



INTEL[®] INSPECTOR

Heinrich Bockhorst, Intel

EuroCC Workshop February 17th 2022

Agenda

- Introduction to Inspector
- GUI usage
- Command line usage
- Results
- Offload support (OpenMP and Sycl)
- Next steps

Debug Memory & Threading Errors on native code

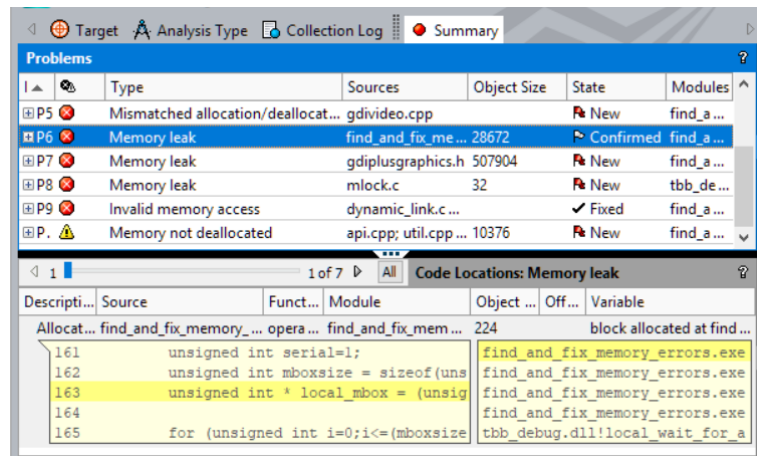
Find and eliminate

- Memory leaks, invalid access...
- Persistent Memory errors
- Races & deadlocks

Simple, Reliable, Accurate

- No special recompiles
- Use any build, any compiler¹
- Analyzes dynamically generated or linked code
- Inspects 3rd party libraries without source
- Command line for automated regression analysis
- Integrated debugger support

¹That follows common OS standards.



Clicking an error instantly displays source code snippets and the call stack

Select analysis and Start

Configure Analysis Type

Analysis Type

Memory Error Analysis

2x-20x
10x-40x
20x-80x

Detect Leaks
Detect Memory Problems
Locate Memory Problems

Analysis Time Overhead

Memory Overhead

Locate Memory Problems

Widest scope memory 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.

- ☒ Detect invalid memory accesses
- ☐ Analyze stack accesses
- ☒ Detect memory leaks upon application exit
- ☒ Enable interactive memory growth detection
- ☒ Enable on-demand memory leak detection

Start

2. Click Start

1. Select Analysis Type

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>`

User Interface Overview

Locate Deadlocks and Data Races

Target Analysis Type Collection Log Summary

Problems

ID	Type	Sources	Modules	State
P1	Data race	memissues_omp.cpp	8ecee7e10398e29e; memissues_omp_exe	New

Filters

Filter	Count
Type	1 item(s)
Source	1 item(s)
Module	1 item(s)
State	1 item(s)

Code Locations: Data race

Description	Source	Function	Module	Variable
Write	memissues_omp.cpp:13	_Z12parallel_sumPjmPm.extracted	8ecee7e10398e29e	0x3842800
<pre>11 // #pragma omp atomic 12 // #pragma omp critical 13 *output += input[i]; // This is a race 14 15 // std::cout << "Hi from thread " << omp_get_thread_num() << "\n"</pre>				
Read	memissues_omp.cpp:13	_Z12parallel_sumPjmPm.extracted	8ecee7e10398e29e	0x3842800
<pre>11 // #pragma omp atomic 12 // #pragma omp critical 13 *output += input[i]; // This is a race 14 15 // std::cout << "Hi from thread " << omp_get_thread_num() << "\n"</pre>				

