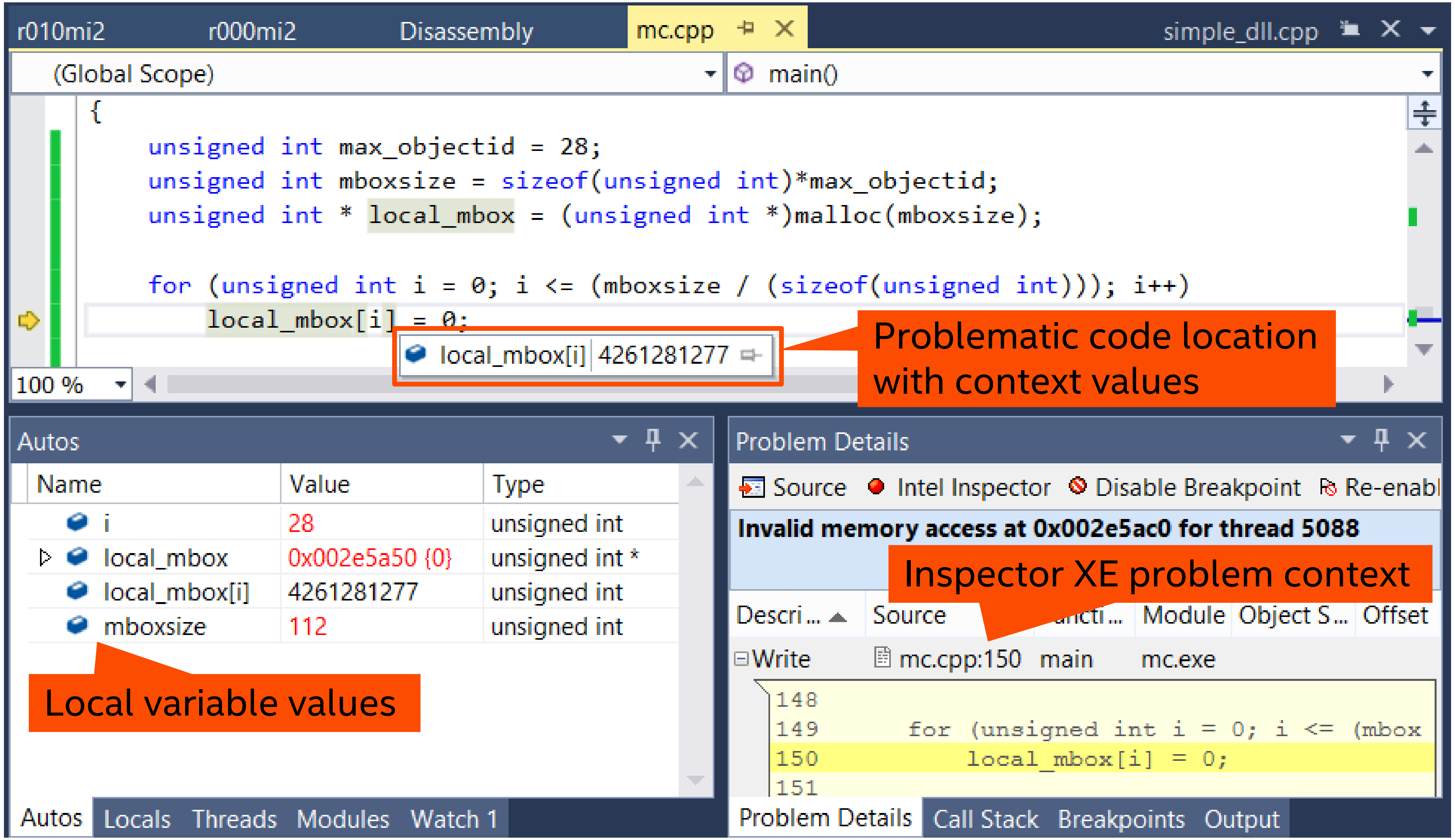


Powerful debugger integration

Right click on a problem

Inspector will set breakpoint, and launch debug session at the place of the problem occurrence

Debug this problem



Collect results and create baseline

```
inspxe-cl -collect mi1 -r r002mi1 -- D:\tests\my_app.exe

inspxe-cl -collect mi1
-module-filter module1.dll,module2.dll -module-filter-
   mode exclude -- D:\tests\my_app.exe

inspxe-cl -collect mi1 -executable-of-interest
   mem_error.exe -- D:\tests\startup_script.bat

inspxe-cl -create-suppression-file “D:\tests\mySup”
   -result-dir r002mi1

inspxe-cl -collect mi1 -suppression-file "D:\tests\mySup"
   -- D:\tests\my_app.exe

inspxe-cl -collect mi1 -baseline-result mi1_base --
   D:\tests\my_app.exe
```

Intel Inspector: User APIs

Enable you to

- Control collection, limit analysis scope
- Specify non-standard synchronization primitives
- Specify custom memory allocation primitives

To use user APIs:

- Include `ittnotify.h`, located at `<install_dir>/include`
- Insert `__itt_*` notifications in your code
- Link to the `libittnotify.lib` file located at `<install_dir>/ <lib32|lib64>`
- Available for C/C++ and Fortran

Collection control APIs

API	Description
`void __itt_suppress_push( unsigned int etype)`	Stop analyzing for errors on the current thread
`void __itt_suppress_pop ( void)`	Resume analysis
`void __itt_suppress_mark_range ( __itt_suppress_mode_t mode, unsigned int etype, void * address, size_t size);`	Suppress or unsuppress error detection for the specific memory range (object).
`void __itt_suppress_clear_range ( __itt_suppress_mode_t mode, unsigned int etype, void * address, size_t size);`	Clear the marked memory range

Using the Intel® Inspector with MPI

Compile the inspector_example.c code with the MPI scripts

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

```
mpirun -n 4 inspxe-cl --result-dir insp_results
   -collect mi1 -- ./insp_example.exe
```

Output is: a results directory for each MPI rank in the job

`ls | grep inspector_results` on Linux

Launch the GUI and view the results for each particular rank

`inspxe-gui inspector_results.<rank#>` on Linux

INTEL® PARALLEL STUDIO XE CLUSTER EDITION OVERVIEW

For Distributed Performance

Intel® Parallel Studio XE

Profiling, Analysis, and Architecture

Intel® Inspector
Memory and Threading Checking

Intel® VTune™ Amplifier
Performance Profiler

Intel® Advisor
Vectorization Optimization and Thread Prototyping

Cluster Tools

Intel® Cluster Checker
Cluster Diagnostic Expert System

Intel® Trace Analyzer and Collector
MPI Profiler

Intel® MPI Library

Performance Libraries

Intel® Data Analytics Acceleration Library
Optimized for Data Analytics & Machine Learning

Intel® Math Kernel Library
Optimized Routines for Science, Engineering, and Financial

