Sanitizers: Instrumentation for Efficient Debugging

Matthias Kretz

Frankfurt Institute for Advanced Studies
Institute for Computer Science
Goethe University Frankfurt

April 30, 2014

HIC for FAIR
Helmholtz International Center

FIAS Frankfurt Institute for Advanced Studies

HGS-HIRe for FAIR
Helmholtz Graduate School for Hadron and Ion Research

Outline

you know valgrind, right?

Think valgrind, but ...

- faster
- more precise
- detects more/different issues
- switched on at compile time:
  - instrumentation added at compile time
  - not at run time as valgrid does

Existing sanitizers:

Address	memory (stack & heap & globals) error detector
Thread	data race detector
Memory	detector of uninitialized reads
Undefined	undefined behavior checker
...	more (`http://clang.llvm.org/docs/UsersManual.html#controlling-code-generation`)

Experimental but interesting (asan):

use-after-return	accessing local variable after the function exit
use-after-scope	accesing local variable after it went out of scope

- sanitzers developed for clang
- ports to GCC ongoing:
  - 4.8 Address & Thread Sanitizer
  - 4.9 Undefined Behavior Sanitizer

Address Sanitizer

fast memory error detector

- Out-of-bounds accesses to heap, stack and globals
- Use-after-free
- Use-after-return (to some extent)
- Double-free, invalid free
- Memory leaks (experimental)
- slowdown: $2\times$

Thread Sanitizer

- detects data races
- slowdown is about $5\times$–$15\times$
- memory overhead is about $5\times$–$10\times$

Memory Sanitizer

- detector of uninitialized reads
- slowdown is $3\times$

Undefined Behavior Sanitizer

- undefined behavior checker
- small runtime cost and no impact on address space layout or ABI
- finds:
  - misaligned pointer or creation of a misaligned reference
  - **`bool`** value which is neither `true` nor `false`
  - out of bounds array indexing (statically)
  - value of an enumerated type which is not in range
  - conversion to, from, or between floating-point types which would overflow the destination
  - floating point division by zero
  - indirect call of a function through a function pointer of the wrong type
  - integer division by zero
  - use of a null pointer or creation of a null reference
  - ...

Address Sanitizer

```
1  int main(int argc, char **argv)
2  {
3    int *array = new int[100];
4    delete[] array;
5    return array[argc];
6  }
```

```
clang++ -O2 -g -fno-omit-frame-pointer -std=c++11
-fsanitize=address -o asan asan.cpp
```

Address Sanitizer

```
1  ==================================================================
2  ==31582==ERROR: AddressSanitizer: heap-use-after-free on address 0x61400000fe44 at pc 0
       x462f52 bp 0x7fff080b22d0 sp 0x7fff080b22c8
3  READ of size 4 at 0x61400000fe44 thread T0
4      #0 0x462f51 in main /home/mkretz/src/sanitize_examples/asan.cpp:5
5      #1 0x2b6a8bdedec4 in __libc_start_main /build/buildd/eglibc-2.19/csu/libc-start.c:287
6      #2 0x462e35 in _start (/home/mkretz/src/sanitize_examples/asan+0x462e35)
7
8  0x61400000fe44 is located 4 bytes inside of 400-byte region [0x61400000fe40,0x61400000ffd0)
9  freed by thread T0 here:
10     #0 0x44d81e in operator delete[](void*) /home/mkretz/src/llvm/projects/compiler-rt/lib/
           asan/asan_new_delete.cc:85
11     #1 0x462f1e in main /home/mkretz/src/sanitize_examples/asan.cpp:4
12     #2 0x2b6a8bdedec4 in __libc_start_main /build/buildd/eglibc-2.19/csu/libc-start.c:287
13
14 previously allocated by thread T0 here:
15     #0 0x44d51e in operator new[](unsigned long) /home/mkretz/src/llvm/projects/compiler-rt
           /lib/asan/asan_new_delete.cc:54
16     #1 0x462f13 in main /home/mkretz/src/sanitize_examples/asan.cpp:3
17     #2 0x2b6a8bdedec4 in __libc_start_main /build/buildd/eglibc-2.19/csu/libc-start.c:287
18
19 SUMMARY: AddressSanitizer: heap-use-after-free /home/mkretz/src/sanitize_examples/asan.cpp
       :5 main
20 Shadow bytes around the buggy address:
21   0x0c287fff9f70: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
22   0x0c287fff9f80: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
23   0x0c287fff9f90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
24   0x0c287fff9fa0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
25   0x0c287fff9fb0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
26 =>0x0c287fff9fc0: fa fa fa fa fa fa fa fa[fd]fd fd fd fd fd fd fd
27   0x0c287fff9fd0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
```

