



Intel[®] Inspector XE

Memory and thread debugger



Agenda

Intro to Intel® Inspector XE

Introduction

Memory problem analysis

Threading problem Analysis

Preparing setup for analysis

Managing analysis results

Advanced Features

Summary

Agenda

Intro to Intel® Inspector XE

Introduction

Memory problem analysis

Threading problem Analysis

Preparing setup for analysis

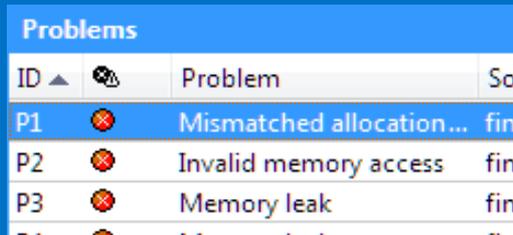
Managing analysis results

Advanced Features

Summary

Motivation for The Inspector XE

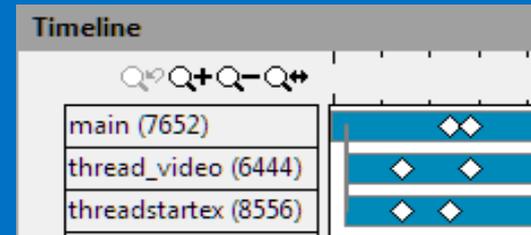
Memory Errors



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

- Invalid Accesses
- Memory Leaks
- Uninitialized Memory Accesses

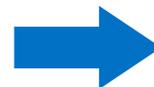
Threading Errors



- Data Races
- Deadlocks
- Cross Stack References

Multi-threading problems

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

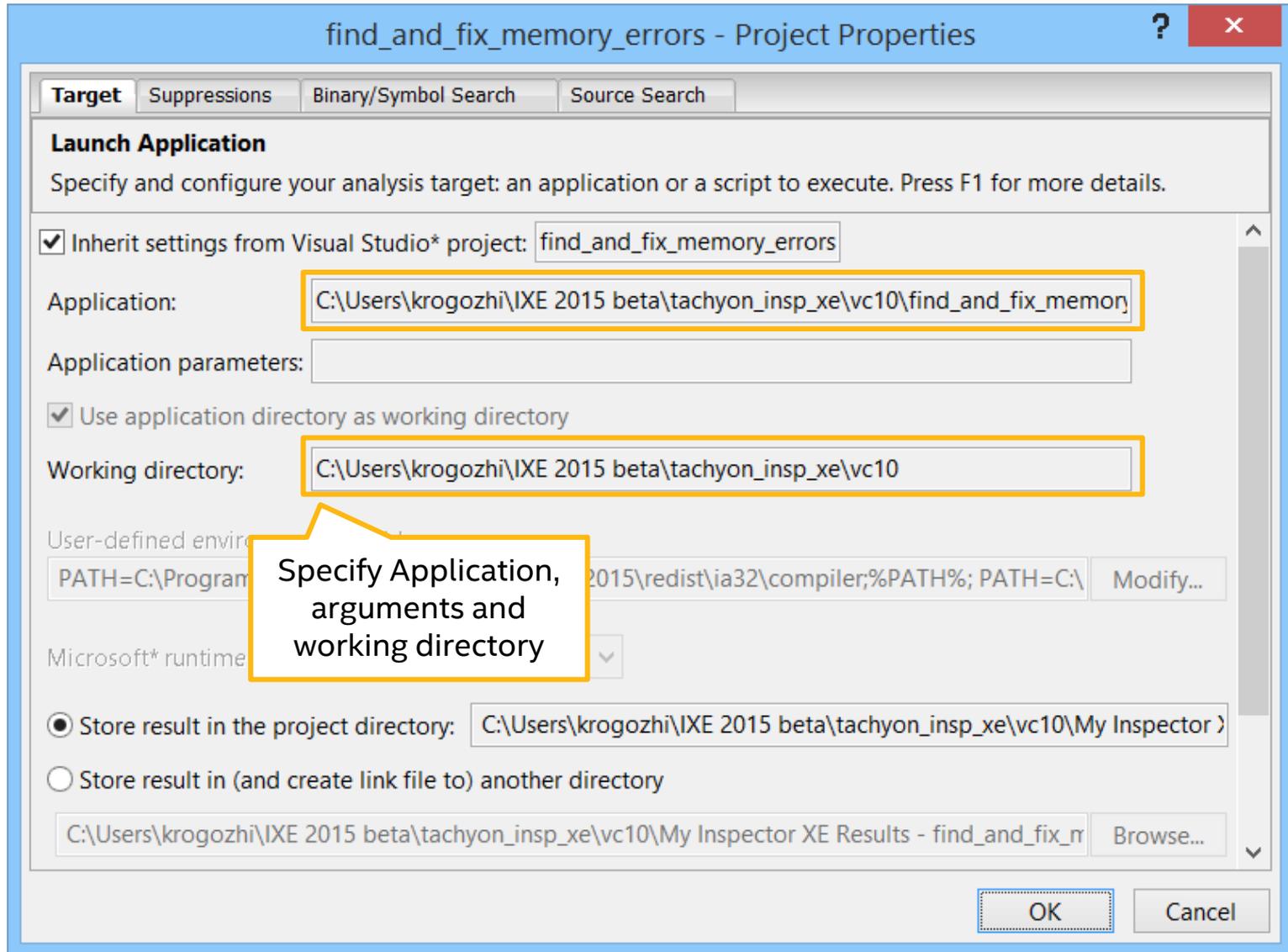


Let the tool do it for you

Key Features at a glance

Feature	Details
Data collection	<ul style="list-style-type: none">• Dynamic Memory and Threading Analysis (including .NET* analysis)• MPI applications analysis
Result analyses	<ul style="list-style-type: none">• GUI data mining: source code analysis, filtering, exploring call paths, etc.• Debugger integration• Result comparison• Problem life cycle management• Command line interface (especially useful for regression testing)