Intel® Integrated Performance Primitives
Image, Signal, and Compression Routines

Intel® Threading Building Blocks
Task-Based Parallel C++ Template Library

Intel® C/C++ and Fortran Compilers

Intel® Distribution for Python
Performance Scripting

Boost Distributed Application Performance with Intel® MPI Library

Performance, Scalability & Fabric Flexibility

Standards Based Optimized MPI Library for Distributed Computing

- Built on open source MPICH Implementation
- Tuned for low latency, high bandwidth & scalability
- Multi-fabric support for flexibility in deployment

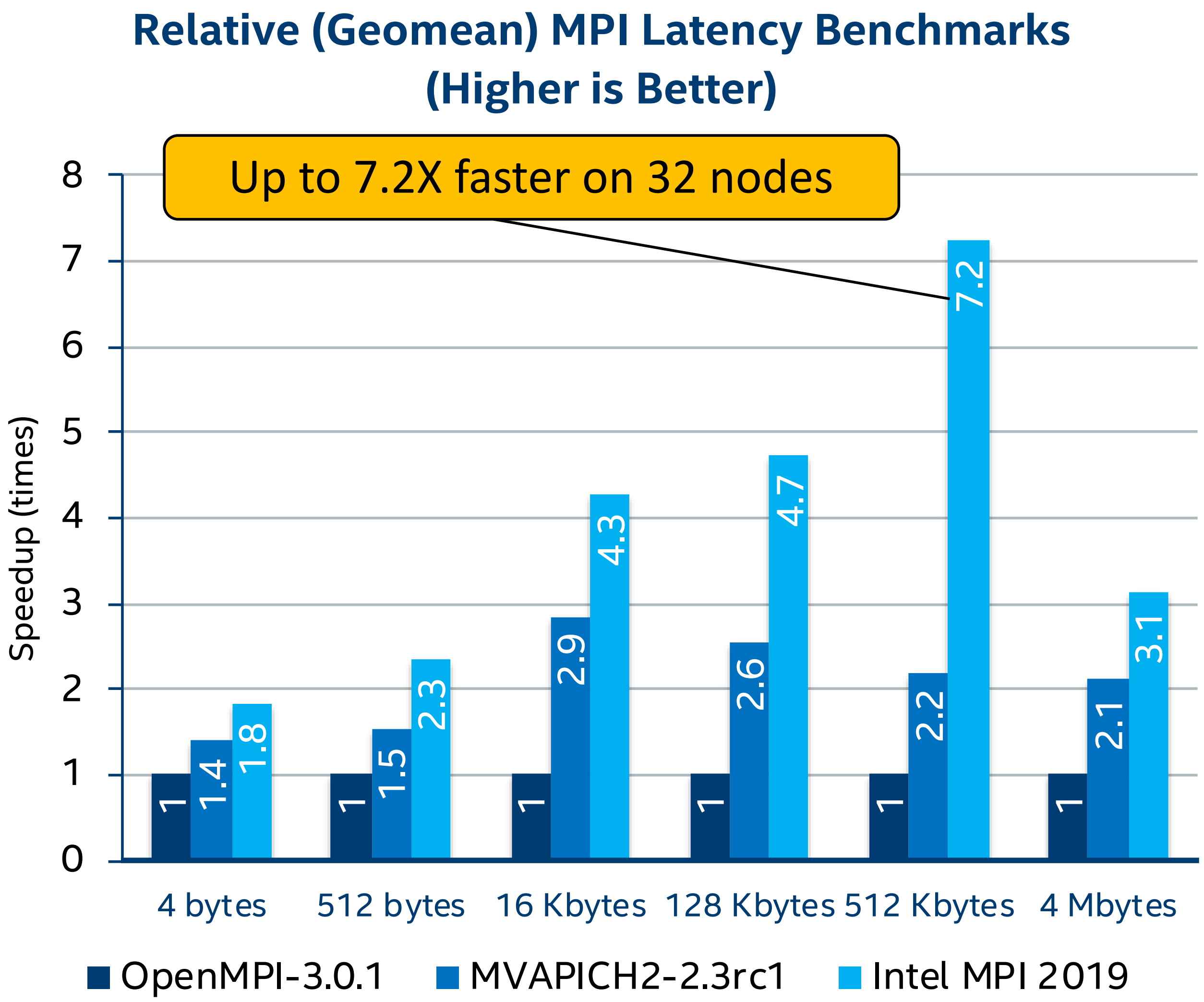
What's New in 2019 Release

- New MPI code base- MPI-CH4 (on the path to Exascale & beyond)
- Greater scalability & shortened CPU paths
- Superior MPI Multi-threaded performance
- Supports the latest Intel® Xeon® Scalable processor

Learn More: software.intel.com/intel-mpi-library

Superior MPI Performance with Intel® MPI Library 2019 on Linux* 64

1,280 Processes, 32 Xeon nodes (Intel® Omni-Path) Linux* 64



Performance results are based on testing as of Sept. 5, 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. 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. For more complete information, see Performance Benchmark Test Disclosure.

Configuration: Testing by Intel as of Sept. 5, 2018.Hardware: Intel® Xeon®Gold 6148 CPU @ 2.40GHz; 192 GB RAM. Interconnect: Intel® Omni-Path Host Fabric InterfaceSoftware: RHEL* 7.4; IFS 10.7.0.0.145; Libfabric internal; Intel® MPI Library 2019; Intel® MPI Benchmarks 2019 (built with Intel® C++ Compiler XE 18.0.2.199 for Linux*);

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. For more complete information about compiler optimizations, see our Optimization Notice.

Command Line Argument Set

Global option

First Host

Second Host

```
$ mpirun –genv OMP_NUM_THREADS 4 –n 6 –host node1 ./exe1 : -n 4 –host node2 ./exe2 :
–n 6 –host node4 ./exe4
```

Third Host

May set both global and local variables for each host

No limit to number of different host or executables

For high numbers of hosts a configuration file is more convenient…

Configuration File

Configuration file allows flexibility and automation

Notice commented out line – simple to change host assignment

```
$ cat theconfigfile
-genv OMP_NUM_THREADS 4
-n 6 –host node1 ./exe1
-n 4 –host node2 ./exe2
# -n 4 –host dead_node3 ./exe3
-n 6 –host node4 ./exe4
```

Launching job is straightforward

```
$ mpirun –configfile theconfigfile
```

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 *[recommended]*
- 5 – Display Intel® MPI Library environment variables
- 6 – Collective operation algorithm controls

I_MPI_HYDRA_DEBUG=1

Turns on Hydra debug output

- Extremely verbose output
- Only turn on if needed

Fabric Selection (v2018 & earlier)

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

- shm – Shared Memory (only valid for intranode)
- dapl – Direct Access Provider Library*
- ofa – Open Fabric Alliance (OFED* verbs)
- tmi – Tag Matching Interface
- tcp – Ethernet/Sockets
- ofi – OpenFabrics Interfaces*

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

Fabric Selection (v2018 vs v2019)

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

- ofi – OpenFabrics Interfaces*

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

Intel® MPI Support of Hybrid Codes

Intel® MPI is strong in mapping and pinning support for MPI processes

Sophisticated defaults or user controlled:

- For pure MPI codes use `I_MPI_PIN_PROCESSOR_LIST`
- For hybrid codes (default, takes precedence over I_MPI_PIN_PROCESSOR_LIST):

  `I_MPI_PIN_DOMAIN` =*<size>[:<layout>]*

<size> =	`omp`	Adjust to OMP_NUM_THREADS
	`auto`	#CPUs/#MPIprocs (default)
	`<n>`	Number
<layout> =	`platform`	According to BIOS numbering
	`compact`	Close to each other
	`scatter`	Far away from each other

Defines mapping and pinning for MPI processes, leaves room for threads on remaining cores!

