FIX INTERMITTENT MULTI-THREADING BUGS

FIND AND SQUASH RACES, DEADLOCKS, AND MEMORY BUGS

Jackson Marusarz– Software Technical Consulting Engineer

What Will Be Covered

Overview

Memory/Thread analysis

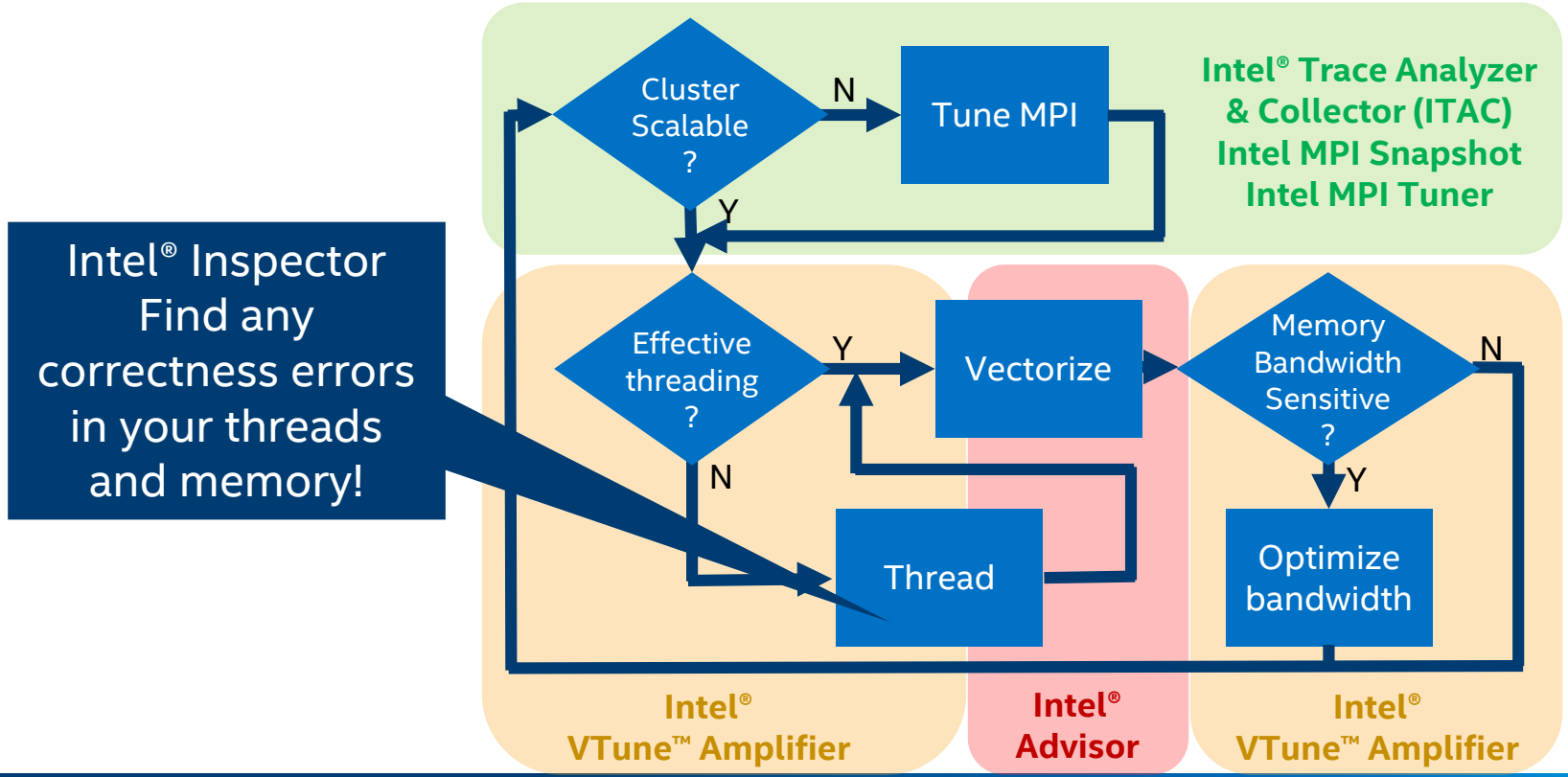
New Features

Deep dive into debugger integrations

Demo

Call to action

Analysis Tools for Diagnosis in Intel® Parallel Studio XE



Correctness Tools Increase ROI By 12%-21%

Cost Factors – Square Project Analysis

CERT: U.S. Computer Emergency Readiness Team, and Carnegie Mellon CyLab

NIST: National Institute of Standards & Technology : Square Project Results

Size and complexity of applications is growing

Reworking defects is 40%-50% of total project effort

Correctness tools find defects during development prior to shipment

Reduce time, effort, and cost to repair

Find errors earlier when they are less expensive to fix

Find & Debug Memory & Threading Errors

Intel® Inspector – Memory & Thread Debugger

Correctness Tools Increase ROI By 12%-21%1

- Errors found earlier are less expensive to fix
- Several studies, ROI% varies, but earlier is cheaper

Diagnosing Some Errors Can Take Months

- Races & deadlocks not easily reproduced
- Memory errors can be hard to find without a tool

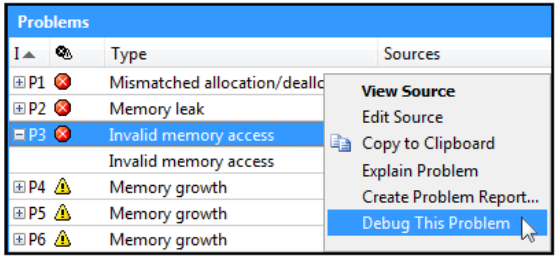
Debugger Integration Speeds Diagnosis

- Breakpoint set just before the problem
- Examine variables & threads with the debugger

Diagnose in hours instead of months

[1] **Cost Factors – Square Project Analysis**
CERT: U.S. Computer Emergency Readiness Team, and Carnegie Mellon CyLab
NIST: National Institute of Standards & Technology : Square Project Results

Debugger Breakpoints



Part of Intel® Parallel Studio XE
For Windows* and Linux*

Intel® Inspector dramatically sped up our ability to track down difficult to isolate threading errors before our packages are released to the field.

