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

Memory & Thread Debugger

Munara Tolubaeva

Software Technical Consulting Engineer
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?

N
Tune MPI

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

Effective threading?

N
Thread

Y
Vectorize

Memory Bandwidth Sensitive?

N
Optimize bandwidth

Y

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.
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
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
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 leak
char *pStr = (char*) malloc(512);
return;
```

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

**Memory growth**
- a block of memory is allocated
- not deallocated, within a specific time segment during application execution.
- Severity level = (Warning)

```
// Memory growth
// Start measuring growth
static char *pStr = malloc(512);
// Stop measuring growth
```
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>Thread 1</td>
<td>Thread 2</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></td>
</tr>
<tr>
<td>Write count</td>
<td>→</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.*
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!

![Image of Fortran analysis tools]

### Problems

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Modules</th>
<th>Object Size</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Memory leak</td>
<td>nqueens_memory.f90 memory_issues.exe</td>
<td>64</td>
<td>New</td>
</tr>
</tbody>
</table>

### Code Locations: Memory leak

- **Description**: Allocation site nqueens_memory.f90:50
- **Source**: NQUEENS
- **Function**: memory_issues.exe
- **Module**: NQUEENS
- **Object Size**: 64

### Code Locations: Data race

- **Description**: Read
- **Source**: nqueens_threading.f90:117
- **Module**: NQUEENS_ip_SETQUEEN

- **Write**: nqueens_threading.f90:117
- **Source**: NQUEENS_ip_SETQUEEN

---

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

14
## 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 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 (tool will change)</td>
</tr>
<tr>
<td>Regression</td>
<td>Error detected with previous state of “Fixed”</td>
</tr>
</tbody>
</table>
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.
Variable name detection for threading analysis
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.
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>Type</td>
<td></td>
</tr>
<tr>
<td>Mismatched allocation/deallocation</td>
<td>View Source</td>
</tr>
<tr>
<td>Memory leak</td>
<td>Edit Source</td>
</tr>
<tr>
<td>Invalid memory access</td>
<td>Copy to Clipboard</td>
</tr>
<tr>
<td>Memory growth</td>
<td>Explain Problem</td>
</tr>
<tr>
<td>Memory growth</td>
<td>Create Problem Report...</td>
</tr>
</tbody>
</table>

Threading Errors

<table>
<thead>
<tr>
<th>Problems</th>
<th>Type</th>
<th>Sources</th>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>Data race</td>
<td>winvideo.h</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Data race</td>
<td>winvideo.h:270</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>Data race</td>
<td>winvideo.h:201</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Data race</td>
<td>winvideo.h:202</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>Data race</td>
<td>winvideo.h:202</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>Data race</td>
<td>winvideo.h:202</td>
<td></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.
Intuitive problem solving using debugger integrations

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

Precise Error Suppression

```c
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

```c
__ittSuppressPush(__ittSuppressThreadingErrors);
/* Any threading errors here are ignored */
__ittSuppressPop();
/* 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**