NB: Naturally extends to hybrid codes on Intel® Xeon Phi™

Default Binding

```
$ export I_MPI_DEBUG=4
$ mpirun -machinefile ./hosts.txt -n 8 ./mpi_hello

[0] MPI startup(): Multi-threaded optimized library
[0] MPID_nem_ofi_init(): used OFI provider: psm2
...
[0] MPI startup(): shm and ofi data transfer modes
...
[0] MPI startup(): Rank   Pid     Node name  Pin cpu
[0] MPI startup(): 0      121023  node0      {0,1,2,3,4,5,6,7,8,9,40,41,42,43,44,45,46,47,48,49}
[0] MPI startup(): 1      121024  node0      {10,11,12,13,14,15,16,17,18,19,50,51,52,53,54,55,56,57,58,59}
[0] MPI startup(): 2      121025  node0      {20,21,22,23,24,25,26,27,28,29,60,61,62,63,64,65,66,67,68,69}
[0] MPI startup(): 3      121026  node0      {30,31,32,33,34,35,36,37,38,39,70,71,72,73,74,75,76,77,78,79}
[0] MPI startup(): 4      246334  node1      {0,1,2,3,4,5,6,7,8,9,40,41,42,43,44,45,46,47,48,49}
[0] MPI startup(): 5      246335  node1      {10,11,12,13,14,15,16,17,18,19,50,51,52,53,54,55,56,57,58,59}
[0] MPI startup(): 6      246336  node1      {20,21,22,23,24,25,26,27,28,29,60,61,62,63,64,65,66,67,68,69}
[0] MPI startup(): 7      246337  node1      {30,31,32,33,34,35,36,37,38,39,70,71,72,73,74,75,76,77,78,79}

Hi from MPI task 0
...
```

Fabric provider

Active transfer modes

Equal distribution of cores among MPI ranks

Using I_MPI_PIN_PROCESSOR_LIST

```
$ export I_MPI_DEBUG=4
$ export I_MPI_PIN_PROCESSOR_LIST=0,1,20,21
$ mpirun -machinefile ./hosts.txt -n 8 ./mpi_hello

[0] MPI startup(): Multi-threaded optimized library
[0] MPID_nem_ofi_init(): used OFI provider: psm2
...
[0] MPI startup(): shm and ofi data transfer modes
...
[0] MPI startup(): Rank   Pid    Node name  Pin cpu
[0] MPI startup(): 0      121023 node0      {0}
[0] MPI startup(): 1      121024 node0      {1}
[0] MPI startup(): 2      121025 node0      {20}
[0] MPI startup(): 3      121026 node0      {21}
[0] MPI startup(): 4      246334 node1      {0}
[0] MPI startup(): 5      246335 node1      {1}
[0] MPI startup(): 6      246336 node1      {20}
[0] MPI startup(): 7      246337 node1      {21}

Hi from MPI task 0
...
```

Fabric provider (no changes)

Active transfer modes (no changes)

Processes bound to specified CPUs, not floating

Using I_MPI_PIN_DOMAIN

```
$ export I_MPI_DEBUG=4
$ export I_MPI_PIN_DOMAIN=omp
$ export OMP_NUM_THREADS=10
$ mpirun -machinefile ./hosts.txt -n 8 ./mpi_hello

[0] MPI startup(): Multi-threaded optimized library
[0] MPID_nem_ofi_init(): used OFI provider: psm2
...
[0] MPI startup(): shm and ofi data transfer modes
...
[0] MPI startup(): Rank   Pid    Node name Pin cpu
[0] MPI startup(): 0      121023 node0     {0,1,2,3,4,40,41,42,43,44}
[0] MPI startup(): 1      121024 node0     {5,6,7,8,9,45,46,47,48,49}
[0] MPI startup(): 2      121025 node0     {10,11,12,13,14,50,51,52,53,54}
[0] MPI startup(): 3      121026 node0     {15,16,17,18,19,55,56,57,58,59}
[0] MPI startup(): 4      246334 node1     {0,1,2,3,4,40,41,42,43,44}
[0] MPI startup(): 5      246335 node1     {5,6,7,8,9,45,46,47,48,49}
[0] MPI startup(): 6      246336 node1     {10,11,12,13,14,50,51,52,53,54}
[0] MPI startup(): 7      246337 node1     {15,16,17,18,19,55,56,57,58,59}

Hi from MPI task 0 OMP thread 0
...
```

Set binding to OMP range

Each MPI task floats on OMP_NUM_THREADS logical processors

Careful in HT systems!

Intel® MPI Benchmarks 2019

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

- Added benchmarks to test new multi-threaded support (IMB-MT) and shared memory transport (IMB-P2P)

Measuring Communication Speed with IMB

The simplest benchmark in IMB is called PingPong

Data packages of different size are sent from rank 0 to rank 1 and back:

```
$ mpirun -n 2 IMB-MPI1 pingpong
```

```
#---------------------------------------------------
# Benchmarking PingPong
# #processes = 2
#---------------------------------------------------
       #bytes #repetitions      t[usec]   Mbytes/sec
            0         1000         1.50         0.00
            1         1000         1.49         0.67
            2         1000         1.49         1.34
            4         1000         1.47         2.71
            8         1000         1.47         5.43
           16         1000         1.69         9.46
           32         1000         1.68        19.04
           64         1000         1.67        38.22
          128         1000         1.73        73.84
          256         1000         1.75       146.12
          512         1000         1.82       281.40
         1024         1000         1.92       533.34
         2048         1000         2.16       949.27
         4096         1000         2.62      1561.88
         8192         1000         3.60      2274.88
        16384         1000         6.58      2490.74
        32768         1000         8.58      3819.34
        65536          640        16.51      3968.86
       131072          320        21.15      6197.71
       262144          160        32.07      8175.19
       524288           80        54.01      9707.91
      1048576           40        95.70     10956.77
      2097152           20       181.25     11570.38
      4194304           10       349.56     11998.90
```

What is mpitune?

Tool to optimize Intel® MPI Library settings

Walks through [application specific] search space of settings and tests performance

Writes out a configuration file with the best settings found

Works in two different modes:

- Cluster Specific
- Application Specific

Cluster Specific Tuning with mpitune

Find optimal values for library tuning knobs on the particular cluster or application environment.