Peter von Kaenel, Director, Software Development, ***Harmonic Inc.***

http://intel.ly/inspector-xe

Debug Memory & Threading Errors

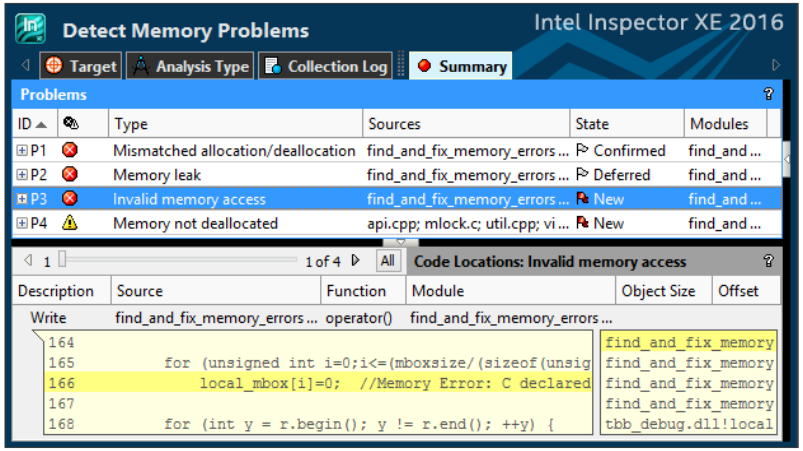
Intel® Inspector

Find and eliminate errors

- Memory leaks, invalid access…
- Races & deadlocks
- C, C++, C#, F# and Fortran (or a mix)

Simple, Reliable, Accurate

- No special recompiles
  Use any build, any compiler[1]
- Analyzes dynamically generated or linked code
- Inspects 3rd party libraries without source
- Productive user interface + debugger integration
- Command line for automated regression analysis



Clicking an error instantly displays source code snippets and the call stack

Fits your existing process

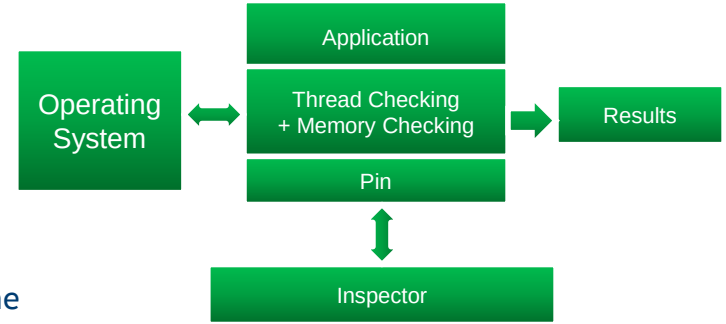
[1]That follows common OS standards.

Intel® Inspector dynamic analysis

Data Collection Techniques

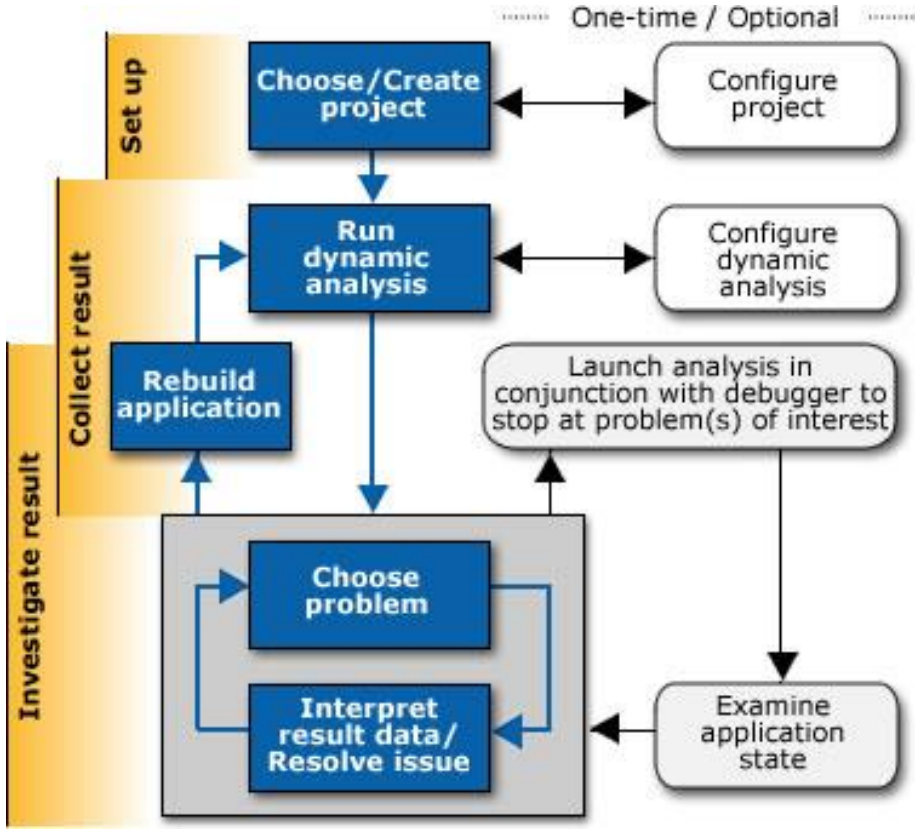
Inspector tracks all memory allocations and threading APIs using a binary instrumentation tool called Pin

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



- OS has to be in the support list
- One process is analysed at a time

Recommended Methodology



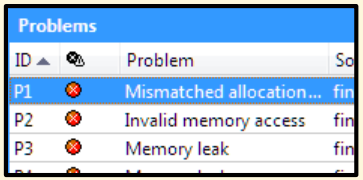
Deliver More Reliable Applications

Intel® Inspector and Intel® Compiler

Intel® Inspector

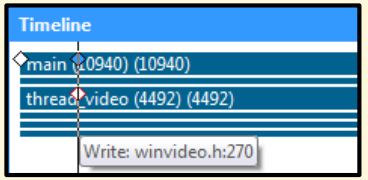
- Dynamic instrumentation
- No special builds
- Any compiler[1]
- Source not required

Memory Errors



- Invalid Accesses
- Memory Leaks
- Uninit. Memory Accesses

Threading Errors

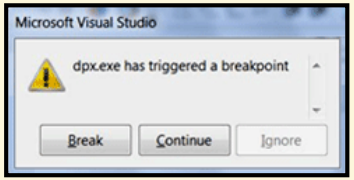


- Races
- Deadlocks
- Cross Stack References

Intel® Compiler

- Pointer checker
- Run time checks
- C, C++

Pointer Errors



- Out of bounds accesses
- Dangling pointers

Find errors earlier with less effort

[1]That follows common OS standards.