GUI	<ul style="list-style-type: none">• Microsoft* Visual Studio IDE integration (2010, 2012 and 2013)• Stand alone GUI on both Windows* and Linux*
Compilers supported	<ul style="list-style-type: none">• Microsoft* Visual* C++ and .NET*• Intel® C/C++ Compiler XE 12.0 or higher• Intel® Visual Fortran Compiler XE 12.0 or higher• gcc
OS	<ul style="list-style-type: none">• Windows* 7, 8, 8.1,• Windows* Server 2008, 2008 R2, 2012• Linux*: RedHat, Fedora, CentOS, SUSE, Debian, Ubuntu
Languages	<ul style="list-style-type: none">• C/C++• C# (.NET 2.0 to 3.5, .NET 4.0 with limitations)• Fortran

Workflow: setup project



Workflow: select analysis and start

Configure Analysis Type Intel Inspector XE 2015

Analysis Type

- Threading Error Analysis
- Memory Error Analysis
- Threading Error Analysis
- Custom Analysis Types

1. Select Analysis Type

Analysis Time Overhead	Memory Overhead
10x-40x	Detect Deadlocks
20x-80x	Detect Deadlocks and Data Races
40x-160x	Locate Deadlocks and Data Races

2. Click Start

Start

- Stop
- Close
- Reset Growth Tracking
- Measure Growth
- Reset Leak Tracking
- Find Leaks

Locate Deadlocks and Data Races Copy

Widest scope threading error analysis type. Maximizes the load on the system and the time and resources required to perform analysis; however, detects the widest set of errors and provides context and maximum detail for those errors. Press F1 for more details.

Terminate on deadlock

Stack frame depth: 16

Scope: Normal

Remove duplicates

Use maximum resources

Project Properties...
Command Line...