- Run it once after installation and each time after a cluster configuration change
- Best configuration is recorded for each combination of communication device, number of nodes, MPI ranks and the process distribution model
- Configuration is stored in Intel ® MPI folders and available to all users

Collect configuration values:
$ mpitune [options]

Reuse recorded values:
$ mpirun –tune ./application

Application Specific Tuning with mpitune

Find optimal values for library tuning knobs on the particular cluster or application environment

- Run it for each application and after application- or cluster configuration change
- Best configuration is recorded for each combination of communication device, number of nodes, MPI ranks and the process distribution model
- Configuration is stored in user's home

Collect configuration values:
$ mpitune [options] \
--application \"mpirun application\"

Reuse recorded values:
$ mpirun –tune ./app.conf \
./application

Example: Cluster Specific MPITUNE

MPITUNE is an executable script. The easiest way is to simply run:

```
$ mpitune
```

We may restrict MPITUNE on full nodes and the default fabric

```
$ mpitune –pr 8:8 –fl shm:dapl
```

hosts should be taken from provided hostfile or the batch system

As the search space is very wide, you may want to limit the number of minimum and maximum ranks per node as well as the number of nodes and the fabric used :

```
$ mpitune -pr <min_ppn>:<max_ppn>      \
          -hr <min_nodes>:<max_nodes> \
          -fl <fabric>                \
          -a \”mpirun …\”
```

MPITUNE – Output

30'Aug'17 14:50:04 | TASK 128 from 273 : Spreadsheet for I_MPI_ADJUST_REDUCE_SCATTER option:

Message size (bytes)	Initial time	Times for algorithms				Tuned vs initial (ex)	Validated time	Validated vs initial (ex)
		Alg 1	Alg 2	Alg 3	Alg 4			
0	0.38	0.37	0.40	*0.35*	0.38	0.92X	0.39	1.03X
4	3.46	*3.51*	51.77	13.81	3.45	1.01X	3.44	0.99X
8	4.05	3.89*	52.01	13.78	*4.14	0.96X	4.02	0.99X
16	4.00	*3.97*	52.04	13.98	4.47	0.99X	4.12	1.03X
32	4.46	*4.42*	52.54	15.25	5.54	0.99X	4.41	0.99X
64	4.85	*4.84*	52.88	16.29	8.22	1.00X	4.84	1.00X
128	5.16	*5.15*	52.98	17.29	11.47	1.00X	5.24	1.02X
256	5.77	*5.67*	54.86	8.04	25.84	0.98X	5.76	1.00X
512	6.07	*5.96*	54.89	8.42	26.49	0.98X	5.81	0.96X
1024	9.14	*9.27	56.15	8.66*	27.60	0.95X	8.81	0.96X
2048	11.03	*11.29	57.73	8.86*	29.85	0.80X	9.16	0.83X
4096	11.26	*11.64	65.04	9.71*	33.99	0.86X	9.98	0.89X
8192	18.05	*18.01	73.03	11.41*	41.31	0.63X	11.57	0.64X
16384	28.02	*28.08	81.30	14.29*	55.85	0.51X	14.45	0.52X
32768	50.47	*50.27	160.89	20.71*	61.36	0.41X	21.04	0.42X
65536	93.98	*93.97	247.74	30.63*	82.26	0.33X	30.90	0.33X
131072	163.70	*164.00	325.70	59.16*	136.41	0.36X	62.99	0.38X
262144	460.85	742.38	*470.05	518.14	426.22*	0.92X	385.39	0.84X
524288	823.88	1746.14	*818.26	1368.98	601.41*	0.73X	565.56	0.69X
1048576	898.08	5993.07	3351.08	5261.42	*970.05*	1.08X	957.37	1.07X
2097152	2003.62	14957.62	6862.75	13167.39	*2115.74*	1.06X	2104.40	1.05X
4194304	3582.84	38222.43	*3671.79*	33904.29	4914.07	1.02X	3631.38	1.01X
AVG	n/a	n/a	n/a	n/a	n/a	0.84X	n/a	0.85X

MPITUNE – Result File

```
$ cat mpiexec_shm_nn_1_np_32_ppn_32.conf

-genv I_MPI_ADJUST_BCAST '1:0-0;10:1-2;8:3-20;10:21-32;1:33-64;9:65-
192;10:193-2582;9:2583-4096;11:4097-25206;10:25207-65536;11:65537-
131072;9:131073-524288;10:524289-1048576;9:1048577-2715592;1:2715593-
2147483647'
-genv I_MPI_ADJUST_BARRIER '7'
-genv I_MPI_ADJUST_GATHER '1:0-1;4:2-12;1:13-64;4:65-947;3:948-2147483647'
```

Note that results files may be empty if the default settings are optimal.

PROFILE & ANALYZE HIGH PERFORMANCE MPI APPLICATIONS WITH INTEL® TRACE ANALYZER & COLLECTOR 2019

Profile, Analyze & Visualize MPI Applications

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

Intel® Trace Analyzer and Collector Overview

Helps the developer to:

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

API and *-tcollect*

-trace

Features

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

Intel® Trace Collector

Trace File (.stf)

Intel® Trace Analyzer

Source Code

Compiler

Objects

Linker

Binary

Runtime

Output

How to Use Intel® Trace Analyzer and Collector

It's Easy…

Step 1

Run your binary and create a tracefile:

