Introduction to Intel® Inspector XE

The Inspector XE workflow and walk thru

Dynamic Memory and Threading Analysis

Static Analysis

Readying your sources and builds

Managing analysis results

Team collaboration

Advanced features
Where are my application’s...

**Memory Errors**
- Invalid Accesses
- Memory Leaks
- Uninitialized Memory Accesses

**Threading Errors**
- Races
- Deadlocks
- Cross Stack References

**Security Errors**
- Buffer overflows and underflows
- Incorrect pointer usage
- Over 250 error types...

- Developing threaded applications can be complex and expensive
- New class of correctness problems are caused by the interaction between concurrent threads

Multi-threading problems are hard to reproduce, difficult to debug and expensive to fix!
## Key Features at a glance

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyses</td>
<td>• Dynamic Memory and Threading Analysis (including .NET* analysis)</td>
</tr>
<tr>
<td></td>
<td>• Static Analysis (with Intel® Compiler)</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++ .NET*</td>
</tr>
<tr>
<td></td>
<td>• Intel® Parallel Composer and Intel® Composer XE</td>
</tr>
<tr>
<td></td>
<td>• gcc</td>
</tr>
<tr>
<td>OS</td>
<td>• Windows XP*, Vista*, 7, 8</td>
</tr>
<tr>
<td></td>
<td>• Linux* (various distros)</td>
</tr>
<tr>
<td>Languages</td>
<td>• C/C++</td>
</tr>
<tr>
<td></td>
<td>• C#</td>
</tr>
<tr>
<td></td>
<td>• Fortran</td>
</tr>
</tbody>
</table>
Standalone GUI for Windows* and Linux*

Configure Analysis Type

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

Locate Memory Problems

Widest scope memory 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.

- Detect resource leaks
- Report still-allocated memory at application exit
- Stack frame depth: 16
- Analyze stack accesses

Start
- Stop
- Set Transaction Start
- Set Transaction End
- Close

Project Properties...
Command Line...
Visual Studio* Integration on Windows*

Locate Memory Problems

Problems

<table>
<thead>
<tr>
<th>ID</th>
<th>Problem</th>
<th>Sources</th>
<th>Modules</th>
<th>Object Size</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>P5</td>
<td>Memory leak</td>
<td>find_and_fix_memory_errors.c</td>
<td>find_and_fix_memory_errors.exe</td>
<td>1232</td>
<td>Not fixed</td>
</tr>
<tr>
<td>P6</td>
<td>Memory leak</td>
<td>find_and_fix_memory_errors.c</td>
<td>find_and_fix_memory_errors.exe</td>
<td>896</td>
<td>Not fixed</td>
</tr>
<tr>
<td>P7</td>
<td>Memory leak</td>
<td>find_and_fix_memory_errors.c</td>
<td>find_and_fix_memory_errors.exe</td>
<td>1008</td>
<td>New</td>
</tr>
<tr>
<td>P8</td>
<td>Memory leak</td>
<td>find_and_fix_memory_errors.c</td>
<td>find_and_fix_memory_errors.exe</td>
<td>672</td>
<td>New</td>
</tr>
</tbody>
</table>

Code Locations: Memory leak

Description: find_and_fix_memory_errors.c

<table>
<thead>
<tr>
<th>Allocation site</th>
<th>Function</th>
<th>Module</th>
<th>Object Size</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int serial=1;</td>
<td>find_and_fix_memory_errors.exe!open</td>
<td>find_and_fix_memory_errors.exe!ex</td>
<td>tbb_debug.dll!local_wait_for_all</td>
<td>1232</td>
</tr>
<tr>
<td>unsigned int mboxsize = sizeof(unsigned int);</td>
<td>find_and_fix_memory Errors.exe!ex</td>
<td>tbb_debug.dll!local_spawn_root</td>
<td>tbb debug.dll!spawn root and wait</td>
<td>161</td>
</tr>
<tr>
<td>unsigned int * local_mbox = (unsigned int);</td>
<td>find_and_fix_memory_errors.exe!open</td>
<td>find_and_fix_memory_errors.exe!ex</td>
<td>tbb_debug.dll!local_wait_for_all</td>
<td>162</td>
</tr>
<tr>
<td>for (unsigned int i=0; i&lt;= (mboxsize/</td>
<td>find_and_fix_memory Errors.exe!ex</td>
<td>tbb_debug.dll!local_spawn_root</td>
<td>tbb debug.dll!spawn root and wait</td>
<td>163</td>
</tr>
<tr>
<td>tbb_debug.dll!spawn root and wait</td>
<td>find_and_fix_memory_errors.exe!open</td>
<td>find_and_fix_memory_errors.exe!ex</td>
<td>tbb_debug.dll!local_wait_for_all</td>
<td>164</td>
</tr>
<tr>
<td>tbb_debug.dll!spawn root and wait</td>
<td>find_and_fix_memory Errors.exe!ex</td>
<td>tbb_debug.dll!local_spawn_root</td>
<td>tbb debug.dll!spawn root and wait</td>
<td>165</td>
</tr>
</tbody>
</table>