Address Sanitizer

```
21     0x0c287fff9f70: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
22     0x0c287fff9f80: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
23     0x0c287fff9f90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
24     0x0c287fff9fa0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
25     0x0c287fff9fb0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
26   =>0x0c287fff9fc0: fa fa fa fa fa fa fa fa[fd]fd fd fd fd fd fd fd
27     0x0c287fff9fd0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
28     0x0c287fff9fe0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
29     0x0c287fff9ff0: fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa
30     0x0c287fffa000: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
31     0x0c287fffa010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
32   Shadow byte legend (one shadow byte represents 8 application bytes):
33     Addressable:           00
34     Partially addressable: 01 02 03 04 05 06 07
35     Heap left redzone:     fa
36     Heap right redzone:    fb
37     Freed heap region:     fd
38     Stack left redzone:    f1
39     Stack mid redzone:     f2
40     Stack right redzone:   f3
41     Stack partial redzone: f4
42     Stack after return:    f5
43     Stack use after scope: f8
44     Global redzone:        f9
45     Global init order:     f6
46     Poisoned by user:      f7
47     ASan internal:         fe
48   ==31582==ABORTING
```

Thread Sanitizer

```
1  #include <thread>
2  int Global;
3  int main()
4  {
5    std::thread t([]() {
6      Global = 42;
7    });
8    Global = 43;
9    t.join();
10   return Global;
11 }
```

```
clang++ -O2 -g -fno-omit-frame-pointer -std=c++11
-fsanitize=thread -o tsan tsan.cpp
```

Thread Sanitizer

```
1  ==31585==Re-execing with stack size limited to 33554432 bytes.
2  ==================
3  WARNING: ThreadSanitizer: data race (pid=31585)
4    Write of size 4 at 0x7f8ef22ffca4 by thread T1:
5      #0 operator() /home/mkretz/src/sanitize_examples/tsan.cpp:6 (tsan+0x00000008533d)
6      #1 void std::_Bind_simple<main::$_0 ()>::_M_invoke<>(std::_Index_tuple<>) /usr/lib/gcc/
           x86_64-linux-gnu/4.8/../../../../include/c++/4.8/functional:1731 (tsan+0
           x00000008533d)
7      #2 std::_Bind_simple<main::$_0 ()>::operator()() /usr/lib/gcc/x86_64-linux-gnu
           /4.8/../../../../include/c++/4.8/functional:1720 (tsan+0x00000008533d)
8      #3 std::thread::_Impl<std::_Bind_simple<main::$_0 ()> >::_M_run() /usr/lib/gcc/x86_64-
           linux-gnu/4.8/../../../../include/c++/4.8/thread:115 (tsan+0x00000008533d)
9      #4 <null> <null>:0 (libstdc++.so.6+0x0000000b1bef)
10
11   Previous write of size 4 at 0x7f8ef22ffca4 by main thread:
12     #0 main /home/mkretz/src/sanitize_examples/tsan.cpp:8 (tsan+0x000000084f18)
13
14   Location is global 'Global' of size 4 at 0x7f8ef22ffca4 (tsan+0x0000016caca4)
15
16   Thread T1 (tid=31589, running) created by main thread at:
17     #0 pthread_create /home/mkretz/src/llvm/projects/compiler-rt/lib/tsan/rtl/
           tsan_interceptors.cc:877 (tsan+0x00000001e91b)
18     #1 <null> <null>:0 (libstdc++.so.6+0x0000000b1e3e)
19     #2 __libc_start_main /build/buildd/eglibc-2.19/csu/libc-start.c:287 (libc.so.6+0
           x000000021ec4)
20
21 SUMMARY: ThreadSanitizer: data race /home/mkretz/src/sanitize_examples/tsan.cpp:6 operator
       ()
22 ==================
23 ThreadSanitizer: reported 1 warnings
```


Memory Sanitizer

```
1  #include <cstdio>
2  int main(int argc, char** argv)
3  {
4    int* a = new int[10];
5    a[5] = 0;
6    if (a[argc]) {
7      printf("xx\n");
8    }
9    return 0;
10 }
```

```
clang++ -O2 -g -fno-omit-frame-pointer -std=c++11
-fsanitize=memory -o msan msan.cpp
```

Memory Sanitizer

```
1  ==31590== WARNING: MemorySanitizer: use-of-uninitialized-value
2      #0 0x7f5a7b6b7916 in main /home/mkretz/src/sanitize_examples/msan.cpp:6
3      #1 0x7f5a7a247ec4 in __libc_start_main /build/buildd/eglibc-2.19/csu/libc-start.c:287
4      #2 0x7f5a7b6b775a in _start (/home/mkretz/src/sanitize_examples/msan+0x5c75a)
5
6  SUMMARY: MemorySanitizer: use-of-uninitialized-value /home/mkretz/src/sanitize_examples/
         msan.cpp:6 main
7  Exiting
```