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

Source & Call Stack details

Source code locations displayed for selected problem

The image shows the Intel Inspector 'Data race' window. It displays two threads involved in a race condition. The top thread is 'Write - Thread TBB Worker Thread (80030) (8ecee7e10398e29e!_Z12parallel_sumPjmPm.extracted - memissues_omp.cpp:13)'. The bottom thread is 'Read - Thread TBB Worker Thread (80080) (8ecee7e10398e29e!_Z12parallel_sumPjmPm.extracted - memissues_omp.cpp:13)'. Both threads are shown with their source code in the left pane and their call stacks in the right pane. The source code for both threads is identical, showing a parallel loop with atomic updates to a shared output variable. Line 13, `*output += input[i]; //This is a race`, is highlighted in both threads, indicating the location of the race condition. The call stacks are empty for both threads.

```
Intel Inspector  
Data race  
Target Analysis Type Collection Log Summary Sources  
Write - Thread TBB Worker Thread (80030) (8ecee7e10398e29e!_Z12parallel_sumPjmPm.extracted - memissues_omp.cpp:13)  
memissues_omp.cpp Disassembly (8ecee7e10398e29e!0x193) Call Stack  
8 #pragma omp parallel for  
9 for(size_t i=0;i<size;i++)  
10 {  
11 //pragma omp atomic  
12 //pragma omp critical  
13 *output += input[i]; //This is a race  
14  
15 //std::cout << "Hi from thread " << omp_get_thread_num() << "\n";  
16 }  
17 }  
Read - Thread TBB Worker Thread (80080) (8ecee7e10398e29e!_Z12parallel_sumPjmPm.extracted - memissues_omp.cpp:13)  
memissues_omp.cpp Disassembly (8ecee7e10398e29e!0x193) Call Stack  
9 #pragma omp parallel for  
10 for(size_t i=0;i<size;i++)  
11 {  
12 //pragma omp atomic  
13 *output += input[i]; //This is a race  
14  
15 //std::cout << "Hi from thread " << omp_get_thread_num() << "\n";  
16 }  
17 }
```

Call Stacks

NEW OFFLOAD FEATURE

Inspector analyses GPU offload code on CPU

Execution targets and analysis scope

Native – Regular application uses known API for threading and memory management.

CPU Offload – code section stored as intermediate representation (SPIR-V) executed on CPU

Native Offload – Code section marked as 'offload' and compiled into native code to execute on host

Analysis scope \ execution target	Native	CPU Offload	Native Offload
Data races	+	+	+
Deadlocks	+	+	+
Locks hierarchy violation	+	-	+
Invalid memory access	+	+	+
Uninitialized memory access	+	+	+
Memory leaks	+	-	+
Kernel and GDI resources leaks	+	-	+
Various allocation/deallocation problems	+	-	+
Memory growth checkpoints	+	-	+

Environment configuration to run kernels on CPU

- Configure DPC++ application to run kernels on CPU device

```
export SYCL_DEVICE_FILTER="opencl:cpu"
```

Check if it works ok before running analysis

- Configure OpenMP application to run kernels on CPU device

```
export OMP_TARGET_OFFLOAD=MANDATORY
```

```
export LIBOMPTARGET_DEVICE_TYPE=cpu
```

Check if it works ok before running analysis

- Disable vectorizer in JIT compiler for more accurate Data Race detection

```
export CL_CONFIG_USE_VECTORIZER=false
```

- Source Inspector vars

```
source /opt/inspxe/inspxe-vars.sh
```

- Choose minimal workload

- Run analysis using either of clients: `inspxe-gui` or `inspxe-cl`

APPLICATION ANALYSIS EXAMPLE

Inspector reports on test applications

Use host pointer on device: source code

```
// ... queue initialization code
```

```
const size_t size = 10;  
uint32_t* array = new uint32_t[size];  
uint64_t* result = new uint64_t;
```

```
// ... array initialization code
```

```
queue.submit([&](cl::sycl::handler &cgh)  
{  
    uint64_t* output = result; // here we use directly local host variable for output  
    cgh.parallel_for<class my_task>(cl::sycl::range<1> { size }, [=](cl::sycl::id<1> idx)  
    {  
        output[0] += array[idx]; // here we use array from host directly  
    });  
});  
queue.wait();
```

DPC++

Use host pointer on device: **execution**

\$./use_host_on_device

```
> RUNNING ON: Intel(R) Xeon(R) Gold 6152 CPU @ 2.10GHz
> Initialize array on host: ptr=0x13a6870
> Result: ptr=0x7ffd9332e2f0
> Starting kernel 1
> Done. Result=0xbcd030
Elapsed: 4.88583 sec
```

\$ inspxe-cl -c mi3 -- ./use_host_on_device

```
Collection started. To stop the collection, either press CTRL-C
or enter from another console window: inspxe-cl -r
/tmp/mtutin/r000mi3 -command stop.
> RUNNING ON: Intel(R) Xeon(R) Gold 6152 CPU @ 2.10GHz
> Initialize array on host: ptr=0x2e3d1c0
> Result: ptr=0x7ffc7a072310
> Starting kernel 1
> Done. Result=0x1a69f00
Elapsed: 116.516 sec
```

83 new problem(s) found

```
1 A host pointer is used on a device problem(s) detected
1 Invalid deallocation problem(s) detected
64 Invalid memory access problem(s) detected
15 Invalid partial memory access problem(s) detected
2 Memory leak problem(s) detected
```

Use host pointer on device: result

The screenshot displays the Intel Inspector interface. The top pane, titled 'Locate Memory Problems', shows a list of 12 memory-related issues. The bottom pane, titled 'Code Locations: A host pointer is used on a device', shows the source code for the first issue, with annotations highlighting specific lines.