Filters

- Severity: 7
- Problem: Invalid memory access
- Warning: 1

Timeline
Agenda

• Intro to Intel® Inspector XE
• **The Inspector XE workflow and walk thru**
• Dynamic Memory and Threading Analysis
• Static Analysis
• Readying your sources and builds
• Managing analysis results
• Team collaboration
• Advanced features
Workflow
Workflow: **Setup Project**

Specify Application, arguments and working directory.
Workflow: Select Analysis and Start

1. Select Analysis Type

2. Click Start
Workflow: **Manage Results and Filter**

- Code locations grouped into Problems to simplify results management.

- Powerful filtration feature!

- Double click on Problem to navigate to source (next slide).
**Workflow: Navigate to sources**

- **Call stacks**
- **Source code panes** annotated for ease of use
Workflow: *Timeline view*

Individual Code Locations are seen in Timeline view in the context of their respective threads.

Hover gives details.
Agenda

• Intro to Intel® Inspector XE
• The Inspector XE workflow and walk thru
• **Dynamic Memory and Threading Analysis**
• Static Analysis
• Readying your sources and builds
• Managing analysis results
• Team collaboration
• Advanced features
Memory 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.

Code path must be executed to be analyzed
Threading Analysis

Dynamic 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

Code path must be executed to be analyzed
Threading Analysis (.NET)

.NET support is only for Windows C#.
- Managed code or mixed mode code

Only Threading Analysis is supported

Memory Checking works only on the native portion
- Does not track any memory in managed heap

Supported .NET versions are from 2.0 to 3.5.
- Limited support for .NET 4.0
Agenda

- Intro to Intel® Inspector XE
- The Inspector XE workflow and walk thru
- Dynamic Memory and Threading Analysis
- Static Analysis
- Readying your sources and builds
- Managing analysis results
- Team collaboration
- Advanced features
Static Analysis (SA)

- SA invokes Intel® Compiler to run in a special mode to perform link time static analysis.
- No benchmark is needed as source code is examined statically across source file boundaries and so is more powerful than intra-file static analysis.
- SA examines all possible execution paths and variable values, not just those that are provoked during testing.
- Program can be single-threaded or multi-threaded.
- **SA feature is in Intel® Parallel Studio XE only and needs both the Intel® Compiler as well as the Inspector XE.**
Running Static Analysis

• Run SA **only** after sources build cleanly

• SA can work on partial program or library but full build is recommended.
  - Results are triaged and fixed in the Inspector XE GUI. You can use regular compiler to generate production binaries.

• Includes monitoring of:
  - Thread and Sync APIs used
  - Memory declaration, allocation, use and liveliness

SA is simple to setup and run.
Run static analysis in few clicks

You don’t need to explicitly change existing build objects or configuration to get SA result on Windows
Cyclomatic Complexity metrics

Reduce complexity to make code more maintainable
### SA vs. The Inspector XE Comparison

<table>
<thead>
<tr>
<th>Comparison attribute</th>
<th>Inspector XE</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis type</td>
<td>Dynamic</td>
<td>Static</td>
</tr>
<tr>
<td>When is code analyzed</td>
<td>Runtime</td>
<td>Compile/Link time</td>
</tr>
<tr>
<td>What code is analyzed</td>
<td>Executed code</td>
<td>All code</td>
</tr>
<tr>
<td>What memory is analyzed</td>
<td>Active and allocated memory only</td>
<td>All declared memory</td>
</tr>
<tr>
<td>Runtime Tests</td>
<td>Tests Needed</td>
<td>No tests needed</td>
</tr>
<tr>
<td>Products needed</td>
<td>Inspector XE</td>
<td>Parallel Studio XE</td>
</tr>
<tr>
<td>Time dilation</td>
<td>Can be invasive</td>
<td>Relatively moderate</td>
</tr>
</tbody>
</table>