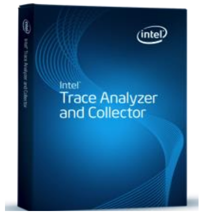
```
$ mpirun -trace -n 2 ./test
```

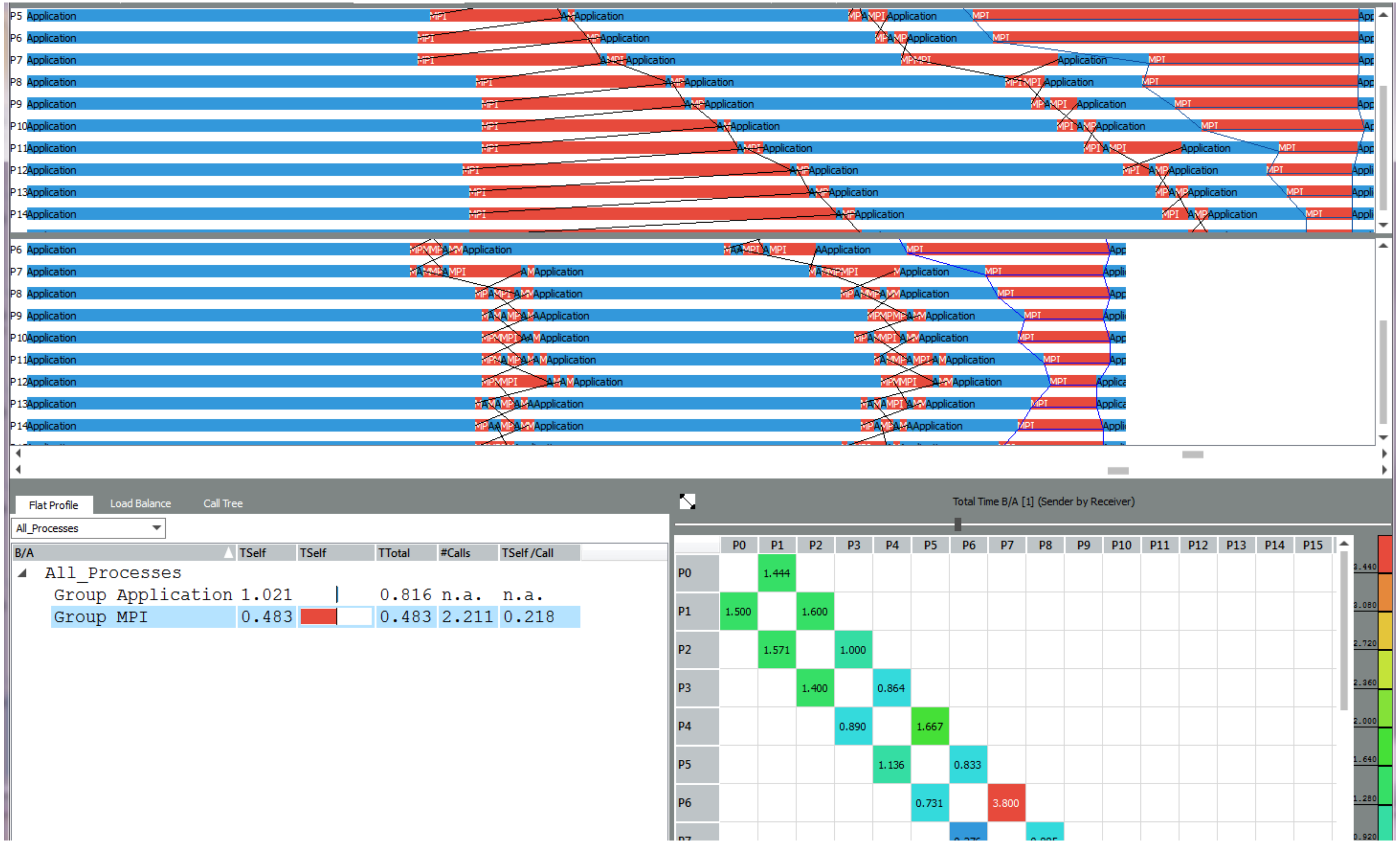
Step 2

View the results:

```
$ traceanalyzer &
```

Intel® Trace Analyzer and Collector



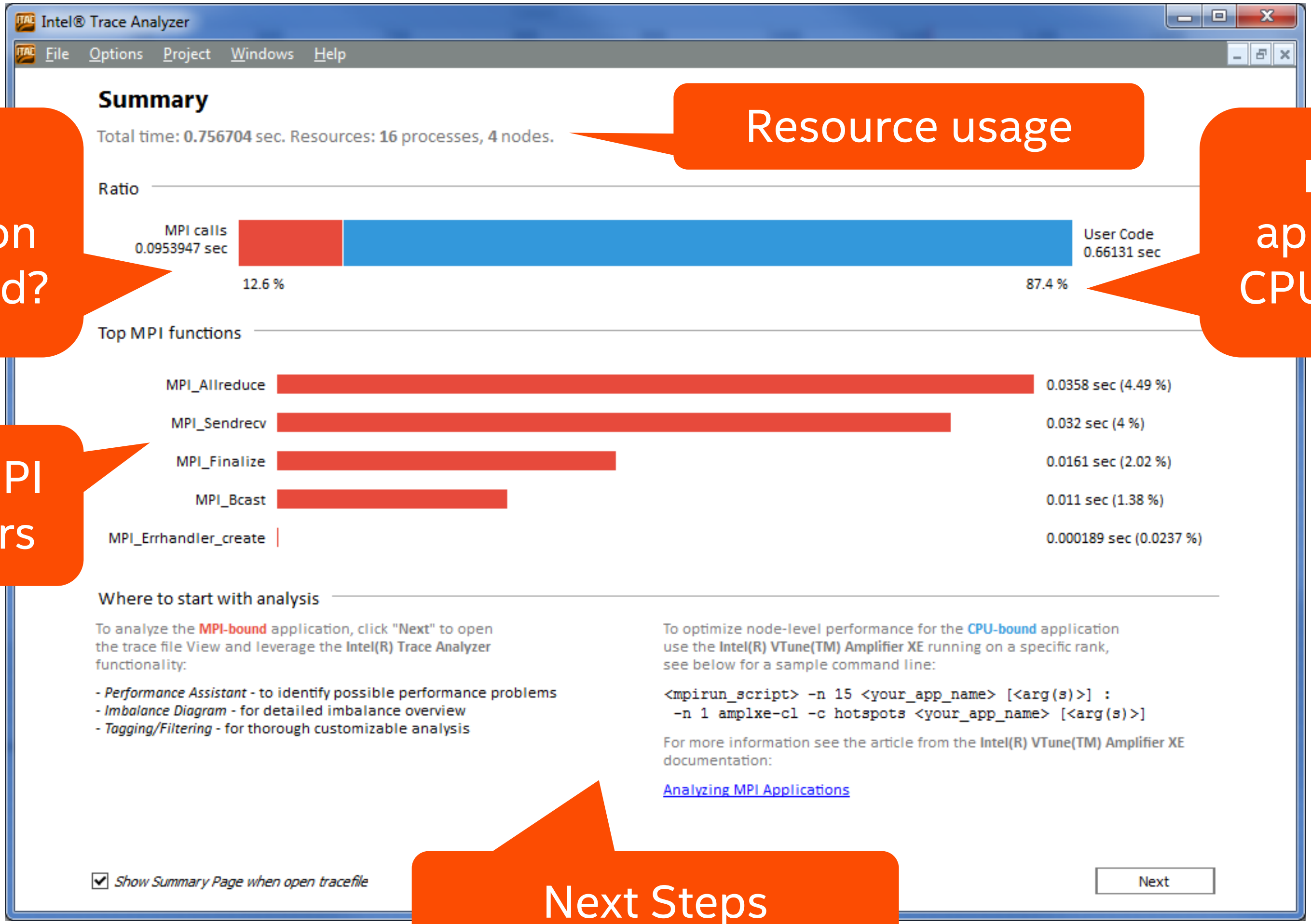


Compare the event timelines of two communication profiles

Blue = Computation
Red = Communication

Chart showing how the MPI processes interact

Summary page shows computation vs. communication breakdown



Is your application MPI-bound?

Resource usage

Is your application CPU-bound?