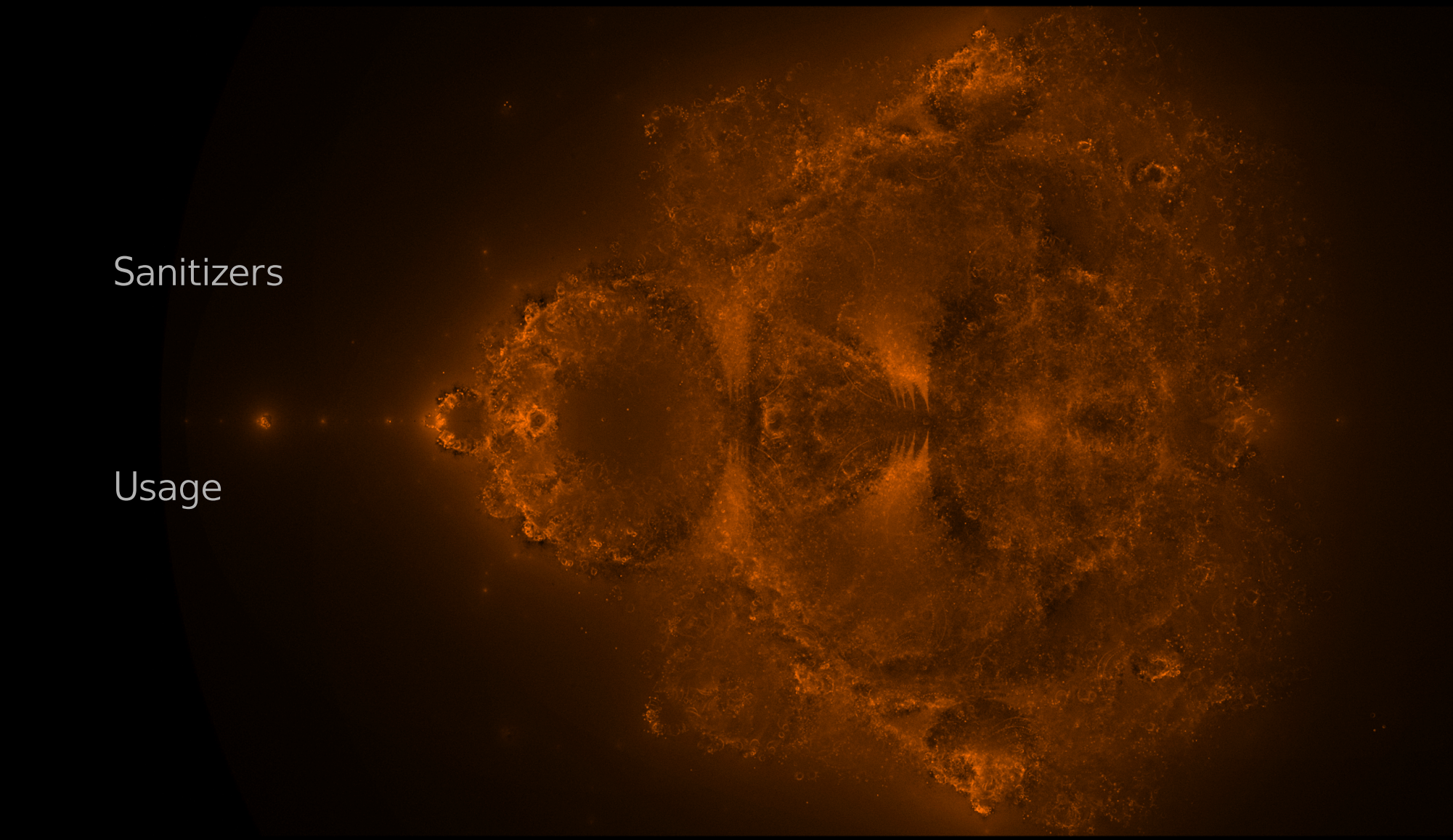
Undefined Behavior Sanitizer

```
1  int main()
2  {
3    float x = 1;
4    float y = 0;
5    x /= y;
6    return x;
7  }
```

```
clang++ -O2 -g -fno-omit-frame-pointer -std=c++11
-fsanitize=undefined -o usan usan.cpp
```

Undefined Behavior Sanitizer

```
1   usan.cpp:5:5: runtime error: division by zero
2   /home/mkretz/src/sanitize_examples/usan.cpp:6: runtime error: value inf is outside the
        range of representable values of type 'int'
```


Sanitizers
Usage