Race Conditions Are Difficult to Diagnose

They only occur occasionally and are difficult to reproduce

Correct

Thread 1	Thread 2		Shared Counter
			0
Read count		←	0
Increment			0
Write count		→	1
	Read count	←	1
	Increment		1
	Write count	→	2

Incorrect

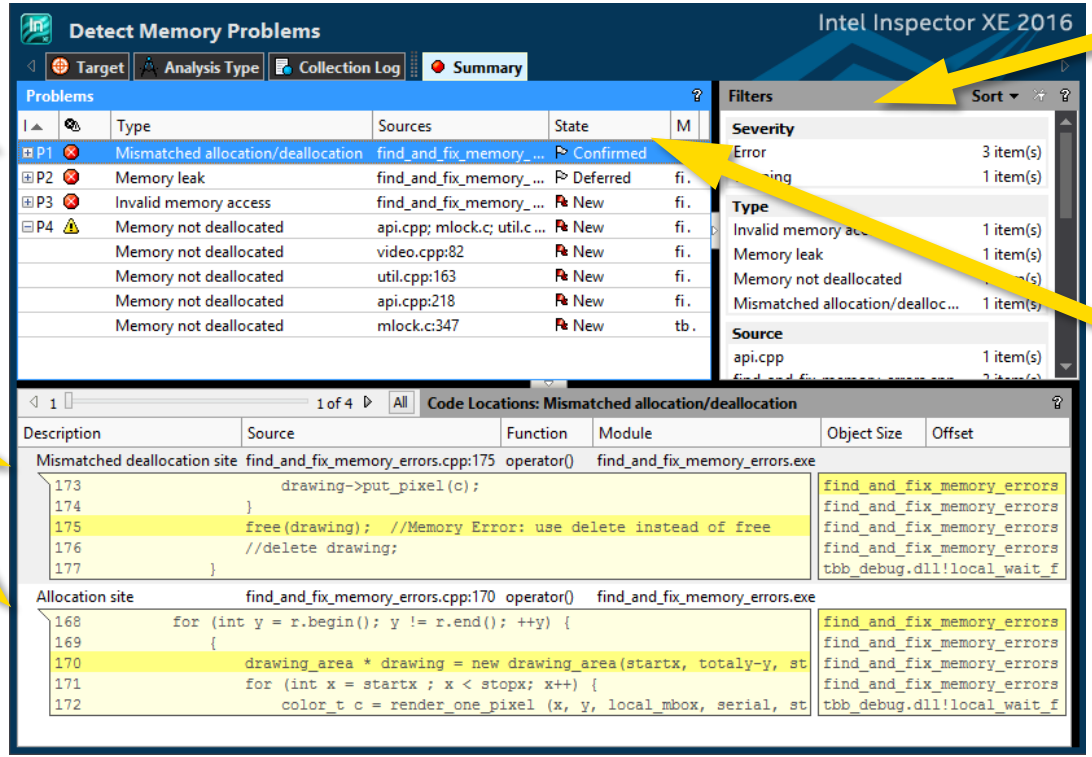
Thread 1	Thread 2		Shared Counter
			0
Read count		←	0
	Read count	←	0
Increment			0
	Increment		0
Write count		→	1
	Write count	→	1

Productive User Interface Saves Time

Intel® Inspector

Select a **problem set**

Code snippets displayed for selected problem



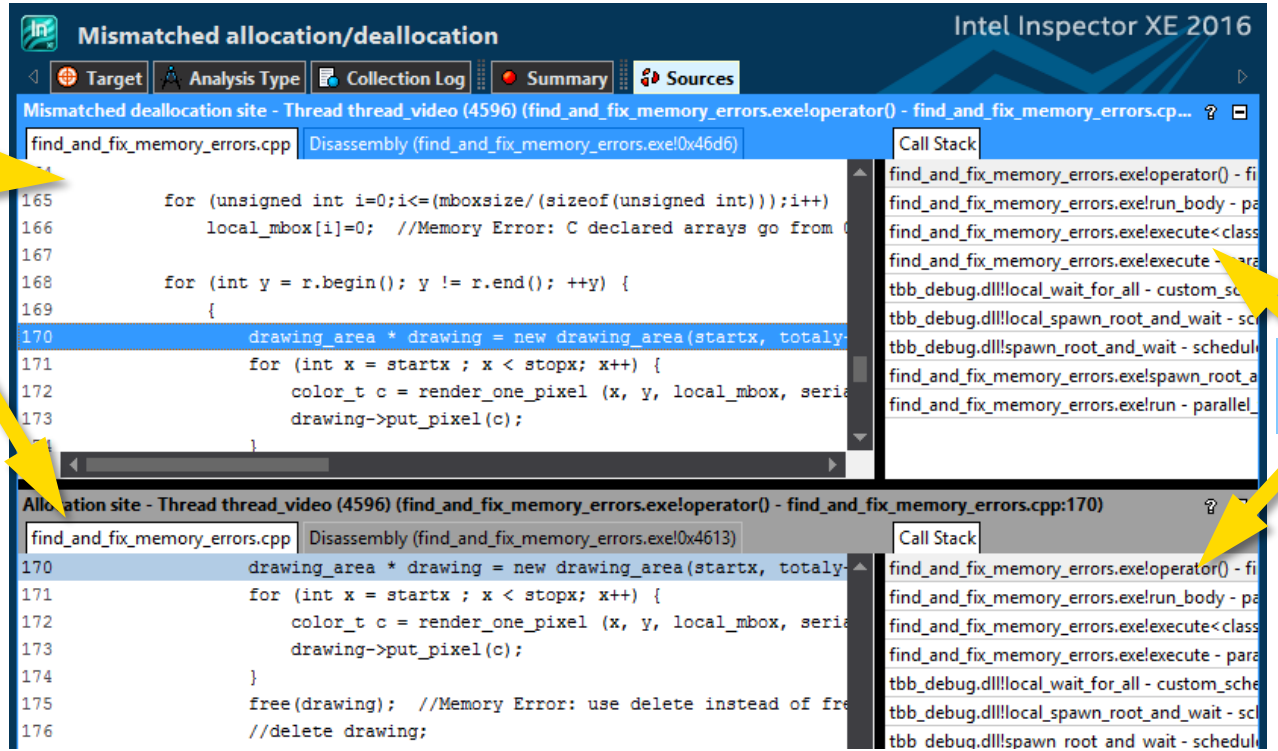
Filters let you focus on a module, or error type, or just the new errors or…

