

HLRS Workshop – 27.10.2022

# Dynamic Debugging with Intel<sup>®</sup> Inspector

Heinrich Bockhorst

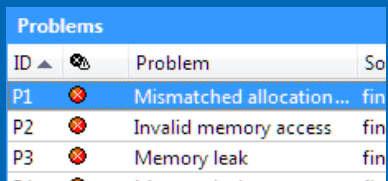


# Agenda

- Introduction to Inspector
- GUI usage
- Command line usage
- Results
- Demo - Next steps

# Motivation for Intel® Inspector

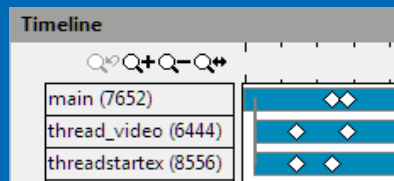
## Memory Errors



ID	Problem	So
P1	Mismatched allocation...	fin
P2	Invalid memory access	fin
P3	Memory leak	fin

- Invalid Accesses
- Memory Leaks
- Uninitialized Memory Accesses

## Threading Errors



- Data Races
- Deadlocks
- Cross Stack References

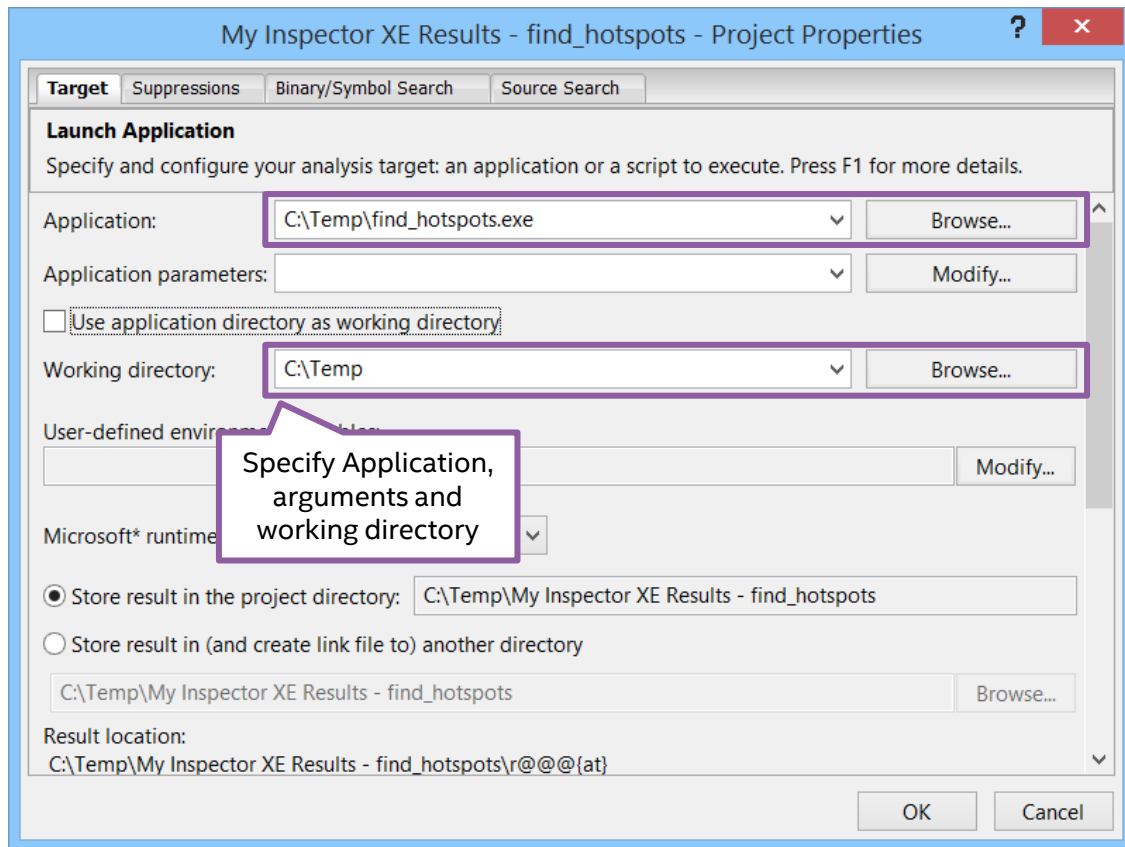
## Multi-threading problems

- Hard to reproduce,
- Difficult to debug
- Expensive to fix



Let the tool do it for you

# Workflow: setup project



# Workflow: select analysis and start

**Configure Analysis Type**

INTEL INSPECTOR 2019

Analysis Type

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

10x-40x Detect Deadlocks

20x-80x Detect Deadlocks and Data Races

40x-160x **Locate Deadlocks and Data Races**

Analysis Time Overhead

Memory Overhead

**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

Use maximum resources

Analyze without debugger

Run an analysis and report all detected problems. Use to view correctness issues without stopping in the debugger to examine them.

