Intermittent Multi-Threading Bugs: Find and Squash Races, Deadlocks, and Memory Bugs

Memory & Thread Debugger

Munara Tolubaeva

Software Technical Consulting Engineer
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 © 2016, 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
Here is What Will Be Covered

• Overview
• Memory/Thread analysis
• Deep dive into debugger integrations
• Inspector 2017 Features
Analysis Tools for Diagnosis

Intel® Parallel Studio XE

Cluster Scalable?

Tune MPI

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

Effective threading?

Vectorize

Memory Bandwidth Sensitive?

Optimize bandwidth

Thread

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

Intel® VTune™ Amplifier

Intel® Advisor

Intel® VTune™ Amplifier

Optimization Notice
Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
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

Intel® Inspector dramatically sped up our ability to track down difficult to isolate threading errors before our packages are released to the field.

Peter von Kaenel, Director, Software Development, Harmonic Inc.

http://intel.ly/inspector-xe

Part of Intel® Parallel Studio For Windows* and Linux*
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 compiler
- 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

Optimization Notice 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

- Set up
  - Choose/Create project
- Run dynamic analysis
  - Configure project
  - Configure dynamic analysis
- Collect result
  - Rebuild application
  - Launch analysis in conjunction with debugger to stop at problem(s) of interest
- Investigate result
  - Choose problem
  - Interpret result data/Resolve issue
  - Examine application state
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)
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
Race Conditions Are Difficult to Diagnose
They only occur occasionally and are difficult to reproduce

<table>
<thead>
<tr>
<th>Correct</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Thread 1</strong></td>
<td><strong>Thread 2</strong></td>
</tr>
<tr>
<td>Read count</td>
<td>✈</td>
</tr>
<tr>
<td>Increment</td>
<td>✈</td>
</tr>
<tr>
<td>Write count</td>
<td>➤</td>
</tr>
<tr>
<td>Read count</td>
<td>✈</td>
</tr>
<tr>
<td>Increment</td>
<td>1</td>
</tr>
<tr>
<td>Write count</td>
<td>➤</td>
</tr>
</tbody>
</table>
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!

**Optimization Notice**

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.*
## Easy Problem Management

Quickly see new problems and regressions

<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 &lt;b&gt;not&lt;/b&gt; change)</td>
</tr>
<tr>
<td>Confirmed</td>
<td>Set by user (tool will &lt;b&gt;not&lt;/b&gt; change)</td>
</tr>
<tr>
<td>Fixed</td>
<td>Set by user (tool &lt;b&gt;will&lt;/b&gt; change)</td>
</tr>
<tr>
<td>Regression</td>
<td>Error detected with previous state of “Fixed”</td>
</tr>
</tbody>
</table>

---

**Optimization Notice**

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.*
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

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

Choose two results of the same analysis type

Result 1:
C:samples\tachyon_insp.xe\vc10\My Inspector Results - find_and_fix_threading_errors\r000ti2\r000ti2.inspxe

Result 2:
C:samples\tachyon_insp.xe\vc10\My Inspector Results - find_and_fix_threading_errors\r001ti2\r001ti2.inspxe

Compared Result

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Sources</th>
<th>Modules</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Data race</td>
<td>find_and_fix_threading_errors.cpp</td>
<td>find_and_fix_threading_errors.exe</td>
<td>Both</td>
</tr>
<tr>
<td>P2</td>
<td>Data race</td>
<td>winvideo.h</td>
<td>find_and_fix_threading_errors.exe</td>
<td>Both</td>
</tr>
</tbody>
</table>
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.
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.
Intuitive problem solving using debugger integrations

```cpp
// Refresh screen picture
bool video::next_frame()
{
    if (!running) return false;

    g_updates++;  // Fast but inaccurate counter. The data race here is benign.
    if (!threaded) while (loop_once(this));
    else if (g_handles[1]) {
        SetEvent(g_handles[1]);
        YIELD_TO_THREAD();
    }
    return true;
}
```

Microsoft Visual Studio* and GNU gdb* or Intel® Debugger (on Linux*)
Work Smarter & Faster
Intel® Inspector - Memory & Thread Debugger

Precise Error Suppression

Precise, easy to edit, team shareable.
Choose which stack frame to suppress.
Eliminate the false, not the real errors.

```c
#include <example.h>
Type = { uninitialized_memory_access }
Stacks = {
    mod=a.out, func=update_x;
    func=main;
}

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 */

Find and diagnose errors with less effort.

Speed-up analysis by limiting its scope.
Analyze only during the execution of the suspected problem.
# 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
Intel Inspector 2017 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
- Support for C++17 std::shared_mutex and Windows SRW Locks during threading analysis
Variable name detection for threading analysis