Problem States: New, Not Fixed, Fixed, Confirmed, Not a problem, Deferred, Regression

Double Click for Source & Call Stack

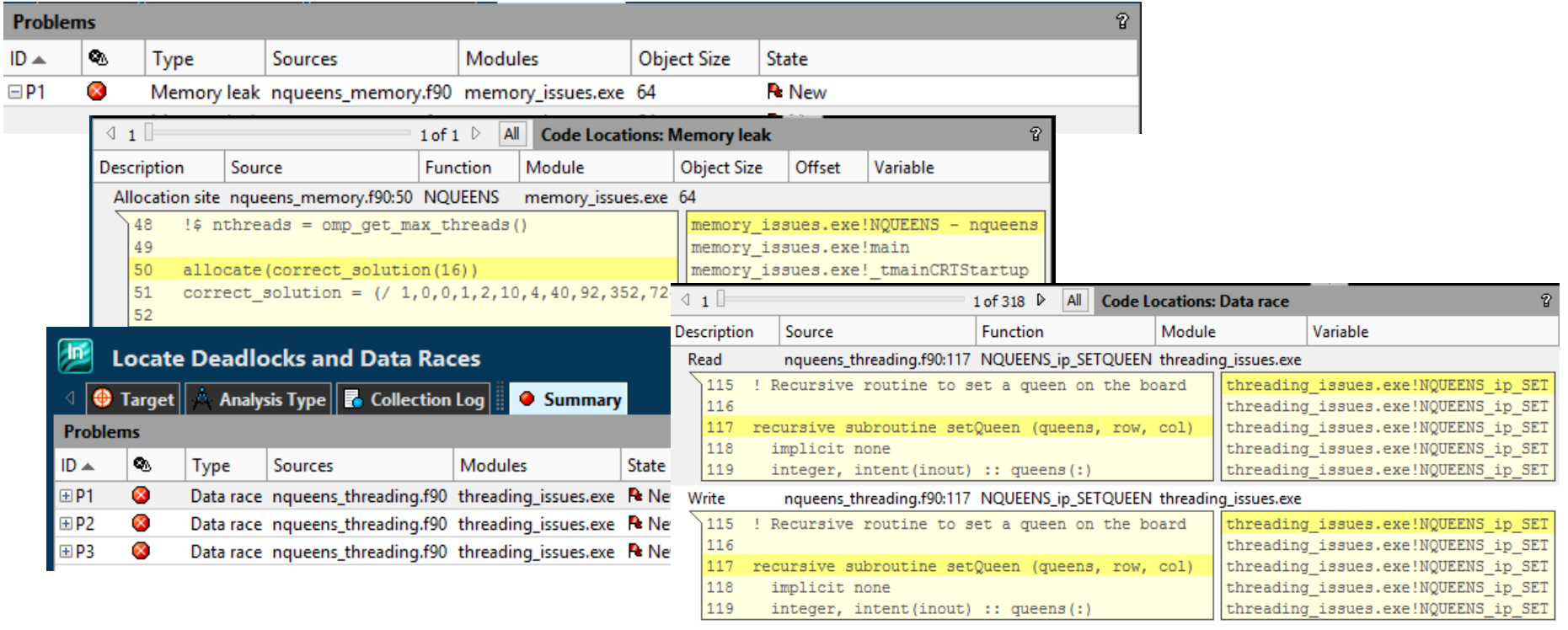
Intel® Inspector

Source code locations displayed for selected problem



Call Stack

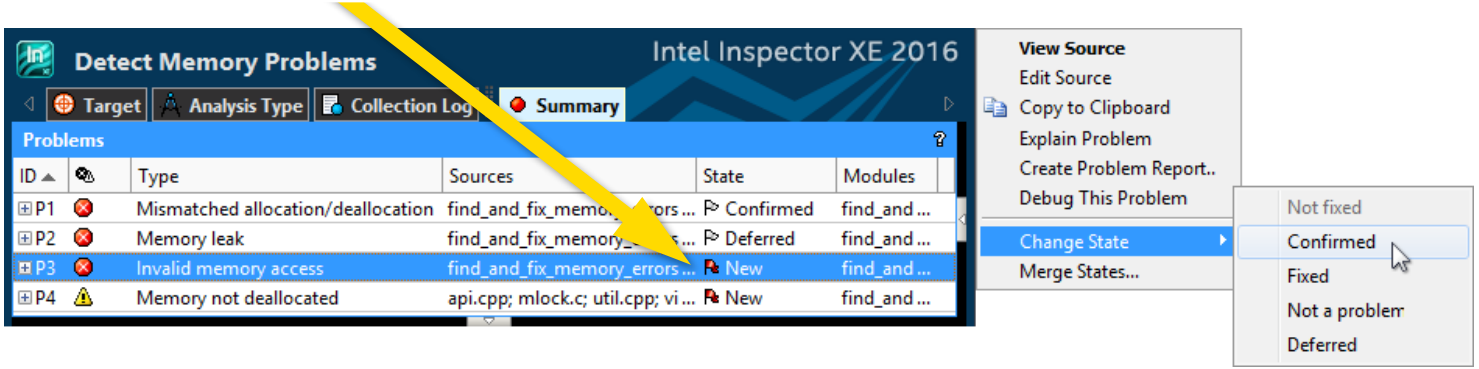
Quickly track down your Fortran issues!