Start

Stop

Close

Reset Growth Tracking

Measure Growth

Reset Leak Tracking

Find Leaks

Project Properties...

Command Line...

1. Select Analysis Type

2. Click Start

# Command Line Interface

## ■ Start analysis

- *Memory:* `inspxe-cl -c mi3 -- <app> [app_args]`
- *Threading:* `inspxe-cl -c ti3 -- <app> [app_args]`

## ■ View results

- `inspxe-cl -report=problems -report-all`
- To open result in GUI, type:  
`inspxe-gui <result folder>`

# Command Line Interface – Intel MPI

- Use gtool flag or environment variable:
  - *flag*: `$ mpirun -gtool "inspxe-cl -c mi3 -r <result_dir>:0" -n N <app> [app_args]`
  - *env*: `$ export I_MPI_GTOOL="inspxe-cl -c mi3 -r <result_dir>:0"`
- Gtool inserts tool on selected ranks
  - Analysis only on rank 0: use `":0"`
  - Analysis on selected ranks: use `":m-n"`
  - Analysis on all ranks: use `":all"`
- In most cases it should be sufficient to do analysis on single rank!

# Workflow: manage results

The screenshot displays the Intel Inspector interface for analyzing deadlocks and data races. The main window is titled "Detect Deadlocks and Data Races" and includes tabs for "Target", "Analysis Type", "Collection Log", and "Summary".

**Problems Table:**

ID	Type	File	Module	State
P1	Data race	find_and_fix_threading_errors.cp...	find_and_fix_threading_errors.exe	New
P2	Data race	winvideo.h	find_and_fix_threading_errors.exe	New
	Data race	winvideo.h:270	find_and_fix_threading_errors.exe	New
	Data race	winvideo.h:270	find_and_fix_threading_errors.exe	New
	Data race	winvideo.h:201; winvideo.h:270	find_and_fix_threading_errors.exe	New

**Filters and Sort:**

- Filters: Data race (2)
- Source: find\_and\_fix\_thre... (1), task\_scheduler\_i... (1), winvideo.h (1)
- Module: (empty)

**Code Locations: Data race**

Module: winvideo.h:270 next\_frame find\_and\_fix\_threading\_errors.exe

```
268 {
269     if(!running) return false;
270     g_updates++; // Fast but inaccurate
271     if(!threaded) while(loop_once(thi
272     else if(g_handles[1]) {
```

**Timeline**

- main (4960)
- thread\_video (4672)
- TBB Worker Thread (2848)
- TBB Worker Thread (1724)
- TBB Worker Thread (6004)
- Read: winvideo.h:270
- Write: winvideo.h:270

**Callouts:**

- "Double click on Problem to navigate to source" points to the P2 row in the Problems table.
- "Code locations grouped into Problems to simplify results management" points to the P2 row and its associated code locations.
- "Powerful filtration feature" points to the Filters and Sort pane.



# Workflow: navigate to sources

Data race

Intel Inspector Call stacks

Target Analysis Type Collection Log Summary Sources

Write - Thread TBB Worker Thread (1724) (find\_and\_fix\_threading\_errors.exe!next\_frame - winvideo.h:270)

```
winvideo.h Disassembly (find_and_fix_threading_errors.exe!0x9257) Call Stack
267 bool video::next_frame()
268 {
269     if(!running) return false;
270     g_updates++; // Fast but inaccurate counter. The data race h
271     if(!threaded) while(loop_once(this));
272     else if(g_handles[1]) {
273         SetEvent(g_handles[1]);
274         YIELD_TO_THREAD();
275     }
```

Problematic line in source code

Read - Thread TBB Worker Thread (6004) (find\_and\_fix\_threading\_errors.exe!next\_frame - winvideo.h:270)

```
winvideo.h Disassembly (find_and_fix_threading_errors.exe!0x924e) Call Stack
267 bool video::next_frame()
270     g_updates++; // Fast but inaccurate
271     if(!threaded) while(loop_once(th
272     else if(g_handles[1]) {
273         SetEvent(g_handles[1]);
274         YIELD_TO_THREAD();
275     }
```