Workflow: manage results

Intel Inspector XE 2015

Detect Deadlocks and Data Races

Target Analysis Type Collection Log Summary

Problems

ID	Type	File	Module	Status
P1	Data race	find_and_fix_threading_errors.cp...	find_and_fix_threading_errors.exe	New
P2	Data race	winvideo.h	find_and_fix_threading_errors.exe	New
	Data race	winvideo.h:270	find_and_fix_threading_errors.exe	New
	Data race	winvideo.h:270	find_and_fix_threading_errors.exe	New
	Data race	winvideo.h:201; winvideo.h:270	find_and_fix_threading_errors.exe	New

Code Locations: Data race

Read winvideo.h:270 next_frame find_and_fix_threading_errors.exe

```
268 {
269     if(!running) return false;
270     g_updates++; // Fast but inaccura
271     if(!threaded) while(loop_once(thi
272     else if(g handles[1]) {
```

Timeline

- main (4960)
- thread_video (4672)
- TBB Worker Thread (2848)
- TBB Worker Thread (1724)
- TBB Worker Thread (6004)

Read: winvideo.h:270
Write: winvideo.h:270

Filters Sort

Data race 2

Source

- find_and_fix_thre... 1
- task_scheduler_i... 1
- winvideo.h 1

Module

Code Locations: Data race

Read winvideo.h:270 next_frame find_and_fix_threading_errors.exe

```
268 {
269     if(!running) return false;
270     g_updates++; // Fast but inaccura
271     if(!threaded) while(loop_once(thi
272     else if(g handles[1]) {
```

Workflow: navigate to sources

Data race

Intel Inspector XE 2015

Target Analysis Type Collection Log Summary Sources

Write - Thread TBB Worker Thread (1724) (find_and_fix_threading_errors.exe!next_frame - winvideo.h:270)

winvideo.h Disassembly (find_and_fix_threading_errors.exe!0x9257) Call Stack

```
267 bool video::next_frame()
268 {
269     if(!running) return false;
270     g_updates++; // Fast but inaccurate counter. The data race h
271     if(!threaded) while(loop_once(this));
272     else if(g_handles[1]) {
273         SetEvent(g_handles[1]);
274         YIELD_TO_THREAD();
275     }
```

Problematic line in source code

Call stacks

Read - Thread TBB Worker Thread (6004) (find_and_fix_threading_errors.exe!next_frame - winvideo.h:270)

winvideo.h Disassembly (find_and_fix_threading_errors.exe!0x924e) Call Stack

```
267 bool video::next_frame()
269     if(!running) return false;
270     g_updates++; // Fast but inaccur
271     if(!threaded) while(loop_once(th
272     else if(g_handles[1]) {
273         SetEvent(g_handles[1]);
274         YIELD_TO_THREAD();
275     }
```

All code locations for a problem

Switch to disassembly for more details

Agenda

Intro to Intel[®] Inspector XE

Introduction

Memory problem analysis

Threading problem Analysis

Preparing setup for analysis

Managing analysis results

Advanced Features

Summary

Memory problem Analysis

Analyzed as software runs

- Data (workload) -driven execution
- Program can be single or multi-threaded
- Diagnostics reported incrementally as they occur

Includes monitoring of:

- Memory allocation and allocating functions
- Memory deallocation and deallocating functions
- Memory leak reporting
- Inconsistent memory API usage

Analysis scope

- Native code only: C, C++, Fortran
- Code path must be executed to be analyzed
- Workload size affects ability to detect a problem

Memory problems

Memory leak

- a block of memory is allocated
- never deallocated
- not reachable (there is no pointer available to deallocate the block)
- Severity level = **(Error)**

```
// Memory leak
```

```
char *pStr = (char*) malloc(512);  
return;
```

Memory not deallocated

- a block of memory is allocated
- never deallocated
- still reachable at application exit (there is a pointer available to deallocate the block).
- Severity level = **(Warning)**

```
// Memory not deallocated
```

```
static char *pStr = malloc(512);  
return;
```