Easy Problem Management

Quickly see new problems and regressions

State	Description
New	Detected by this run
Not Fixed	Previously seen error detected by this run
Not a Problem	Set by user (tool will not change)
Confirmed	Set by user (tool will not change)
Fixed	Set by user (tool will change)
Regression	Error detected with previous state of “Fixed”

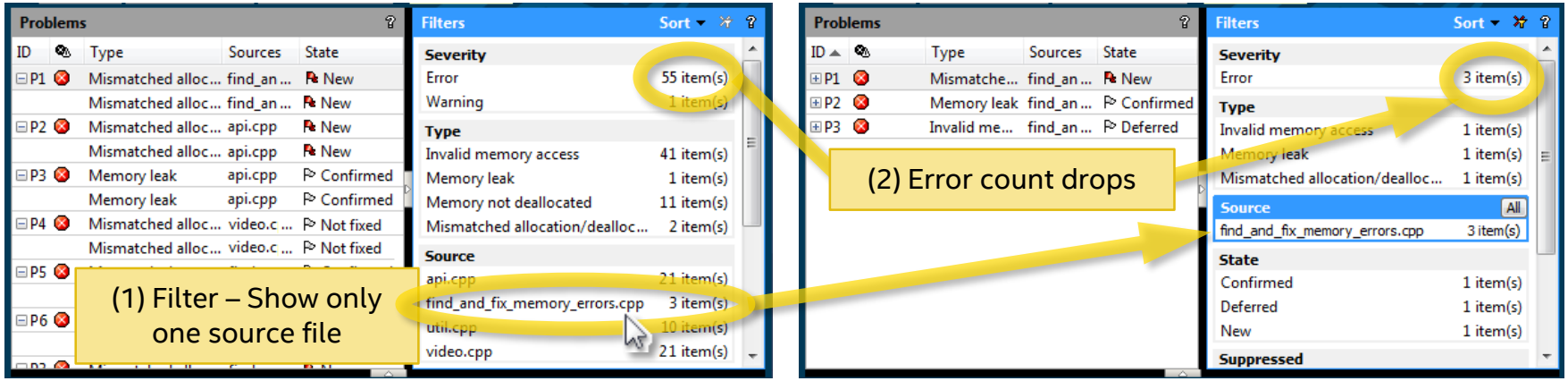


Filtering - Focus on What's Important

Example: See only the errors in one source file

Before – All Errors

After – Only errors from one source file



Tip: Set the "Investigated" filter to "Not investigated" while investigating problems.
This removes from view the problems you are done with, leaving only the ones left to investigate.

Incrementally Diagnose Memory Growth

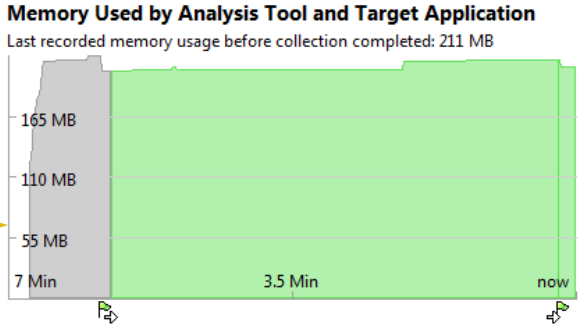
Intel® Inspector

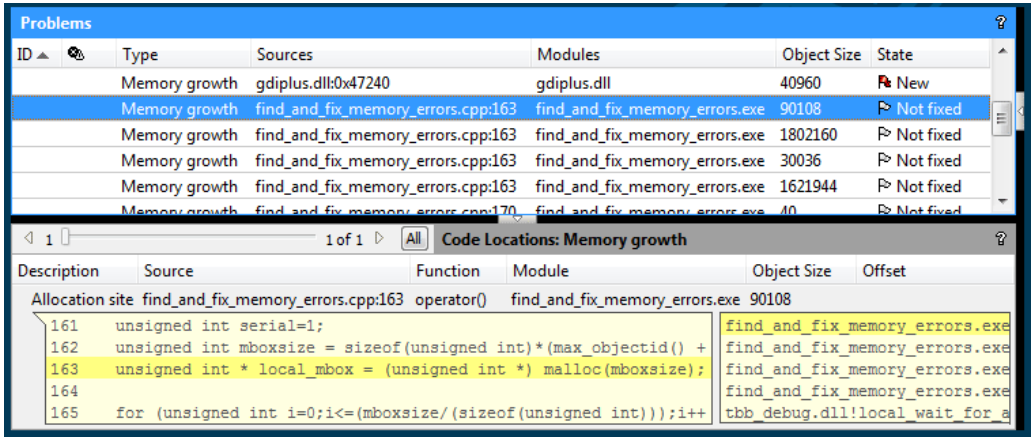
As your app is running…

Memory usage graph plots memory growth

Select a cause of memory growth

See the code snippet & call stack





Speed diagnosis of difficult to find heap errors

Automate Regression Analysis

Command Line Interface

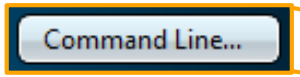
inspxe-cl is the command line:

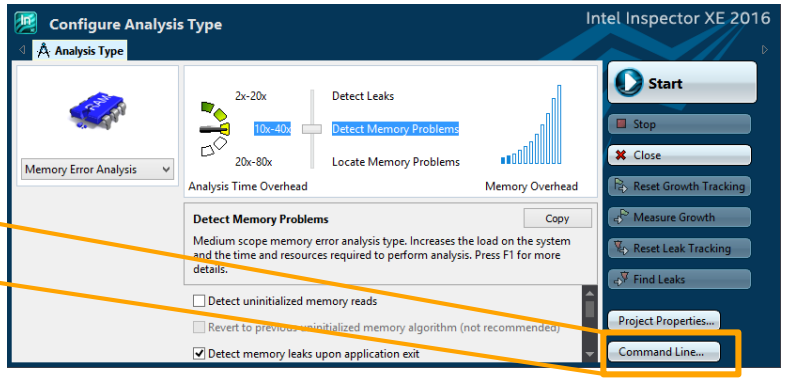
- **Windows:** `C:\Program Files\Intel\Inspector XE \bin[32|64]\inspxe-cl.exe`
- **Linux:** `/opt/intel/inspector_xe/bin[32|64]/inspxe-cl`