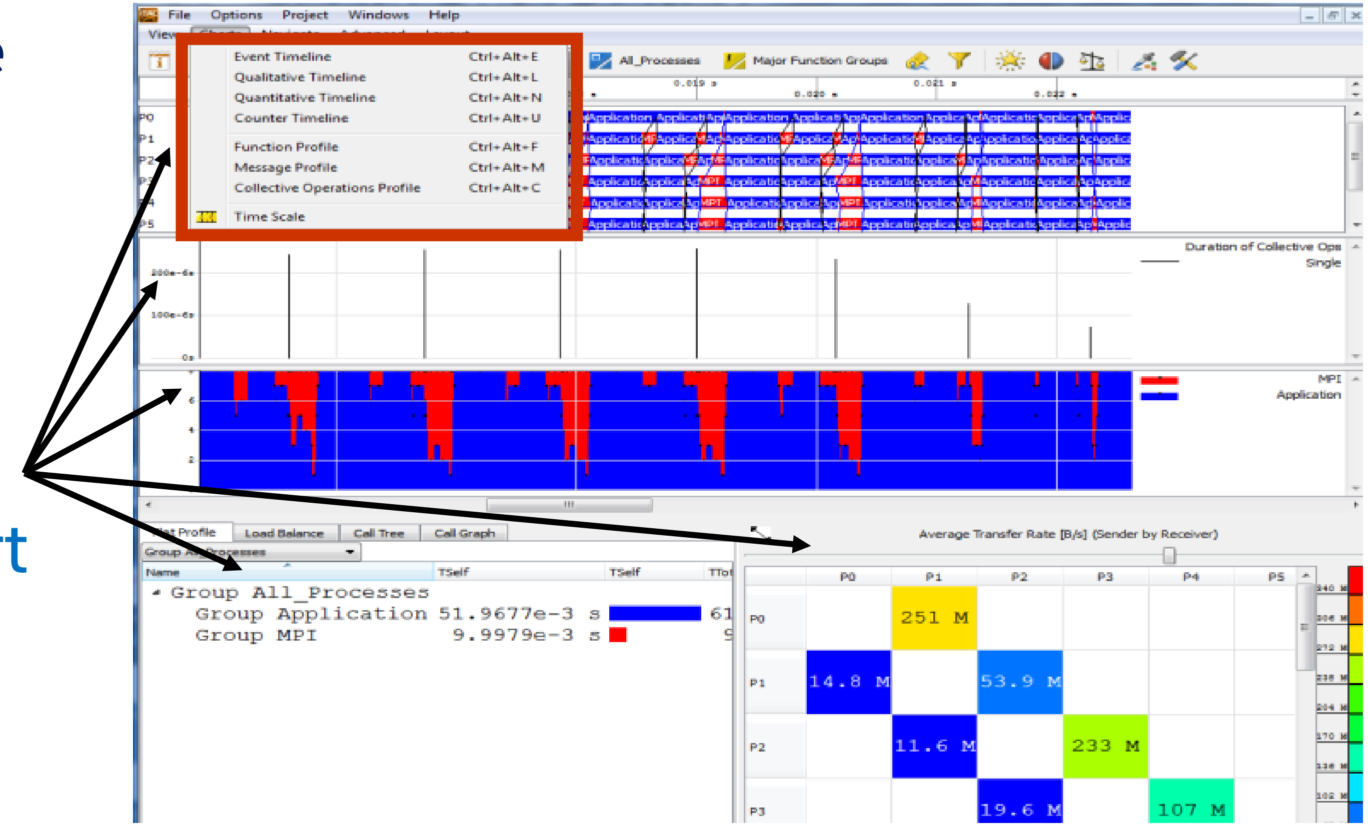
Largest MPI consumers

Next Steps

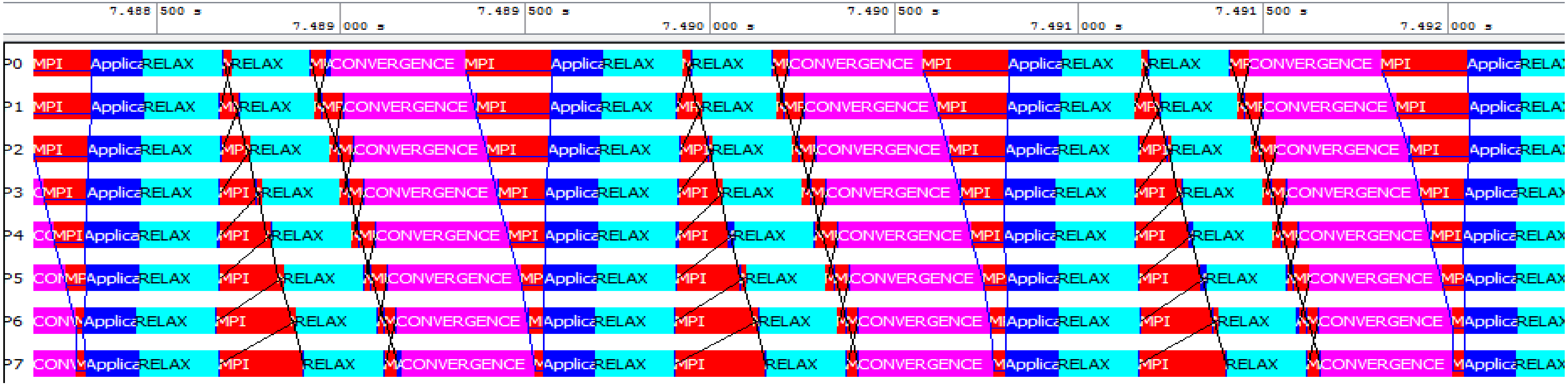
Views and Charts

- 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



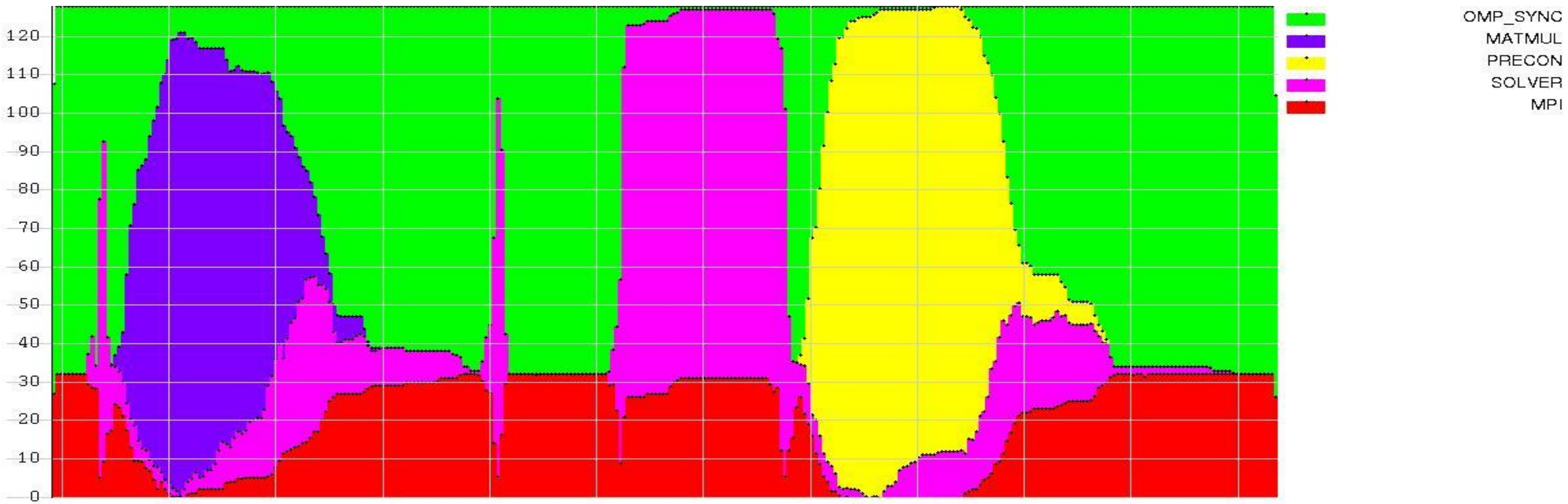
Event Timeline



- 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

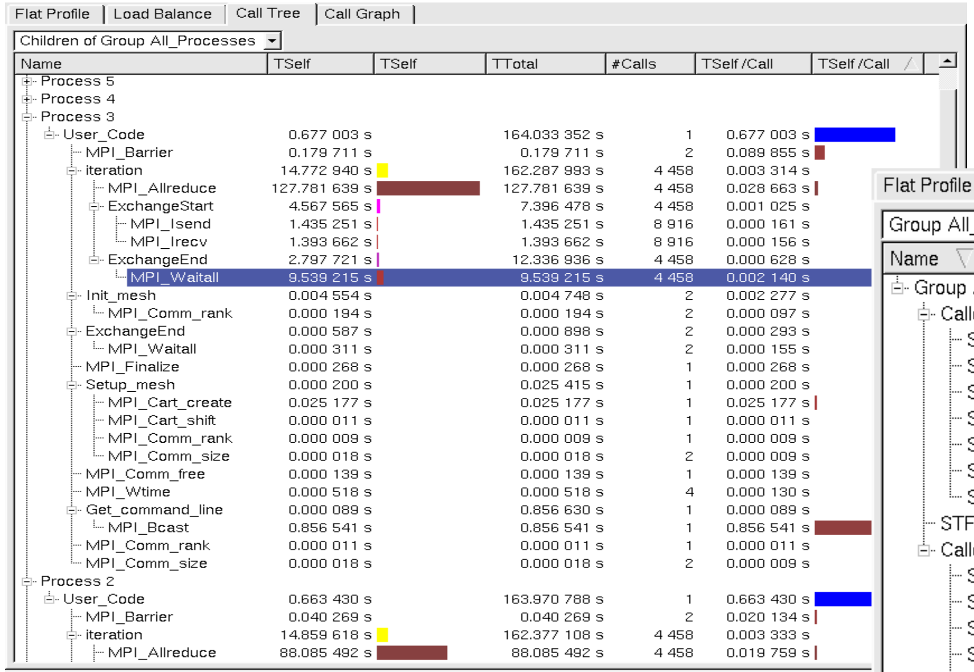
Children of Group All_Threads

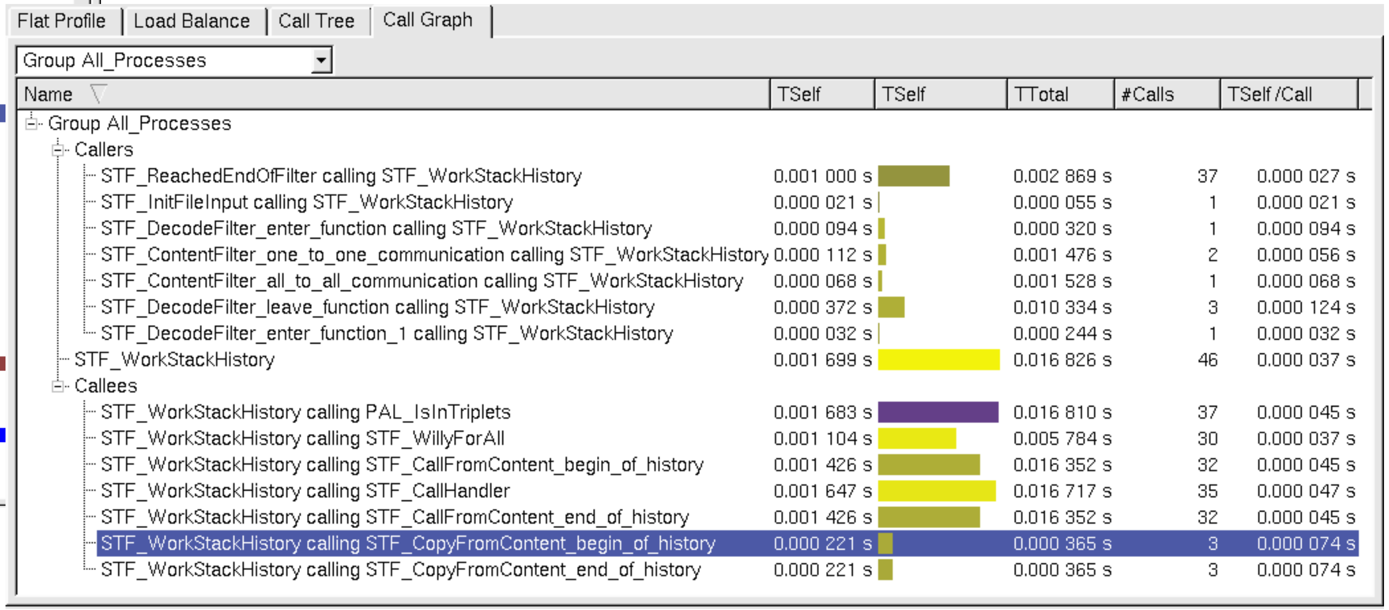
Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
MPI_Comm_dup					
MPI_Waitall					
Process 31 Thread 0	0.913 338 s		0.913 338 s	1 546	0.000 591 s
Process 24 Thread 0	0.801 994 s		0.801 994 s	1 546	0.000 519 s
Process 28 Thread 0	0.756 392 s		0.756 392 s	1 546	0.000 489 s
Process 23 Thread 0	0.721 329 s		0.721 329 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

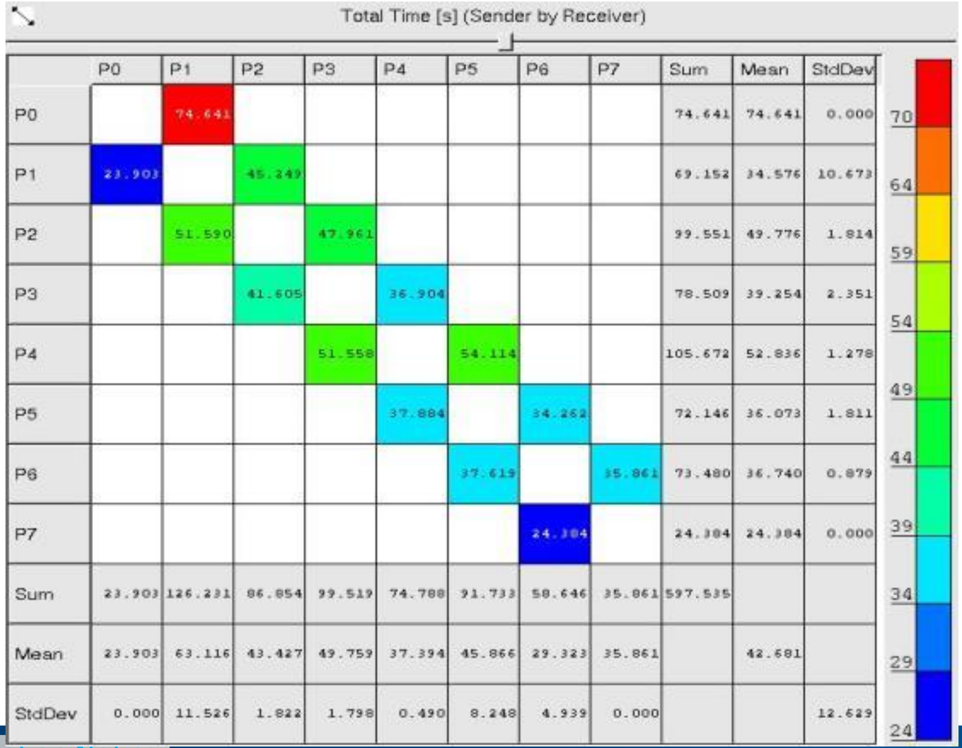
- Tree: Call-stack
- Graph: Calling dependencies