Memory growth

- a block of memory is allocated
- not deallocated, within a specific time segment during application execution.
- Severity level = **(Warning)**

```
// Memory growth
```

```
// Start measuring growth  
static char *pStr = malloc(512);  
// Stop measuring growth
```

Memory problems

Uninitialized memory access

- Read of an uninitialized memory location

```
// Uninitialized Memory Access

void func()
{
    int a;
    int b = a * 4;
}
```

Invalid Memory Access

- Read or write instruction references memory that is logically or physically invalid

```
// Invalid Memory Access

char *pStr = (char*) malloc(20);
free(pStr);
strcpy(pStr, "my string");
```

Kernel Resource Leak

- Kernel object handle is created but never closed

```
// Kernel Resource Leak

HANDLE hThread = CreateThread(0,
    8192, work0, NULL, 0, NULL);
return;
```

GDI Resource Leak

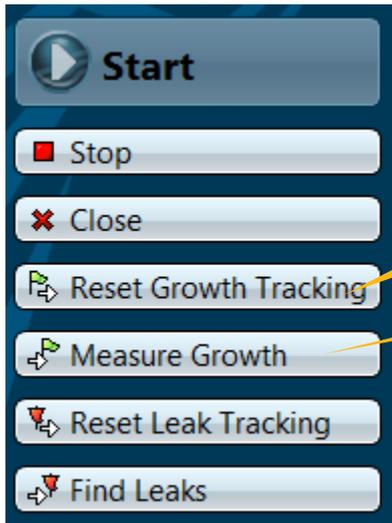
- GDI object is created but never deleted

```
// GDI Resource Leak

HPEN pen = CreatePen(0, 0, 0);
return;
```

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

Detect Memory Problems

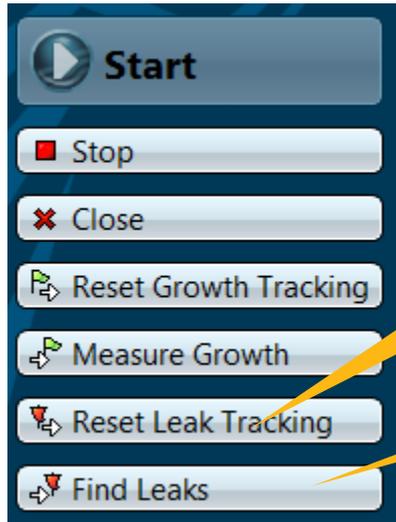
Target Analysis Type Collection Log Summary

Problems						
I	Type	Sources	Modules	Object ...	State	
P	Memory leak	ixe_mem_growth.cpp	ixe_mem_growth.e...	144	New	
P	Memory growth	[Unknown]; ix...	Unknown; ix...	272	New	
	Start memory growth det...	[Unknown]	Unknown		Not fixed	
	Memory growth	ixe_mem_growth.cpp:7	ixe_mem_growth.e...	272	New	
	End memory growth det...	[Unknown]	Unknown		Not fixed	

Code Locations: Memory growth

Description	Source	Function	Module	Object Size	Offset
Allocation site	ixe_mem_growth.cpp:7	transaction	ixe_mem_growth.exe	272	
5	{				ixe_mem_growth.exe!transaction
6	char *str;				ixe_mem_growth.exe!main - ix...
7	str = (char*) malloc(16);				ixe_mem_growth.exe!_tmainCRTSta
8	}				ixe_mem_growth.exe!mainCRTStart
9					kernel32.dll!BaseThreadInitThun

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

Detect Memory Problems

Target Analysis Type Collection Log Summary

ID	Type	Sources	Modules	Object Size	State
P1	Memory leak	ixe_mem_growth.cpp	ixe_mem_growth.exe	192	New
	Memory leak	ixe_mem_growth.cpp:7	ixe_mem_growth.exe	192	New
P2	Memory growth [Unknown];	ixe_mem_gr...	Unknown; ixe_mem_gr...	368	New