Help:

```
inspxe-cl -help
```

Set up command line with GUI





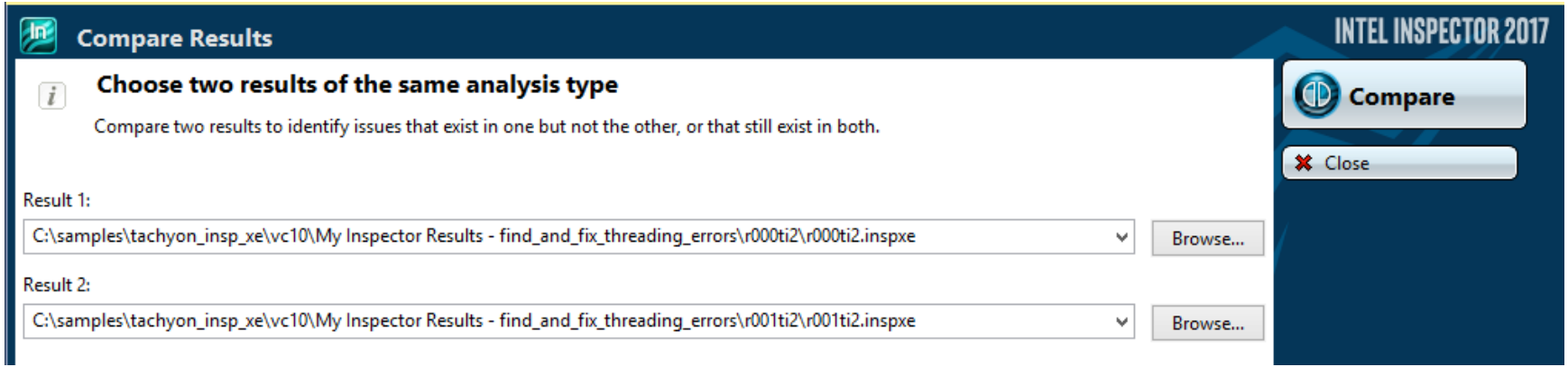
Command examples:

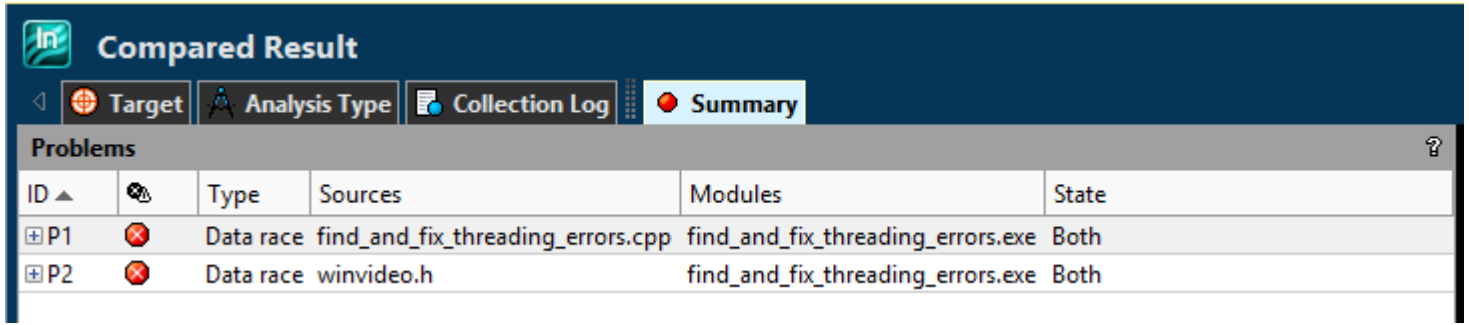
1. `inspxe-cl -collect-list`
2. `inspxe-cl -collect ti2 -- MyApp.exe`
3. `inspxe-cl -report problems`

Send results file to developer to analyze with the UI

Compare results and see what has changed

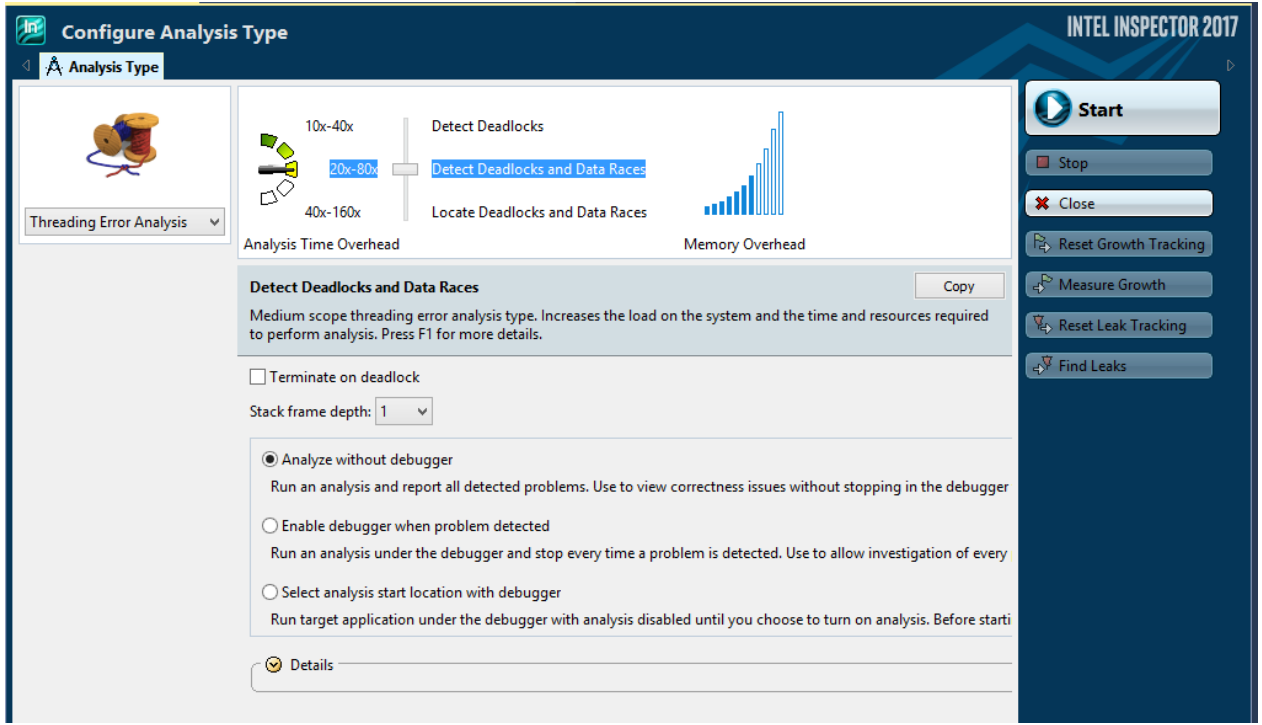
Ideal for regression testing





Find problems quicker!