ID	Type	Sources	Modules	Object S...	State
P1	A host pointer is used on...	use_host_on_device.cpp	use_host_on_device		Ne.
P2	Memory leak	observer_proxy.cpp	libtbb.so.12	24	Ne.
P3	Invalid memory access	[Unknown]; basic_string.h	ld-linux-x86-64.so.2; use_host_on_device		Ne.
P4	Invalid memory access	[Unknown]; char_traits.h	ld-linux-x86-64.so.2; use_host_on_device		Ne.
P5	Invalid memory access	[Unknown]; cxa_finalize.c	ld-linux-x86-64.so.2; libc.so.6		Ne.
P6	Invalid memory access	[Unknown]; dl-error-skeleton.c; pthread_once.c	ld-linux-x86-64.so.2; libc.so.6; libpthread.so.0		Ne.
P7	Invalid memory access	[Unknown]; dl-error-skeleton.c; new_allocator.h; ...	ld-linux-x86-64.so.2; libc.so.6; libpthread.so.0; use...		Ne.
P8	Invalid memory access	dl-error-skeleton.c; dl-sym.c	libc.so.6		Ne.
P9	Invalid memory access	[Unknown]; dlerror.c	ld-linux-x86-64.so.2; libdl.so.2		Ne.
P10	Invalid memory access	[Unknown]; dlerror.c	ld-linux-x86-64.so.2; libdl.so.2		Ne.
P11	Invalid memory access	[Unknown]; dlerror.c	ld-linux-x86-64.so.2; libdl.so.2		Ne.
P12	Invalid memory access	[Unknown]; dlerror.c	ld-linux-x86-64.so.2; libdl.so.2		Ne.

Code Locations: A host pointer is used on a device

Description	Source	Function	Module	Object Size	Offset	Variable
A host pointer is used on a device	use_host_on_device.cpp:30	main	use_host_on_device			block allocated at use_host_on_device.cpp:19
28	std::cout << "> Starting kernel 1" << std::endl;					use_host_on_device!main - use_host_on_device.cpp:30
29	queue.submit([&](cl::sycl::handler &cgh)					libc.so.6! _libc_start_main - libc-start.c:308
30	{					use_host_on_device!_start
31	uint64_t* output = result; // here we use directly local host variable for output					
32						
Allocation site	use_host_on_device.cpp:20	main	use_host_on_device			block allocated at use_host_on_device.cpp:20
18	const size_t size = 10;					use_host_on_device!main - use_host_on_device.cpp:20
19	uint32_t* array = new uint32_t[size];					libc.so.6! _libc_start_main - libc-start.c:308
20	uint64_t* result = new uint64_t;					use_host_on_device!_start
21						
22	std::cout << "> Initialize array on host: ptr=" << (void*)array << std::endl;					
Allocation site	use_host_on_device.cpp:19	main	use_host_on_device			block allocated at use_host_on_device.cpp:19
17						use_host_on_device!main - use_host_on_device.cpp:19
18	const size_t size = 10;					libc.so.6! _libc_start_main - libc-start.c:308
19	uint32_t* array = new uint32_t[size];					use_host_on_device!_start
20	uint64_t* result = new uint64_t;					
21						

Passing kernel arguments here

Object allocation site

CURRENT OFFLOAD PROBLEM TYPES LISTED IN BACKUP

Remark on problem cases

- Set of correctness problems was build based on our learnings of offload compute runtimes: OpenCL, SYCL/DPC++, OpenMP offload
- Some of the problem were reported by customer support engineers based on real use cases (DPC++ mostly)
- (!) If you think some specific scenario is missing, please let us know

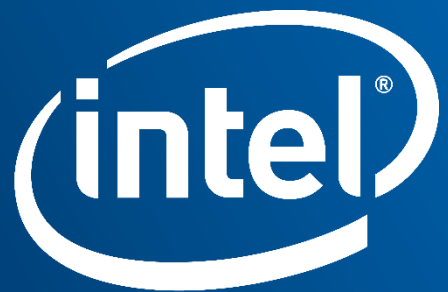
Next steps

Memory analysis

- Improve memory analysis quality – reduce false reports
- Eliminate or suppress errors reported in runtime libraries
- If new kind of memory problem arrive, evaluate if we can detect it