All code locations for a problem

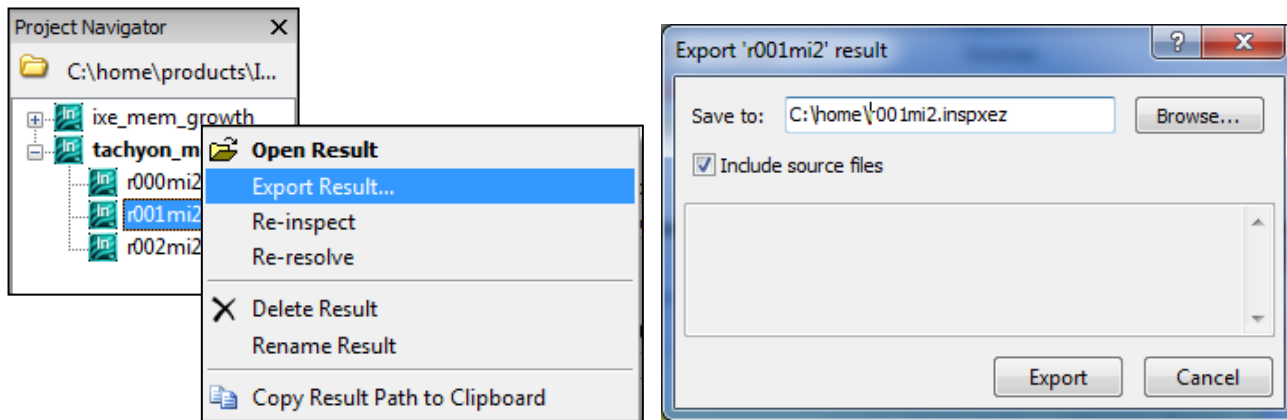
Switch to disassembly for more details

# Exporting results

Save results with sources – copy and browse anywhere  
without setting search paths

CLI: `inspxe-cl -export -archive-name r000mi2.inspxez  
-include-sources -result-dir r000mi2`

GUI:



# Work on remote computer

- Working with GUI on remote system might be not possible
- Result generated by CLI on remote system can be exported to archive (plain file)
- Transfer archive from remote system to local desktop/laptop with local Inspector installation. Linux results can be analyzed by Windows Inspector

# Demo – Hands on

- Try Inspector on DevCloud or local system
- Playbook will provide some sample command lines
- More information:

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/inspector.html>

- Video:

<https://www.intel.com/content/www/us/en/developer/videos/introduction-to-intel-inspector.html>

# Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](https://www.Intel.com/PerformanceIndex).

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

# Backup

Additional information

# Using the Intel® Inspector with MPI (mpich etc)

- 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 inspector_results.<rank#> on Linux
```

# 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)**

```
// Memory leak
```

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

```
// Memory not deallocated
```

```
static char *pStr = malloc(512);  
return;
```

```
// Memory growth
```

```
// Start measuring growth  
static char *pStr = malloc(512);  
// Stop measuring growth
```



# Memory problems

## ■ Uninitialized memory access

- Read of an uninitialized memory location

## ■ Invalid Memory Access

- Read or write instruction references memory that is logically or physically invalid

## ■ Kernel Resource Leak

- Kernel object handle is created but never closed

## ■ GDI Resource Leak

- GDI object is created but never deleted

```
// Uninitialized Memory Access
```

```
void func()
{
    int a;
    int b = a * 4;
}
```

```
// Invalid Memory Access
```

```
char *pStr = (char*) malloc(20);
free(pStr);
strcpy(pStr, "my string");
```

```
// Kernel Resource Leak
```

```
HANDLE hThread = CreateThread(0,
    8192, work0, NULL, 0,
    NULL);
return;
```

```
// GDI Resource Leak
```

```
HPEN pen = CreatePen(0, 0, 0);
return;
```

# Data race

```
CRITICAL_SECTION cs;           // Preparation
int *p = malloc(sizeof(int)); // Allocation Site
*p = 0;
InitializeCriticalSection(&cs);
```

## Write -> Write Data Race

Thread #1

```
*p = 1; // First Write
```

Thread #2

```
EnterCriticalSection(&cs)
;
*p = 2; // Second Write
LeaveCriticalSection(&cs)
;
```

## Read -> Write Data Race

Thread #1

```
int x;
x = *p; // Read
```

Thread #2

```
EnterCriticalSection(&cs)
;
*p = 2; // Write
LeaveCriticalSection(&cs)
;
```

intel®