Code Locations: Memory leak

Description	Source	Function	Module	Object Size	Offset
Allocation site	ixe_mem_growth.cpp:7	transaction	ixe_mem_growth.exe	192	
5	{		ixe_mem_growth.exe!transaction		
6	char *str;		ixe_mem_growth.exe!main - ixe_		
7	str = (char*) malloc(16);		ixe_mem_growth.exe!_tmainCRTSt		
8			ixe_mem_growth.exe!mainCRTStar		
9	malloc(4);		kernel32.dll!BaseThreadInitThu		

Agenda

Intro to Intel® Inspector XE

Introduction

Memory problem analysis

Threading problem Analysis

Preparing setup for analysis

Managing analysis results

Advanced Features

Summary

Threading problem Analysis

Analyzed as software runs

- Data (workload) -driven execution
- Program needs to be multi-threaded
- Diagnostics reported incrementally as they occur

Includes monitoring of:

- Thread and Sync APIs used
- Thread execution order
 - Scheduler impacts results
- Memory accesses between threads

Analysis scope

- Native code: C, C++, Fortran
- Managed or mixed code: C# (.NET 2.0 to 3.5, .NET 4.0 with limitations)
- Code path must be executed to be analyzed
- Workload size doesn't affect ability to detect a problem

Data race

```
CRITICAL_SECTION cs;           // Preparation
int *p = malloc(sizeof(int)); // Allocation Site
*p = 0;
InitializeCriticalSection(&cs);
```

Write -> Write Data Race

Thread #1

```
*p = 1; // First Write
```

Thread #2

```
EnterCriticalSection(&cs);
*p = 2; // Second Write
LeaveCriticalSection(&cs);
```

Read -> Write Data Race

Thread #1

```
int x;
x = *p; // Read
```

Thread #2

```
EnterCriticalSection(&cs);
*p = 2; // Write
LeaveCriticalSection(&cs);
```

Deadlock

```
CRITICAL_SECTION cs1;  
CRITICAL_SECTION cs2;  
int x = 0;  
int y = 0;  
InitializeCriticalSection(&cs1); // Allocation Site (cs1)  
InitializeCriticalSection(&cs2); // Allocation Site (cs2)
```

Thread #1

```
EnterCriticalSection(&cs1);  
x++;  
    EnterCriticalSection(&cs2);  
    y++;  
    LeaveCriticalSection(&cs2);  
LeaveCriticalSection(&cs1);
```

Thread #2

```
EnterCriticalSection(&cs2);  
y++;  
    EnterCriticalSection(&cs1);  
    x++;  
    LeaveCriticalSection(&cs1);  
LeaveCriticalSection(&cs2);
```

Deadlock

1. EnterCriticalSection(&cs1); in thread #1
2. EnterCriticalSection(&cs2); in thread #2

Lock Hierarchy Violation

1. EnterCriticalSection(&cs1); in thread #1
2. EnterCriticalSection(&cs2); in thread #1
3. EnterCriticalSection(&cs2); in thread #2
4. EnterCriticalSection(&cs1); in thread #2

Agenda

Intro to Intel® Inspector XE

Introduction

Memory problem analysis

Threading problem Analysis

Preparing setup for analysis

Managing analysis results

Advanced Features

Summary

Prepare build for analysis

Compile

- Use dynamically linked thread-safe runtime libraries
`/MDd` on Windows
- Generate symbolic information
`/ZI` on Windows
- Disable optimization
`/Od` on Windows

Link

- Preserve symbolic information
`/DEBUG` on Windows
- Specify relocatable code sections
`/FIXED:NO` on Windows