Interactive debugging support



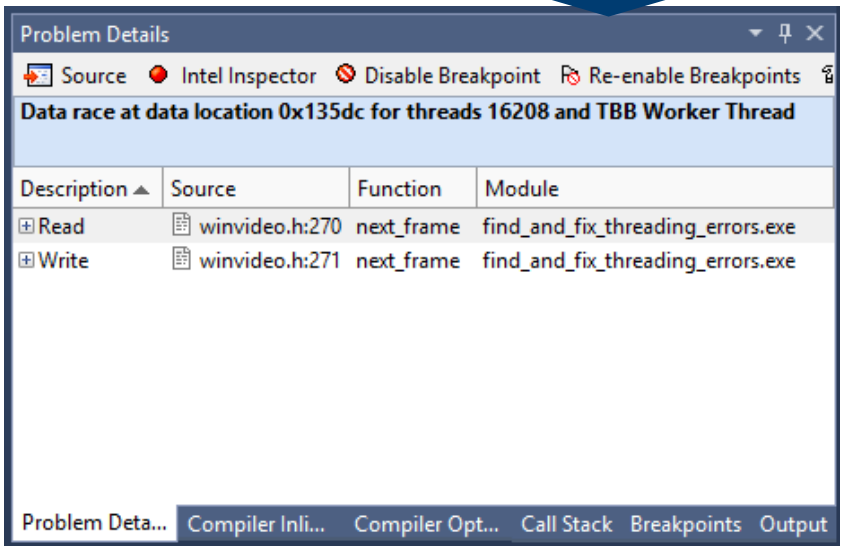
3 debugging modes supported

1. Analyze without debugger
2. Enable debugger when problem detected
3. Start analysis when a debug breakpoint is hit.

Intuitive problem solving using debugger integrations

```
//! Refresh screen picture
bool video::next_frame()
{
    if(!running) return false;
    g_updates++; // Fast but inaccurate counter. The data race here is beni
    if(!threaded) while(loop_once(this));
    else if(g_handles[1]) {
        SetEvent(g_handles[1]);
        YIELD_TO_THREAD();
    }
    return true;
}
```

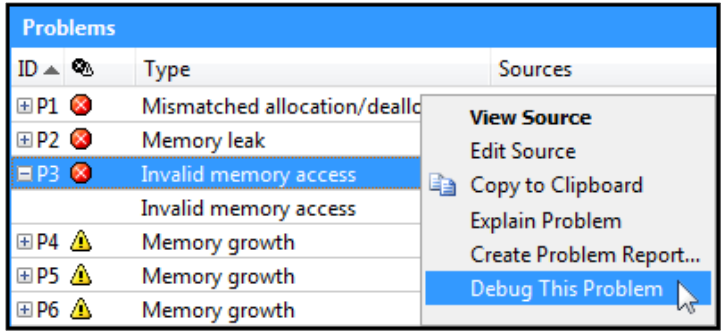
Microsoft Visual Studio* and GNU gdb* or Intel® Debugger (on Linux*)



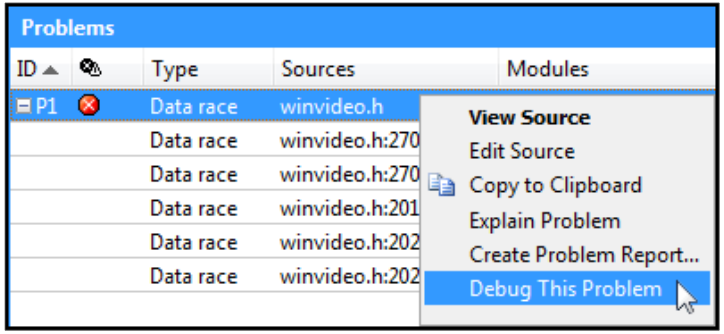
Break At Just The Right Time

Intel® Inspector - Memory & Thread Debugger

Memory Errors



Threading Errors



Break into the debugger just before the error occurs.

Examine the variables and threads.

Diagnose the problem.

Save time. Find and diagnose errors with less effort.

Work Smarter & Faster

Intel® Inspector - Memory & Thread Debugger

Precise Error Suppression

```
Suppression = {
    Name = "Example";
    Type = { uninitialized_memory_access }
    Stacks = {
        {
            mod=a.out, func=update_x;
            func=main;
```

Precise, easy to edit, team shareable.

Choose which stack frame to suppress.

Eliminate the false, not the real errors.

Pause/Resume Collection

```
__itt_suppress_push(__itt_suppress_threading_errors);
    /* Any threading errors here are ignored */
__itt_suppress_pop();
    /* Any threading errors here are seen */
```

Speed-up analysis by limiting its scope.

Analyze only during the execution of the suspected problem.

Find and diagnose errors with less effort.

Work Smarter & Faster

Intel® Inspector - Memory & Thread Debugger

Precise Error Suppression

```
Suppression = {
    Name = "Example";
    Type = { uninitialized_memory_access }
    Stacks = {
        {
            mod=a.out, func=update_x;
            func=main;
```

Precise, easy to edit, team shareable.

Choose which stack frame to suppress.

Eliminate the false, not the real errors.

Pause/Resume Collection

```
__itt_suppress_push(__itt_suppress_threading_errors);
    /* Any threading errors here are ignored */
__itt_suppress_pop();
    /* Any threading errors here are seen */
```

Speed-up analysis by limiting its scope.

Analyze only during the execution of the suspected problem.

Find and diagnose errors with less effort.

Productive Memory & Threading Debugger

Intel® Inspector

	Memory Analysis	Threading Analysis
View Context of Problem		
Stack	✓	✓
Multiple Contributing Source Locations	✓	✓
Collapse multiple “sightings” to one error (e.g., memory allocated in a loop, then leaked is 1 error)	✓	✓
Suppression, Filtering, and Workflow Management	✓	✓
Visual Studio* Integration (Windows*)	✓	✓
Command line for automated tests	✓	✓
Time Line visualization	✓	✓
Memory Growth during a transaction	✓	
Trigger Debugger Breakpoint	✓	✓

