INTERMITTENT MULTI-THREADING BUGS: FIND AND SQUASH RACES, DEADLOCKS, AND MEMORY BUGS

Memory & Thread Debugger
Here is What Will Be Covered

Overview
Memory/Thread analysis
New Features
Deep dive into debugger integrations
Demo
Call to action
Analysis Tools for Diagnosis

Intel® Parallel Studio XE

- Intel® Trace Analyzer & Collector (ITAC)
- Intel MPI Snapshot
- Intel MPI Tuner

Intel® Inspector
Find any correctness errors in your threads and memory!

Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
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
- Correctness tools find defects during development prior to shipment
- Reworking defects is 40%-50% of total project effort
- 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

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 compiler¹
- 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

¹That 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
Race Conditions Are Difficult to Diagnose
They only occur occasionally and are difficult to reproduce

<table>
<thead>
<tr>
<th></th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Shared Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Correct</strong></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Read count</td>
<td></td>
<td>← 0</td>
<td></td>
</tr>
<tr>
<td>Increment</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Write count</td>
<td>→ 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read count</td>
<td>← 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increment</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write count</td>
<td>→ 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Shared Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Incorrect</strong></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Read count</td>
<td></td>
<td>← 0</td>
<td></td>
</tr>
<tr>
<td>Increment</td>
<td></td>
<td></td>
<td>← 0</td>
</tr>
<tr>
<td>Write count</td>
<td></td>
<td>→ 1</td>
<td></td>
</tr>
<tr>
<td>Read count</td>
<td>← 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increment</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Write count</td>
<td>→ 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Correct:
- Read count: 0
- Increment: 0
- Write count: 1

Incorrect:
- Read count: 0
- Increment: 0
- 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!
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

(1) Filter – Show only one source file

(2) Error count drops

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

Microsoft Visual Studio* and GNU gdb* or Intel® Debugger (on Linux*)
Break At Just The Right Time
Intel® Inspector - Memory & Thread Debugger

Memory Errors

<table>
<thead>
<tr>
<th>Problems</th>
<th>Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Type</td>
</tr>
<tr>
<td>P1</td>
<td>Mismatched allocation/deallocation</td>
</tr>
<tr>
<td>P2</td>
<td>Memory leak</td>
</tr>
<tr>
<td><strong>P3</strong></td>
<td><strong>Invalid memory access</strong></td>
</tr>
<tr>
<td>P4</td>
<td>Memory growth</td>
</tr>
<tr>
<td>P5</td>
<td>Memory growth</td>
</tr>
<tr>
<td>P6</td>
<td>Memory growth</td>
</tr>
</tbody>
</table>

Threading Errors

<table>
<thead>
<tr>
<th>Problems</th>
<th>Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Type</td>
</tr>
<tr>
<td>P1: Data race</td>
<td>winvideo.h</td>
</tr>
</tbody>
</table>

- 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.
## Productive Memory & Threading Debugger

**Intel® Inspector**

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

**Easier & Faster Debugging of Memory & Threading Errors**
WHAT'S NEW
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.
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.
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