Prior to using Inspector XE, sources should compile & link cleanly

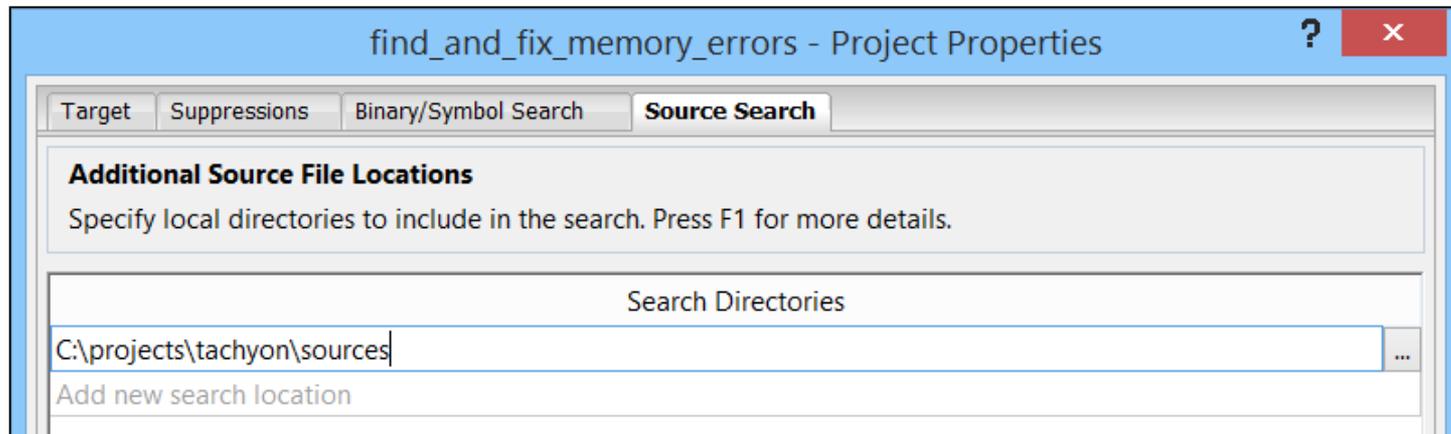
Search directories

Inspector XE needs to locate paths to:

- Binary files
- Symbol files
- Source files

No need for extra search directories configuration if:

- Binary, symbol and source files were not modified and moved
- Results are collected and viewed on the same machine



Correctness analyses overhead

Inspector XE tracks

- Thread and Sync APIs
- Memory accesses

Inspector XE 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!

Agenda

Intro to Intel® Inspector XE

Introduction

Memory problem analysis

Threading problem Analysis

Preparing setup for analysis

Managing analysis results

Advanced Features

Summary

Include and Exclude modules

The image shows a screenshot of the 'My Inspector XE Results - find_and_fix_threading_errors - Project Properties' dialog box. The 'Target' tab is active, showing the 'Launch Application' section. A yellow callout box points to the 'Advanced' section, specifically the 'Modules' options. Another yellow callout box points to the 'Modify...' button. A third yellow callout box points to the 'Edit Module' dialog box, which is open and shows a list of modules with 'C:\home\tbb_debug.dll' selected. The 'Edit Module' dialog box has a 'Delete' button and 'OK' and 'Cancel' buttons. The main dialog box has 'OK' and 'Cancel' buttons at the bottom.

3. Choose modules you want to include or exclude from analysis

1. There are two options:

- Include modules of interest
- Exclude unnecessary modules

2. Press Modify

Filtering - focus on what is important

Problems

I	Type	Sources	Modules	Object...	St.
P1	Mismatched allocation/de...	find_and_fix_memory...	find_and_fix_mem...		
P2	Memory leak	find_and_fix_memory...	find_and_fix_mem...	28672	
P3	Invalid memory access	find_and_fix_memory...	find_and_fix_mem...		
P4	Memory not deallocated	api.cpp; util.cpp; vide...	find_and_fix_mem...	10376	
	Memory not deallocated	video.cpp:82	find_and_fix_mem...	8192	
	Memory not deallocated	util.cpp:163	find_and_fix_mem...	1808	
	Memory not deallocated	api.cpp:218	find_and_fix_mem...	376	