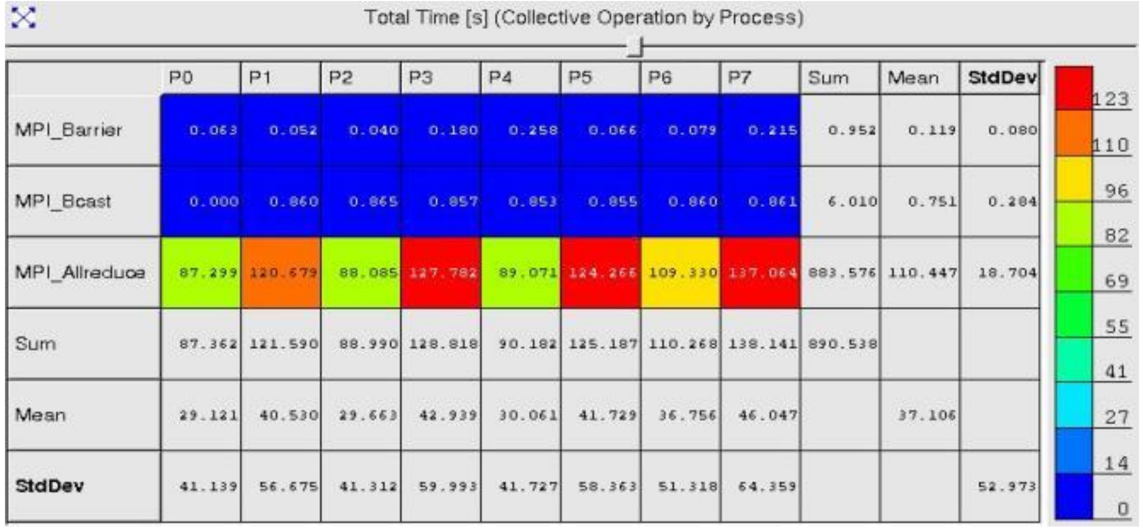
Communication Profiles

- 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



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

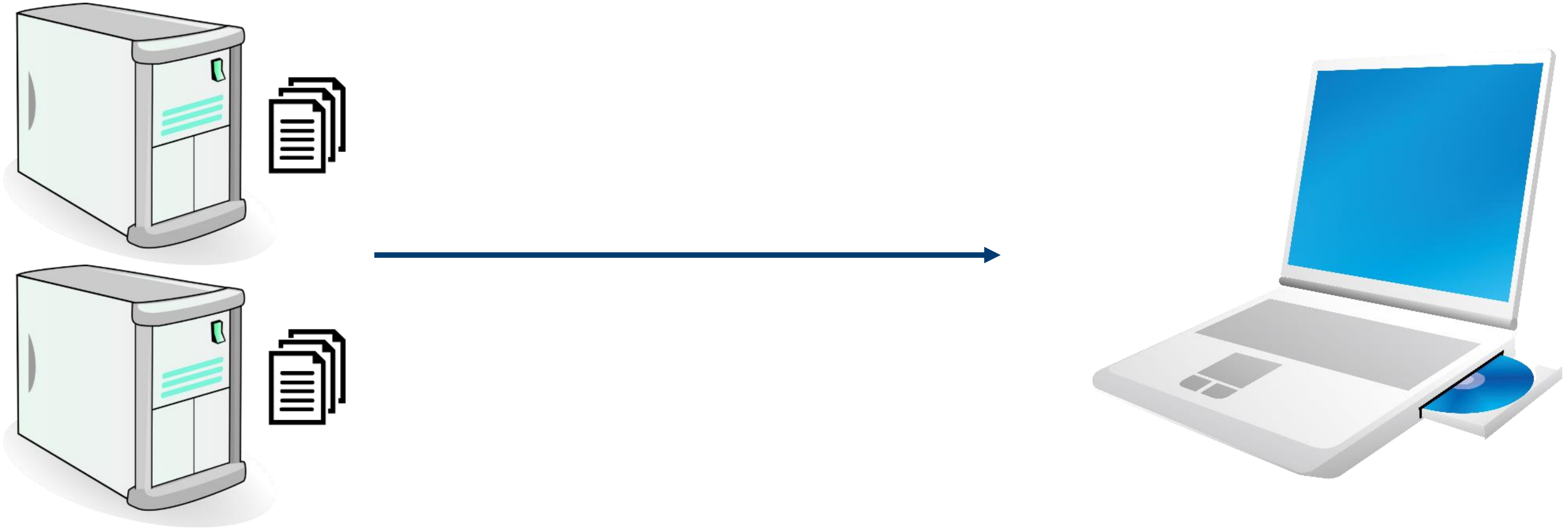
BACKUP

Platform Profiler

Collect and manage data from multiple targets easily

Data Collector

Platform Profiler Server
Web-based user interface



Data Collection “Targets”

Platform Profiler Server

Web-based UI
Chrome works best

Workflow overview

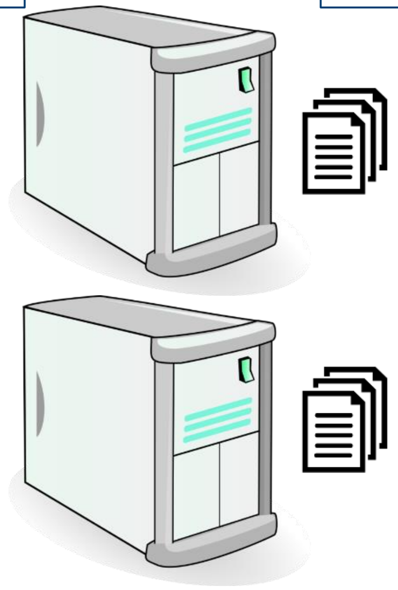
1. Install VTune and launch the Platform Profiler Server

2. Install data collector on “targets” (Linux or Windows)
Ivy Bridge or Later

3. Collect data

4. Upload data

5. View and analyze results



Data Collection “Targets”

Platform Profiler Server

Web-based UI
Chrome works best

intel®

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. For more complete information visit www.intel.com/benchmarks.

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.

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