Threading analysis

- Add annotations into SYCL barrier pass to make barriers visible to the tool
=> detect potential deadlocks in SYCL code
- Add annotations into OpenMP runtime to recognize its custom locks
=> avoid false reports in OpenMP kernels



Software

Backup

Additional information

Memory problem types

#	Problem class	Example	Status
1	Memory leak	<pre>clBuff = clCreateBuffer(...) //clReleaseMemObject(clBuff)</pre>	Done
2	Invalid deallocation	<pre>ptr = malloc(...) clBuff = clCreateBuffer(USE_HOST_PTR, ptr, ...) //clReleaseMemObject(clBuff) free(ptr);</pre>	Done
3	Uninitialized read	<pre>ptr = malloc(...) clBuff = clCreateBuffer(USE_HOST_PTR, ptr, ...) read of clBuff (in the kernel)</pre>	Done

Memory problem types

#	Problem class	Example	Status
4	Invalid access	<pre>ptr = malloc(...) clBuff = clCreateBuffer(USE_HOST_PTR, ptr, ...) clReleaseMemObject(clBuff) and/or free(ptr) read\write of clBuff (in the kernel)</pre>	Done
5	Missing allocation	<pre>clReleaseMemObject(ptr) //release invalid handle //here OCL will throw an exception</pre>	Done
6	Invalid arguments	<pre>b1 = clCreateBuffer(NULL, CL_MEM_COPY_HOST_PTR) b2 = clCreateBuffer(random_ptr, CL_MEM_COPY_HOST_PTR)</pre>	Done

Memory problem types

#	Problem class	Example	Status
7	Invalid read/write (undefined behavior)	<pre>ptr = malloc() b = clCreateBuffer(ptr, CL_MEM_USE_HOST_PTR) for(i=0; ...) ptr[i] = i; //undefined behavior // start kernel here</pre>	Not ready yet
8	Invalid buffer region	<pre>uint32_t* array = new uint32_t[1024]; cl::sycl::buffer<uint32_t, 1> inputBuf(array + 512, 1024); queue.submit([&](cl::sycl::handler &cgh) { // ... });</pre>	Done
9	Incorrect API use	<pre>void *deviceArray = sycl::malloc_device(...); memcpy(deviceArray, input, sizeInBytes); // Correct variant should be h.memcpy(...) inside the kernel</pre>	Temporary disabled

Memory problem types

#	Problem class	Example	Status
10	Incorrect pointer type	<pre>queue.submit([&](cl::sycl::handler &cgh) { // here we use host variable for output uint64_t* output = my_result; cgh.parallel_for<...>(..., [=](cl::sycl::id<1> idx) { output[idx] += array[idx]; }); });</pre>	Done
11	Invalid pointer (invalid access)	<pre>queue.submit([&](cl::sycl::handler &cgh) { // here invalid pointer is used as output uint64_t* output = random_ptr; cgh.parallel_for<...>(..., [=](cl::sycl::id<1> idx) { output[idx] += array[idx]; }); });</pre>	Done
12	Invalid kernel argument	Data allocated on one device/queue used in another one	Done

Threading problem types

#	Problem class	Example	Status
1	Data race	<pre>cgh.parallel_for(...){ if(idx.get(0) > 0 && idx.get(0) < ArraySize - 1) { uint32_t s = array2_acc[idx-1] + array2_acc[idx] + array2_acc[idx + 1]; array2_acc[idx] = (s / 3) % ArraySize; } });</pre>	Partially ready

Data race on shared data is reported, but it has limitations:

- DPC++ barriers and OpenMP synchronizations are ignored. Tool will report false races even if work-items are synchronized.
- Data races are not detected on variables defined in kernel local memory.

Will be supported in
upcoming releases

Threading problem types

#	Problem class	Example	Status
2	Potential deadlock	Conditional barrier causes deadlock if only part of the work items reach it <code>size_t local_id = item.get_local_linear_id(); if (local_id < 5) item.barrier(sycl::access::fence_space::global_and_local);</code>	Not ready yet

DPC++ barriers are not intercepted by the tool on CPU target device:

- DPC++ compiler applies code transformations that eliminate barrier 'call'
- Requires code mark-ups to be visible to the tool

Will be supported in
upcoming releases