Filter - Show only one source file

Filters Sort

Source	Count
api.cpp	1
find_and_fix_memory_error...	3
util.cpp	1
video.cpp	1

Module

Module	Count
find_and_fix_memory_error...	4

State

Problems

I	Type	Sources	Modules	Object...	St.
P4	Memory not deallocated	api.cpp; util.cpp; vide...	find_and_fix_mem...	10376	
	Memory not deallocated	video.cpp:82	find_and_fix_mem...	8192	
	Memory not deallocated	util.cpp:163	find_and_fix_mem...	1808	
	Memory not deallocated	api.cpp:218	find_and_fix_mem...	376	

Only related errors are shown

Filters Sort

Warning	Count
Warning	1

Type

Type	Count
Memory not deallocated	1

Source All

Source	Count
api.cpp	1 item(s)

Module

Module	Count
find_and_fix_memory_error...	1

State

Suppressions: manage false errors

Choose problem type

Suppressions are marked or hidden entirely

Choose stack frames to match the rule

Type	Sources	Modules	Objec...	St..
Memory leak	ixe_mem_growth.cpp	ixe_mem_growt...	144	
Memory growth	[Unknown]; ixe_mem...	Unknown; ixe_m...	272	

Use in Rule	Module	Function	Source	Line
<input checked="" type="checkbox"/>	ixe_mem_grow...	transaction	ixe_mem_gro ...	7
<input type="checkbox"/>	ixe_mem_grow...	main	ixe_mem_gro ...	16
<input type="checkbox"/>	ixe_mem_grow...	_tmainCRTStart...	crtexe.c	555
<input type="checkbox"/>	ixe_mem_grow...	mainCRTStartup	crtexe.c	370
<input type="checkbox"/>	kernel32.dll	BaseThreadInit...	* (any)	* (any)

- Suppressions are saved in one or more files
- Tool suppresses all files from specified folder(s)

Agenda

Intro to Intel® Inspector XE

Introduction

Memory problem analysis

Threading problem Analysis

Preparing setup for analysis

Managing analysis results

Advanced Features

Summary

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

2x-20x Detect Leaks
10x-40x Detect Memory Problems
20x-80x Locate Memory Problems

Analysis Time Overhead

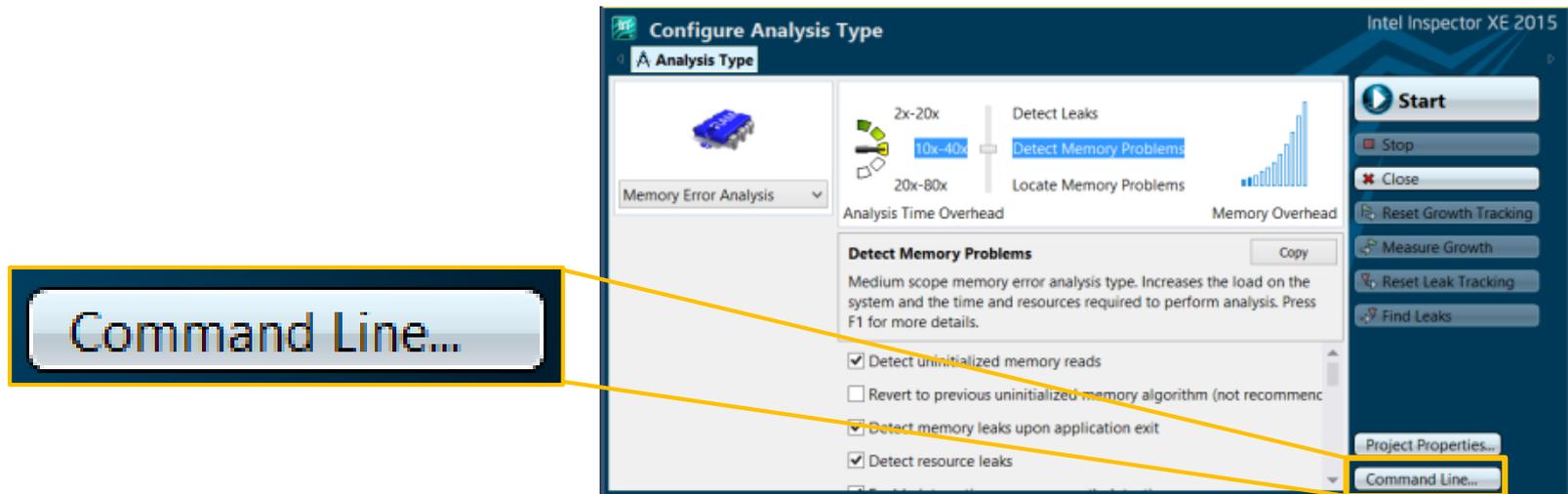
Detect Memory Problems Copy

Medium scope memory error analysis type. Increases the load on the system and the time and resources required to perform analysis. Press F1 for more details.

- Analyze without debugger
Run an analysis and report all detected problems. Use to view correctness issues without stopping in the debugger to examine them.
- Enable debugger when problem detected
Run an analysis under the debugger and stop every time a problem is detected. Use to allow investigation of every problem detected.
- Select analysis start location with debugger
Run target application under the debugger with analysis disabled until you choose to turn on analysis. Before starting, set a code breakpoint to stop execution prior to where you want analysis to begin. Sele...

Command Line Interface

- inspxe-cl is the command line:
 - windows: `C:\Program Files\Intel\Inspector XE\bin32\inspxe-cl.exe`
 - Linux: `/opt/intel/inspector_xe/bin64/inspxe-cl`
- Help:
`inspxe-cl -help`
- Set up command line with GUI



Automated regression testing

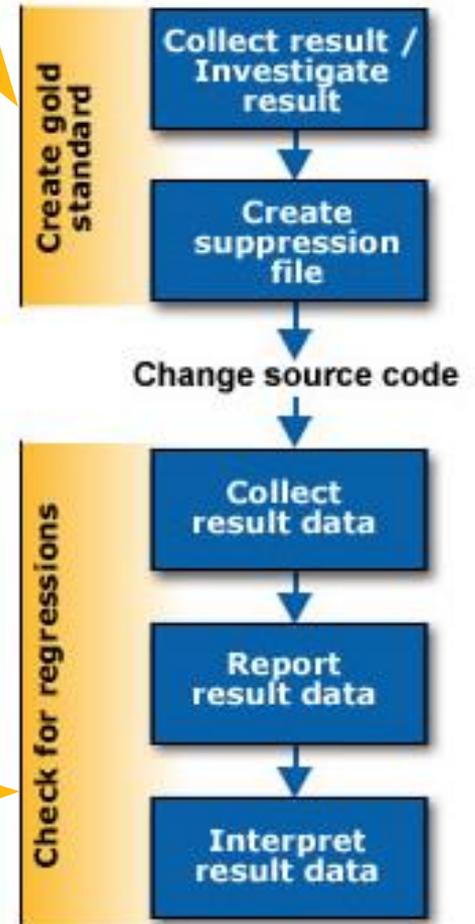
Data collection from script

- Command line interface (CLI) for running analysis
- Child process analysis

Reporting CLI

- Exporting results (pack and send)
- Text reports: XML, CSV and plain text
- Detect new problems automatically

Create a baseline



Check for regressions

Using the Intel® Inspector XE 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 Inspector XE: Summary

Advanced correctness checking

- Find issues that traditional testing misses
- Dynamic memory and threading error detection

Automated regression

- Command line interface
- Suitable for scripting

Wide analysis capabilities

- GUI data management
- Debugger integration

Ship high quality software products!

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.

Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