Static and Dynamic Analyses complement each other!
Agenda

• Intro to Intel® Inspector XE
• The Inspector XE workflow and walk thru
• Dynamic Memory and Threading Analysis
• Static Analysis
• **Readying your sources and builds**
• Managing analysis results
• Team collaboration
• Advanced features
Correctness analyses dilate time & memory

Adds calls to library to record information using PIN based *Just in time* instrumentation.
- Thread and Sync APIs
- Memory accesses

Increases *execution time* and *memory consumed* (*potentially significantly*)

Use *small* data sets (workloads)
- Execution time and space is *expanded*
- Multiple runs over different paths yield best results

The Inspector XE dilates both time and memory consumed significantly!
Workload Guidelines

Execute problem code once per thread to be identified

Use smallest possible working data set
- Minimize data set size
  - Smaller image sizes
- Minimize loop iterations or time steps
  - Simulate minutes rather than days
- Minimize update rates
  - Lower frames per second

Scale down workload to speed up analysis!
Prepare your 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
Recommended Analysis Sequence

Run Static Analysis

Fix Problems

Run Memory Analysis

Fix Problems

Run Threading Analysis

Fix Problems

Cleaner code!

Use both Static and Dynamic analyses for cleaner code!
Agenda

• Intro to Intel® Inspector XE
• The Inspector XE workflow and walk thru
• Dynamic Memory and Threading Analysis
• Static Analysis
• Readying your sources and builds
• **Managing analysis results**
• Team collaboration
• Advanced features
Filtering lets you focus on what’s important

Before – All Errors

After – Only errors from one source file

1. Filter – Show only one source file

2. Error count drops
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
## Problem State Lifecycle
Makes problems easier to manage

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>Detected by this run</td>
</tr>
<tr>
<td>Not Fixed</td>
<td>Previously seen error detected by this run</td>
</tr>
<tr>
<td>Not a Problem</td>
<td>Set by user - tool will not change</td>
</tr>
<tr>
<td>Confirmed</td>
<td>Set by user - tool will not change</td>
</tr>
<tr>
<td>Fixed</td>
<td>Set by user</td>
</tr>
<tr>
<td>Regression</td>
<td>Error detected with previous state of “Fixed”</td>
</tr>
<tr>
<td>Deferred</td>
<td>Set by user</td>
</tr>
</tbody>
</table>
Suppressions: manage false errors

- Suppressions are saved in one or more files
- Tool suppresses all files from specified folder(s)
- Private & Public suppression folders

Team sharing of suppression files reduces false error count
Agenda

• Intro to Intel® Inspector XE
• The Inspector XE workflow and walk thru
• Dynamic Memory and Threading Analysis
• Static Analysis
• Readying your sources and builds
• Managing analysis results
• **Team collaboration**
• Advanced features
Team collaboration

• Share comments with the team
  – Provide a way to export meaningful and sufficient problem description as a plain text
• Share public suppression rules with the team
• Prevent simultaneous result/project opening by two or more users
Merge arbitrary results

• Merge states from another result to current one
• Incorporate states from other users
Agenda

• Intro to Intel® Inspector XE
• The Inspector XE workflow and walk thru
• Dynamic Memory and Threading Analysis
• Static Analysis
• Readying your sources and builds
• Managing analysis results
• Team collaboration
• Advanced features
Integrated Debugger Support

• 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
  – Intel® Debugger
Analyze Memory Growth
Transactional Applications

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 deallocated during the time period
Child Program Analysis

Running a top level script is the norm for some Linux apps. For such cases, a different *Child Program* can be analyzed (not necessarily the app launched by Inspector XE).

Limitations:

- Only the first instance of *Child Program* will be analyzed by Inspector XE analysis.
- *Child Program* name is the one shown in Windows Task Manager or the name shown in “ps –aef” on Linux.
- Multi-process analysis is not supported for .NET applications.

Child Program Analysis is very useful in multi-process scenario
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:**
  - inspxe-cl -collect-list
  - inspxe-cl -collect ti2 -- MyApp.exe
  - inspxe-cl -report problems

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

To generate a report:

\texttt{inspxe-cl \textasciitilde R=<report-type> \textless results directory name\textgreater}

Sample commands:

\texttt{inspxe-cl \textasciitilde report-list}
\texttt{inspxe-cl \textasciitilde report=summary}
\texttt{inspxe-cl \textasciitilde report=problems}

Example:

\texttt{cd /home/user/testProgram/r000mi}
\texttt{cd ..}
\texttt{inspxe-cl \textasciitilde R=observations r000mi}

Report generation is very convenient to use from command line.
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 insp_results.<rank#> on Linux
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 © Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, Core,
VTune, and Cilk 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