Easier & Faster Debugging of Memory & Threading Errors

WHAT'S NEW

Intel® Inspector 2017 Beta

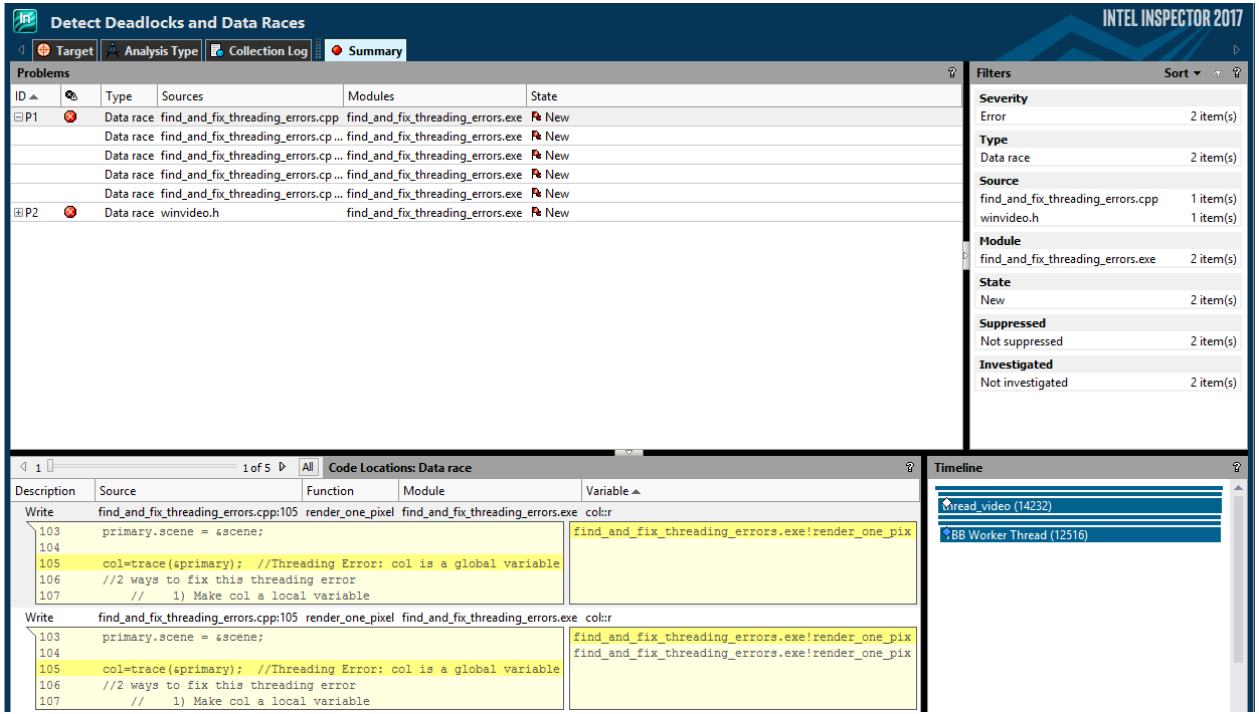
New Features

Support for Intel® Xeon Phi™ processor (codename: Knights Landing)

Support for C++11 synchronization primitives during threading analysis

Variable name detection for threading analysis

Variable name detection for threading analysis



Memory & Threading Debugger Saves Time

Intel® Inspector

"We struggled for a week with a crash situation, the corruption was identified but the source was really hard to find. Then we ran **Intel® Inspector** and immediately found the array out of bounds that occurred long before the actual crash. We could have saved a week!"

Mikael Le Guerroué,
Senior Codec Architecture Engineer,
Envivio

"**Intel® Inspector** is quite fast and intuitive compared to products we have used in the past. We can now run our entire batch of test cases (~750) which was not feasible previously. **Intel® Inspector** easily completed tests that failed due to lack of virtual memory on another product."

Gerald Mattauch
Senior Software Developer
***Siemens AG**, Healthcare Sector*

Intel® Inspector has dramatically sped up our ability to find/fix memory problems and track down difficult to isolate threading errors before our packages are released to the field.

Peter von Kaenel, Director,
Software Development,
Harmonic Inc.

More Case Studies

DEMO

Intel® Inspector 2017 Beta

Call to Action

Modernize your Code

- To get the most out of your hardware, you need to modernize your code with vectorization and threading.
- Taking a methodical approach such as the one outlined in this presentation, and taking advantage of the powerful tools in Intel® Parallel Studio XE, can make the modernization task dramatically easier.
- Join the Intel® Parallel Studio 2017 Beta at Intel® Parallel Studio XE 2017 Beta

Intel® Inspector XE benchmark

Configuration information

7zip benchmark configuration info – Configuration Info – SW Versions: 7zip 9.22beta (Windows), 9.20(Linux); Microsoft Visual Studio* 10.0 (Windows), GCC 4.4.6 (Linux 64-bit), GCC 4.3.4 (Linux 32-bit); Hardware: Intel® Core™ i7 CPU 965 @ 3.20GHz, 6GB Memory; OS: SUSE Linux Enterprise Server 11 SP2, x86, kernel 3.0.13-0.27-pae; Red Hat Enterprise Linux Server 6.3, x86_64, kernel 2.6.32-279.el6.x86_64; Windows 7, x86; Windows 8, x86_64;

blender benchmark configuration info – Configuration Info – SW Versions: blender 2.69; Intel® C++ Compiler 14.0.0; Hardware: Intel® Core™ i7 CPU 965 @ 3.20GHz, 6GB Memory; OS: SUSE Linux Enterprise Server 11 SP2, x86, kernel 3.0.13-0.27-pae; Red Hat Enterprise Linux Server 6.3, x86_64, kernel 2.6.32-279.el6.x86_64; Windows 7, x86; Windows 8, x86_64;

firefox benchmark configuration info – Configuration Info – SW Versions: firefox 20.0; Microsoft Visual Studio 11.0 (Windows 64-bit), Microsoft Visual Studio 10.0 (Windows 32-bit), GCC 4.4.6 (Linux); Hardware: Intel® Core™ i7 CPU 965 @ 3.20GHz, 6GB Memory; OS: SUSE Linux Enterprise Server 11 SP2, x86, kernel 3.0.13-0.27-pae; Red Hat Enterprise Linux Server 6.3, x86_64, kernel 2.6.32-279.el6.x86_64; Windows 7, x86; Windows 8, x86_64;

Legal Disclaimer & Optimization Notice

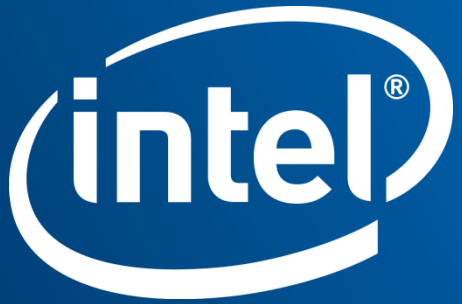
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel®