Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Static Analysis
5. Readying your sources and builds
6. Managing analysis results
7. Team collaboration
8. Advanced features
Intel® Inspector XE

Is a *debugging tool* for software. “Correctness Analyzer”

Has an intuitive GUI. Provides powerful results management, navigation, and filtering.

Easy to use *one-click* help for diagnostics (Possible causes and solution suggestions)

Finds threading bugs in OpenMP*, Cilk™ Plus, Intel® Threading Building Blocks, Win32* and Posix Threads threaded software

Locates bugs quickly that can take days to find using traditional methods and tools
- Isolates problems, not the symptoms
- Bug does not have to occur to find it!

The Inspector XE has a comprehensive portfolio of analyses and an easy to use GUI for effective and efficient results management.
Motivation for Inspector XE

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>
</table>
| **Analyses**          | • Dynamic Memory and Threading Analysis  
                        • Static Analysis (with Intel® Compiler)                                                                                             |
                        • Stand alone GUI on both Windows and Linux                                                                                      |
| **Compilers supported** | • Microsoft* Visual* C++ .NET*  
                        • Intel® Parallel Composer and Intel® Composer XE  
                        • Gcc                                                                                                                            |
| **OS**                | • Windows XP, Vista, 7, 8  
                        • Linux (various distros)                                                                                                                                 |
| **Languages**         | • C/C++  
                        • C#  
                        • Fortran                                                                                                                          |
Standalone GUI for Windows* and Linux*
Visual Studio* Integration on Windows*

Locate Memory Problems

Problems

ID  Problem         Sources                           Modules             Object Size  State  
P5   Memory leak    find_and_fix_memory_errors.c...  find_and_fix_memo...  1232    Not fixed
P6   Memory leak    find_and_fix_memory_errors.c...  find_and_fix_memo...  896     Not fixed
P7   Memory leak    find_and_fix_memory_errors.c...  find_and_fix_memo...  1008    New
P8   Memory leak    find_and_fix_memory_errors.c...  find_and_fix_memo...  672     New

Code Locations: Memory leak

Description: find_and_fix_memory_errors.exe!op
Function: find_and_fix_memory_errors.exe!op
Module: find_and_fix_memory_errors.exe!op
Object Size: 1232
Offset: 161

161 unsigned int serial=1;
162 unsigned int mboxsize = sizeof(unsigned int);
163 unsigned int * local_mbox = (unsigned int *)malloc(mboxsize);
164 for (unsigned int i=0; i<=(mboxsize/)
165 1)

Timeline

Ready
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Static Analysis
5. Readying your sources and builds
6. Managing analysis results
7. Team collaboration
8. Advanced features
Workflow: Setup Project

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

1. Select Analysis Type

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

2. Click Start

**Locate Deadlocks and Data Races**

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
- Details
Workflow: Manage Results and Filter

- Powerful filtration feature!
- Code locations grouped into Problems to simplify results management
- Double click on Problem to navigate to source (next slide)
Workflow: Navigate to sources

Source code panes annotated for ease of use

Call stacks
Workflow: *Timeline view*

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

Hover gives details.
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Static Analysis
5. Readying your sources and builds
6. Managing analysis results
7. Team collaboration
8. 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#.

Only Threading Analysis is supported for managed code or mixed mode code.

Memory Checking works only on the native portion of a mixed-mode application (does not track any memory in managed heap).

Analysis can be performed using the Inspector XE integrated within Visual Studio* or using Windows Standalone Inspector GUI.

Supported .NET versions are from 2.0 to 3.5.

Limited support for .NET 4.0
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Static Analysis
5. Readying your sources and builds
6. Managing analysis results
7. Team collaboration
8. 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 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
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
## Static Analysis vs. 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

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Static Analysis
5. Readying your sources and builds
6. Managing analysis results
7. Team collaboration
8. 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
  - E.g. smaller image sizes

- Minimize loop iterations or time steps
  - E.g. simulate minutes rather than days

- Minimize update rates
  - E.g. 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
  Don’t use –static on Linux
- Generate symbolic information
  /ZI on Windows
  -g on Linux
- Disable optimization
  /Od on Windows
  -O0 on Linux

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

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Static Analysis
5. Readying your sources and builds
6. Managing analysis results
7. Team collaboration
8. Advanced features
Filtering

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 marked (shown) or hidden entirely

Choose stack matching rule

Be specific or select group of similar problems

Suppressions are saved in one or more files

Tool suppresses all files from specified folder(s)

Team sharing of suppression files reduces false error count
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Static Analysis
5. Readying your sources and builds
6. Managing analysis results
7. Team collaboration
8. Advanced features
Team collaboration

Share comments with the team

- Provides a way to export meaningful and sufficient problem description as plain text

Share public suppression rules with the team

Prevent simultaneous result/project opening by two
Merge arbitrary results

Merge states from another result to current one

Incorporate states from other users
Agenda

1. Intro to Intel® Inspector XE
2. The Inspector XE workflow and walk thru
3. Dynamic Memory and Threading Analysis
4. Static Analysis
5. Readying your sources and builds
6. Managing analysis results
7. Team collaboration
8. 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 Application 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

### Analyze Application 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

#### 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></td>
<td>Memory growth</td>
<td>[Unknown]; test.cpp</td>
<td>Unknown; test.exe</td>
<td>144</td>
<td>Not fixed</td>
</tr>
<tr>
<td>P1</td>
<td>Transaction start</td>
<td>[Unknown]</td>
<td>Unknown</td>
<td>144</td>
<td>Not fixed</td>
</tr>
<tr>
<td></td>
<td>Memory growth</td>
<td>test.cpp:15</td>
<td>test.exe</td>
<td>144</td>
<td>Not fixed</td>
</tr>
<tr>
<td></td>
<td>Transaction end</td>
<td>[Unknown]</td>
<td>Unknown</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Code Locations: Memory growth

<table>
<thead>
<tr>
<th>Description</th>
<th>Source</th>
<th>Function</th>
<th>Module</th>
<th>Object Size</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation site</td>
<td>test.cpp:15</td>
<td>transaction</td>
<td>test.exe</td>
<td>144</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>{</td>
<td>char *str;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>char *str;</td>
<td>malloc(16);</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>str = (char*) malloc(16);</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Other brands and names are the property of their respective owners.*
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 scenarios*
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

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{inspex-cl -R=<report-type> <results directory name>}

Sample commands:

\texttt{inspex-cl -report-list}
\texttt{inspex-cl -report=summary}
\texttt{inspex-cl -report=problems}

Example:

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

\textit{Report generation is very convenient to use from command line.}
Using the Intel® Inspector XE with Hybrid  MPI + Threads

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
```
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
Legal Disclaimer

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.

Performance tests and ratings are measured using specific computer systems and/or components
and reflect the approximate performance of Intel products as measured by those tests. Any
difference in system hardware or software design or configuration may affect actual performance.
Buyers should consult other sources of information to evaluate the performance of systems or
components they are considering purchasing. For more information on performance tests and on

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino
Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386,
Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside,
NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel
XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium
Inside, skool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon
Inside are trademarks of Intel Corporation in the U.S. and other countries.

http://intel.com/software/products

*Other names and brands may be claimed as the property of others.