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
- 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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data collection</td>
<td>• Dynamic Memory and Threading Analysis (including .NET* analysis)</td>
</tr>
<tr>
<td></td>
<td>• MPI applications analysis</td>
</tr>
<tr>
<td>Result analyses</td>
<td>• GUI data mining: source code analysis, filtering, exploring call paths, etc.</td>
</tr>
<tr>
<td></td>
<td>• Debugger integration</td>
</tr>
<tr>
<td></td>
<td>• Result comparison</td>
</tr>
<tr>
<td></td>
<td>• Problem life cycle management</td>
</tr>
<tr>
<td></td>
<td>• Command line interface (especially useful for regression testing)</td>
</tr>
<tr>
<td>GUI</td>
<td>• Microsoft* Visual Studio IDE integration (2010, 2012 and 2013)</td>
</tr>
<tr>
<td></td>
<td>• Stand alone GUI on both Windows* and Linux*</td>
</tr>
<tr>
<td>Compilers supported</td>
<td>• Microsoft* Visual* C++ and .NET*</td>
</tr>
<tr>
<td></td>
<td>• Intel® C/C++ Compiler XE 12.0 or higher</td>
</tr>
<tr>
<td></td>
<td>• Intel® Visual Fortran Compiler XE 12.0 or higher</td>
</tr>
<tr>
<td></td>
<td>• gcc</td>
</tr>
<tr>
<td>OS</td>
<td>• Windows* 7, 8, 8.1,</td>
</tr>
<tr>
<td></td>
<td>• Linux*: RedHat, Fedora, CentOS, SUSE, Debian, Ubuntu</td>
</tr>
<tr>
<td>Languages</td>
<td>• C/C++</td>
</tr>
<tr>
<td></td>
<td>• C# (.NET 2.0 to 3.5, .NET 4.0 with limitations)</td>
</tr>
<tr>
<td></td>
<td>• Fortran</td>
</tr>
</tbody>
</table>
Workflow: setup project

Specify Application, arguments and working directory
Workflow: select analysis and start

1. Select Analysis Type

2. Click Start
Workflow: manage results

- Code locations grouped into Problems to simplify results management
- Powerful filtration feature
- Double click on Problem to navigate to source
Workflow: navigate to sources

- Problematic line in source code:
  ```c
  bool video::next_frame()
  {
      if(!running) return false;
      g_updates++; // Fast but inaccurate counter. The data race h
      if(!threaded) while(loop_once(this));
      else if(g_handles[1]) {
          SetEvent(g_handles[1]);
          YIELD_TO_THREAD();
  }
  ```

- Call stacks

- All code locations for a problem

- Switch to disassembly for more details
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 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 growth
• a block of memory is allocated
• not deallocated, within a specific time segment during application execution.
• Severity level = (Warning)

// Memory leak
char *pStr = (char*) malloc(512);
return;

// Memory not deallocated
static char *pStr = malloc(512);
return;

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

Uninitialized memory access
• Read of an uninitialized memory location

Invalid Memory Access
• Read or write instruction references memory that is logically or physically invalid

Kernel Resource Leak
• Kernel object handle is created but never closed

GDI Resource Leak
• GDI object is created but never deleted

// Uninitialized Memory Access
void func()
{
    int a;
    int b = a * 4;
}

// Invalid Memory Access
char *pStr = (char*) malloc(20);
free(pStr);
strcpy(pStr, "my string");

// Kernel Resource Leak
HANDLE hThread = CreateThread(0, 8192, work0, NULL, 0, NULL);
return;

// 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
On-demand leak detection

Analysis Results:
- Memory Leak shown during run time
- Check code regions between points 'A' and 'B' for leaks
- Check daemon processes for leaks
- Check crashing processes for leaks
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](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

1. There are two options:
   - Include modules of interest
   - Exclude unnecessary modules

2. Press Modify

3. Choose modules you want to include or exclude from analysis
Filtering - focus on what is important

Filter – Show only one source file

<table>
<thead>
<tr>
<th>Problems</th>
<th>Type</th>
<th>Sources</th>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Mismatched allocation/deallocation</td>
<td>find_and_fix_memory_error.cpp</td>
<td>find_and_fix_memory_error.cpp</td>
</tr>
<tr>
<td>P2</td>
<td>Memory leak</td>
<td>find_and_fix_memory_error.cpp</td>
<td>find_and_fix_memory_error.cpp</td>
</tr>
<tr>
<td>P3</td>
<td>Invalid memory access</td>
<td>find_and_fix_memory_error.cpp</td>
<td>find_and_fix_memory_error.cpp</td>
</tr>
<tr>
<td>P4</td>
<td>Memory not deallocated</td>
<td>api.cpp; util.cpp; video.cpp</td>
<td>find_and_fix_memory_error.cpp</td>
</tr>
</tbody>
</table>

Only related errors are shown

<table>
<thead>
<tr>
<th>Problems</th>
<th>Type</th>
<th>Sources</th>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>Memory not deallocated</td>
<td>api.cpp; util.cpp; video.cpp</td>
<td>find_and_fix_memory_error.cpp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>video.cpp:82</td>
<td>find_and_fix_memory_error.cpp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>util.cpp:163</td>
<td>find_and_fix_memory_error.cpp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>api.cpp:218</td>
<td>find_and_fix_memory_error.cpp</td>
</tr>
</tbody>
</table>

Source

<table>
<thead>
<tr>
<th>Source</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>api.cpp</td>
<td>1</td>
</tr>
<tr>
<td>find_and_fix_memory_error.cpp</td>
<td>3</td>
</tr>
<tr>
<td>util.cpp</td>
<td>1</td>
</tr>
<tr>
<td>video.cpp</td>
<td>1</td>
</tr>
</tbody>
</table>
Suppressions: manage false errors

- 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
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
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
  
  ```bash
  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
  
  ```bash
  ls | grep inspector_results
  ``` on Linux

- Launch the GUI and view the results for each particular rank
  
  ```bash
  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!
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